TI FORTH

INSTRUCTION MANUAL

2ND LES EDITION

This document originally prepared in 1983 by

Leslie O'Hagan

Leon Tietz

John T. Yantis

—Edited and Expanded by Lee Stewart (2012 & 2013)

Dedication

This diskette-based Forth Language system for the Texas Instruments TI-99/4A Home Computer was adapted by Leon Tietz and Leslie O'Hagan of the TI Corporate Engineering Center from Ed Ferguson's TMS9900 implementation of the Forth Interest Group (FIG) standard kernel. This system was placed in the public domain "as is" by Texas Instruments on December 21, 1983, by sending one copy of this *TI Forth Instruction Manual* and the TI Forth System diskette to each of the TI-recognized TI-99/4A Home Computer User Groups as of that date. There were no more copies made, and none are available from Texas Instruments. TI Forth had not undergone the testing and evaluation normally given a product which is intended for distribution at the time TI withdrew from the Home Computer market. Although both the diskette and this manual may contain errors and omissions, TI Forth for the TI-99/4A Home Computer ***will not be supported by*** TI in any way, shape, form or fashion. What is contained in this manual and on the accompanying TI Forth System diskette is all that exists of this system, and is its sole reference.

Texas Instruments Incorporated (hereinafter "TI") hereby relinquishes any and all proprietary claims to the software language known as "TI Forth" to the public for free use thereof, without reservations on the part of TI. It should be understood that the TI Forth software language is not subject to any warranties of fitness, either express or implied, by TI, and TI makes no representations as to the fitness of the TI Forth software language for any intended application by the user. Any use of the TI Forth software language is specifically at the discretion of the user who assumes the entire responsibility for such use.

Table of Contents

1 Introduction

The Forth language was invented in 1969 by Charles Moore and has continually gained acceptance. The last several years have shown a dramatic increase in this language's following due to the excellent compatibility between Forth and mini- and microcomputers. Forth is a threaded interpretive language that occupies little memory, yet, maintains an execution speed within a factor of two of assembly language for most applications. It has been used for such diverse applications as radio telescope control to the creation of word processing systems. The Forth Interest Group (FIG) is dedicated to the standardization and proliferation of the Forth language. TI Forth is an extension of the fig-Forth dialect of the language. The fig-Forth language is in the public domain. Nearly every currently available mini- and microcomputer has a Forth system available on it, although some of these are not similar to the FIG version of the language.

The address for the Forth Interest Group is:

Forth Interest Group
P. O. BOX 1105
San Carlos, CA 94070

This document will cover some of the fundamentals of Forth and then show how the language has been extended to provide easy access to the diverse features of the TI-99/4A Computer. The novice Forth programmer is advised to seek additional information from such publications as:

Starting FORTH (1st Ed.)
by Leo Brodie
published by Prentice Hall

Using FORTH
by Forth Inc.

Invitation to FORTH
by Katzan
published by Petrocelli Books

In order to utilize all the capabilities of the TI-99/4A, it is necessary to understand its architecture. It is recommended that the user who wants to use Forth for graphics, music, access to Disk Manager functions or files have a working knowledge of this architecture. This information is available in the *Editor/Assembler Manual* accompanying the Editor/Assembler Command Module. All the capabilities addressed in that document are possible in Forth and most have been provided by easy-to-use Forth words that are documented in this manual.

Forth is designed around a virtual machine with a stack architecture. There are two stacks: The first is referred to variously as the data stack, parameter stack or stack. The second is the return stack. The act of programming in Forth is the act of defining procedures called "words", which are defined in terms of other more basic words. The Forth programmer continues to do this until a single word becomes the application desired. Since a Forth word must exist before it can be referenced, a bottom up programming discipline is enforced. The language is structured and contains no GOTO statements. Successful Forth programming is best achieved by designing top down and programming bottom up.

Bottom-up programming is inconvenient in most languages due to the difficulty in generating drivers to adequately test each of the routines as they are created. This difficulty is so severe that bottom-up programming is usually abandoned. In Forth, however, each routine can be tested interactively from the console and it will execute identically to the environment of being called by another routine. Words take their parameters from the stack and place the results on the stack. To test a word, the programmer can type numbers at the console. These are put on the stack by the Forth system. Typing the word to be tested causes it to be executed and when complete, the stack contents can be examined. By writing only relatively small routines (words) all the boundary conditions of the routine can easily be tested. Once the word is tested (debugged) it can be used confidently in subsequent word definitions.

The Forth stack is 16 bits wide. [*Editor's Note:* In Forth, a 16-bit value is known as a *cell*; hence, the stack is one cell wide.] When multi-precision values are stored on the stack they are always stored with the most significant part most accessible. The width of the return stack is implementation dependent as it must contain addresses so that words can be nested to many levels. The return stack in TI Forth is 16 bits wide.

Disk drives in TI Forth are numbered starting with 0 and are abbreviated with "DR" preceding the drive number: DR0, DR1, etc. Other TI languages (TI BASIC, TI Extended BASIC, TI Assembler, etc.) and software refer to disk drives starting with 1 and the abbreviation "DSK" preceding the disk (drive) number: DSK1, DSK2, etc. From this you can see that DR0 and DSK1 refer to the same disk drive. When referring to the disk drives by device names, they will always be DSK1, …, such as part of a complete file reference, *e.g.*, DSK1.MYFILE.

Keyboard key names in this document will be offset with "<>" and set in the italicized font of the following examples: ***<ENTER>***, ***<CTRL+V>***, ***<FCTN+4>***, ***<BREAK>*** and ***<CLEAR>***. Incidentally, the last three key names listed refer to the same key.

1.1 Editor's Note—1st LES Edition

The source for this document was a series of sixteen files named A, B, C, …, P in TI-Writer format, which I had purchased from the MANNERS (Mid-Atlantic Ninety-NinERS) TI Users Group shortly after TI put TI Forth into the public domain. I do not know who deserves the credit for originating these files; but, it was always my understanding they came from TI and that the printed document we all received with the TI Forth system was prepared in and printed from TI Writer. However, the A – P files have differences from the printed document. I have attempted to correct those differences; but, I have also taken the liberty of elaborating on the original in an effort to make it easier to understand and to correct known bugs. I have added a new chapter, "12 TI Forth Dictionary Entry Structure" and three new appendices, "J TI Forth Bugs", "K Diskette Format Details" and "L TI Forth System for Larger Disks".

Though I have been careful with my additional coding, as with anything else in this document, you assume responsibility for any use you make of it. Please, feel free to contact me with comments and corrections at *lee@stewkitt.com*.

—Lee Stewart
February, 2012
Silver Run, MD

1.2 Editor's Note—2nd LES Edition

This second edition includes numerous corrections to the first edition as well as many additions:

- A note of explanation in § 7.7 on output of exponential format.
- A better explanation of the status ("screen offset") and flag/status bytes in § 8.4 .
- A much expanded Chapter 9 "The TI Forth 9900 Assembler", including § 9.10 that explains how to translate to machine code from assembly code words defined with **CODE** and **;CODE** .
- Some corrections and additions to Appendix G "Assembly Source for CODEd Words".
- More bugs corrected in Appendix J .
- A discussion in a new Appendix M "Notes on Radix-100 Notation" of the notation used for floating-point numbers in the TI-99/4A.
- A new Appendix N "Adding True Lowercase Character Sets".
- Appendix O "TMS9900 Assembly Source Code for TI Forth" annotates the source code and discusses how to build the TI Forth system from scratch.

—Lee Stewart
May, 2013
Silver Run, MD

1.3 Starting Forth

To operate the TI Forth System, you must have the following equipment:

TI-99/4A Console
Monitor
Memory Expansion
Disk Controller
1 (or more) Disk Drives
Editor/Assembler Module
RS232 Interface (optional)
Printer (optional)

See the manuals accompanying each item for proper assembly of the TI-99/4A system.

To begin, power up the system. The TI Color-Bar screen should appear on your monitor. (If it does not, power down and recheck all connections.) Press any key to continue. A new screen will appear displaying a choice between TI BASIC and the Editor/Assembler. To use Forth, select the Editor/Assembler.

On the next screen choose the **LOAD AND RUN** option. The computer will ask for a **FILE NAME**. After placing your TI Forth System disk in the first drive, type "DSK1.FORTH" and press ***<ENTER>***.

The TI Forth welcome screen will display a list of load options (or elective blocks). Each option loads all routines necessary to perform a particular group of tasks:

Load Option	Loads Forth Words Necessary to:	Chapter
-SYNONYMS	Perform VDP reads and writes. Random number generators and the disk formatting routine are also loaded.	5
-EDITOR	Run the regular, 40-column TI Forth editor.	3
-COPY	Copy Forth screens[1] and Forth disks. String store routines are also loaded.	5
-DUMP	Execute **DUMP** and **VLIST**.	5
-TRACE	Trace the execution of Forth words.	5
-FLOAT	Use floating-point arithmetic.	7
-VDPMODES	Change display screen to any of the 6 available VDP modes.	6
-TEXT	Change display screen to Text mode.	6
-GRAPH1	Change display screen to Graphics mode.	6
-MULTI	Change display screen to Multicolor mode.	6
-GRAPH2	Change display screen to Graphics2 (bit-map) mode.	6
-SPLIT	Change display screen to either of the two Split modes.	6
-FILE	Utilize the file I/O capabilities of the TI-99/4A.	8
-PRINT	Send output to an RS232 device.	8
-64SUPPORT	Run the 64-column TI Forth editor.	3
-CODE	Write assembly code in hexadecimal.	9
-ASSEMBLER	Write routines in TI Forth Assembler.	9
-GRAPH	Utilize the graphics capabilities of the TI-99/4A.	6
-BSAVE	Save dictionary overlays to diskette.	11
-CRU	Access the Forth equivalents of TI-Assembler mnemonics: LDCR, STCR, SBO, SBZ and TB.	11

To load a particular package, simply type its name exactly as it appears in the list. For example, to load the graphics package, type **-GRAPH** and press ***<ENTER>***. You may load more than one package at a time.

The list of load options may be displayed at any time by typing the word **MENU** and pressing ***<ENTER>***. See Appendix F for a detailed list of what each option loads.

1 A Forth screen is also called a block and consists of 16 lines of 64 characters for a total of 1024 characters. When a Forth screen is loaded from disk, 1024 characters are copied from the disk into a VDP RAM buffer. This is explained in more detail later in this document.

2 Getting Started

This chapter will familiarize you with the most common words (instructions) in the Forth Interest Group version of Forth (fig-Forth). The purpose is to permit those users that have at least an elementary knowledge of some Forth dialect to easily begin to use TI Forth. Those with no Forth experience should begin by reading a book such as *Starting FORTH, (1st Ed.)* by Leo Brodie. Appendix C is designed to be used with this particular text and lists differences between the Forth language described in the book (poly-Forth) and TI Forth.

A word in Forth is any sequence of characters delimited by blanks or a carriage return (***<ENTER>***). In this document, all Forth words will be set in a bold mono-spaced font that distinguishes the digit '**0**' from the capital letter '**O**' and will always be followed by a blank, even when punctuation such as a period or a comma follows. For example, **DUP** is such a Forth word and is shown also at the end of this sentence to demonstrate this practice: **DUP** . This obviously looks odd; but, this notation is necessary to avoid ambiguity when discussing Forth words because many of them either end in or, in fact, are such punctuation marks themselves. For example, the following, space-delimited character strings are all Forth words:

```
. : , ' ! ; C, C! ;CODE ? ." !"
```

The following convention will be used when referring to the stack in Forth:

(n_1 n_2 --- n_3)

This diagram shows the stack contents before and after the execution of a word. In this case the stack contains two values, n_1 and n_2, before execution of a word. The execution is denoted by "---" and the stack contents after execution is n_3. The most accessible stack element is always on the right. In this example, n_2 is more accessible than n_1. There may be values on the stack that are less accessible than n_1 but these are unaffected by the execution of the word in question.

The return stack may also be indicated beside the parameter stack (the stack) with a preceding "R:", especially when both stacks are involved, as follows:

(*n* ---) (R: --- *n*)

In addition, the following symbols are used as operands for clarity:

SOME SYMBOLS USED IN THIS DOCUMENT

n, n_1, ...	16-bit signed numbers
d, d_1, ...	32-bit signed double numbers
u	16-bit unsigned number
ud	32-bit unsigned double number
addr, $addr_1$, ...	memory addresses
b	8-bit byte (in right half of word)
c	7-bit character (in right end of word)
flag	Boolean flag (0 = false, non-0 = true)
|	separates alternate results

2.1 Stack Manipulation

The following are the most common stack manipulation words:

DUP	(n --- n n)	Duplicate top of stack
DROP	(n ---)	Discard top of stack
SWAP	(n_1 n_2 --- n_2 n_1)	Exchange top two stack items
OVER	(n_1 n_2 --- n_1 n_2 n_1)	Make copy of second item on top
ROT	(n_1 n_2 n_3 --- n_2 n_3 n_1)	Rotate third item to top
-DUP	(n --- n n | n)	Duplicate only if non-zero
>R[2]	(n ---) (R: --- n)	Move top item on stack to return stack
R>	(--- n) (R: n ---)	Move top item on return stack to stack
R	(--- n) (R: n --- n)	Copy top item of return stack to stack

2.2 Arithmetic and Logical Operations

The following are the most common arithmetic and logical operations:

+	(n_1 n_2 --- n_3)	Add
D+	(d_1 d_2 --- d_3)	Add double precision numbers
-	(n_1 n_2 --- n_3)	Subtract (n_1 - n_2)
1+	(n_1 --- n_2)	Increment by 1
2+	(n_1 --- n_2)	Increment by 2
1-	(n_1 --- n_2)	Decrement by 1
2-	(n_1 --- n_2)	Decrement by 2
*****	(n_1 n_2 --- n_3)	Multiply
/	(n_1 n_2 --- n_3)	Divide (n_1 / n_2)
MOD	(n_1 n_2 --- n_3)	Modulo (remainder from n_1 / n_2)
/MOD	(n_1 n_2 --- *rem quot*)	Divide giving remainder & quotient
***/MOD**	(n_1 n_2 n_3 --- *rem quot*)	n_1 * n_2 / n_3 with 32 bit intermediate
***/**	(n_1 n_2 n_3 --- n_4)	Like ***/MOD** but giving *quot* only

2 **>R** and **R>** must be used with caution as they may interfere with the normal address stacking mechanism of Forth. Make sure that each **>R** in your program has an **R>** to match it in the same word definition.

U*	($ud_1\ u_1$ --- ud_2)	Unsigned * with double product
U/	($u_1\ u_2$ --- *urem uquot*)	Unsigned **/** with remainder
MAX	($n_1\ n_2$ --- $n_1 \mid n_2$)	Maximum
MIN	($n_1\ n_2$ --- $n_1 \mid n_2$)	Minimum
ABS	(n --- $\|n\|$)	Absolute value
DABS	(d --- $\|d\|$)	Absolute value of 32-bit number
MINUS	(n_1 --- n_2)	Leave two's complement
DMINUS	(d_1 --- d_2)	Leave two's complement of 32-bits
AND	($n_1\ n_2$ --- n_3)	Bitwise logical AND n_3
OR	($n_1\ n_2$ --- n_3)	Bitwise logical OR n_3
XOR	($n_1\ n_2$ --- n_3)	Bitwise logical exclusive OR n_3
SWPB	(n_1 --- n_2)	Swap the bytes of n_1 producing n_2
SRC	($n_1\ n_2$ --- n_3)	Shift n_1 right circular n_2 bits giving n_3
SRL	($n_1\ n_2$ --- n_3)	Shift n_1 right logical n_2 bits giving n_3
SRA	($n_1\ n_2$ --- n_3)	Shift n_1 right arithmetic n_2 bits giving n_3
SLA	($n_1\ n_2$ --- n_3)	Shift n_1 left arithmetic n_2 bits giving n_3

2.3 Comparison Operations

The following are the most common comparisons:

<	($n_1\ n_2$ --- *flag*)	True if n_1 less than n_2 (signed)
=	($n_1\ n_2$ --- *flag*)	True if top two numbers are equal
>	($n_1\ n_2$ --- *flag*)	True if n_1 greater than n_2
0<	(n --- *flag*)	True if top number is negative
0=	(n --- *flag*)	True if top number is 0 (*i.e.* NOT)
U<	($u_1\ u_2$ --- *flag*)	Unsigned integer compare

2.4 Memory Access Operations

The following operations are used to inspect and modify memory locations anywhere in the computer:

@	(*addr* --- *n*)	Replace word address by its contents
!	(*n addr* ---)	Store *n* at address (store a word)

C@	(*addr* --- *b*)	Fetch the byte at *addr*
C!	(*b addr* ---)	Store *b* at address (store a byte)
?	(*addr* ---)	Print the contents of address
+!	(*n addr* ---)	Add *n* to contents of address
CMOVE	(*from_addr to_addr u* ---)	Block move *u* bytes.
FILL	(*addr u b* ---)	Fill *u* bytes with *b* beginning at *addr*
ERASE	(*addr u* ---)	Fill *u* bytes beginning at *addr* with 0s
BLANKS	(*addr u* ---)	Fill *u* bytes with blanks beginning at *addr*

2.5 Control Structures

The following sets of words are used to implement control structures in Forth. They are used to create all looping and conditional structures. These structures may be nested to any depth. If they are nested improperly an error message will be generated at compile time and the word definition will be aborted.

DO … LOOP		**DO** sets up a loop with a loop counter. The stack contains the first and final values of the loop counter. The loop is executed at least once. **LOOP** causes a return to the word following **DO** unless termination is reached.
DO	(*end*+1 *start* ---)	
I	(--- *n*)	Used between **DO** and **LOOP**. Places value of loop counter on stack.
J	(--- *n*)	Used when **DO LOOP**s are nested. Places value of next outer loop counter on the stack.
LEAVE	(---)	Causes loop to terminate at next **LOOP** or **+LOOP**.
DO … +LOOP		**DO** as above. **+LOOP** adds top stack value to loop counter (index)
DO	(*end*+1 *start* ---)	
+LOOP	(*n* ---)	
IF … ENDIF		IF tests the top of stack and if non-zero (true), the words between **IF** and **ENDIF** are executed. Otherwise, they are skipped and execution resumes after **ENDIF**.
IF	(*flag* ---)	
IF … ELSE … ENDIF		**IF** tests the top of stack and if non-zero (true), the words between **IF** and **ELSE** are executed. If the top of the stack is zero (false), the words between **ELSE** and **ENDIF** are executed. Execution then continues after **ENDIF** .
IF	(*flag* ---)	
THEN		May be used as a synonym for **ENDIF** .
BEGIN … UNTIL		Loop which executes the words between **BEGIN** and

UNTIL (*flag* ---)	**UNTIL** until the top of stack when tested by **UNTIL** is non-zero (true).
END	May be used as a synonym for **UNTIL** .
BEGIN … AGAIN	Creates an infinite loop continually re-executing the words between **BEGIN** and **AGAIN**[3].
BEGIN … WHILE … REPEAT **WHILE** (*flag* ---)	Executes words between **BEGIN** and **WHILE** leaving *flag* which is tested by **WHILE**. If *flag* is non-zero (true), executes words between **WHILE** and **REPEAT** , then jumps back to **BEGIN**. If *flag* is zero (false), continues execution after the **REPEAT** .
CASE n_1 **OF … ENDOF** n_2 **OF … ENDOF** **…** n_m **OF … ENDOF** **…** **ENDCASE** **CASE** (*n* ---)	Looks for a number (n_1, n_2, …, n_m) matching *n*. If there is a match, executes the code between the **OF … ENDOF** set that immediately follows the matching number, proceeding then to the code following **ENDCASE** . If there is no match, the code after the last **ENDOF** is executed, with **ENDCASE** dropping *n* from the stack. Execution then continues after **ENDCASE** . Code after the last **ENDOF** may use *n*, which is still available; but, it must not consume *n*. Otherwise, **ENDCASE** will drop whatever was under *n*, adversely affecting program logic and possibly causing a stack underflow.

2.6 Input and Output to/from the Terminal

The most common type of terminal input is simply to enter a number at the terminal. This number will be placed on the stack. The number which is input will be converted according to the number base stored at **BASE** . **BASE** is also used during numeric output.

DECIMAL	(---)	Sets the base to Decimal (Base 10)
HEX	(---)	Sets the base to Hexadecimal (Base 16)
BASE	(--- *addr*)	System variable containing number base. To set some base (*e.g.*, Octal) use the following sequence: **8 BASE !**
.	(*n* ---)	Print a signed number
U.	(*u* ---)	Print an unsigned number
.R	(n_1 n_2 ---)	Print n_1 right-justified in field of width n_2
D.	(*d* ---)	Print double-precision number
D.R	(*d n* ---)	Print double-precision number right-justified in field of width *n*
CR	(---)	Perform a Carriage Return/Line Feed

3 This loop may be exited by executing **R> DROP** one level below.

SPACE	(---)	Type 1 space
SPACES	(*n* ---)	Type *n* spaces
."	(---)	Print a string terminated by **"**
TYPE	(*addr n* ---)	Type *n* characters from *addr* to terminal
COUNT	(*addr* --- *addr*+1 *n*)	Move string length from *addr* to stack
?TERMINAL	(--- *flag*)	Test if ***<BREAK>*** (***<CLEAR>*** on TI-99/4A) pressed
?KEY	(--- *n*)	Read keyboard. If no key pressed, *n* = 0 else *n* = ASCII keycode.
KEY	(--- *c*)	Wait for a keystroke and put its ASCII value on the stack.
EMIT	(*c* ---)	Type character from stack to terminal
EXPECT	(*addr n* ---)	Read *n* characters (or until **CR**) from terminal to *addr*
WORD	(*c* ---)	Read one word from input stream delimited by *c*

2.7 Numeric Formatting

Advanced numeric formatting control is possible with the following words:

NUMBER	(*addr* --- *d*)	Convert string at *addr* to *d* number
<#	(---)	Start output string conversion
#	(d_1 --- d_2)	Convert next, least-significant digit of d_1 leaving d_2
#S	(*d* --- 0 0)	Convert all significant digits from right to left
SIGN	(*n d* --- *d*)	Insert sign of *n* into number
#>	(*d* --- *addr u*)	Terminate conversion, ready for **TYPE**
HOLD	(*c* ---)	Insert ASCII character *c* into string

2.8 Disk-Related Words

The following words assist in maintaining source code on disk as well as implementing the Forth virtual memory capability:

LIST	(*n* ---)	List screen *n* to terminal
LOAD	(*n* ---)	Compile or execute screen *n*
BLOCK	(*n* --- *addr*)	Leave address of block *n*, reading it from disk if necessary
B/BUF	(--- *n*)	Constant giving disk block size in bytes
BLK	(--- *addr*)	User variable containing current block number (contains 0 for terminal input)

SCR	(--- *addr*)	User variable containing screen number most recently referenced by **LIST** or **EDIT**
UPDATE	(---)	Mark last buffer accessed as updated
FLUSH	(---)	Write all updated buffers to disk
EMPTY-BUFFERS	(---)	Erase all buffers

2.9 Defining Words

The following are defining words. They are used not only to create new Forth words but in the case of **<BUILDS … DOES>** and **<BUILDS … ;CODE** to create new defining words.

: xxx	(---)	Begin colon definition of **xxx**
;	(---)	End colon definition
VARIABLE xxx	(*n* ---)	Create variable with initial value *n*
xxx	(--- *addr*)	Returns address when executed
CONSTANT xxx	(*n* ---)	Create constant with value *n*
xxx	(--- *n*)	Returns *n* when executed
CODE xxx	(---)	Begin definition of assembly language primitive named **xxx**
<BUILDS … ;CODE		Create new defining word with execution-time assembly/machine code routine
<BUILDS … DOES>		Create new defining word with execution-time high level Forth routine

2.10 Miscellaneous Words

The following words are relatively common but don't fit well in any of the above categories:

CONTEXT	(--- *addr*)	Leave address of pointer to context vocabulary (searched first)
CURRENT	(--- *addr*)	Leave address of pointer to current vocabulary (new definitions placed there)
FORTH	(---)	Set **CONTEXT** to main Forth vocabulary
DEFINITIONS	(---)	Set **CURRENT** to **CONTEXT**
VOCABULARY xxx	(---)	Define new vocabulary
(	(---)	Begin comment. Terminated by **)**
FORGET xxx	(---)	Forget all definitions back to and including **xxx**
ABORT	(---)	Error termination

' xxx	(--- *addr*)	Leave address of **xxx** . If compiling compile address. (tick)
HERE	(--- *addr*)	Leaves address of next unused byte in the dictionary
PAD	(--- *addr*)	Leaves address of scratch area
IN	(--- *addr*)	User variable containing offset into input buffer
SP@	(--- *addr*)	Leaves address of top stack item
ALLOT	(*n* ---)	Leave *n*-byte gap in dictionary
,	(*n* ---)	Compile *n* into the dictionary (comma)

Several Forth screens on the TI Forth System disk serve special purposes. Forth screen 0 must not be modified because it is used by the disk Device Service Routine (DSR) to locate the object code of the Forth kernel. Forth screen 3 is the **BOOT** screen (see **BOOT** in Appendix D), and Forth screens 4 and 5 contain error messages used by several Forth words. Any disk placed in drive 0 (DR0) must contain a copy of Forth screens 4 and 5.

Many additional words are available in TI Forth. The user should consult the remaining chapters in this manual as well as the glossary (Appendix D) and Appendix F for a complete description. Most of these words are disk-resident and must be loaded by the user via the load options, which are viewable by typing **MENU** , before they become available.

3 How to Use the Forth Editor

Words introduced in this chapter:

CLEAR	**FLUSH**
ED@	**TEXT**
EDIT	**WHERE**

In the Forth language, programs are divided into screens or blocks. Each Forth screen is 16 lines of 64 characters and has a number associated with it. A TI-99/4A disk holds 90 Forth screens (numbered 0 – 89), however, Forth screen 0 is special and is usually not used. A program may occupy as many Forth screens as necessary.

You must read Chapter 5, "System Synonyms and Miscellaneous Utilities" and correctly format your data disk before using the editor. Disks initialized by the disk manager are acceptable. After loading Forth from the System disk, place the System disk in DR1 (2nd drive) and your Forth disk in DR0 (1st drive). It is necessary to copy Forth screens 4 and 5 from the Forth System disk onto your Forth disk. These screens contain the error messages. If you have a two-drive system, see the instructions for **SCOPY** and **SMOVE** in Chapter 5 for directions on how to do this.

If you have a one-drive system, however, this procedure is more complicated. The following diagram illustrates the process used to copy parts of a Forth disk or an entire Forth disk with a one drive system.

START: With original diskette in your drive and type:

```
FLUSH
```

LOOP: Type these lines:

```
scr BLOCK DROP UPDATE
  •
  •
  •
scr BLOCK DROP UPDATE
```

} up to 5 screens because the system has 5 disk buffers

Switch to backup diskette and type:

```
FLUSH
```

Go back to LOOP if you need to copy more screens.

Now you are ready to begin editing your Forth disk.

CAUTION: Do *not* edit your System disk. It is a hybrid disk containing both TI-99/4A files and Forth screens. Editing the disk may destroy its integrity!

3.1 Forth Screen Layout Caveat

As indicated above, Forth screens are laid out in 16 lines of 64 characters each. However, you should be aware that the lines have no actual delimiters, *i.e.*, there are no carriage-return or line-feed characters at the end of a Forth-screen line. This means that one line wraps around to the next line with no intervening white-space such that a word ending on one line will be concatenated with a word that starts on the next line if there is no intervening space. This will usually be nonsense to the system and generate an error message when the screen is loaded, indicating that the unintended word has not been defined. Worse, it can result in an unintended existing word such as **-DUP** instead of **- DUP** or **+LOOP** instead of **+ LOOP** .

3.2 The Two TI Forth Editors

There are two Forth editors available on the TI Forth System disk. The first, which is loaded by **-EDITOR**, operates in **TEXT** mode. It will be referred to as the 40-column editor[4]. Each Forth screen is displayed in two halves (left and right) in normal sized characters.

The second, which is loaded by **-64SUPPORT**, operates in bit map mode. It allows you to view an entire Forth screen at once; however, the characters are very small. It will be referred to as the 64-column editor.

Only one editor may be in memory at any time. Load whichever you prefer. Editing instructions are identical for each.

3.3 Editing Instructions

Initialization fills each Forth screen with non-printable characters. These characters appear as solid white squares on the terminal when you are using the 40-column editor and as unidentifiable characters in the 64-column editor. A Forth screen must be filled with blanks before it can be used. Typing a Forth screen number and **CLEAR** will fill a Forth screen with blanks.

```
1 CLEAR
```

will prepare Forth screen 1 for use by the editor.

You may begin writing on Forth screen 1 or on any Forth screen you wish. To bring a Forth screen from the disk into the editor, type the Forth screen number followed by the word **EDIT**.

```
1 EDIT
```

The above instruction will bring the contents of Forth screen 1 into view. If you did not **CLEAR** the screen before entering the editor, the screen will appear to be a block of undefined characters. You must exit the editor temporarily and clear the screen on the disk before you can write to it. To exit the editor, press the ***<BACK>*** (***<FCTN+9>***) function key on your keyboard. To clear the screen, type the screen number and the word **CLEAR** .

To re-enter the editor, you do *not* have to type **1 EDIT** again. A special Forth word,

```
ED@
```

4 The 40-column Forth editor may only be used when the computer is in **TEXT** mode (see Chapter 6). For example, if the 40-column editor is loaded, don't type **EDIT** while you are in **SPLIT** or **SPLIT2** mode.

will return you to the last screen you were editing.

Upon entering the editor, the cursor is located in column 1 of line 0. It is customary to use line 0 for a comment describing the contents of that screen. Type a comment that says "**PRACTICE SCREEN**" or something to that effect. Do not forget that all comments must begin with a "**(** "[5] and end with a "**)**".

If you are using the 40-column editor, you have probably noticed that only 35 columns (of the 64 available columns) are visible on your terminal. To see the rest of the screen, type any characters on line 1 until you reach the right margin. Now type a few more characters. Notice that the screen is now displaying columns 30 – 64. Press ***<ENTER>*** to move to the beginning of the next line.

The function keys on your keyboard each perform a special editing function.

key	function
<FCTN+S>, (←)	moves the cursor one position to the left.
<FCTN+D>, (→)	moves the cursor one position to the right.
<FCTN+E>, (↑)	moves the cursor up one position.
<FCTN+X>, (↓)	moves the cursor down one position.
<DELETE> (***<FCTN+1>***)	deletes the character on which the cursor is placed.
<INSERT> (***<FCTN+2>***)	inserts a space to the left of the cursor moving the rest of the line right one space. Characters may be lost off the end of the line.
<AID> (***<FCTN+7>***)	erases from the cursor to the end of a line and saves the erased characters in **PAD**. They may be placed at the beginning of a new line by pressing ***<REDO>***. ***<REDO>*** inserts a line just above where the cursor is and places the contents of **PAD** there.
<BEGIN> (***<FCTN+5>***)	**40-column editor:** moves the cursor 29 positions to the right if the cursor is on the left half of a Forth screen. Otherwise, it moves the cursor 29 positions to the left. This key can be used to toggle between the left and right half of a screen. **64-column editor:** places the cursor in the upper left corner
<ERASE> (***<FCTN+3>***) ***<REDO>*** (***<FCTN+8>***)	are used in combination to pick up lines and move them elsewhere on the screen. ***<ERASE>*** picks up one line while erasing it from view. ***<REDO>*** inserts this line just above the line on which the cursor is placed. Both ERASE and ***<REDO>*** may be used repeatedly to erase several lines from view or to insert multiple copies of a line.
<CTRL+8>	will insert a blank line just above the line the cursor is on.
<CTRL+V>	will tab forward by words.
<FCTN+V>	will tab backwards by words.

5 The left parenthesis *must* be followed by at least 1 space. Press ***<ENTER>*** to move to the next line.

Experiment with these features until you feel you understand each of their functions. Erase the line you typed from the screen and type a sample program for practice.

The Forth editor allows you to move forward or backward a screen without leaving the editor. Pressing ***<CLEAR>*** (***<FCTN+4>***) will read in the succeeding screen. Pressing ***<PROCEED>*** ***(<FCTN+6>)*** will read in the preceding screen.

If an error occurs during a **LOAD** command, typing the word **WHERE** will bring you back into the editor and place the cursor at the exact point the error occurred.

The word **FLUSH** is used to force the disk buffers that contain data no longer consistent with the copy on disk to be written to the disk. Use this word at the end of an editing session to be certain your changes are written to the disk.

One last note about Forth screens: Though your word definitions can span more than one screen, you should try to insure that any given word is defined on a single screen. This aids in clarity and the good Forth-programming practice of keeping word definitions short.

3.4 Changing Foreground/Background Colors of 64-Col Editor

The white-on-black color scheme of the 64-column editor can be changed to whatever foreground/background pair you would like by changing system screen 54, where **GRAPHICS2** is defined. A more pleasant combination is black on gray. To effect that, change the color table fill value **0F0** (white on transparent) on line 6 to **010** (black on transparent) and **0F1** (white on black) on line 13 to **0FE** (white on gray)—the left byte doesn't matter except in text mode.

You can also change the default colors for text mode to something other than white on dark blue when typing **TEXT** after leaving the 64-column editor by changing **0F4** on line 9 of system screen 51 to another color pair, with the foreground color in the left byte and the background color in the right byte, *e.g.*, **01E** for black on gray.

4 Memory Maps

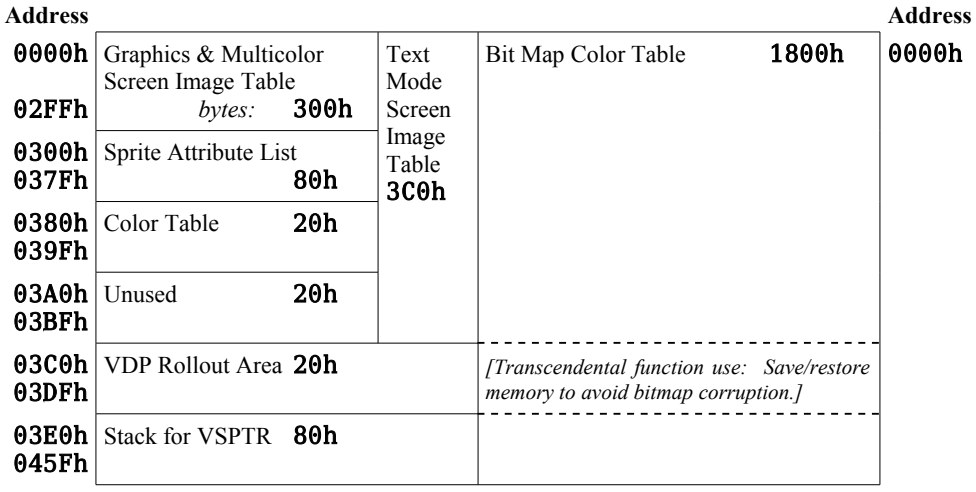
The following diagrams illustrate the memory allocation in the TI-99/4A system. For more detailed information, see the *Editor/Assembler Manual*.[6]

The VDP memory can be configured in many ways by the user. The TI Forth system provides the ability to set up this memory for each of the VDP's 4 modes of operation (Text, Graphics, Multicolor and Graphics2). The allocation of memory for these modes is shown on the VDP Memory Map. The first three modes are shown on the left half of the figure, the Graphics2 mode on the right half. The area at **03C0h** is used by the transcendental functions in all modes for a rollout area. If transcendentals are used during Graphics2 (bit-map) mode, this portion of the color table must be saved by the user before using the transcendental function and restored afterward. Note that the VDP RAM is accessed from the 9900 only through a memory mapped port and is not directly in the processor's address space.

The only CPU RAM on a true 16-bit data bus is in the console at **8300h**. Because this is the fastest RAM in the system, the Forth Workspace and the most frequently executed code of the interpreter are placed in this area to maximize the speed of the TI Forth system. The use of the remainder of the RAM in this area is dictated by the TI-99/4A's resident operating system.

The 32K byte memory expansion is divided into an 8K piece at **2000h** and a 24K piece at **A000h**. The small piece contains BIOS and utility support for TI Forth as well as 5K of disk buffers, the Return Stack, and the User Variable area. The large piece of this RAM contains the dictionary, the Parameter Stack and the Terminal Input Buffer.

4.1 VDP Memory Map



6 Hexadecimal (base 16) notation for integers in this manual is indicated when a string of 1 – 4 hexadecimal digits (**0** – **9**, **A** – **F**) is followed by '**h**'. For example, **2F0Eh** is a hexadecimal integer equivalent in value to decimal integer 12046 and **Ah** is decimal 10. The '**h**' is never typed into the Forth terminal or on Forth screens. It is used in this manual only to avoid confusion. The notation used in the *Editor/Assembler Manual* (use of a preceding '>' instead of a trailing '**h**') is only used in Chapter 9 for the conventional assembler examples, where it is required as input to the Editor/Assembler module.

Address				Address
0460h 077Fh	PABS etc.	320h		
0780h 07FFh	Sprite Motion Table	80h		
0800h 0BFFh	Pattern Descriptor Table Sprite Descriptor Table 0 – 127	 400h		
0C00h 0FFFh	128 – 255	400h		
1000h 13FFh	Forth's Disk Buffer (4 sectors)	 400h		
1400h	Unused	21D8h		17FFh
			Bit Map Screen Image Table 300h	1800h 1AFFh
			PABS etc. C0h	1B00h
			Stack for VSPTR 40h	1BFFh
			Forth's Disk Buffer (4 sectors) 400h	1C00h 1FFFh
35D7h			Bit Map Pattern Descriptor Table 1800h	2000h 37FFh
35D8h	Disk Buffering Region for 3 Simultaneous Disk Files	A28h	Sprite Attribute List 80h	3800h 387Fh
			Sprite Attribute Descriptors (Optional & based at 3800h) 15Ah	3880h 39D9h
3FFFh			Disk Buffer Region for 2 Disk Files 626h	39DAh 3FFFh

4.2 CPU Memory

Address	
0000h 1FFFh	Console ROM
2000h 3FFFh	Low Memory Expansion Loader, Your Program, REF/DEF Table
4000h 5FFFh	Peripheral ROMs for DSRs
6000h 7FFFh	Unavailable—ROM in Command Modules
8000h 9FFFh	Memory-mapped Devices for VDP, GROM, SOUND, SPEECH. CPU RAM at 8300h – 83FFh
A000h FFFFh	High Memory Expansion Your Program

4.3 CPU RAM Pad

Address[7]	
8300h 831Fh	Forth's Workspace
8320h 832Dh	–FREE– Eh
832Eh 8347h	Forth's Inner Interpreter, etc.
8348h 8349h	–FREE– 2
834Ah 8351h	FAC (Floating Point Accumulator)
8354h	Floating Point Error
8355h	Floating Point String↔Number Conversion Options
8356h 8357h	Subroutine Pointer for DSRs (use these 3 bytes)
835Ch 8363h	ARG (Floating Point Argument Register)
836Eh 836Fh	VSPTR (Value Stack Pointer)
8370h 8371h	Highest Available Address of VDP RAM
8372h	Least Significant Byte of Data Stack Pointer
8373h	Least Significant Byte of Subroutine Stack Pointer
8374h	Keyboard Number to be Scanned
8375h	ASCII Keycode Detected by Scan Routine
8376h	Joystick Y-status
8377h	Joystick X-status
8379h	VDP Interrupt Timer
837Ah	Number of Sprites that can be in Automotion
837Bh	VDP Status Byte Bit 0[8] On during VDP Interrupt Bit 1 On when 5 Sprites on a Line Bit 2 On when Sprite Coincidence Bits 3-7 Number of 5th Sprite on a Line
837Ch	GPL Status Byte Bit 0 High Bit Bit 1 Greater than Bit Bit 2 On when Keystroke Detected (COND) Bit 3 Carry Bit Bit 4 Overflow Bit
837Dh	VDP Character Buffer
837Eh	Current Screen Row Pointer
837Fh	Current Screen Column Pointer
8380h	Default Subroutine Stack
83A0h	Default Data Stack

7 Locations omitted are not used by Forth, but may be used by system routines.

8 Bit 0 = high order bit.

Address	
83C0h	Random Number Seed (Begin Interrupt Workspace)
83C2h	Flag Bit 0 Disable All of the Following Bit 1 Disable Sprite Motion Bit 2 Disable Auto Sound Bit 3 Disable System Reset Key (Quit)
83C4h	Link to ISR Hook
83CAh	Console Keyboard Debonce
83CCh	Sound List Pointer (VDP RAM)
83CEh	Sound List Initiation (set to 01h) & Countdown Byte
83D0h	Search Pointers for GROM & ROM
83D4h	Contents of VDP Register 1
83D6h	Screen Timeout Counter
83D8h	Return Address Saved by Scan Routine
83DAh	Player Number Used by Scan Routine
83E0h 83FFh	Begin GPL Workspace

4.4 Low Memory Expansion

2000h 200Fh	XML Vectors	0010h bytes
2010h 3423h	Forth Disk Buffers	1414h
3424h 397Fh	99/4 Support for Forth	055Ch
3980h 39FFh	User Variable Area	0080h
3A00h 3CD9h	Assembler Support	020Ah
3CDAh ↑ ↑ 3FFFh	Return Stack	0326h

4.5 High Memory Expansion

A000h BC7Fh	Resident Forth Vocabulary	1C80h
BC80h ↓ ↓ ↑ ↑ FF9Fh	User Dictionary Space Parameter Stack	4320h
FFA0h FFF1h	Terminal Input Buffer	0052h

5 System Synonyms and Miscellaneous Utilities

Words introduced in this chapter:

!"	**MYSELF**	**UNTRACE**
.S	**RANDOMIZE**	**VAND**
: (traceable)	**RND**	**VFILL**
CLS	**RNDW**	**VLIST**
DISK-HEAD	**SCOPY**	**VMBR**
DSRLNK	**SEED**	**VMBW**
DTEST	**SMOVE**	**VOR**
DUMP	**TRACE**	**VSBR**
FORMAT-DISK	**TRIAD**	**VSBW**
FORTH-COPY	**TRIADS**	**VWTR**
GPLLNK	**TROFF**	**VXOR**
INDEX	**TRON**	**VMLLNK**

Several utilities are available to give you simple access to many resources of the TI-99/4A Home Computer. These are defined as system synonyms.

Also included in this chapter are several disk utilities, special trace routines, random number generators and a special routine that allows recursion.

The descriptions that follow in tabular form include the abbreviation “instr” for “instruction”.

5.1 System Synonyms

The system synonyms are loaded by typing the TI Forth **MENU** option, **-SYNONYMS** . These utilities allow you to

- change the display;
- access the Device Service Routines for peripheral devices such as RS232 interfaces and disk drives;
- link your program to GPL and Assembler routines; and
- perform operations on VDP memory locations.

5.1.1 VDP RAM Read/Write

The first group of instructions enables you to read from and write to VDP RAM. Each of the following Forth words implements the Editor/Assembler utility with the same name.

VSBW (*b vaddr* ---)

Writes a single byte to VDP RAM. It requires 2 parameters on the stack: a byte *b* to be written and a VDP address *vaddr*.

base	*byte*	*vaddr*	instr
HEX	**A3**	**380**	**VSBW**

The above line, when interpreted will change the base to hexadecimal, push **A3h** and **380h** onto the stack and, when **VSBW** executes, places the value **A3h** into VDP address **380h**.

VMBW (*addr vaddr count* ---)

Writes multiple bytes to VDP RAM. You must first place on the stack a source address at which the bytes to be written are located. This must be followed by a VDP address (or destination) and the number of bytes to be written.

base	*addr*	*vaddr*	*count*	instr
HEX	**PAD**	**808**	**4**	**VMBW**

reads 4 bytes from PAD and writes them into VDP RAM beginning at **808h**.

VSBR (*vaddr --- byte*)

Reads a single byte from VDP RAM and places it on the stack. A VDP address is the only parameter required.

base	*vaddr*	instr
HEX	**781**	**VSBR**

places the contents of VDP address **781h** on the stack.

VMBR (*vaddr addr count* ---)

Reads multiple bytes from VDP and places them at a specified address. You must specify the VDP source address, a destination address and a byte count.

base	*vaddr*	*addr*	*count*	instr
HEX	**300**	**PAD**	**20**	**VMBR**

reads 32 bytes beginning at **300h** and stores them into PAD.

VFILL (*vaddr count byte* ---)

If you wish to fill a group of consecutive VDP memory locations with a particular byte, a **VFILL** instruction is available. You must specify a beginning VDP address, a count and the byte you wish to write into each location.

base	*vaddr*	*count*	*byte*	instr
HEX	**300**	**20**	**0**	**VFILL**

fills 32 (**20h**) locations, starting at **300h**, with zeroes.

5.1.2 Extended Utilities: GPLLNK, XMLLNK AND DSRLNK

The next group of instructions allows you to implement the Editor/Assembler instructions GPLLNK, XMLLNK and DSRLNK. To assist the user, the Forth instructions have the same names as the Editor/Assembler utilities. Consult the *Editor/Assembler Manual* for more details.

GPLLNK (*addr* ---)

Allows you to link your program to Graphics Programming Language (GPL) routines. You must place on the stack the address of the GPL routine to which you wish to link.

base	*addr*	instr
HEX	**16**	**GPLLNK**

branches to the GPL routine located at **16h** which loads the standard character set into VDP RAM. It then returns to your program.

XMLLNK (*addr* ---)

Allows you to link a Forth program to a routine in ROM or to branch to a routine located in the Memory Expansion unit. The instruction expects to find a ROM address on the stack.

base	*addr*	instr
HEX	**800**	**XMLLNK**

accesses the Floating Point multiplication routine, located in ROM at **800h**, and returns to your program.

DSRLNK (---)

Links a Forth program to any Device Service Routine (DSR) in ROM. Before this instruction is used, a Peripheral Access Block (PAB) must be set up in VDP RAM. A PAB contains information about the file to be accessed. See the *Editor/Assembler Manual* and Chapter 8 of this manual for additional setup information. **DSRLNK** needs no parameters on the stack.

The Editor/Assembler version of DSRLNK also allows linkage with a subroutine, but the TI Forth version does not. If you need this functionality, you might define the following word in decimal mode (**BASE** contains **Ah**):

```
: DSRLNK-SP 10 14 SYSTEM ;
```

See the *Editor/Assembler Manual* for details on this form of the call to the DSRLNK utility. You will also need to consult the DSR's specifications because this form of access is at a lower level, with each subroutine often requiring information that differs from the PAB set up for **DSRLNK**.

5.1.3 VDP Write-Only Registers

The VDP contains 8 special write-only registers. In the Editor/Assembler, a VWTR instruction is used to write values into these registers. The Forth word **VWTR** implements this instruction.

VWTR (*b n* ---)

VWTR requires 2 parameters; a byte *b* to be written and a VDP register number *n*.

base	*b*	*n*	instr
HEX	**F5**	**7**	**VWTR**

The above instruction writes **F5h** into VDP write only register number 7. This particular register controls the foreground and background colors in text mode. Executing the above instruction will change the foreground color to white and the background color to light blue.

5.1.4 VDP RAM Single-Byte Logical Operations

VAND , **VOR** and **VXOR** (*b vaddr* ---)

The Forth instructions **VAND** , **VOR** and **VXOR** greatly simplify the task of performing a logical operation on a single byte in VDP RAM. Normally, 3 programming steps would be required: a read from VDP RAM, an operation, and a write back into VDP RAM. The above instructions get the job done in a single step. Each of these words requires 2 parameters, a byte *b* to be used as the second operand and the VDP address *vaddr* at which to perform the operation. The result of the operation is placed back into *vaddr*.

base	*b*	*vaddr*	instr
HEX	**F0**	**804**	**VAND**
HEX	**F0**	**804**	**VOR**
HEX	**F0**	**804**	**VXOR**

Each of the above instructions reads the byte stored at **804h** in VDP RAM, performs an AND, OR or XOR on that byte and **F0h**, and places the result back into VDP RAM at **804h**.

5.2 Disk Utilities

The TI Forth system was designed to be used with 90 screens per disk, *i.e.*, with 90 KB, single-sided, single-density (SSSD) disks. The system easily scales up to other disk formats[9], except for some of the disk utilities in this section: **FORTH-COPY** , **DTEST** , **DISK-HEAD** and **FORMAT-DISK** are hardwired to use 90 KB disks. **FORTH-COPY** and **DTEST** require minor changes in the word

9 See Appendix K for a detailed discussion of disk format.

definitions to change the 90-screen limit per disk (See Forth screen 39). Changing **DISK-HEAD** to work is a lot more complicated! It requires low-level knowledge of the format of TI disks to modify its definition (See Forth screen 40). **FORMAT-DISK** is part of the resident TI Forth vocabulary, making it wiser to use other means of formatting disks rather than attempting to re-write the definition for this word, but see Appendix L.

5.2.1 Disk Formatting Utility

FORMAT-DISK (*n* ---)

FORMAT-DISK is one of the system utilities loaded by the **-SYNONYMS** option. Any disk that you wish to use with the Forth system must first be properly formatted. Place the disk in a disk drive and place the number of that disk drive on the stack. TI Forth numbers disk drives beginning with 0, therefore, if the new disk is in the first drive, put a 0 on the stack, *etc*. Next, type **FORMAT-DISK** .

```
0 FORMAT-DISK
```

will initialize the disk in DR0, thus preparing it for use by the Forth system. Disks initialized by the TI Disk Manager are properly formatted and may be used. **FORMAT-DISK** assumes 90 KB, SSSD TI disks.

5.2.2 Disk and Screen Copying Utilities

The disk and screen copying utilities are loaded by the **-COPY** option.

DISK-HEAD (---)

The TI Forth System disk, or any disk which contains a copy of Forth screens 0 thru 19 of the System disk, may be copied with the TI Disk Manager. Any other disk may be copied with the TI Disk Manager only after a special header has been written on it by the TI Forth word **DISK-HEAD** . Please note that you *must* reset the value of the user variable **DISK_LO** to zero *before* using **DISK-HEAD** . This word writes the volume name "FORTH" on the disk and creates a single file named "SCREENS" of type "DIS/FIX128", *i.e.*, display-type, 128-byte, fixed-length records. The file is set up to fill all available space on the disk.

Any Forth disk (system or screens-only), which can be copied by the TI Disk Manager, can also be accessed from TI BASIC. If you access a Forth disk that contains the Forth kernel, the only file you should access is named "SYS-SCRNS" and record 0 of the file will be located on line 4 of screen 19. Records of length = 128 bytes will proceed thru record 565, which is located on line 14 of screen 89. Record 566 then wraps to line 4 of screen 1. The file ends with record 623 located on line 6 of screen 8.

A Forth disk which does not contain the kernel may also be accessed by TI BASIC, but the location of the records will be different. The file created by **DISK-HEAD** above, named "SCREENS", will begin on line 8 of screen 8 and continue thru record 651 located on line 14 of screen 89. Record 652 begins on line 12 of screen 0 and the file ends with record 713 on line 6 of screen 8.

FORTH-COPY (---)

To copy an entire 90 KB, SSSD Forth disk without using the TI Disk Manager, you must place the new disk in DR0 and the source disk in DR1. Typing **FORTH-COPY** will copy the entire contents of the disk in DR1 onto the disk in DR0. Please note that you *must* reset the value of the user variable **DISK_LO** to zero *before* using **FORTH-COPY** . This will allow you to copy screen 0. This is accomplished by executing the following instruction:

```
0 DISK_LO !
```

Using **FORTH-COPY** to copy Forth disks that have higher capacity than 90 KB, *e.g.*, 180 KB or 360 KB, requires rewriting the definition of **FORTH-COPY** , as well as changing **DISK_SIZE** and **DISK_HI** to accommodate the new disk sizes (see Appendix L).

SCOPY (scr_1 scr_2 ---)

You can copy the contents of a single Forth screen from one screen location to another without destroying the original copy by using the **SCOPY** instruction. A source screen number scr_1 and a destination screen number scr_2 must be specified.

base	scr_1	scr_2	instr
DECIMAL	**5**	**17**	**SCOPY**

will write the contents of screen 5 over the contents of screen 17 without erasing screen 5. The old contents of screen 17 will be destroyed.

SMOVE (scr_1 scr_2 *count* ---)

The **SMOVE** instruction acts as a multiple **SCOPY**. It allows you to copy a group of Forth screens with a single instruction. You must designate a beginning source screen, a beginning destination screen, and the number of screens you wish to copy. When using **SMOVE**, overlapping screen ranges may be used without user concern. The order of the copy is adjusted so that the entire group of screens is moved intact.

base	scr_1	scr_2	*count*	instr
DECIMAL	**11**	**36**	**7**	**SMOVE**

will copy screens 11 - 17 over screens 36 - 42 without erasing screens 11 - 17.

Both the **SCOPY** and **SMOVE** instructions can be used to copy screens from one disk drive to another. Assuming that **DISK_SIZE** (a user variable which contains the number of screens per disk) is at its default value of 90, screens 0 - 89 are contained on the disk in DR0, screens 90 -179 are located on the disk in DR1, *etc*. ***Note:*** To copy screens from one disk drive to another, you must reset the user variable **DISK_HI**. If you are using two disk drives, its value must be 180 (2 · 90). This is accomplished by executing the following instruction:

```
180 DISK_HI !
```

Therefore, to copy screen 6 on DR0 to screen 20 on DR1, you would type:

base	scr_1	scr_2	instr
DECIMAL	**6**	**110**	**SCOPY**

The **SMOVE** instruction is handled in the same manner. Simply use an offset of **DISK_SIZE** to specify which disk drives you wish to copy to and from.

DTEST (---)

If you have reason to suspect that a 90 KB, SSSD disk has a bad sector or is in some way damaged, a non-destructive disk test is available. The **DTEST** instruction will attempt to read each screen from the disk in DR0. Please note that you *must* reset the value of the user variable **DISK_LO** to zero *before* using **DTEST** . A screen number will be displayed on your monitor as each screen is read. If execution stops before screen 89 is reached, the problem lies in the last screen displayed. To correct the problem, **CLEAR** that screen and write to it again. This correction will work if the disk surface is intact and if the formatting information has not been damaged. **DTEST** can be rewritten to accommodate more capacious disks (see **FORTH-COPY** above and Appendix L).

5.3 Listing Utilities

There are three words on the TI Forth System disk (loaded by the **-PRINT** option) which make listing information from a Forth disk very simple.

TRIAD (*scr* ---)

The first, called **TRIAD**, requires a Forth screen number on the stack. When executed, it will print to an RS232 device the three screens which contain the specified screen, beginning with a screen number evenly divisible by three. Screens that contain non-printable information will be skipped. If your RS232 printer is not on Port 1 and set at 9600 Baud, you must modify the word **SWCH** on your System disk.

TRIADS (scr_1 scr_2 ---)

The second instruction, called **TRIADS**, may be thought of as a multiple **TRIAD**. It expects a beginning and an ending screen number on the stack. **TRIADS** performs as many **TRIADS** as necessary to cover the specified range of screens.

INDEX (scr_1 scr_2 ---)

The **INDEX** instruction allows you to list to your terminal line 0 (the comment line) of each of a specified range of screens. **INDEX** expects a beginning and an ending screen number on the stack. If you wish to temporarily stop the flow of output in order to read it before it scrolls off the screen, simply press any key. Press any key to start up again. Press ***<BREAK>*** (***<CLEAR>*** or ***<FCTN+4>***) to exit execution prematurely.

5.4 Debugging

5.4.1 Dump Information to Terminal

Choosing the **-DUMP** option loads three useful TI Forth words for getting information for debugging purposes.

VLIST (---)

The Forth word **VLIST** lists to your terminal the names of all words currently defined in the **CONTEXT** vocabulary. This instruction requires no parameters and may be halted and started again by pressing any key as with **INDEX** in the previous section.

DUMP (*addr count* ---)

The **DUMP** instruction allows you to list portions of memory to your terminal. **DUMP** requires two parameters: an address and a byte count. For example,

base	*addr*	*count*	instr
HEX	**2F26**	**100**	**DUMP**

will list 256 (**100h**) bytes of memory beginning at address **2F26h** to your terminal. Press any key to temporarily stop execution in order to read the information before it scrolls off the screen. Press any key to continue. To exit this routine permanently, press ***<BREAK>***.

.S (---)

The Forth word **.S** allows you to view the parameter stack contents. It may be placed inside a colon definition or executed directly from the keyboard. The word **SP!** should be typed before executing a routine that contains **.S** . This will clear any garbage from the stack. The **|** symbol is printed to represent the bottom of the stack. The number appearing farthest from the **|** is the most accessible stack element.

5.4.2 Tracing Word Execution

This section is based on the following article available at *www.forth.org* :

Paul van der Eijk. 1981. Tracing Colon-Definitions. *Forth Dimensions* **3:** 58.

A special set of instructions allows you to trace the execution of any colon definition. Executing the **TRACE** instruction will cause all following colon definitions to be compiled in such a way that they can be traced. In other words, the Forth word **:** takes on a new meaning. To stop compiling under the **TRACE** option, type **UNTRACE**. When you have finished debugging, recompile the routine under the **UNTRACE** option.

After instructions have been compiled under the **TRACE** option, you can trace their execution by typing the word **TRON** before using the instruction. **TRON** activates the trace. If you wish to execute the same instruction without the trace, type **TROFF** before using the instruction.

The actual trace will print the word being traced, along with the stack contents, each time the word is encountered. This shows you what numbers are on the stack just before the traced word is executed. The **|** symbol is used to represent the bottom of the stack. The number printed closest to the **|** is the least accessible while the number farthest from the **|** is the most accessible number on the stack. Here is a sample **TRACE** session:

```
DECIMAL
TRACE ok                    (compile next definition with TRACE option)
: CUBE DUP DUP * * ;        (routine to be traced)
UNTRACE OK                  (don't compile next definition with TRACE option)
: TEST CUBE ROT CUBE ROT CUBE ; ok
TRON ok                     (want to execute with a TRACE)
```

```
5 6 7 TEST              (put parameters on stack and execute TEST)
CUBE                    (TRACE begins)
| 5 6 7                 (stack contents upon entering CUBE)
CUBE
| 6 343 5               (stack contents upon entering CUBE)
CUBE
| 343 125 6 ok
```

5.4.3 Recursion

Normally, a Forth word cannot call itself before the definition has been compiled through to a **;** because the **SMUDGE** bit is set. To allow recursion, TI Forth includes the special word **MYSELF .**

MYSELF (---)

The **MYSELF** instruction places the CFA of the word currently being compiled into its own definition thus allowing a word to call itself.

The following, more complex, **TRACE** example uses a recursive factorial routine for illustration:

```
DECIMAL ok
TRACE ok                (compile following definition under TRACE option)
: FACT DUP 1 > IF DUP 1 - MYSELF * ENDIF ; ok
UNTRACE ok
TRON ok
5 FACT                  (put parameter on stack and execute FACT)
FACT                    (TRACE begins)
| 5
FACT
| 5 4
FACT
| 5 4 3
FACT
| 5 4 3 2
FACT
| 5 4 3 2 1 ok
.S                      (check final stack contents)
| 120 ok
```

Each time the traced **FACT** routine calls itself, a **TRACE** is executed.

5.5 Random Numbers

Two different random number functions are available in TI Forth.

RND (n_1 --- n_2)

The first, **RND ,** generates a positive random integer between 0 and a specified range n_1.

base	n_1	instr
DECIMAL	**13**	**RND**

will place on the stack an integer greater than or equal to 0 and less than 13.

RNDW (--- *n*)

The second random number function, **RNDW ,** generates a random word (2 bytes). No range is specified for **RNDW .**

```
RNDW
```

will place on the stack a number from **0** to **FFFFh**.

RANDOMIZE (---)

To guarantee a different sequence of random numbers each time a program is run, the **RANDOMIZE** instruction must be used. **RANDOMIZE** places an unknown seed into the random number generator.

SEED (n ---)

To place a known seed into the random number generator, the **SEED** instruction is used. You must specify the seed value.

```
4 SEED
```

will place the value 4 into the random number generator seed location.

5.6 Miscellaneous Instructions

!" (*addr* ---)

This word is loaded by the **-COPY** option to be used by **DISK-HEAD** , but is available for your use. It stores a string at a specified address, but does not store the character count, which you would need to use **TYPE** . **!"** expects to find an address on the stack and must be followed by a string terminated with a **"** . The following instruction places the string "HOW ARE YOU?" at address **PAD** :

```
PAD !" HOW ARE YOU?"
```

CLS (---)

CLS is loaded by the **-SYNONYMS** option. Use this word to clear the display screen. **CLS** clears the display screen by filling the screen image table with blanks. The screen image table runs from **SCRN_START** to **SCRN_END** . **CLS** may be used inside a colon definition or directly from the keyboard. **CLS** will not clear bit-map displays or sprites.

6 An Introduction to Graphics

Words introduced in this chapter:

#MOTION	**GRAPHICS**	**SPLIT2**
BEEP	**GRAPHICS2**	**SPRCOL**
CHAR	**HCHAR**	**SPRDIST**
CHARPAT	**HONK**	**SPRDISTXY**
COINC	**JOYST**	**SPRGET**
COINCALL	**LINE**	**SPRITE**
COINCXY	**MAGNIFY**	**SPRPAT**
COLOR	**MCHAR**	**SPRPUT**
DELALL	**MINIT**	**SSDT**
DELSPR	**MOTION**	**TEXT**
DOT	**MULTI**	**UNDRAW**
DRAW	**SCREEN**	**VCHAR**
DTOG	**SPCHAR**	
GCHAR	**SPLIT**	

6.1 Graphics Modes

The TI Home Computer possesses a broad range of graphics capabilities. Four screen modes are available to the user:

1) **Text Mode—**Standard ASCII characters are available, and new characters may be defined. All characters have the same foreground and background color. The screen is 40 columns by 24 lines. Text mode is used by the Forth 40-column screen editor.

2) **Graphics Mode—**Standard ASCII characters are available, and new characters may be defined. Each character set may have its own foreground and background color.

3) **Multicolor Mode—**The screen is 64 columns by 48 rows. Each standard character position is now 4 smaller boxes which can each have a different color. ASCII characters are not available and new characters cannot be defined.

4) **Bit-Map Mode (Graphics2)—**This mode is available only on the TI-99/4A. Bit-map mode allows you to set any pixel on the screen and to change its color within the limits permitted by the 9918a. The screen is 256 columns by 192 rows. Graphics2 mode is used by the 64-column editor.

Sprites (moving graphics) are available in all modes except text. The sprite automotion feature is not available in graphics2, split, or split2 modes.

Two unique graphics modes have been created by using graphics2 mode in a non-standard way. Split and split2 modes allow you to display text while creating bit-map graphics. Split mode sets the top two thirds of the screen in graphics2 mode and places text on the last third. Split2 sets the

top one sixth of the screen as a text window and the rest in graphics2 mode. These modes provide an interactive bit map graphics setting. That is, you can type bit map instructions and watch them execute without changing modes.

You may place the computer in the above modes by executing one of the following instructions:

TEXT (---)

GRAPHICS (---)

MULTI (---)

GRAPHICS2 (---)

SPLIT (---)

SPLIT2 (---)

6.2 Forth Graphics Words

Many Forth words have been defined to make graphics handling much easier for the user. As many words are mentioned, an annotation will appear underneath them denoting which of the modes they may be used in (T G M B). These denote text, graphics, multicolor and bit-mapped (graphics2, split, split2) modes, respectively.

In several instruction examples, a base (**HEX** or **DECIMAL**) is specified. This does not mean that you must be in a particular base in order to use the instruction. It merely illustrates that some instructions are more easily written in hexadecimal than in decimal. It also avoids ambiguity.

6.3 Color Changes

The simplest graphics operations involve altering the color of the screen and of character sets. There are 32 character sets (0 – 31), each containing 8 characters. For example, character set 0 consists of characters 0 – 7, character set 1 consists of characters 8 – 15, etc. Sixteen colors are available on the TI Home Computer.

Color	Hex Value	Color	Hex Value
transparent	0	medium red	8
black	1	light red	9
medium green	2	dark yellow	A
light green	3	light yellow	B
dark blue	4	dark green	C
light blue	5	magenta	D
dark red	6	gray	E
cyan	7	white	F

SCREEN (*color* ---)

The Forth word **SCREEN** following one of the above table values will change the screen color to that value. The following example changes the screen to light yellow:

base	*color*	instr	
HEX	**B**	**SCREEN**	or
DECIMAL	**11**	**SCREEN**	
		(G)	

For text mode, the color of the foreground also needs to be set and should be different from the background color so that text is visible. The foreground color must be in the leftmost 4 bits of the byte passed to **SCREEN**. It is easier to compose the byte in hexadecimal than decimal because each half of the byte is one hexadecimal digit. To set the foreground to black (**1**) and the background to light yellow (**Bh**), the following sequence will do the trick:

```
HEX 1B SCREEN
```

COLOR (*fg bg charset* ---)

The foreground and background colors of a character set may also be easily changed:

base	*fg*	*bg*	*charset*	instr	
HEX	**4**	**D**	**1A**	**COLOR**	or
DECIMAL	**4**	**13**	**26**	**COLOR**	
				(G)	

The above instruction will change character set 26 (characters 208 – 215) to have a foreground color of dark blue and a background color of magenta.

6.4 Placing Characters on the Screen

HCHAR (*col row count char* ---)

To print a character anywhere on the screen and optionally repeat it horizontally, the **HCHAR** instruction is used. You must specify a starting column and row position as well as the number of repetitions and the ASCII code of the character you wish to print.

Keep in mind that both rows and columns are numbered from zero !!!

For example,

base	*col*	*row*	*count*	*char*	instr	
HEX	**A**	**11**	**5B**	**2A**	**HCHAR**	or
DECIMAL	**10**	**17**	**91**	**42**	**HCHAR**	
					(T G)	

will print a stream of 91 *s, starting at column 10 row 17, that will wrap from right to left on the screen.

VCHAR (*col row count char* ---)

To print a vertical stream of characters, the word **VCHAR** is used in the same format as **HCHAR** . These characters will wrap from the bottom of the screen to the top.

GCHAR (*col row* --- *char*)

The Forth word **GCHAR** will return on the stack the ASCII code of the character currently at any position on the screen. If the above **HCHAR** instruction were executed and followed by

base	*col*	*row*	instr	
HEX	**F**	**11**	**GCHAR**	or
DECIMAL	**15**	**17**	**GCHAR**	
			(T G)	

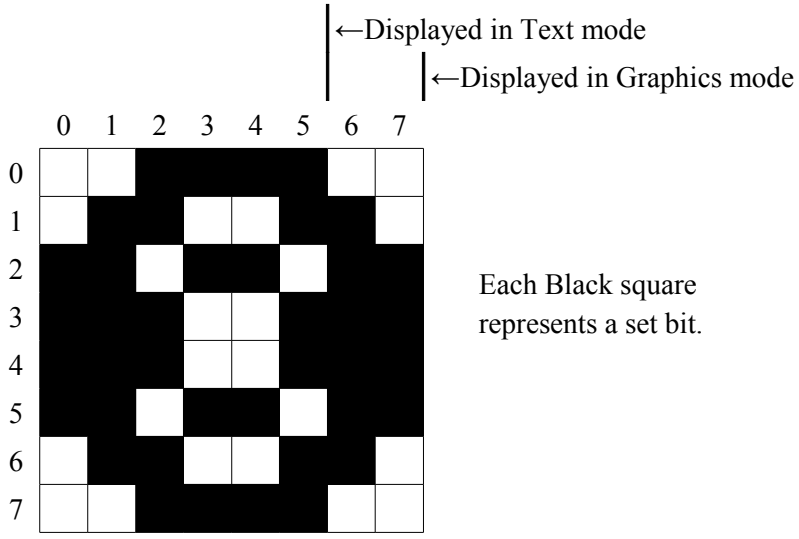
2Ah or 42 would be left on the stack.

6.5 Defining New Characters

Each character in graphics mode is 8 x 8 pixels in size. Each row makes up one byte of the 8-byte character definition. Each set bit (1) takes on the foreground color while the others remain the background color.

In text mode, characters are defined in the same way, but only the left 6 bits of each row are displayed on the screen.

For example,



this character is defined:

	3C66h	DBE7h	E7DBh	663Ch
Rows	0 – 1	2 – 3	4 – 5	6 – 7

CHAR (n_1 n_2 n_3 n_4 *char* ---)

The Forth word **CHAR** is used to create new characters. To assign the above pattern to character number 123, you would type

base	n_1	n_2	n_3	n_4	*char*	instr	
HEX	**3C66**	**DBE7**	**E7DB**	**663C**	**7B**	**CHAR**	or
DECIMAL	**15426**	**56295**	**59355**	**26172**	**123**	**CHAR**	
						(T G)	

As you can see, it is more natural to use this instruction in **HEX** than in **DECIMAL** .

CHARPAT (*char* --- n_1 n_2 n_3 n_4)

To define another character to look like character 65 (“A”), for example, you must first find out what the pattern code for “A” is. To accomplish this, use the **CHARPAT** instruction. This instruction leaves the character definition on the stack in the proper order for a **CHAR** instruction. Study this line of code:

HEX	**41**	**CHARPAT**	**7E**	**CHAR**	or
DECIMAL	**65**	**CHARPAT**	**126**	**CHAR**	
				(T G)	

The above instructions place on the stack the character pattern for “A” and assigns the pattern to character 126. Now both character 65 and 126 have the same shape.

6.6 Sprites

Sprites are moving graphics that can be displayed on the screen independently and/or on top of other characters. Thirty-two sprites are available.

6.6.1 Magnification

Sprites may be defined in 4 different sizes or magnifications:

Magnification Factor	Description
0	Causes all sprites to be single size and unmagnified. Each sprite is defined only by the character specified and occupies one character position on the screen.

Magnification Factor	Description
1	Causes all sprites to be single size and magnified. Each sprite is defined only by the character specified, but this character expands to fill 4 screen positions.
2	Causes all sprites to be double size and unmagnified. Each sprite is defined by the character specified along with the next 3 characters. The first character number must be divisible by 4. This character becomes the upper left quarter of the sprite, the next characters are the lower left, upper right, lower right respectively. The sprite fills 4 screen positions.
3	Causes all sprites to be double size and magnified. Each sprite is defined by 4 characters as above, but each character is expanded to occupy 4 screen positions. The sprite fills 16 positions.

The default magnification is 0.

MAGNIFY (*n* ---)

To alter sprite magnification, use the Forth word **MAGNIFY** .

n	instr
2	**MAGNIFY**
	(G M B)

will change all sprites to double size and unmagnified.

6.6.2 Sprite Initialization

SSDT (*vaddr* ---)

Before you begin defining sprites, you must execute the Forth word **SSDT** which roughly translates, "set Sprite Descriptor Table." Before executing this instruction, the computer must be set into the VDP mode you wish to use with sprites. Recall that *sprites are not available in text mode*.

You have a choice of overlapping your sprite character definitions with the standard characters in the Pattern Descriptor Table (see VDP Memory Map in Chapter 4) or moving the Sprite Descriptor Table elsewhere in memory. This move is highly recommended to avoid confusion. **2000h** is usually a good location, but any available 2K (**800h**) boundary will do.

base	*vaddr*	instr
HEX	**2000**	**SSDT**
DECIMAL	**8192**	**SSDT**

or

(G M B)

will move the Sprite Descriptor Table to **2000h**. Use the value **800h** with the **SSDT** instruction if you do not want to move the Descriptor Table.

Note: Whether or not you choose to move the table, you must execute this instruction before you can use sprites in your program!!!

6.6.3 Using Sprites in Bit-Map Mode

SATR (--- *vaddr*)

When using sprites in any of the bit-map modes (graphics2, split, split2), a little extra work is required. After entering the desired VDP mode, the location of the Sprite Attribute List must be changed to **3800h** as follows.

```
HEX 3800 ' SATR !
```
[10]

The base address of the Sprite Descriptor Table must also be changed using the **SSDT** instruction. It will be based at the same address as the Sprite Attribute List (**3800h**), but only a few character numbers will be available for sprite patterns. **SPCHAR** may only be used to define patterns 16 – 58. (See following section for information on **SPCHAR**.)

3800h	Sprite Attribute List **0080h**
3880h	Sprite Patterns 16-58 (based at **3800h**)
39D9h	**015Ah**

6.6.4 Creating Sprites

The first task involved in creating sprites is to define the characters you will use to make them. These definitions will be stored in the Sprite Descriptor Table mentioned in the above section.

SPCHAR (n_1 n_2 n_3 n_4 *char* ---)

A word identical in format to **CHAR** is used to store sprite character patterns. If you are using a magnification factor of 2 or 3, do not forget that you must define 4 consecutive characters for *each* sprite. In this case, the character # of the first character must be a multiple of 4.

10 Bug fix: See Appendix J.

base	n_1	n_2	n_3	n_4	*char*	instr
HEX	**0F0F**	**2424**	**F0F0**	**4242**	**0**	**SPCHAR**
DECIMAL	**3855**	**9252**	**61680**	**8770**	**0**	**SPCHAR**
						(G M B)

or

defines character 0 in the Sprite Descriptor Table. If your Pattern and Sprite Descriptor Tables overlap, use character numbers below 127 with caution.

SPRITE (*dotcol dotrow color char spr* ---)

To define a sprite, you must specify the dot column and dot row at which its upper left corner will be located, its color, a character number and a sprite number (0 – 31).

base	*dotcol*	*dotrow*	*color*	*char*	*spr*	instr
HEX	**6B**	**4C**	**5**	**10**	**1**	**SPRITE**
DECIMAL	**107**	**76**	**5**	**16**	**1**	**SPRITE**
						(G M B)

or

defines sprite #1 to be located at column 107 and row 76, to be light blue and to begin with character 16. Its size will depend on the magnification factor.

Once a sprite has been created, changing its pattern, color or location is trivial.

SPRPAT (*char spr* ---)

base	*char*	*spr*	instr
HEX	**14**	**1**	**SPRPAT**
DECIMAL	**20**	**1**	**SPRPAT**
			(G M B)

or

will change the pattern of sprite #1 to character number 20.

SPRCOL (*color spr* ---)

base	*color*	*spr*	instr
HEX	**C**	**2**	**SPRCOL**
DECIMAL	**12**	**2**	**SPRCOL**
			(G M B)

or

will change the color of sprite #2 to dark green.

SPRPUT (*dotcol dotrow spr* ---)

base	*dotcol*	*dotrow*	*spr*	instr	
HEX	**28**	**4F**	**1**	**SPRPUT**	or
DECIMAL	**40**	**79**	**1**	**SPRPUT**	
				(G M B)	

will place sprite #1 at column 40 and row 79.

6.6.5 Sprite Automotion

In graphics or multicolor mode, sprites may be set in automotion. That is, having assigned them horizontal and vertical velocities and set them in motion, they will continue moving with no further instruction. Sprite automotion is only available in graphics and multicolor modes.

Velocities from 0 to **7Fh** are positive velocities (down for vertical and right for horizontal), and from **FFh** to **80h** are taken as two's complement negative velocities.

MOTION (*xvel yvel spr* ---)

base	*xvel*	*yvel*	*spr*	instr	
HEX	**FC**	**6**	**1**	**MOTION**	or
DECIMAL	**-4**	**6**	**1**	**MOTION**	
				(G M)	

will assign sprite #1 a horizontal velocity of -4 and a vertical velocity of 6, but will not actually set them into motion.

#MOTION (*n* ---)

After you assign each sprite you want to use a velocity, you must execute the word **#MOTION** to set the sprites in motion. **#MOTION** expects to find on the stack the highest sprite number you are using + 1.

n	instr
6	**#MOTION**
	(G M)

will set sprites #0 – #5 in motion.

n	instr
0	**#MOTION**

will stop all sprite automotion, but motion will resume when another **#MOTION** instruction is executed.

SPRGET (*spr --- dotcol dotrow*)

Once a sprite is in motion, you may wish to find out its horizontal and vertical position on the screen at a given time.

spr	instr
2	**SPRGET**

(G M B)

will return on the stack the horizontal position of sprite #2 underneath the vertical position. The sprite does *not* have to be in automotion to use this instruction.

6.6.6 Distance and Coincidences between Sprites

It is possible to determine the distance *d* between two sprites or between a sprite and a point on the screen. This capability comes in handy when writing game programs. The actual value returned by each of the Forth words, **SPRDIST** and **SPRDISTXY** , is d^2. Distance *d* is the hypotenuse of the right triangle formed by joining the line segments, *d*, $x_2 - x_1$ (the horizontal ***x***-distance difference in dot columns) and y_2 - y_1 (the vertical ***y***-distance difference in dot rows). The squared distance between the two sprites or the sprite and screen point is calculated by squaring the ***x***-distance difference and adding that to the square of the the ***y***-distance difference, *i.e.*, $d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$.

SPRDIST (spr_1 spr_2 --- *n*)

spr_1	spr_2	instr
2	**4**	**SPRDIST**

(G M B)

returns on the stack the square of the distance between sprite #2 and sprite #4.

SPRDISTXY (*dotcol dotrow spr --- n*)

base	*dotcol*	*dotrow*	*spr*	instr
DECIMAL	**65**	**21**	**5**	**SPRDISTXY**

(G M B)

returns the square of the distance between sprite #5 and the point (65,21).

A coincidence occurs when two sprites become positioned directly on top of one another. That is, their upper left corners reside at the same point. Because this condition rarely occurs when sprites are in automotion you can set a tolerance limit for coincidence detection. For example, a tolerance of 3 would report a coincidence whenever the two sprites upper left corners came within 3 dot positions of each other.

COINC (spr_1 spr_2 *tol --- flag*)

To find a coincidence between two sprites, the Forth word **COINC** is used.

spr_1	spr_2	*tol*	instr
7	**9**	**2**	**COINC**

(G M B)

will detect a coincidence between sprites #7 and #9 if their upper left corners passed within 2 dot positions of each other. If a coincidence is found, a true flag is left on the stack. If not, a false flag is left.

COINCXY (*dotcol dotrow spr tol --- flag*)

Detecting a coincidence between a sprite and a point is similar.

base	*dotcol*	*dotrow*	*spr*	*tol*	instr
DECIMAL	**63**	**29**	**8**	**3**	**COINCXY**

(G M B)

will detect a coincidence between sprite #8 and the point (63,29) with a tolerance of 3. A true or false flag will again be left on the stack.

Both of the above instructions will detect a coincidence between non-visible parts of the sprites. That is, you may not be able to *see* the coincidence.

COINCALL (--- *flag*)

Another instruction is used to detect only *visible* coincidences. It, however, will not detect coincidences between a select two sprites, but will return a true flag when any two sprites collide. This instruction is **COINCALL** , and requires no arguments.

6.6.7 Deleting Sprites

As you might have noticed, sprites do not go away when you clear the rest of the screen with **CLS** . Special instructions must be used to remove sprites from the display,

DELSPR (*spr* ---)

spr	instr
2	**DELSPR**

(G M B)

will remove sprite #2 from the screen by altering its description in the sprite Attribute List (see VDP Memory Map in Chapter 4). It does not remove the velocity of sprite #2 from the Sprite Motion Table, nor does it alter the number of sprites the computer thinks it is dealing with. In other words, if you were to redefine sprite #2, it would immediately begin moving with whatever speed the old sprite #2 had.

DELALL (---)

```
DELALL
```

(G M B)

on the other hand, will remove all sprites from the screen, and from memory. **DELALL** needs no parameters. Only the Sprite Descriptor Table will remain intact after this instruction is executed.

6.7 Multicolor Graphics

Multicolor mode allows you to display kaleidoscopic graphics. Each character position on the screen consists of 4 smaller squares which can each be a different color. A cluster of these characters produces a kaleidoscope when the colors are changed rapidly.

MINIT (---)

After entering multicolor mode, it is necessary to initialize the screen. The **MINIT** instruction will accomplish this. It needs no parameters.

When in multicolor mode, the columns are numbered 0 – 63 and rows are numbered 0 – 47. A multicolor character is ¼ the size of a standard character; therefore more of them fit across and down the screen.

MCHAR (*color col row* ---)

To define a multicolor character, you must specify a color and a position (column, row) and then execute the word **MCHAR** :

base	*color*	*col*	*row*	instr	
HEX	**B**	**1A**	**2C**	**MCHAR**	or
DECIMAL	**11**	**26**	**44**	**MCHAR**	

The above instruction will place a light yellow square at (26,44).

To change a character's color, simply define a different color **MCHAR** with the same position. In other words, cover the existing character.

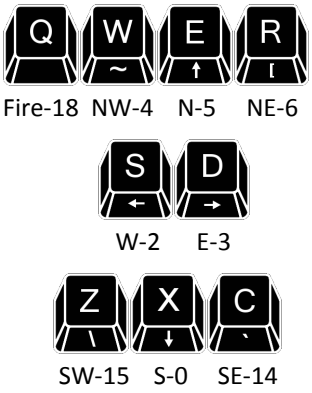
6.8 Using Joysticks

JOYST (n_1 --- *char* n_2 n_3)

The **JOYST** instruction allows you to use joysticks in your Forth program. **JOYST** requires only one parameter, *viz.*, a keyboard number n_1. The keyboard number tells the computer which joystick or which side of the keyboard to scan for input. When keyboard #1 is specified (n_1 = 1), both joystick #1 and the left side of the keyboard are scanned. When keyboard #2 is specified (n_1 = 2), joystick #2 and the right side of the keyboard are scanned. A "Key Pad" exists on each side of the keyboard and may be used in place of joysticks. Map directions (N, S, E, W, NE, etc.) are used on the diagrams below to indicate the corresponding display-screen directions (up, down, right, left, diagonally-up-and-right, *etc.*) The following diagrams show which keys have which function.

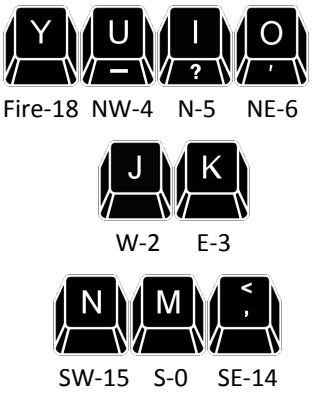
When Joystick #1 is specified, these keys on the left side of the keyboard are valid ▪▪▶

The function of each key is indicated below the key and is followed by the character code returned as *char* on the stack.



When Joystick #2 is specified, these keys on the right side of the keyboard are valid ▪▪▶

The function of each key is indicated below the key and is followed by the character code returned as *char* on the stack.



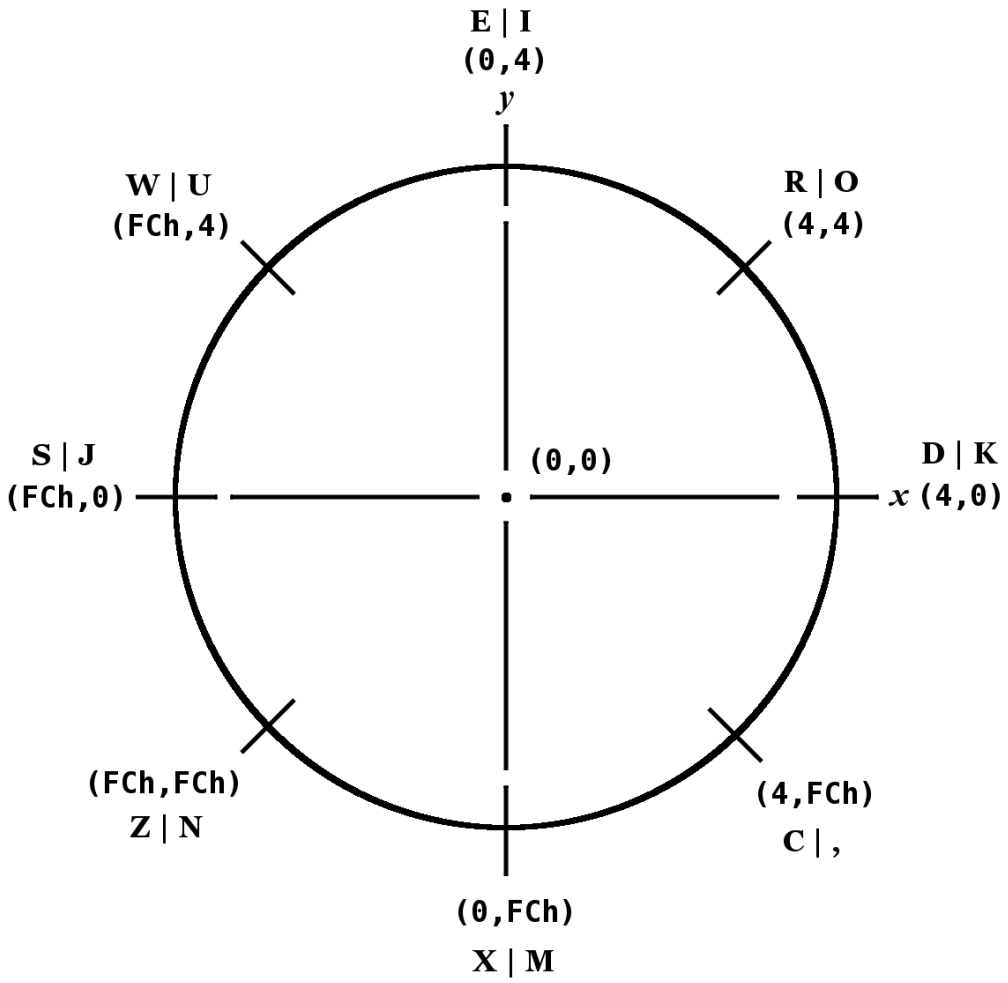
The **JOYST** instruction returns 3 numbers on the stack: a character code *char* on the bottom of the stack, an ***x***-joystick status n_2 and a ***y***-joystick status n_3 on top of the stack. The joystick positions are illustrated in the diagram that follows.

FCh equals decimal 252. The capital letters and ',' separated by '|' indicate which keys on the left and right side of the keyboard return these values. *Note:* The character value of all fire buttons is 18 (**12h**).

If no key is pressed, the returned values will be a character code of 255 (**FFh**), and the current ***x***- and ***y***-joystick positions. If a valid key is pressed, the character code of that key will be returned along with its translated directional meaning (see diagram).

If an illegal key is pressed, three zeroes will be returned. If the fire button is pressed, a character code of 18 (**12h**) along with two zeroes will be returned.

If you are using **JOYST** in a loop, do not forget to **DROP** or otherwise use the three numbers left on the stack before calling **JOYST** again. A stack overflow will likely result if you do not.



Joystick positions and values

6.9 Dot Graphics

High resolution (dot) graphics are available in graphics2, split and split2 modes. In graphics2 mode, it is possible to independently define each of the 49152 pixels on the screen. Split and split2 modes allow you to define the upper two thirds or the lower five sixths of the pixels.

Three dot drawing modes are available:

DRAW (---)

plots dots in the 'on' state.

UNDRAW (---)

plots dots in the 'off' state.

DTOG (---)

toggles dots between the 'on' and 'off' state. If the dot is 'on', **DTOG** will turn it 'off' and vice versa.

DMODE (--- *addr*)

The value of a variable called **DMODE** controls which drawing mode you are in. If **DMODE** = 0, you are in **DRAW** mode. If **DMODE** = 1, you are in **UNDRAW** mode, and if **DMODE** = 2, you are in **DTOG** mode.

DOT (*dotcol dotrow*---)

To actually plot a dot on the screen, the **DOT** instruction is used. You must specify the dot column and dot row of the pixel you wish to plot:

base	*dotcol*	*dotrow*	instr
DECIMAL	**34**	**12**	**DOT**

will plot or unplot, depending on the value of **DMODE** , a dot at position (34,12).

DCOLOR (--- *addr*)

The default color for dots is white on transparent. The screen color default is black. To alter the foreground and background color of the dots you plot, you must modify the value of the variable **DCOLOR** . The value of **DCOLOR** should be two hexadecimal digits where the first digit specifies the foreground color and the second specifies a background color. Why do you need a background color for a dot? There is a simple explanation. Each dot represents one bit of a byte in memory. Any bit in the byte that is turned 'on' displays the foreground color while the others take on the background color. Usually, you would specify the background color to be transparent.

LINE ($dotcol_1$ $dotrow_1$ $dotcol_2$ $dotrow_2$ ---)

The Forth instruction **LINE** allows you to easily plot a line between *any* two points on the bit-map portion of the screen. You must specify a dot column and a dot row for each of the two points.

base	$dotcol_1$	$dotrow_1$	$dotcol_2$	$dotrow_2$	instr
DECIMAL	**23**	**12**	**56**	**78**	**LINE**

The above instruction will plot a line from left to right between (23,12) and (56,78). The line instruction calls **DOT** to plot each point therefore, you must preset **DMODE** and **DCOLOR** before using **LINE** .

6.10 Special Sounds

Two special sounds can be used to enhance your graphics application. To use these noises in your program, simply type the name of the sound you want to hear. No parameters are needed.

BEEP (---)

The first is called **BEEP** and produces a pleasant high pitched sound.

HONK (---)

The other, called **HONK** , produces a less pleasant low tone.

6.11 Constants and Variables Used in Graphics Programming

The following constants and variables are defined in the graphics routines. The value of **COLTAB** , **PDT** , **SATR** , **SMTN** and **SPDTAB** must be changed if you are operating in graphics2, split or split2 mode. See the VDP Memory Map in Chapter 4.

name	type	description	default
COLTAB	constant	VDP address of Color Table	**380h**
DMODE	variable	Dot graphics drawing mode	**0**
PDT	constant	VDP address of Pattern Descriptor Table	**800h**
SATR	constant	VDP address of Sprite Attribute Table	**300h**
SMTN	constant	VDP address of Sprite Motion Table	**780h**
SPDTAB	constant	VDP address of Sprite Descriptor Table	**800h**

7 The Floating Point Support Package

Words introduced in this chapter:

>ARG	**FO<**	**FMUL**
>F	**FO=**	**FOVER**
>FAC	**F<**	**FSUB**
?FLERR	**F=**	**FSWAP**
ATN	**F>**	**INT**
COS	**F@**	**LOG**
EXP	**FAC->S**	**PI**
F!	**FAC>**	**S->F**
F*	**FAC>ARG**	**S->FAC**
F+	**FADD**	**SETFL**
F-	**FDIV**	**SIN**
F->S	**FDUP**	**SQR**
F.	**FF.**	**TAN**
F.R	**FF.R**	**VAL**
F/	**FLERR**	

The floating point package is designed to make it easy to use the Radix 100 floating point package available in ROM in the TI-99/4A console. Normal use of these routines does not require the user to understand the implementation. For those users desiring to improve the efficiency of these operations by optimizing the code for this implementation, the details are given in the latter portion of this chapter.

7.1 Floating Point Stack Manipulation

The floating point numbers in the TI-99/4A occupy 4 16-bit words[11] (8 bytes) each. In order to simplify stack manipulations with these numbers, the following stack manipulation words are presented:

FDUP (f --- f f)

FDROP (f ---)

FOVER (f_1 f_2 --- f_1 f_2 f_1)

FSWAP (f_1 f_2 --- f_2 f_1)

11 This use of the term "word" here is different from a Forth word. It refers to the largest memory unit the TMS9900 CPU can address. It is equal to 2 bytes or 16 bits.

7.2 Floating Point Fetch and Store

Floating point numbers can be stored and fetched by using

F! (*f addr* ---)

F@ (*addr* --- *f*)

The user must ensure that adequate storage is allocated for these numbers (*e.g.*, **0 VARIABLE nnnn 6 ALLOT** could be used. **VARIABLE** allots 2 bytes.)

7.3 Floating Point Conversion Words

The following words put floating point numbers on the stack so that the above operations can be used:

S->F (*n* --- *f*)

A 16-bit number can be converted to floating point by using **S->F** . It functions by replacing the 16-bit number on the stack by a floating point number of equal value.

F->S (*f* --- *n*)

This is the inverse of **S->F** . It starts with a floating point number on the stack and leaves a 16-bit integer.

7.4 Floating Point Number Entry

In addition, the word

>F (--- *f*)

can be used from the console or in a colon definition to convert a string of characters to a floating point number. Note that **>F** is independent of the current value of **BASE** .

The string is always terminated by a blank or carriage return. The following are examples:

```
>F 123          or  123 S->F
>F 123.46
>F -123.46
>F 1.23E-006
>F 9.88E+091
>F 0            or    0 S->F
```

7.5 Floating Point Arithmetic

Floating point arithmetic can now be performed on the stack just as it is with integers. The four arithmetic operators are:

F+ (f_1 f_2 --- f_3)

F- (f_1 f_2 --- f_3) Puts on the stack the result (f_3) of $f_1 - f_2$.

F* (f_1 f_2 --- f_3) Puts on the stack the result (f_3) of f_1 **x** f_2.

F/ (f_1 f_2 --- f_3) Puts on the stack the result (f_3) of f_1 / f_2.

PI (--- f)

The word **PI** is a constant available to place 3.141592653590 on the stack.

7.6 Floating Point Comparison Words

Comparisons between floating point numbers and testing against zero are provided by the following words. They are used just like their 16-bit counterparts except that the numbers tested are floating point.

F0< (f --- *flag*) *flag* is true if f on stack is negative

F0= (f --- *flag*) *flag* is true if f on stack is zero

F> (f_1 f_2 --- *flag*) *flag* is true if $f_1 > f_2$

F= (f_1 f_2 --- *flag*) *flag* is true if $f_1 = f_2$

F< (f_1 f_2 --- *flag*) *flag* is true if $f_1 < f_2$

7.7 Formatting and Printing Floating Point Numbers

F. (f ---)

The word **F.** is used to print the floating point number on the top of the stack to the terminal. The format used is identical to that used by BASIC:

1) Integers representable exactly are printed without a trailing decimal,
2) Fixed point format is used for numbers in range and
3) Exponential format (scientific notation) is used for very large or very small numbers.

F.R (f n ---)

If the floating point numbers are to be output in a table the word **F.R** can be used to right justify it in a field of width n where n is a 16-bit word added to the top of the stack for this purpose.

Two additional words are used for more specific formatting:

FF. (f n_1 n_2 ---)

FF. requires two integers on the stack above the floating point number f. They control the maximum number of digits (n_1) to convert and the number of digits (n_2) following the decimal point.

FF.R (f n_1 n_2 n_3 ---)

FF.R adds the printing field width (n_3), in which the output is right justified. As for **FF.** , n_1 is the maximum number of digits to convert and n_2 is the number of digits following the decimal point.

Editor's Note: It should be noted that the exponential format of the output string allows for just two digits for the power of ten. It is puzzling that TI did this because the exponent can be as high as 127 and as low as -128. This means that perfectly legitimate three-digit exponents appear as "**" in the output!

7.8 Transcendental Functions

The following transcendental functions are also available:

INT	(f_1 --- f_2)	Returns largest integer not larger than input
^	(f_1 f_2 --- f_3)	f_3 is f_1 raised to the f_2 power
SQR	(f_1 --- f_2)	f_2 is the square root of f_1
EXP	(f_1 --- f_2)	f_2 is e (2.71828...) raised to the f_1 power
LOG	(f_1 --- f_2)	f_2 is the natural log of f_1
COS	(f_1 --- f_2)	f_2 is the cosine of f_1 (in radians)
SIN	(f_1 --- f_2)	f_2 is the sin of f_1 (in radians)
TAN	(f_1 --- f_2)	f_2 is the tangent of f_1 (in radians)
ATN	(f_1 --- f_2)	f_2 is the arctangent (in radians) of f_1

Caution! A conflict exists when using transcendentals and floating point prints while in bit-map mode. The contents of the VDP Rollout Area (**3C0h** – **3DFh**) must be saved before transcendentals or floating point prints are executed and restored upon completion.

Note: The transcendentals also use the area known as the stack for the value stack pointer, VSPTR (See VDP Memory Map in Chapter 4). This area is pointed to by **836Eh** (VSPTR).

7.9 Interface to the Floating Point Routines

The remainder of this chapter will address the interface to the floating point routines in the console in greater detail and is not necessary for most floating point operations.

The floating point routines use two memory locations in the console CPU RAM as floating point registers. They are called FAC (for floating point accumulator) and ARG (for argument register). Forth has two constants with these same names that can be used to access these locations directly:

FAC	(--- *addr*)	constant that puts the address of FAC on the stack.
ARG	(--- *addr*)	constant that puts the address of ARG on the stack.

The words **>FAC** and **>ARG** move floating point data from the stack to these locations.

>FAC	(f ---)	moves f to FAC.
>ARG	(f ---)	moves f to ARG.
FAC>	(--- f)	is used to move data from FAC to the stack.

SETFL (f_1 f_2 ---)

Each of the binary floating point operations requires that two numbers be moved from the stack to FAC and ARG. **SETFL** does this by calling **>FAC** and **>ARG** to place f_2 in FAC and f_1 in ARG.

The words **FADD** , **FSUB** , **FMUL** and **FDIV** each use the values in FAC and ARG and leave the result in FAC as they perform the floating point arithmetic functions.

FADD (---)

FSUB (---)

FMUL (---)

FDIV (---)

When conversion from 16-bit integer to floating point is performed by **S->F** , it is done in the FAC. If the user does not desire the result to be copied from FAC to the stack, the word **S->FAC** can be used instead:

S->FAC (*n* ---)

S->FAC moves a 16-bit integer (*n*) to the FAC, where it converts it to a floating point number.

Several miscellaneous words include:

FAC->S (--- *n*) converts the contents of FAC to a 16-bit integer on the stack.

FAC>ARG (---) copies the contents of FAC to ARG.

VAL (---)

VAL converts a string at PAD to a floating point number in FAC. **VAL** expects the first byte at PAD to be the character count. There must not be any leading spaces in the string.

FLERR (--- *n*)

FLERR is used to fetch the contents of the floating point error register (**8354h**) to the stack. See the *Editor/Assembler Manual* for more information.

?FLERR (---)

?FLERR issues an appropriate error message if the last floating point operation resulted in an error.

8 Access to File I/O Using TI-99/4A Device Service Routines

Words introduced in this chapter:

APPND	**INPT**	**RLTV**
CHAR-CNT!	**INTRNL**	**RSTR**
CHAR-CNT@	**LD**	**SCRTCH**[12]
CHK-STAT	**N-LEN!**	**SET-PAB**
CLR-STAT	**OPN**	**SQNTL**
CLSE	**OUTPT**	**STAT**
DLT	**PAB-ADDR**	**SV**
DOI/O	**PAB-BUF**	**SWCH**
DSPLY	**PAB-VBUF**	**UNSWCH**
F-D"	**PABS**	**UPDT**
FILE	**PUT-FLAG**	**VRBL**
FXD	**RD**	**WRT**
GET-FLAG	**REC-LEN**	
I/OMD	**REC-NO**	

This chapter will explain the means by which different types of data files native to the TI-99/4A are accessed with TI Forth. To further illustrate the material, two commented examples have been included in this chapter. The first (§ 8.6) demonstrates the use of a relative disk file and the second (§ 8.7) a sequential RS232 file.

A group of Forth words has been included in this version of TI Forth to permit a Forth program to reference common data with BASIC or Assembly Language programs. These words implement the file system described in the *TI BASIC Manual* and the *Editor/Assembler Manual*. Note that the diskette on which you received your TI Forth system is ***not*** a standard diskette and that you should perform file I/O to/from disks only if they are initialized by the Disk Manager and do ***not*** contain Forth screens.

8.1 The Peripheral Access Block (PAB)

Before any file access can be achieved, a Peripheral Access Block (PAB) must be set up that describes the device and file to be accessed. Most of the words in this chapter are designed to make manipulation of the PAB as easy as possible.

A PAB consists of 10 bytes of VDP RAM plus as many bytes as the device name to be accessed. An area of VDP RAM has been reserved for this purpose (consult the VDP Memory Map in Chapter 4). The user variable **PABS** points to the beginning of this region. ***Do not*** use the first 2

12 **SCRTCH** , though defined in TI Forth, was never implemented in any DSR for the TI-99/4A. Its use will result in a file I/O error.

bytes of this area as they are used by Forth in its Forth-style disk access. Adequate space is provided for many PABs in this area. More information on the details of a PAB are available in the *Editor/Assembler Manual*, page 293*ff*. The following diagram illustrates the structure of a PAB:

Byte 0 I/O Opcode	Byte 1 Flag/Status
Bytes 2 & 3 Data Buffer Address in VDP	
Byte 4 Logical Record Length	Byte 5 Character Count
Bytes 6 & 7 Record Number	
Byte 8 Screen Offset	Byte 9 Name Length
Byte 10+ File Descriptor • • •	

8.2 File Setup and I/O Variables

All Device Service Routines (DSRs) on the TI-99/4A expect to perform data transfers to/from the VDP RAM. Since Forth is using CPU RAM it means that the data will be moved twice in the process of reading or writing a file. Three variables are defined in the file I/O words to keep track of these memory areas.

PAB-ADDR (--- *addr*)

Holds address in VDP RAM of first byte of the PAB.

PAB-BUF (--- *addr*)

Holds address in CPU RAM of first byte in Forth's memory where allocation has been made for this buffer.

PAB-VBUF (--- *addr*)

Holds address in VDP RAM of the first byte of a region of adequate length to store data temporally while it is transferred between the file and Forth. The area of VDP RAM which is used for this purpose is labeled "Unused" on the VDP Memory Map in Chapter 4. If working in bit-map mode, be cautious where **PAB-VBUF** is placed.

FILE ($vaddr_1$ *addr* $vaddr_2$ ---)

The word **FILE** is a defining word and permits you to create a word which is the name by which the file will be known. A decision must be made as to the location of each of the buffers before the word **FILE** may be used. The values to be used for those locations are contained in the above variables and are placed on the stack in the above order followed by **FILE** and the file name (not necessarily the device name). For example:

Using The Defining Word, FILE

0 VARIABLE MY-BUF 78 ALLOT	(Create 80 character buffer)
PABS @ 10 +	(PAB starts 10 bytes into region for PABS: **PAB-ADDR**)
MY-BUF	(Location of **PAB-BUF**)
6000	(A free area for **PAB-VBUF**)
FILE JOE	(Whenever the word **JOE** is executed, the file I/O variables, **PAB-ADDR** , **PAB-BUF** , **PAB-VBUF** , will be set as defined here.)
JOE	(Use the word before using any other file I/O words)

SET-PAB (---)

The word that creates the PAB skeleton is **SET-PAB** . It creates a PAB at the address shown in **PAB-ADDR** and zeroes it except for the buffer address slot. Into this it places the contents of the variable **PAB-VBUF** .

8.3 File Attribute Words

Files on the TI-99/4A have various characteristics that are indicated by keywords. The following table describes the available options. The example in the back of the chapter will be helpful in that it shows at what time in the procedure these words are used. Use only the attributes which apply to your file and ignore the others. Remember, if you are using multiple files, then the file referenced is the file whose name word was most recently executed.

	Options From		
Attribute Type	**BASIC**	**Forth**	**Description**
File Type	SEQUENTIAL	**SQNTL***	Records may only be accessed in sequential order
	RELATIVE	**RLTV**	Accessed in sequential or random order. Records must be of fixed length
Record Type	FIXED	**FXD***	All records in the file are the same length
	VARIABLE	**VRBL**	Records in the same file may have different lengths

Attribute Type	Options From BASIC	Options From Forth	Description
Data Type	DISPLAY	**DSPLY***	File contains printable or displayable characters
	INTERNAL	**INTRNL**	File contains data in machine or binary format
Mode of Operation	INPUT	**INPT**	File contents can be read from but not written to
	OUTPUT	**OUTPT**	File contents can be written to but not read from
	UPDATE	**UPDT***	File contents can be written to and read from
	APPEND	**APPND**	Data may be added to end of file but cannot be read

* Default if attribute is not specified

REC-LEN (*b* ---)

To specify the record length for a file, the desired length byte *b* should be on the stack when the word **REC-LEN** is executed. The length will be placed in the current PAB.

F-D" (---)

Every file must have a name to specify the device and file to be accessed. This is performed with the **F-D"** word which enters the File Description in the PAB. **F-D"** must be followed by a string describing the file and terminated by a **"** mark. Here are a few examples of the use of **F-D"** :

```
F-D" RS232.BA=9600"
F-D" DSK2.FILE-ABC"
```

8.4 Words that Perform File I/O

The actual I/O operations are performed by the following words. The table gives the usual BASIC keyword associated with the corresponding Forth word. Here, as in the previous table, the Forth words are spelled differently than the BASIC words to avoid conflict with one or more existing Forth words.

From BASIC	From Forth	DSR Opcode
OPEN	**OPN**	0
CLOSE	**CLSE**	1
READ	**RD**	2
WRITE	**WRT**	3
RESTORE	**RSTR**	4

From BASIC	From Forth	DSR Opcode
LOAD	**LD**	5
SAVE	**SV**	6
DELETE	**DLT**	7
SCRATCH	**SCRTCH**[13]	8
STATUS	**STAT**	9

OPN (---)

opens the file specified by the currently selected PAB, which is pointed to by **PAB-ADDR** .

CLSE (---)

closes the file whose PAB is pointed to by **PAB-ADDR** .

REC-NO (*n* ---)

Before using the **RD** , **WRT** and **SCRTCH**[14] instructions with a relative file, you must place the desired, zero-based record number *n* into the PAB. To do this, place the record number *n* on the stack and execute the word **REC-NO** . If your file is Sequential, you need not do this.

RD (--- *n*)

The **RD** instruction will transfer the contents of the next record from the current file into your **PAB-BUF** and leave a character count *n* on the stack.

WRT (*n* ---)

takes a character count *n* from the stack and moves that number of characters from the **PAB-BUF** to the current file.

RSTR (*n* ---)

takes a record number *n* from the stack and repositions (restores) a relative file to that record for the next access.

SCRTCH[15] (*n* ---)

is used to remove a relative record. It requires a record number *n* on the stack.

LD (*n* ---)

used to load a program file of maximum *n* bytes into VDP RAM at the address specified in **PAB-VBUF** . **OPN** and **CLSE** need not be used.

SV (*n* ---)

used to save *n* bytes of a program file from VDP RAM at the address specified in **PAB-VBUF** . **OPN** and **CLSE** need not be used.

13 See footnote 12, page 52.

14 *idem*

15 *idem*

DLT (---)

is used to delete the file whose PAB is pointed to by **PAB-ADDR** .

STAT (--- *b*)

returns the status byte *b* (PAB+8, labeled "Screen Offset" in the PAB diagram above) of the current device/file from the PAB pointed to by **PAB-ADDR** after calling the DSR's STATUS opcode (9), which actually gets the status and writes it to PAB+8. Incidentally, the term "Screen Offset" for PAB+8 is from its use by the cassette interface, which must put prompts on the screen, to get the offset of screen characters with respect to their normal ASCII values. The table below, excerpted from the *Editor/Assembler Manual*, p. 298, shows the meaning of each bit of the status byte:

	Status Byte Information When Value is	
Bit	**1**	**0**
0	File does not exist.	File exists. If device is a printer or similar, always 0.
1	Protected file.	Unprotected file.
2		Reserved for future use. Always 0.
3	INTERNAL data type.	DISPLAY data type or program file.
4	Program file.	Data file.
5	VARIABLE record length.	FIXED record length.
6	At physical end of peripheral. No more data can be written.	Not at physical end of peripheral. Always 0 when file not open.
7	End of file (EOF). Can be written if open in APPEND, OUTPUT or UPDATE modes. Reading will cause an error.	Not EOF. Always 0 when file not open.

The words that follow are available for the advanced user and their utility can be worked out by examining their definitions on Forth screen 68*ff*. They are lower-level words that are used in the definitions of the above file I/O words.

GET-FLAG (--- *b*)

retrieves to the stack the flag/status byte *b* from the current PAB. The high-order 3 bits are used for DSR error return, except for "bad device name". With the "bad device name" error, this error return will be 0; but, the GPL status byte (**837Ch**) will have the COND bit set (**20h**). The low-order 5 bits are set by routines that set the file type prior to calling **OPN** , which reads these bits. See table below for the meaning of each bit of the flag/status byte:

Flag/Status Byte of PAB (Byte 1)

Bits	Contents	Meaning
0–2	Error Code	0 = no error. Error codes are decoded in table below.
3	Record Type	0 = fixed-length records; 1 = variable-length records.
4	Data Type	0 = DISPLAY; 1 = INTERNAL.
5–6	Mode of Operation	0 = UPDATE; 1 = OUTPUT; 2 = INPUT; 3 = APPEND.
7	File Type	0 = sequential file; 1 = relative file.

Error Codes in Bits 0–2 of Flag/Status Byte of PAB

Error Code	Meaning
0	No error unless bit 2 of status byte at address **837Ch** is set (then, bad device name).
1	Device is write protected.
2	Bad OPEN attribute such as incorrect file type, incorrect record length, incorrect I/O mode or no records in a relative record file.
3	Illegal operation; *i.e.*, an operation not supported on the peripheral or a conflict with the OPEN attributes.
4	Out of table or buffer space on the device.
5	Attempt to read past the end of file. When this error occurs, the file is closed. Also given for non-extant records in a relative record file.
6	Device error. Covers all hard device errors such as parity and bad medium errors.
7	File error such as program/data file mismatch, non-existing file opened in INPUT mode, *etc.*

PUT-FLAG (*b* ---)

writes the flag/status byte *b* on the stack to the current PAB to clear the error bits and set the file type prior to calling **OPN** . See table after **GET-FLAG** for the meaning of each bit.

CLR-STAT (---)

clears the error code in bits 0–2 of the flag/status byte of the current PAB.

CHK-STAT (---)

checks the error code in bits 0–2 of the flag/status byte of the current PAB. If it is not 0, an appropriate error message is printed.

I/OMD (--- *b*)

gets the flag/status byte *b* of the current PAB, clears the I/O mode bits (5 & 6) and leaves it on the stack in preparation for setting the I/O mode with an I/O word.

CHAR-CNT! (*n* ---)

stores the character count *n* in the current PAB prior to a write operation. **CHAR-CNT!** is used by **WRT** .

CHAR-CNT@ (--- *n*)

retrieves the character count *n* from the current PAB of the last read operation. It is used by **RD** .

N-LEN! (*b* ---)

stores in the current PAB the length byte *b* of the file descriptor associated with the current PAB. For "DSK1.MYFILE", this would be 11.

DOI/O (*n* ---)

executes the **DSRLNK** word with the I/O opcode *n* on the stack. The current PAB must be updated with the information required by opcode *n* before executing **DOI/O** . See Section 18.2.1 of the *Editor/Assembler Manual* for details or consult the definitions on Forth screen 68*ff.* of the I/O words, **OPN** , **CLSE** , **RD** , **WRT** , **RSTR** , **SCRTCH**[16] , **LD** , **SV** , **DLT** and **STAT** , all of which use this low-level word in their definitions.

Examples of file I/O in use are available on Forth screen 72*ff.*, which defines the Alternate I/O capabilities for printing to the RS232 interface.

8.5 Alternate Input and Output

When using alternate input or output devices, the 1-byte buffer in VDP memory must be the byte immediately preceding the PAB for **ALTIN** or **ALTOUT** .

The words

SWCH (---) and

UNSWCH (---)

make it possible to send output that would normally go to the monitor to an RS232 printer. For example, the LIST instruction normally outputs to the monitor. By typing

```
SWCH 45 LIST UNSWCH
```

you can list Forth screen 45 to the printer. If your RS232 printer is not on port 1 and set at 9600 baud, you must modify the word **SWCH** on your system disk.

The user variables

ALTIN (--- *vaddr*) and

ALTOUT (--- *vaddr*)

contain values which point to the current input and output devices. The value of **ALTIN** is 0 if input is coming from the keyboard, else its value is a pointer to the VDP address where the PAB for the alternate input device is located. The value of **ALTOUT** is 0 if the output is going to the monitor. Otherwise, it contains a pointer to the PAB of the alternate output device.

16 See footnote 12, page 52.

8.6 File I/O Example 1: Relative Disk File

Instruction	Comment
HEX	Change number base to hexadecimal
0 VARIABLE BUFR 3E ALLOT	Create space for a 64 byte buffer which will be the **PAB-BUF**
PABS @ A +	PAB starts 10 bytes into **PABS** . This will be the **PAB-ADDR**
BUFR 1700	Place the **PAB-BUF** and **PAB-VBUF** on stack in preparation for **FILE**
FILE TESTFIL	Associates the name **TESTFIL** with these three parameters
TESTFIL	File name must be executed before using any other File I/O words
SET-PAB	Create PAB skeleton
RLTV	Make **TESTFIL** a relative file
DSPLY	Records will contain printable information
40 REC-LEN	Record length is 64 (**40h**) bytes
F-D" DSK2.TEST"	Will create the file descriptor "DSK2.TEST" in the PAB for **TESTFIL** .
OPN	Open the file. This will create the file on disk unless it already exists.

To write more than one record to the file, it is necessary to write a procedure. This routine may be composed on a Forth screen beforehand and loaded at this time.

: FIL-WRT TESTDATA	**TESTDATA** is assumed to be the beginning memory address of the information to be written to the file
10 0 DO	Want to write 16 (**10h**) records
DUP	Duplicate address
BUFR 40 CMOVE	Move 64 bytes of information into the **PAB-BUF**
I REC-NO	Place record number into PAB
40 WRT	Write one 64-byte record to the disk
40 +	Increment address for next record
LOOP DROP	Clear stack
;	End definition
FIL-WRT	Execute writing procedure
4 REC-NO RD	Choose a record number to read (4 is chosen here) to verify correct output. A byte count will be left on the stack and the read information will be in **BUFR**
BUFR 40 DUMP	Print out the read information to the monitor. (**DUMP** routines must be loaded)
CLSE	Close the file

8.7 File I/O Example 2: Sequential RS232 File

Instruction	Comment
HEX	Change number base to hexadecimal
0 VARIABLE MY-BUF 4E ALLOT	Create a 80 character **PAB-BUF**
PABS @ 30 +	Skip all previous PAB. This will be the **PAB-ADDR**
MY-BUF 1900	Place the **PAB-BUF** and **PAB-VBUF** on stack in preparation for **FILE**
FILE PRNTR	Associates the name **PRNTR** with these three parameters
PRNTR	File name must be executed before using any other File I/O words
SET-PAB	Create a PAB skeleton
DSPLY	**PRNTR** will contain printable information
SQNTL	**PRNTR** may be accessed only in sequential order
VRBL	Records may have variable lengths
50 REC-LEN	Maximum record length is 80 char.
F-D" RS232.BA=9600"	**PRNTR** will be an RS232 file. Baud rate = 9600.
OPN	Open the file

A procedure is necessary to write more than one record to a file. A file-write routine may be composed on a Forth screen beforehand and loaded at this time. The following is a simple example:

: PRNT FILE-INFO	**FILE-INFO** is assumed to be the beginning memory address of the information to be sent to the printer
20 0 DO	Will write 32 records
DUP	Duplicate address
MYBUF 50 CMOVE	Move 80 characters from **FILE-INFO** to **MY-BUF**
50 WRT	Write one record to printer
50 +	Increment address on stack
LOOP DROP	Clear stack
;	End definition
PRNT	Execute write program
CLSE	Close the file called **PRNTR**

9 The TI Forth 9900 Assembler

The assembler supplied with your TI Forth system is typical of assemblers supplied with fig-Forth systems. It provides the capability of using all of the opcodes of the TMS9900 as well as the ability to use structured assembly instructions. It uses no labels. The complete Forth language is available to the user to assist in macro type assembly, if desired. The assembler uses the standard Forth convention of Reverse Polish Notation for each instruction. For example the instruction to add register 1 to register 2 is:

```
1 2 A,
```

As can be seen in the above example, the 'add' instruction mnemonic is followed by a comma. Every opcode in this Forth assembler is followed by a comma. The significance is that when the opcode is reached during the assembly process, the instruction is compiled into the dictionary. The comma is a reminder of this compile operation. It also serves to assist in differentiating assembler words from the rest of the words in the TI Forth language. A complete list of Forth-style instruction mnemonics is given in the next section.

9.1 TMS9900 Assembly Mnemonics

```
A,          JEQ,        RSET,
AB,         JGT,        RTWP,
ABS,        JH,         S,
AI,         JHE,        SB,
ANDI,       JL,         SBO,
B,          JLE,        SBZ,
BL,         JLT,        SETO,
BLWP,       JMP,        SLA,
C,          JNC,        SOC,
CB,         JNE,        SOCB,
CI,         JNO,        SRA,
CKOF,       JOC,        SRC,
CKON,       JOP,        SRL,
CLR,        LDCR,       STCR,
COC,        LI,         STST,
CZC,        LIMI,       STWP,
DEC,        LREX,       SWPB,
DECT,       LWPI,       SZC,
DIV,        MOV,        SZCB,
IDLE,       MOVB,       TB,
INC,        MPY,        X,
INCT,       NEG,        XOP,
INV,        ORI,        XOR,
```

These words are available when the assembler is loaded. Only the word **C,** conflicts with the existing Forth vocabulary.

Most assembly code in Forth will probably use Forth's workspace registers. The following table describes the register allocation. The user may use registers 0 through 7 for any purpose. They are used as temporary registers only within Forth words which are themselves written in TMS9900 assembly code.

9.2 Forth's Workspace Registers

Register Name	Usage
0	These registers are available. They are used only within Forth words written in **CODE** .
1	
2	
3	
4	
5	
6	
7	
UP	Points to base of User Variable area
SP	Parameter Stack Pointer
W	Inner Interpreter current Word pointer
11	Linkage for subroutines in **CODE** routines
12	Used for CRU instructions
IP	Interpretive Pointer
RP	Return Stack Pointer
NEXT	Points to the next instruction fetch routine

9.3 Loading and Using the Assembler

The TI Forth TMS9900 Assembler is located on Forth screens 75 – 82 and is loaded by typing the menu word **-ASSEMBLER** . Loading the assembler first ensures that **CODE** and **;CODE** are in the dictionary and loads Forth screen 74 if they are not. When the assembler is loaded, it is loaded into the Assembler vocabulary. To use the assembler, it must be the context vocabulary, which may be effected by typing **ASSEMBLER** or by using the words **CODE** or **;CODE** , each of which makes Assembler the context vocabulary. After defining words that use **CODE** or **;CODE** , it is advisable to execute **FORTH** to restore the context vocabulary to Forth, unless such use is immediately followed by **:** (beginning a colon definition), which restores the context vocabulary to the current vocabulary (usually Forth). The important point is that Forth must be the context vocabulary before the Forth word **C,** is intended because **C,** is the only Assembler vocabulary word that conflicts with a Forth vocabulary word of the same name.

Assembly definitions either begin with **CODE** or end with **;CODE** . Each are followed by assembly mnemonics or the machine-code equivalent. **CODE** is used in the following way:

```
CODE EXAMPLE <assembly mnemonics>
```

This defines a Forth word named **EXAMPLE** with an execution procedure defined by the assembly mnemonics that follow **EXAMPLE** . The assembly code should end with **NEXT,** so the TI Forth interpreter can get to the next word in the input stream. There are several examples using **CODE** in the sections that follow.

;CODE is used with **<BUILDS** to create the execution procedure of a new defining word very much like the word **DOES>** except that **;CODE** does not cause the PFA of newly defined words to be left on the stack for the consumption of the code following **;CODE** as is the case with **DOES>** . **;CODE** is used as follows:

```
: DEF-WRD <BUILDS … ;CODE <assembly mnemonics>
```

Just as with **CODE** , assembly code following **;CODE** should end with **NEXT,** . Later when the newly created defining word **DEF-WRD** is executed in the following form, a new word is defined:

```
DEF-WRD TEST
```

This will create the word **TEST** which has as its execution procedure the code following **;CODE** . An example using **;CODE** is shown in § 9.9 .

9.4 TI Forth Assembler Addressing Modes

We will now introduce those words that permit this assembler to perform the various addressing modes of which the TMS9900 is capable. Each of the remaining examples will show both the Forth assembler code for various instructions and the more conventional method of coding the same instructions.

The word **NEXT,** is defined as (see § 9.4.6 for definition of ***NEXT**)

```
: NEXT, *NEXT B, ;
```

and is equivalent to the following assembly code:

```
B     *R15
```

9.4.1 Workspace Register Addressing

The registers in the Forth code below are referenced directly by number:

Forth	Conventional Assembler		
CODE EX1		DEF	EX1
1 2 A,	EX1	A	R1,R2
3 INC,		INC	R3
3 FFFC ANDI,		ANDI	R3,>FFFC
NEXT,		B	*R15

9.4.2 Symbolic Memory Addressing

Symbolic addressing is done with the **@()** word. It is used after the address.

Forth	Conventional Assembler		
`0 VARIABLE VAR1`	VAR1	BSS	2
`5 VARIABLE VAR2`	VAR2	DATA	5
`CODE EX2`		DEF	EX2
`VAR2 @() 1 MOV,`	EX2	MOV	@VAR2,R1
`1 2 SRC,`		SRC	R1,2
`1 VAR1 @() S,`		S	R1,@VAR1
`VAR2 @() VAR1 @() SOC,`		SOC	@VAR2,@VAR1
`NEXT,`		B	*R15

9.4.3 Workspace Register Indirect Addressing

Workspace Register Indirect addressing is done with the ***?** word. It is used after the register number to which it pertains.

Forth	Conventional Assembler		
`HEX 2000 CONSTANT XRAM`	XRAM	EQU	>2000
`CODE EX3`		DEF	EX3
`1 XRAM LI,`	EX3	LI	R1,XRAM
`1 *? 2 MOV,`		MOV	*R1,R2
`NEXT,`		B	*R15

9.4.4 Workspace Register Indirect Auto-increment Addressing

Workspace Register Indirect Auto-increment addressing is done with the ***?+** word. It is also used after the register to which it pertains.

Forth	Conventional Assembler		
`HEX 2000 CONSTANT XRAM`	XRAM	EQU	>2000
`CODE EX4`		DEF	EX4
`1 XRAM LI,`	EX4	LI	R1,XRAM
`1 *?+ 2 MOV,`		MOV	*R1+,R2
`NEXT,`		B	*R15

9.4.5 Indexed Memory Addressing

The final addressing type is Indexed Memory addressing. This is performed with the **@(?)** word used after the Index and register as shown below:

Forth	Conventional Assembler		
`HEX 2000 CONSTANT XRAM`	`XRAM`	`EQU`	`>2000`
`CODE EX5`		`DEF`	`EX5`
`XRAM 1 @(?) 2 MOV,`	`EX5`	`MOV`	`@XRAM(R1),R2`
`DECIMAL`			
`XRAM 22 + 2 @(?)`		`MOV`	`@XRAM+22(R2),@XRAM+26(R2)`
`XRAM 26 + 2 @(?) MOV,`			
`NEXT,`		`B`	`*R15`

9.4.6 Addressing Mode Words for Special Registers

In order to make addressing modes easier for the **W** , **RP** , **IP** , **SP** , **UP** and **NEXT** registers, the following words are available and eliminate the need to enter the register name separately:

Register Address	Indirect	Indexed	Indirect Auto-increment
`W`	`*W`	`@(W)`	`*W+`
`RP`	`*RP`	`@(RP)`	`*RP+`
`IP`	`*IP`	`@(IP)`	`*IP+`
`SP`	`*SP`	`@(SP)`	`*SP+`
`UP`	`*UP`	`@(UP)`	`*UP+`
`NEXT`	`*NEXT`	`@(NEXT)`	`*NEXT+`

9.5 Handling the Forth Stacks

Both the parameter stack and the return stack grow downward in memory. This means that removing a cell from the top of either stack requires *incrementing* the stack pointer after consuming the cell's value. Conversely, adding a cell requires *decrementing* the stack pointer. The Forth Assembler word ***SP+** references the contents of the top cell of the parameter stack and then increments the stack pointer **SP** to reduce the size of the stack by one cell. The following code copies the contents of the stack's top cell to register 0 and reduces the stack by one cell:

```
*SP+ 0 MOV,
```

The following code adds a cell to the top of the stack and copies the contents of register 1 to the new cell:

```
SP DECT,
1 *SP MOV,
```

The same procedures obtain for the return stack using ***RP+** , **RP** and ***RP** ; but, be very careful if you must manipulate the return stack.

9.6 Structured Assembler Constructs

This assembler also permits the user to write structured code, *i.e.*, code that does not use labels. This is done in a manner very similar to the way that Forth implements conditional constructs. The major difference is that rather than taking a value from the stack and using it as a true/false flag, the processor's condition register is used to determine whether or not to jump. The following structured constructs are implemented:

```
IF, … ENDIF,
IF, … ELSE, … ENDIF,
BEGIN, … UNTIL,
BEGIN, … AGAIN,
BEGIN, … WHILE, … REPEAT,
```

The three conditional words in the previous list (**IF,** , **UNTIL,** and **WHILE,**) must each be preceded by one of the jump tokens in the next section.

9.7 Assembler Jump Tokens

Token	Comment	Conventional Assembler Used	Machine Code Generated
EQ	True if =	**JNE**	**1600h**
GT	True if signed >	**JGT $+1 JMP**	**1501h 1000h**
GTE	True if signed > or =	**JLT**	**1100h**
H	True if unsigned >	**JLE**	**1200h**
HE	True if unsigned > or =	**JL**	**1A00h**
L	True if unsigned <	**JHE**	**1400h**
LE	True if unsigned < or =	**JH**	**1B00h**
LT	True if signed <	**JLT $+1 JMP**	**1100h 1000h**
LTE	True if signed < or =	**JGT**	**1500h**
NC	True if No Carry	**JOC**	**1800h**
NE	True if equal bit not set	**JEQ**	**1300h**
NO	True if No overflow	**JNO $+1 JMP**	**1901h 1000h**
NP	True if Not odd Parity	**JOP**	**1C00h**
OC	True if Carry bit is set	**JNC**	**1700h**
OO	True if Overflow	**JNO**	**1900h**
OP	True if Odd Parity	**JOP $+1 JMP**	**1C00h 1000h**

9.8 Assembly Example for Structured Constructs

The following example is designed to show how these jump tokens and structured constructs are used:

Forth	Conventional Assembler		
`( GENERALIZED SHIFTER )`	`* GENERALIZED SHIFTER`		
`CODE SHIFT`		`DEF`	`SHIFT`
`*SP+ 0 MOV,`	`SHIFT`	`MOV`	`*SP+,R0`
`NE IF,`		`JEQ`	`L3`
`*SP 1 MOV,`		`MOV`	`*SP,R1`
`0 ABS,`		`ABS`	`R0`
`GTE IF,`		`JLT`	`L1`
`1 0 SLA,`		`SLA`	`R1,0`
`ELSE,`		`JMP`	`L2`
`1 0 SRL,`	`L1`	`SRL`	`R1,0`
`ENDIF,`			
`1 *SP MOV,`	`L2`	`MOV`	`R1,*SP`
`ENDIF,`			
`NEXT,`	`L3`	`B`	`*R15`

One word of caution is in order. The structured constructs shown above do not check to ensure that the jump target is within range (+127, -128 words). This will be a problem only with very large assembly language definitions and will violate the Forth philosophy of small, easily understood words.

9.9 Assembly Example Using ;CODE

Before giving an example of defining a TI Forth defining word with **;CODE** , an explanation of why you might want to use it in the first place is in order.

The defining words that are part of the TI Forth kernel are **:** (paired with **;**), **VARIABLE** , **CONSTANT** , **USER** , **VOCABULARY** , **<BUILDS** (paired with **DOES>** or **;CODE** [not in the kernel]) and **CREATE** . The defining words **CODE** and **;CODE** are defined on the system disk and must be loaded with **-CODE** or **-ASSEMBLER** to be used. Of course, Most words you would ever need to define can be created with the first three (**:** , **VARIABLE** and **CONSTANT**). However, you too can use **<BUILDS** and **CREATE** , the same words used for defining most of the above, for the eventuality that these do not suffice.

In TI Forth, it is not useful to use **CREATE** on the command line unless you really know what you are doing because it creates a dictionary header in which the smudge bit is set and the code field points at the parameter field with no storage allotted for it. This means that the parameter field must be allotted with executable code (or the code field changed to point to some) and the smudge bit must be reset so a dictionary search can find the word. The same discussion obtains for **<BUILDS** except for the smudge bit because **<BUILDS** is defined in TI Forth as

```
: <BUILDS CREATE SMUDGE ;
```
(**SMUDGE** toggles the smudge bit.)

This situation is made easier by using **<BUILDS** , **DOES>** and **;CODE** within colon definitions as

```
: NEW_DEFINING_WORD <BUILDS … DOES> … ;
```

or

```
: NEW_DEFINING_WORD <BUILDS … ;CODE …
```

You simply replace the first "…" with words you want to execute when **NEW_DEFINING_WORD** is compiling a new word, *e.g.*, to reserve space for and store a value in the first cell of the parameter field using **,** . You then replace the second "…" with code to be executed when the new word actually executes. It will be this code to which the code field of the new word will point.

Here, now, is an example of the use of **;CODE** in the definition of a defining word, *i.e.*, a word that creates new words:

CONSTANT is a TI Forth word that defines a word, the value of which is pushed to the stack when the word is executed.

```
9 CONSTANT XXX
```

defines the word **XXX** with 9 in its parameter field and the address of the execution code of **CONSTANT** in its code field. TI Forth defines **CONSTANT** in high-level Forth essentially as

```
: CONSTANT <BUILDS , DOES> @ ;
```

Using **;CODE** , it could also be defined with Assembler code as

: CONSTANT	Start colon definition of **CONSTANT** .
<BUILDS	**CONSTANT** will create a dictionary header for the word appearing after it in the input stream when **CONSTANT** is executed. The new word's CFA will point to the address immediately following the CFA. This will be the new word's PFA, but no space will be allocated for the PFA.
,	Expects a number on the stack, which it will store at the PFA of the new word, allocating space for it.
;CODE	The new word's CFA will be changed to point to machine code that follows **;CODE** here in **CONSTANT** . The following machine code is what will run when the new word is executed:
SP DECT,	Make space on the stack.
***W *SP MOV,**	Copy current (newly defined) word's parameter field contents to the stack. [**W** (R10) contains the current word's PFA.]
NEXT,	Return to the interpreter.

which, once you know the machine code, can be coded without the Assembler loaded as

```
HEX
: CONSTANT <BUILDS , ;CODE 0649 , C65A , 045F ,
```

For **CONSTANT** , the first, high-level definition is easier to understand. They are both the same length. They both create words of the same length. However, there may come a time when only Assembler will do your bidding and **;CODE** offers that facility.

9.10 Using CODE and ;CODE without the Assembler

TI Forth words using **CODE** or **;CODE** can be written without the 2250-byte overhead of the TI Forth Assembler by using the machine code equivalent to assembly code. The editor may well write a TI Forth program soon to do the dirty work; but, for now you must endure the painful procedure below to get the job done. Until you have tested and debugged your work, it is probably best to work with one Forth word at a time on a Forth screen.

1. Write, test and debug your Forth word using the TI Forth Assembler. Here, we'll use **EX5** from § 9.4.5 for the **CODE** example and **CONSTANT** (renamed **CONST2** to avoid confusion) from § 9.9 for the **;CODE** example.
2. Ensure that the TI Forth Assembler is loaded.
3. Ensure that the dump routines are loaded by executing **-DUMP** .
4. Load the screen that contains the definition of your Forth word and continue with (5) in the appropriate section below.

9.10.1 CODE without the Assembler

Refer to the example in § 9.4.5 for the following:

5. Use **'** to find the PFA of **EX5** and dump from the PFA to the end of the word:

   ```
   HERE ' EX5 SWAP OVER - DUMP
   ```

   will dump this to the screen:

   ```
   E42C  C0A1  2000 C8A2  2016  .. ... .
   E434  201A  045F              ..
   ok
   ```

   The column at the left indicates the addresses in RAM where the hexadecimal cells to the right are located. The 8-character, right-hand column is their ASCII representation.
6. The last cell should be **045Fh**, corresponding to the **NEXT,** instruction.
7. Write the high-level part of the word (**CODE EX5**) followed by the machine code after **EX5** using the dump above to compile the hexadecimal value for each cell with **,** starting with the first cell (parameter field) and ending with **045Fh** as follows:

   ```
   HEX
   CODE EX5 C0A1 , 2000 , C8A2 , 2016 , 201A , 045F ,
   ```
8. If all the code was assembly code, you're done. Otherwise, you need to replace values that can vary from one load to the next, such as variables, named constants and dictionary entries, with the high-level code used in the word's assembly language definition. In the above example, the constant **XRAM** was used, so we need to replace the value **2000h** with the reference that put it there. In this case **XRAM** is used three times to get the cells with **2000h**, **2016h** and **201Ah**. We need to replace the **2000h** with **XRAM** , the **2016h** with **XRAM 16 +** and the **201Ah** with **XRAM 1A +** to get

   ```
   HEX
   CODE EX5 C0A1 , XRAM , C8A2 , XRAM 16 + , XRAM 1A + , 045F ,
   ```

   which can now be entered on a Forth screen to be loaded with only **CODE** in the dictionary (use **-CODE** to ensure it's loaded) and without the Assembler overhead.
9. You should test your new version of the word to verify it is identical to the original assembly version.

9.10.2 ;CODE without the Assembler

We need to do more work with **;CODE** than we did with **CODE** above. We must find the CFA of **(;CODE)** that **;CODE** compiled into our word and retrieve the machine code that follows it. Refer to the example in § 9.9 (which we've renamed here as **CONST2** to avoid confusion) for the following:

5. Use **'** and **CFA** to find the CFA of **(;CODE)** so you can find the cell within the definition of **CONST2** that contains it:

   ```
   HEX ' (;CODE) CFA U.
   ```

   will display this on the screen:

   ```
   BA6A  ok
   ```

6. Use **'** to find the PFA of **CONST2** and dump from the PFA to the end of the word:

   ```
   HERE ' CONST2 SWAP OVER - DUMP
   ```

   will dump this to the screen:

   ```
   E424  B998  A992  BA6A 0649  .....j.I
   E42C  C65A  045F             .Z._
   ok
   ```

   The column at the left indicates the addresses in RAM where the hexadecimal cells to the right are located. The 8-character, right-hand column is their ASCII representation.

7. The last cell should be **045Fh**, corresponding to the **NEXT,** instruction.

8. Write the high-level part of the word through **;CODE** followed by the machine code after **BA6Ah** [the CFA of **(;CODE)** we found above in (5)]. Use the dump above for guidance to compile with **,** the hexadecimal value for each cell as follows:

   ```
   HEX
   : CONSTANT <BUILDS , ;CODE 0649 , C65A , 045F ,
   ```

   which can now be entered on a Forth screen to be loaded with only **;CODE** in the dictionary (use -**CODE** to ensure it's loaded) and without the Assembler overhead.

9. If all the code was assembly code, as it is here, you're done. Otherwise, you need to replace values that can vary from one load to the next, such as variables, named constants and dictionary entries, with the high-level code used in the word's assembly language definition. See (8) in § 9.10.1 for an example with a named constant.

10. You should test your new version of the word to verify it is identical to the original assembly version.

10 Interrupt Service Routines (ISRs)

The TI-99/4A has the built-in ability to execute an interrupt routine every 1/60 second. This facility has been extended by the TI Forth system so that the routine to be executed at each interrupt period may be written in Forth rather than in assembly language. This is an advanced programming concept and its use depends on the user's knowledge of the TI-99/4A.

The user Variables **ISR** and **INTLNK** are provided to assist the user in using ISRs. Initially, they each contain the address of the link to the Forth ISR handler, To correctly use User Variable **ISR** the following steps should be followed:

10.1 Installing a Forth Language Interrupt Service Routine

1) Create and test a Forth routine to perform the function.
2) Determine the Code Field Address (**CFA**) of the routine in (1).
3) Write the **CFA** from (2) into **ISR** .
4) Write the contents of **INTLNK** into **83C4h** (33732).

The ISR linkage mechanism is designed so that your interrupt service routine will be allowed to execute immediately after each time the Forth system executes the "NEXT" instruction (as it does at the end of each code word). In addition, the **KEY** routine has been coded so that it also executes "NEXT" after every keyscan whether or not a key has been pressed. The "NEXT" instruction is actually coded in TI Assembler as "**B *NEXT**" or "**B *R15**" because workspace register 15 (**R15** or **NEXT**) contains the address of the next instruction to be executed. This executes the same procedure as the TI Forth Assembler word **NEXT,** (see Chapter 9).

Before installing an ISR you should have some idea of how long it takes to execute, keeping in mind that for normal behavior it should execute in less than 16 milliseconds. ISRs that take longer than that may cause erratic sprite motion and sound because of missed interrupts. In addition it is possible to bring the Forth system to a slow crawl by using about 99% of the processor's time for the ISR.

The ISR capability has obvious applications in game software as well as for playing background music or for spooling screens from disk to printer while other activities are taking place. This final application will require that disk buffers and user variables for the spool task be separate from the main Forth task or a very undesirable cross-fertilization of buffers may result. In addition it should be mentioned that disk activity causes all interrupt service activity to halt.

ISRs in Forth can be written as either colon definitions or as **CODE** definitions. The former permits very easy routine creation, and the latter permits the same speed capabilities as routines created by the Editor/Assembler. Both types can be used in a single routine to gain the advantages of both.

10.2 An Example of an Interrupt Service Routine

An example of a simple ISR is given below. This example also illustrates some of the problems associated with ISRs and how they can be circumvented. The problems are:

1) A contention for PAD between a normal Forth command and the ISR routine.
2) Long execution time for the ISR routine. (Even simple routines, especially if they include output conversion routines or other words that nest Forth routines very deeply, will not complete execution in 1/60 second.)

These problems are overcome by moving PAD in the interrupt routine to eliminate the interference between the foreground and the background task. The built-in number formatting routines are quite general and hence pay a performance penalty. This example performs this conversion rather crudely, but fast enough that there is adequate time remaining in each 1/60 second to do meaningful computing.

```
0 VARIABLE TIMER                  (TIMER will hold the current count)
: UP 100 ALLOT ;                  (move HERE and thus PAD up 100 bytes)
: DOWN -100 ALLOT DROP[17] ;      (restore PAD to its original location)
: DEMO UP                         (move PAD to avoid conflict)
    1 TIMER +! TIMER @            (increment TIMER , leave on stack)
    PAD DUP 5 +                   (ready to loop from PAD + 5 down to PAD + 1)
    DO
      0 10 U/                     (make positive double, get 1st digit)
      SWAP 48 +                   (generate ASCII digit)
      I C!                        (store to PAD )
    -1 +LOOP                      (decrement loop counter)
    PAD 1+ SCRN_START @ 5 VMBW    (write to screen)
    DOWN ;                        (restore PAD location)
```

10.3 Installing the ISR

To install this ISR the following code may be executed:

```
INTLNK @          (get the ISR 'hook' to the stack)
' DEMO CFA        (get CFA of the word to be installed as ISR)
ISR !             (place it in user variable ISR )
HEX 83C4 !        (put ISR 'hook' into console interrupt service routine)
                  (Note: the CFA must be in user variable ISR before
                    writing to 83C4h)
```

17 Bug Fix: See Appendix J for the source of the fix. It might be clearer why **DROP** is necessary if it were placed after **+LOOP** instead of in the definition of **DOWN** : After the first pass through the loop in **DEMO** , the remainder from **U/** is consumed, but the quotient is left for the next pass through the loop and, of course, remains on the stack when the loop exits. **DROP** cleans up the stack.

To reverse the installation of the ISR one can either write a 0 to **83C4h** or place the **CFA** of **NOP** (a do-nothing instruction) in user variable **ISR** .

10.4 Some Additional Thoughts Concerning the Use of ISRs

ISRs are uninterruptible. Interrupts are disabled by the code that branches to your ISR routine and they are not enabled until just before branching back to the foreground routine. *Do not enable interrupts in your interrupt routine.*

1) Caution must be exercised when using PABs, changing user variables or using disk buffers in an ISR, as these activities will likely interfere with the foreground task unless duplicate copies are used in the two processes.
2) An ISR must never expect nor leave anything on the stacks. It may however use them in the normal manner during execution.
3) Disk activity disables interrupts as do most of the other DSRs in the TI-99/4A. An ISR that is installed will not execute during the time interval in which disk data transfer is active. It will resume after the disk is finished. Note that it is possible to **LOAD** from disk while the ISR is active. It will wait for about a second each time the disk is accessed. The dictionary will grow with the resultant movement of **PAD** without difficulty.

11 Potpourri

Your TI Forth system has a number of additional features that will be discussed in this chapter. These include a facility to save and load binary images of the dictionary so that applications need not be recompiled each time they are used. Also available are a group of CRU (Communications Register Unit) instructions and a version of **MESSAGE** that does not require a disk to display the standard error messages.

11.1 BSAVE and BLOAD

BSAVE (*addr* scr_1 --- scr_2)

The word **BSAVE** is used to save binary images of the dictionary. **BSAVE** requires two entries on the stack:

1) The lowest memory address *addr* in the dictionary image to be saved to disk.
2) The Forth screen number scr_1 to which the saved image will be written.

BSAVE will use as many Forth screens as necessary to save the dictionary contents from the address given on the stack to **HERE**. These are saved with 1000 bytes per Forth screen until the entire image is saved. **BSAVE** returns on the stack the number scr_2 of the first available Forth screen after the image.

Each Forth screen of the saved image has the following format:

Byte #	Contents
0–1	Address at which the first image byte of this Forth screen will be placed.
2–3	DP for this memory image.
4–5	Contents of **CURRENT** .
6–7	Contents of **CURRENT @** .
8–9	Contents of **CONTEXT** .
10–11	Contents of **CONTEXT @** .
12–13	Contents of **VOC-LINK** .
14	The letter ‘t’.
15	The letter ‘i’.
16–23	Not used.
24–1023	Up to 1000 bytes of the memory image.

BLOAD (*scr* --- *flag*)

BLOAD is part of your TI Forth kernel and does not have to be loaded before you can use it. It reverses the **BSAVE** process and makes it possible to bring in an entire application in seconds. **BLOAD** expects a Forth screen number *scr* on the stack. Before performing the **BLOAD** function the 14th and 15th bytes are checked to see that they contain the letters “ti”.

If they do, the load proceeds and **BLOAD** returns a flag of 0 on the stack signifying a successful load. If the letters "ti" are not found, then the **BLOAD** is not performed and a flag of 1 is returned. This facility permits a conditional binary load to be performed and if it fails (wrong disk, *etc*.), other actions can be performed.

Because the **BLOAD** and **BSAVE** facility is designed to start the save (and hence the load) at a user-supplied address, a complete overlay structure can be implemented. ***Very important:*** The user must ensure that when part of the dictionary is brought in, the remainder of the dictionary (older part) is identical to that which existed when the image was saved.

11.1.1 Customizing How TI Forth Boots Up

You may find that you use the same **MENU** choices frequently and would like to load them automatically and quickly each time you boot TI Forth. You can do this by using the Forth word **TASK** as a reference point for **BSAVE** . A no-operation word or null definition, **TASK** is the last word defined in the resident Forth vocabulary of TI Forth and the last word that *cannot* be forgotten using **FORGET** . Its definition is simply

```
: TASK ;
```

Its address can be used to **BSAVE** a personalized TI Forth system disk by using **' TASK** as the address on the stack for **BSAVE** . If part of your personalized system includes the 64-column editor, you can use the 9 screens starting with screen 21 to save your system image:

```
' TASK 21 BSAVE .
```

(*Be sure to back up the original disk before trying this!*). It is important that you ensure that this procedure does not compromise Forth system screens you may need for your new personalized system. The **.** after **BSAVE** will report the next available screen from the value left on the stack. Subtracting 21 from that number will tell you how many screens it took to save the binary image in the above **BSAVE** line.

You now need to add the code to load what you have just saved the next time you boot your system. You can also do a little housecleaning by erasing superfluous material from screens 3 and 20:

On Forth screen 3:

- Erase lines 3 – 11. These definitions will be redundant.
- Replace **20 LOAD** on line 2 with **21 BLOAD** to load the rest of the system from **TASK** forward the next time you boot up TI Forth.

On Forth screen 20:

- Erase lines 0 – 8.
- On line 0, put something like: **(MENU CHOICES)** , to indicate the purpose of lines 9 – 15. You need to keep those lines because **MENU** will list them to the screen regardless of how they read.

11.1.2 An Overlay System with BSAVE/BLOAD

As mentioned above, you can implement a complete overlay structure using **BSAVE** and **BLOAD** . It can be a bit tedious to set up, however, because you must ensure that the dictionary structure older than what you load with **BLOAD** is identical to what it was when the binary image was saved with **BSAVE** . If your application always uses **TASK** as the reference point, as in the previous section, for saving and loading all overlays you set up for your application, the situation is actually pretty simple. If, on the other hand, you wish to have the most efficiently running application possible with minimum load/reload times, you will want to load as overlays only those parts of your application that can be considered mutually exclusive or, at least, not redundant functions.

Such an application might be set up as follows:

1. Anticipate screens where overlays will be saved with **BSAVE** .
2. Set up storage (variables, arrays, ...) that is common to two or more overlays.
3. Set up the overlay-loading mechanism in your application to use **BLOAD** to load them. The following example illustrates such a mechanism using the **CASE … ENDCASE** construct:

```
0 VARIABLE OVLY ( track current ovly# )
: OVLY_LD ( ovly# --- )
      DUP
      CASE
         1 OF 120 BLOAD ENDOF
         2 OF 130 BLOAD ENDOF
         3 OF 140 BLOAD ENDOF
         ( no overlay change if we get here!)
         -1 SWAP ( ENDCASE will DROP top number)
      ENDCASE
      ( 2 cells to here unless fell thru. Top cell: -1|0|1)
      CASE
          -1 OF ." No choice for overlay " . CR ENDOF
           0 OF OVLY ! ENDOF ( Success! Save new #)
           1 OF ." Failed to load overlay " . CR ENDOF
      ENDCASE ;
```

4. Program a method for determining which overlay is needed for a particular function or set of functions and use **OVLY** to determine whether that overlay needs to be loaded.
5. As the last word of your application before any overlays, define **OVERLAYS** as a null definition to be a reference point for **BSAVE** and make it unforgettable:

```
: OVERLAYS ;
' OVERLAYS NFA FENCE !
```

6. Begin each overlay with the following null definition as a **FORGET** reference point for loading the next overlay source screen prior to saving its binary image with **BSAVE** :

```
: OVLY_STRT ;
```

7. After the successful load (with **BLOAD**) of an overlay, set **OVLY** to its number as in the example in (3) above.

After programming and debugging the application, save the application and its overlays as follows:

1. Remove all system components from the dictionary that are not required by your application and that are newer than **TASK** . To start with a dictionary with only resident words:
   a) Execute **-DUMP** to load the definition for **VLIST** .
   b) Execute **VLIST** to get the name of the word immediately following **TASK** . Remember that **VLIST** lists the dictionary from **HERE** back to older words.
   c) **FORGET** that word to leave only the resident dictionary. If the word following **TASK** , *i.e.*, listed just before **TASK** by **VLIST** , is **XXX** , then execute **FORGET XXX** .
2. Load all system components required to run your application. At the very least, you will need to load **-BSAVE** to use **BSAVE** to save the binary images for your application and its overlays, even though your application will never need it.
3. Load application.
4. Load first overlay.
5. **BSAVE** application using the address of **TASK** to a free Forth screen:

```
' TASK 110 BSAVE .
```

6. **BSAVE** first overlay using the address of **OVERLAYS** to a free Forth screen:

```
' OVERLAYS 120 BSAVE .
```

7. For each overlay following the first do the following:
   a) **FORGET OVLY_STRT**
   b) **100 LOAD** (100 should be where the Forth screen for next overlay resides.)
   c) **' OVERLAYS 130 BSAVE .** (Obviously, 130 should be a different screen for each additional overlay.)

11.1.3 An Easier Overlay System in Source Code

The above **BSAVE/BLOAD** method for setting up an overlay system can be very difficult to maintain because of the unforgiving nature of **BLOAD** . Any changes in the application other than the overlay section will almost certainly necessitate re-saving *all* of the overlays. An easier method to maintain is one such as described in *Starting FORTH (1st Ed.)*, p. 80*ff*. It will be necessarily slower to load overlays because it uses source screens. You can still save a binary image of the application as above with the first, presumably most used, overlay to minimize load time; but, it still may be better for software changes to **BSAVE** the application without an overlay.

Because you are not using **BSAVE** to save the overlays, you can dispense with one of the null definitions. Let us say you are using **OVERLAYS** , as the word to **FORGET** each time another overlay is loaded. **OVERLAYS** will now separate the main application from the current overlay and should, of course, be the last word of the main application. **OVERLAYS** should obviously not be made unforgettable! The first Forth screen of each overlay should begin with

```
FORGET OVERLAYS          : OVERLAYS ;
```

You can use the same mechanism (**OVLY_LD**) as in the previous section for loading the overlays; but, you will need to change all instances of **BLOAD** to **LOAD** and, of course, the screens will be text screens, not binary images. You will also need to change the code that expects a flag on the stack from **BLOAD** because **LOAD** does not leave a flag.

11.2 Conditional Loads

CLOAD (*scr* ---)

The word **CLOAD** has been included in your system to assist in easily managing the process of loading the proper support routines for an application without compiling duplicates of support routines into the dictionary.

CLOAD calls the words **<CLOAD>** , **WLITERAL** , and **SLIT** . Their functions are described briefly as follows:

<CLOAD> (---)

performs the primary **CLOAD** function and is executed or compiled by **CLOAD** depending on **STATE** .

SLIT (--- *addr*)

is a word designed to handle string literals during execution. Its purpose is to put the address of the string on the stack and step the Forth Instruction Pointer over it.

WLITERAL (---)

is used to compile **SLIT** and the desired character string into the current dictionary definition. See the TI Forth Glossary (Appendix D) for more detail.

To use **CLOAD** , there must always be a Forth screen number on the stack. The word **CLOAD** must be followed by the word whose conditional presence in the dictionary will determine whether or not the Forth screen number on the stack is loaded.

```
27 CLOAD FOO
```

This instruction, for example, will load Forth screen 27 only if a dictionary search, **(FIND)** , fails to find **FOO** . **FOO** should be the last word loaded by the command **27 LOAD** .

It is also possible to use **CLOAD** to abort the loading of a Forth screen. This is done by using the command:

```
0 CLOAD TESTWORD
```

If this line of code were located on Forth screen 50, and the word **TESTWORD** was in the present dictionary, the load would abort just as if a **;S** had been encountered.

Caution must be exercised when using **BASE->R** and **R->BASE** with **CLOAD** as these will cause the return stack to be polluted if a **LOAD** is aborted and the **BASE->R** is not balanced by a **R->BASE** at execution time.

11.3 Memory Resident Messages

message (---)

If the user desires, he may elect to use a version of **MESSAGE** which is provided on the system disk (Forth screen 84). This version is spelled with lower case **message**. The purpose of this version is to avoid having to place the messages on the diskette in DR0. The code to install this version is supplied on the same Forth screens with the routines. Installing **message** will remove the 5th disk buffer from the system and use that memory for storing the error messages. It will then place a patch in the old version of message to cause it to branch to the new routine. Caution must be exercised if **COLD** is executed with the new version in place, as **COLD** will restore the 5th buffer but will not unpatch the old version of **MESSAGE**. After performing the **COLD**, you must reinstall the new **message** or unpatch the old version of **MESSAGE** prior to the system using the word **MESSAGE**. Failure to do this will cause a crash. To repatch **MESSAGE**, the first two words in the parameter field must be restored to be the CFAs of **WARNING** and **@**.

11.4 CRU Words

LDCR (n_1 n_2 *addr* ---)

STCR (n_1 *addr* --- n_2)

TB (*addr* --- *flag*)

SBO (*addr* ---)

SBZ (*addr* ---)

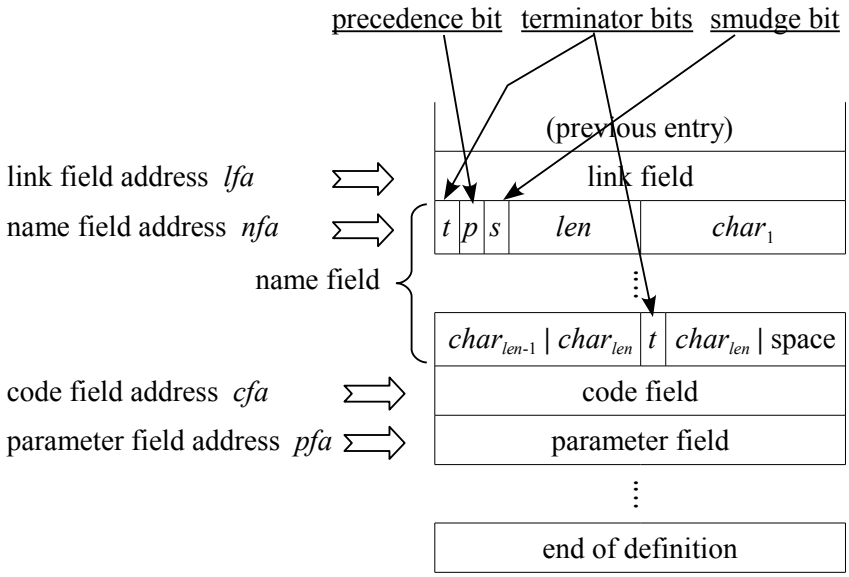
The above five words have been included to assist in performing CRU (Communications Register Unit) related functions. They allow the Forth programmer to perform the **LDCR**, **STCR**, **TB**, **SBO** and **SBZ** operations of the TMS9900 without using the Assembler. The functions of these words will be apparent when someone familiar with these instructions on the TMS9900 examines their definitions in the Glossary (Appendix D). Also, see the *Editor/Assembler Manual* for greater detail.

12 TI Forth Dictionary Entry Structure

[*Editor's Note: As with several of the appendices in this document, this chapter was added by the editor.*]

The structure of an entry (a Forth *word*) in the TI Forth dictionary is briefly described in this chapter to give the reader a better understanding of TI Forth and how its dictionary may differ from other Forth implementations.

The dictionary entries are shown here schematically as a stack of single cells of 16 bits each:



At the least, each entry contains a link field (1 cell), a name field (1 – 16 cells), a code field (1 cell) and a parameter field ($n \geq 1$ cells).

12.1 Link Field

The link field is the first field in a definition. It contains the address of the name field of the immediately preceding word in the vocabulary list to which the word belongs in the dictionary. The address of this field is termed the link field address *lfa* and may be retrieved by pushing the *pfa* (see § 12.4) onto the stack and executing **LFA** .

12.2 Name Field

The name field follows the link field and may be as long as 16 cells (32 bytes). The name field address *nfa* points to this field and may be retrieved by pushing the *pfa* (see § 12.4) onto the stack and executing **NFA** .

The first byte is the length byte. The three highest bits of the length byte are the beginning terminator bit (**80h**), the precedence bit (**40h**) and the smudge bit (**20h**). These are shown in the above figure as *t*, *p* and *s*, respectively. That leaves 5 bits for the character-length *len* of the name, which is the reason that TI Forth words have a maximum length of 31 characters. The name field in TI Forth always occupies an even number of bytes, *i.e.*, it begins and ends on a cell boundary. The last byte of the name field will be either the last character of the name or a space and will have the highest bit (**80h**) set as the ending terminator bit.

To clarify the above diagram a bit, when the name is only one character long, the first character is obviously the last character and the ending terminator bit will be set in that byte, which results in a name field occupying just one cell.

The terminator bits are flags used by **TRAVERSE** (*q.v.*) to find the beginning or end of the name field, given the address of one end and the direction (+1|-1) to search.

The precedence bit is used to indicate that a word should be executed rather than compiled during compilation. It is set by **IMMEDIATE** , which sets the precedence bit for the most recently completed definition.

The smudge bit is used to hide|unhide a word from a dictionary search during compilation. If the smudge bit is set (**20h**), **'** , **-FIND** and **(FIND)** will not find the word. During compilation, the smudge bit is toggled by **SMUDGE** or similar code and toggled again by **;** or similar termination code.

12.3 Code Field

The code field immediately follows the last cell of the name field. The code field address *cfa* points to this field and may be retrieved by pushing the *pfa* (see § 12.4) onto the stack and executing **CFA** . The code field contains the address of the machine-code routine that TI Forth will run when it executes this word and depends on the nature of the word's definition. The following table shows common situations:

Word Defined by	Code Field Contains Address of	What the Runtime Code Does
VARIABLE	Runtime code of **VARIABLE**	Pushes word's *pfa* onto stack
CONSTANT	Runtime code of **CONSTANT**	Pushes contents of word's *pfa* onto stack
:	Runtime code of **:**	Executes the list of previously defined words, the addresses of which are stored beginning at this word's *pfa*
CODE	*pfa* of word	Executes machine code stored beginning at this word's *pfa*

12.4 Parameter Field

The parameter field follows the code field. The parameter field address *pfa* points to this address, which can be retrieved by using **'** :

```
' cccc
```

where **cccc** is the name of the Forth word for which you desire the *pfa*.. If the word is not found, however, you will get an error message as well as two values on the stack that indicate the character offset and screen number (0 for terminal) of the error. **-FIND** (*q.v.*) will also return the *pfa* along with the length byte of the name field and *true* if the word is found in the dictionary or just *false* if it is not found. It is used the same way as **'** ; but, more work is required if all you want is the *pfa*, so it is more suited to colon definitions:

```
-FIND cccc DROP DROP
```

If you know only the *nfa*, you can retrieve the *pfa* by executing **PFA** .

The contents of the parameter field depend on the type of word defined. The following table shows common situations:

Word Defined by	Parameter Field Contains
VARIABLE	Value of variable
CONSTANT	Value of constant
:	Mostly a list of the addresses (usually their *cfa*s) of previously defined words that comprise this word's definition
CODE	Machine code comprising this word's runtime code

Appendix A ASCII Keycodes (Sequential Order)

Character			ASCII Code hex	ASCII Code decimal	Character			ASCII Code hex	ASCII Code decimal
NUL	<CTRL+,>		00h	0	SP			20h	32
SOH	<CTRL+A>	<FCTN+7>	01h	1	!			21h	33
STX	<CTRL+B>	<FCTN+4>	02h	2	"	<FCTN+P>		22h	34
ETX	<CTRL+C>	<FCTN+1>	03h	3	#			23h	35
EOT	<CTRL+D>	<FCTN+2>	04h	4	$			24h	36
ENQ	<CTRL+E>	<FCTN+=>	05h	5	%			25h	37
ACK	<CTRL+F>	<FCTN+8>	06h	6	&			26h	38
BEL	<CTRL+G>	<FCTN+3>	07h	7	'	<FCTN+O>		27h	39
BS	<CTRL+H>	<FCTN+S>	08h	8	(			28h	40
HT	<CTRL+I>	<FCTN+D>	09h	9	)			29h	41
LF	<CTRL+J>	<FCTN+X>	0Ah	10	*			2Ah	42
VT	<CTRL+K>	<FCTN+E>	0Bh	11	+			2Bh	43
FF	<CTRL+L>	<FCTN+6>	0Ch	12	,			2Ch	44
CR	<CTRL+M>		0Dh	13	-			2Dh	45
SO	<CTRL+N>	<FCTN+5>	0Eh	14	.			2Eh	46
SI	<CTRL+O>	<FCTN+9>	0Fh	15	/			2Fh	47
DLE	<CTRL+P>		10h	16	0	<CTRL+0>		30h	48
DC1	<CTRL+Q>		11h	17	1	<CTRL+1>		31h	49
DC2	<CTRL+R>		12h	18	2	<CTRL+2>		32h	50
DC3	<CTRL+S>		13h	19	3	<CTRL+3>		33h	51
DC4	<CTRL+T>		14h	20	4	<CTRL+4>		34h	52
NAK	<CTRL+U>		15h	21	5	<CTRL+5>		35h	53
SYN	<CTRL+V>		16h	22	6	<CTRL+6>		36h	54
ETB	<CTRL+W>		17h	23	7	<CTRL+7>		37h	55
CAN	<CTRL+X>		18h	24	8			38h	56
EM	<CTRL+Y>		19h	25	9	<FCTN+Q>	<FCTN+.>	39h	57
SUB	<CTRL+Z>		1Ah	26	:	<FCTN+/>		3Ah	58
ESC	<CTRL+.>		1Bh	27	;	<CTRL+/>		3Bh	59
FS	<CTRL+;>		1Ch	28	<	<FCTN+0>		3Ch	60
GS	<CTRL+=>		1Dh	29	=	<FCTN+;>		3Dh	61
RS	<CTRL+8>		1Eh	30	>	<FCTN+B>		3Eh	62
US	<CTRL+9>		1Fh	31	?	<FCTN+H>	<FCTN+I>	3Fh	63

…continued from previous page—

Character		ASCII Code hex	ASCII Code decimal	Character		ASCII Code hex	ASCII Code decimal
@	<FCTN+J>	40h	64	`	<FCTN+C>	60h	96
A	<FCTN+K>	41h	65	a		61h	97
B	<FCTN+L>	42h	66	b		62h	98
C	<FCTN+M>	43h	67	c		63h	99
D	<FCTN+N>	44h	68	d		64h	100
E		45h	69	e		65h	101
F	<FCTN+Y>	46h	70	f		66h	102
G		47h	71	g		67h	103
H		48h	72	h		68h	104
I		49h	73	i		69h	105
J		4Ah	74	j		6Ah	106
K		4Bh	75	k		6Bh	107
L		4Ch	76	l		6Ch	108
M		4Dh	77	m		6Dh	109
N		4Eh	78	n		6Eh	110
O		4Fh	79	o		6Fh	111
P		50h	80	p		70h	112
Q		51h	81	q		71h	113
R		52h	82	r		72h	114
S		53h	83	s		73h	115
T		54h	84	t		74h	116
U		55h	85	u		75h	117
V		56h	86	v		76h	118
W		57h	87	w		77h	119
X		58h	88	x		78h	120
Y		59h	89	y		79h	121
Z		5Ah	90	z		7Ah	122
[	<FCTN+R>	5Bh	91	{	<FCTN+F>	7Bh	123
\	<FCTN+Z>	5Ch	92	|	<FCTN+A>	7Ch	124
]	<FCTN+T>	5Dh	93	}	<FCTN+G>	7Dh	125
^		5Eh	94	~	<FCTN+W>	7Eh	126
_	<FCTN+U>	5Fh	95	DEL	<FCTN+V>	7Fh	127

Appendix B ASCII Keycodes (Keyboard Order)

	ASCII Code			ASCII Code	
Control Key	hex	decimal	Function Key	hex	decimal
<CTRL+1>	31h	49	<FCTN+1>	03h	3
<CTRL+2>	32h	50	<FCTN+2>	04h	4
<CTRL+3>	33h	51	<FCTN+3>	07h	7
<CTRL+4>	34h	52	<FCTN+4>	02h	2
<CTRL+5>	35h	53	<FCTN+5>	0Eh	14
<CTRL+6>	36h	54	<FCTN+6>	0Ch	12
<CTRL+7>	37h	55	<FCTN+7>	01h	1
<CTRL+8>	1Eh	30	<FCTN+8>	06h	6
<CTRL+9>	1Fh	31	<FCTN+9>	0Fh	15
<CTRL+0>	30h	48	<FCTN+0>	3Ch	60
<CTRL+=>	1Dh	29	<FCTN+=>	05h	5
<CTRL+Q>	11h	11	<FCTN+Q>	39h	57
<CTRL+W>	17h	23	<FCTN+W>	7Eh	126
<CTRL+E>	05h	5	<FCTN+E>	0Bh	11
<CTRL+R>	12h	18	<FCTN+R>	5Bh	91
<CTRL+T>	14h	20	<FCTN+T>	5Dh	93
<CTRL+Y>	19h	25	<FCTN+Y>	46h	70
<CTRL+U>	15h	21	<FCTN+U>	5Fh	95
<CTRL+I>	09h	9	<FCTN+I>	3Fh	63
<CTRL+O>	0Fh	15	<FCTN+O>	27h	39
<CTRL+P>	10h	16	<FCTN+P>	22h	34
<CTRL+/>	3Bh	59	<FCTN+/>	3Ah	58

…continued from previous page—

	ASCII Code			ASCII Code	
Control Key	**hex**	**decimal**	**Function Key**	**hex**	**decimal**
<CTRL+A>	01h	1	*<FCTN+A>*	7Ch	124
<CTRL+S>	13h	19	*<FCTN+S>*	08h	8
<CTRL+D>	04h	4	*<FCTN+D>*	09h	9
<CTRL+F>	06h	6	*<FCTN+F>*	7Bh	123
<CTRL+G>	07h	7	*<FCTN+G>*	7Dh	125
<CTRL+H>	08h	8	*<FCTN+H>*	3Fh	63
<CTRL+J>	0Ah	10	*<FCTN+J>*	40h	64
<CTRL+K>	0Bh	11	*<FCTN+K>*	41h	65
<CTRL+L>	0Ch	12	*<FCTN+L>*	42h	66
<CTRL+;>	1Ch	28	*<FCTN+;>*	3Dh	61
<CTRL+Z>	1Ah	26	*<FCTN+Z>*	5Ch	92
<CTRL+X>	18h	24	*<FCTN+X>*	0Ah	10
<CTRL+C>	03h	3	*<FCTN+C>*	60h	96
<CTRL+V>	16h	22	*<FCTN+V>*	7Fh	127
<CTRL+B>	02h	2	*<FCTN+B>*	3Eh	62
<CTRL+N>	0Eh	14	*<FCTN+N>*	44h	68
<CTRL+M>	0Dh	13	*<FCTN+M>*	43h	67
<CTRL+,>	00h	0	*<FCTN+,>*	38h	56
<CTRL+.>	1Bh	27	*<FCTN+.>*	39h	57

Appendix C Differences between *Starting FORTH (1st Ed.) and* TI Forth

Page	Word	Changes Required
10	**BACKSPACE**	*<FCTN+S>* produces a backspace on the TI 99/4A.
10	**OK**	TI Forth automatically prints a space before "ok".
16		The TI Forth dictionary can store names up to 31 characters in length.
18	**^**	Not a special character in TI Forth.
18	**."**	Will execute inside or outside a colon definition in TI Forth.
42	**/MOD**	Uses signed numbers in TI Forth. Remainder has sign of dividend.
42	**MOD**	Uses signed numbers in TI Forth. Remainder has sign of dividend.
50	**.S**	This word is available on the TI Forth disk. The TI Forth version prints a vertical bar (|) followed by the stack contents. The stack contents will be printed as unsigned numbers. To use the definition shown you must make the following change because of vocabulary differences: in place of **'S** use **SP@ 2-** : `: .S CR SP@ 2- S0 @ . -2 +LOOP ;`
52	**2SWAP**	This word is not in TI Forth but can be created with the following definition: `: 2SWAP ROT >R ROT R> ;`
52	**2DUP**	This word is not in TI Forth but can be created with the following definition: `: 2DUP OVER OVER ;`
52	**2OVER**	This word is not in TI Forth but can be created with the following definition: `: 2OVER SP@ 6 + @ SP@ 6 + @ ;`
52	**2DROP**	This word is not in TI Forth but can be created with the following definition: `: 2DROP DROP DROP ;`
57		When you redefine a word that is already in the dictionary, TI Forth will issue a message saying "WORD isn't unique". In the example, a message saying "GREET isn't unique" would appear.
60		TI Forth supports 90 screens per disk, numbered 0 – 89.
63-82		The TI Forth Editor is different (much better) than the editor described in this section. Read the section of this *TI Forth Instruction Manual* describing the Editor.

Page	Word	Changes Required
83	**DEPTH**	See comments for page 50.
84	**COPY**	TI Forth has a disk based word **SCOPY** (screen copy) which is exactly like **COPY** , *e.g.*, `: COPY SCOPY ;`
84-5		Ignore Editor words.
89*ff.*	**THEN**	**THEN** is in the TI Forth vocabulary and is a synonym for the word **ENDIF** . Many people find **ENDIF** less confusing than **THEN** .
91	**0>**	This word is not in TI Forth but can be created with the following definition: `: 0> 0 > ;`
91	**NOT**	This word is not in TI Forth, but can be created with the following definition: `: NOT 0= ;`
101	**?DUP**	This word is identical to **-DUP** in TI Forth. Use the following definition if necessary: `: ?DUP -DUP ;`
101*ff.*	**ABORT"**	As with the Forth-79 Standard, TI Forth provides **ABORT** instead of **ABORT"** .
102	**?STACK**	In TI Forth this word automatically calls **ABORT** and prints the appropriate error message.
107	**2***	This word is not in TI Forth, but can be created with the following definition: `: 2* DUP + ;`
107	**2/**	This word is not in TI Forth, but can be created with the following definition: `: 2/ 1 SRA ;`
108	**NEGATE**	This word is not in TI Forth, but can be created with the following definition: `: NEGATE MINUS ;`
110	**I**	This word exists in TI Forth but also has a duplicate definition, **R** . **I** and **R** are identical in function.
110	**I'**	This word is not in TI Forth, but can be created with the following definition: (*Note*: **R** is a synonym for **I** .) `: I' R> R SWAP >R ;`

Page	Word	Changes Required
112		If you will notice, there is a **.** (print) missing in the **QUADRATIC** definition. You must add a **.** after the last **+** to make **QUADRATIC** work correctly.
112		Ignore the last two paragraphs. They do not apply.
131		Just a reminder! You must define **2DUP** and **2DROP** before the **COMPOUND** example may be used.

132		There is a mistake in the second definition of **TABLE**. It should look like this:
		`: TABLE CR 11 1 DO`
		`11 1 DO I J * 5 U.R LOOP CR LOOP ;`

134		When you execute the **DOUBLING** example, an extra number will be printed after 16384. This is because **+LOOP** behaves a little differently in TI Forth.
136		In the definition of **COMPOUND** , the **CR** should precede **SWAP** instead of **LOOP** .
137	**XX**	When an error is detected in TI Forth, the stack is cleared but then the contents of **BLK** and **IN** are saved on the stack to assist in locating the error. The stack may be completely cleared with the word **SP!** .
142	**PAGE**	This word is not in TI Forth, but can be created with the following definition: `: PAGE CLS 0 0 GOTOXY ;`
161	**U/MOD**	This word is not in TI Forth, but can be created with the following definition: `: U/MOD U/ ;`
161	**/LOOP**	This word is not in TI Forth.
162	**OCTAL**	**OCTAL** does not exist in TI Forth. See p. 163 for definition.
164-5		Numbers in TI Forth may only be punctuated with periods. Commas, slashes and other marks are not permitted. Any number containing a period (**.**) is considered double-length. In later examples using **D.** and **UD.** , replace all punctuation in the inputs with decimal points. It is recommended that you not place more than one decimal place in each number if you want valid output.
166	**UD.**	This word is already defined in TI Forth.
173	**D-**	This word is not in TI Forth, but can be created with the following definition: `: D- DMINUS D+ ;`

Page	Word	Changes Required
173	**DNEGATE**	This word is not in TI Forth, but can be created with the following definition: `: DENEGATE DMINUS ;`

173	**DMAX**	This word is not in TI Forth, but can be created with the following definition:
		`: DMAX 2OVER 2OVER D- SWAP DROP 0<`
		`IF 2SWAP ENDIF`
		`2DROP ;`

173	**DMIN**	This word is not in TI Forth, but can be created with the following definition:
		`: DMIN 2OVER 2OVER 2SWAP D- SWAP DROP 0<`
		`IF 2SWAP ENDIF`
		`2DROP ;`

173	**D=**	This word is not in TI Forth, but can be created with the following definition: `: D= D- 0= SWAP 0= AND ;`
173	**D0=**	This word is not in TI Forth, but can be created with the following definition: `: D0= 0. D= ;`
173	**D<**	This word is not in TI Forth, but can be created with the following definition: `: D< D- SWAP DROP 0<;`

173	**DU<**	This word is not in TI Forth, but can be created with the following definition:
		`: DU< ROT SWAP OVER OVER`
		`U<`
		`IF` *(determined less using high order halves)*
		`DROP DROP DROP DROP 1`
		`ELSE` *(test if high halves equal)*
		`=`
		`IF` *(equal so just test low halves)*
		`U<`
		`ELSE` *(test fails)*
		`DROP DROP 0`
		`ENDIF`
		`ENDIF ;`

Page	Word	Changes Required
174	**M+**	This word is not in TI Forth, but can be created with the following definition: **: M+ 0 D+ ;**
174	**M/**	This word is different in TI Forth and can be changed with the following definition: **: M/ M/ SWAP DROP ;**
174	**M*/**	Not available in TI Forth because no triple precision arithmetic has been included. This could be created using either a relatively complicated colon definition or by using the Assembler included with TI Forth.
183*ff.*		Variables in TI Forth are required to be initialized at creation, thus the word **VARIABLE** takes the top item on the stack and places it into the variable as its initial value. For example, **12 VARIABLE DATE** both creates the variable **DATE** and initializes it to 12. If desired, the advanced user can use the words **<BUILDS** and **DOES>** to create a new defining word, **VARIABLE** , which has exactly the behavior of **VARIABLE** as used in this section. The code to do this is: **: VARIABLE <BUILDS 0 , DOES> ;**
193	**2VARIABLE**	This word is not in TI Forth, but can be created with the following definition: **: 2VARIABLE <BUILDS 0. , , DOES> ;** This definition does not require a number to be on the stack when it is executed.
193	**2!**	This word is not in TI Forth, but can be created with the following definition: **: 2! >R R ! R> 2+ ! ;**
193	**2@**	This word is not in TI Forth, but can be created with the following definition: **: 2@ >R R 2+ @ R> @ ;**
193	**2CONSTANT**	This word is not in TI Forth, but can be created with the following definition: **: 2CONSTANT <BUILDS , , DOES> 2@ ;** This definition does *not* require a number on the stack.
199		You must place a 0 on the stack before executing **VARIABLE COUNTS 10 ALLOT** . This, however, initializes only the first element of the array COUNTS to 0. You must execute either the **FILL** or **ERASE** instruction at the bottom of the page to properly initialize the array.

Page	Word	Changes Required
204	**DUMP**	TI Forth already has a dump instruction which must be loaded from the disk. Dumps are always printed in hexadecimal. See Appendix D for location of **DUMP** .
207	**CREATE**	The **CREATE** word of TI Forth behaves somewhat differently. Hackers should consult fig-Forth documentation.
216	**EXECUTE**	Because this word operates a little differently in TI Forth, it must be preceded by the word **CFA** . The example should read: **' GREET CFA EXECUTE**
217		The example illustrating indirect execution must be modified to work in TI Forth: **' GREET CFA POINTER ! POINTER @ EXECUTE**
218	**[']**	In TI Forth, this word is unnecessary as the word **'** will take the following word of a definition when used in a definition.
219	**NUMBER**	In TI Forth, **NUMBER** is always able to convert double precision numbers.
219	**'NUMBER**	TI Forth does not use **'NUMBER** to locate the **NUMBER** routine.
220		In TI Forth, the name field is variable length and contains up to 31 characters. Also, the link field precedes the name field in TI Forth.
225	**EXIT**	This word is **;S** in TI Forth. **;S** is the word compiled by **;** so to create **EXIT** we might use: **: EXIT [COMPILE] ;S ; IMMEDIATE**
225	**I**	In TI Forth, the interpreter pointer is called **IP** , not **I** .
232		See Chapter 1 in this *TI Forth Instruction Manual* for instructions for loading elective blocks.
232	**RELOAD**	This instruction is not available in TI Forth.
233	**H**	This word is **DP** (dictionary pointer) in TI Forth.
235	**'S**	In TI Forth, **SP@** is used instead of **'S** .
240		See Appendix E in this *TI Forth Instruction Manual* for a complete list of user variables.
240	**>IN**	This word is **IN** in TI Forth.
245	**LOCATE**	TI Forth does not support **LOCATE** .
256	**COPY**	In TI Forth, this word is **SCOPY** . **SCOPY** is disk resident. See Appendix D for location.
259	**[']**	Change the **[']** to **'** in the bottom example. In TI Forth, **'** will compile the address of the next word in the colon definition.
261	**>TYPE**	Unnecessary in non-multiprogramming systems. Not present in TI Forth.

Page	Word	Changes Required
265	**RND**	TI Forth has two disk resident random number generators: **RND** and **RNDW** . See Appendix D for locations and descriptions. See also definitions for **SEED** and **RANDOMIZE** .
266	**MOVE**	In TI Forth, **MOVE** moves *u* words in memory, not *u* bytes. **MOVE** can be redefined to conform to *Starting FORTH (1st Ed.)*: `: MOVE 2/ MOVE ;`
266	**<CMOVE**	Not present in TI Forth. Must be created with the Assembler if required. This word is used only when the source and destination regions of a move overlap and the destination is higher than the source.
270	**WORD**	In TI Forth, the word **WORD** does not leave an address on the stack.

270	**TEXT**	This word is not available in TI Forth, but can be defined as follows:
		`: TEXT PAD 72 BLANKS PAD HERE - 1-`
		`DUP ALLOT MINUS SWAP WORD ALLOT ;`
		If you want the count to also be stored at PAD, remove the **1-** from the definition.

277	**>BINARY**	This is named **(NUMBER)** in TI Forth.
277		Because **WORD** does not leave an address on the stack, it is necessary to redefine **PLUS** as follows: `: PLUS 32 WORD DROP NUMBER + ." = " . ;`
279	**NUMBER**	This definition of **NUMBER** is not compatible with TI Forth.
281	**-TEXT**	Not in TI Forth. Use the definition on page 282.
292		TI Forth uses the word pair **<BUILDS … DOES>** to define a new defining word. **<BUILDS** calls **CREATE** as part of its function.

297		To create a byte **ARRAY** in TI Forth:
		`: ARRAY <BUILDS OVER , * ALLOT`
		`DOES> DUP @ ROT * + + 2+ ;`

298		Just a reminder! Don't forget to define **2*** *before* trying the example at the bottom of the page. Also, replace the word **CREATE** with **<BUILDS** .
301	**(DO)**	This is the runtime behavior of **DO** just as listed. **2>R** is not used, however.
301	**DO**	The given definition of **DO** is not compatible with TI Forth. TI Forth's definition of **DO** is much more complex because of compile-time error checking.
303	**(LITERAL)**	The TI Forth name for this word is **LIT** .
306		TI Forth remains in compilation mode until a **;** is typed.

Appendix D The TI Forth Glossary

TI Forth words appear in this glossary on the left of the entry line for that word and in the order of the ASCII collating sequence, which is displayed as a handy reference at the bottom of each page of this appendix. The Forth screen on which the word is defined is right-justified on the entry line along with the **MENU** choice that will load its definition. If the word is part of the core system, it is listed as "RESIDENT". The stack effects are listed on the second line. The stack effects on the return stack may also be shown. These will be indicated by "R:" following the "(" as in the following: "(R: *n* ---)", which would mean that a 16-bit number *n* is removed from the top of the return stack after the word being described is executed.

D.1 Explanation of Some Terms and Abbreviations

Term/Abbreviation	Meaning
addr, addr$_1$*, ...*	memory address
b	byte
col	column position
cccc, nnnn, xxxx	string representation
cfa	code field address
char	ASCII character code
count	count (length)
d, d$_1$*, d*$_2$*, ...*	signed double-precision number
dotcol, dotcol$_1$*, dotcol*$_2$*, ...*	dot column position
dotrow, dotrow$_1$*, dotrow*$_2$*, ...*	dot row position
drive	refers to DR0, DR1, DR2 (DSK1, DSK2, DSK3)
flag	Boolean flag
false	Boolean false flag (value = 0)
f, f$_1$*, f*$_2$*, ...*	floating point number
lfa	link field address
n, n$_1$*, n*$_2$*, ...*	signed single-precision number
nfa	name field address
pfa	parameter field address
row	row position
rem	remainder
scr	screen number
spr	sprite number
true	Boolean true flag (value ≠ 0)
tol	tolerance limit
u	unsigned single-precision number
ud	unsigned double-precision number
vaddr	VDP address

D.2 TI Forth Word Descriptions

! RESIDENT

(*n addr* ---)

Store 16 bits of *n* at address. Pronounced “store”.

!" SCR 39 -COPY

(*addr* ---)

A string terminated with a **"** must follow this word. This string will be stored at the specified address; however, the character count is not stored.

!CSP RESIDENT

(---)

Save the stack position in user variable **CSP** . Used as part of the compiler security.

RESIDENT

(d_1 --- d_2)

Generate from a double number d_1, the next ASCII character which is placed in an output string. Result d_2 is the quotient after division by the value in **BASE** , and is maintained for further processing. Used between **<#** and **#>** . See **#S** .

#> RESIDENT

(*d --- addr count*)

Terminates numeric output conversion by dropping *d*, leaving the text address and character count suitable for **TYPE** .

#MOTION SCR 59 -GRAPH

(*n* ---)

Sets sprite numbers 0 to *n* - 1 in automotion.

#S RESIDENT

(d_1 --- d_2)

Generates ASCII text from d_1 in the text output buffer, by the use of **#** , until a zero double number d_2 results. Used between **<#** and **#>** .

' RESIDENT

(--- *pfa*)

Used in the form:

```
' nnnn
```

Leaves the parameter field address of dictionary word **nnnn** . As a compiler directive, executes in a colon definition to compile the address of a literal. If the

word is not found after a search of **CONTEXT** and **CURRENT** , an appropriate error message is given. Pronounced “tick”.

(RESIDENT

(---)

Used in the form:

```
( cccc)
```

Ignore a comment that will be delimited by a right parenthesis on the same Forth screen. May occur during execution or in a colon definition. A blank after the leading parenthesis is required.

(+LOOP) RESIDENT

(*n* ---)

The runtime procedure compiled by **+LOOP** , which increments the loop index by *n* and tests for loop completion. See **+LOOP** .

(.") RESIDENT

(---)

The runtime procedure, compiled by **."** ,which transmits the following in-line text to the selected output device. See **."** .

(;CODE) RESIDENT

(---)

The runtime procedure, compiled by **;CODE** , that rewrites the code field of the most recently defined word to point to the machine code sequence following **;CODE** . See **;CODE** .

(ABORT) RESIDENT

(---)

Executes after an error when **WARNING** < 0. This word normally executes **ABORT** , but may be redefined (with care) to execute a user’s alternative procedure.

(DO) RESIDENT

(---)

The runtime procedure complied by **DO** which moves the loop control parameters to the return stack. See **DO** .

(DOES>) RESIDENT

(---)

The run time procedure compiled by **DOES>** .

(FIND) RESIDENT

($addr_1$ $addr_2$ --- *false* | *pfa b true*)

Searches the dictionary starting at the name field address $addr_2$, matching to the text at $addr_1$. Returns parameter field address *pfa*, length byte *b* of name field, and *true* for a good match. If no match is found, only *false* is left.

(LINE) RESIDENT

(*n scr* --- *addr count*)

Convert the line number *n* and the Forth screen *scr* to the disk buffer address *addr* containing the data and the number of characters *count*. If *count* is 64, the full-line text length is indicated.

(LOOP) RESIDENT

(---)

The runtime procedure compiled by **LOOP** , which increments the loop index and tests for loop completion. See **LOOP** .

(NUMBER) RESIDENT

(d_1 $addr_1$ --- d_2 $addr_2$)

Convert the ASCII text beginning at $addr_1$ + 1 with respect to **BASE** . The new value is accumulated into double number d_1, being left as d_2. $addr_2$ is the address of the first unconvertible digit. Used by **NUMBER** .

(OF) RESIDENT

(---)

The run time procedure compiled by **OF** .

***** RESIDENT

(n_1 n_2 --- n_3)

Leave the signed product of two signed numbers.

***/** RESIDENT

(n_1 n_2 n_3 --- n_4)

Leave the ratio $n_4 = n_1 * n_2 / n_3$, where all are signed numbers. Retention of an intermediate 31-bit product permits greater accuracy than would be available with the sequence :

```
n1 n2 * n3 /
```

***/MOD** RESIDENT

(n_1 n_2 n_3 --- *rem quot*)

Leave the quotient *quot* and remainder *rem* of the operation n_1*n_2/n_3. A 31-bit intermediate product is used as for ***/** .

+ RESIDENT

(n_1 n_2 --- n_3)

Leave the sum of n_1 + n_2 as n_3.

+! RESIDENT

(*n addr* ---)

Add *n* to the value at the address. Pronounced "plus store".

+- RESIDENT

(n_1 n_2 --- n_3)

Apply the sign of n_2 to n_1, which is left as n_3.

+BUF RESIDENT

($addr_1$ --- $addr_2$ *flag*)

Advance the disk buffer address $addr_1$ to the address of the next buffer $addr_2$. Boolean flag is false when $addr_2$ is the buffer presently pointed to by user variable **PREV** .

+LOOP RESIDENT

Runtime: (n_1 ---) Compilation: (*addr* n_2 ---)

Used in a colon-definition in the form:

DO … n_1 +LOOP

At run time, **+LOOP** selectively controls branching back to the corresponding **DO** based on n_1, the loop index and the loop limit. The signed increment n_1 is added to the index and the total compared to the limit. The branch back to **DO** occurs until the new index is equal to or greater than the limit (n_1 > 0), or until the new index is equal to or less than the limit (n_1 < 0). Upon exiting the loop, the parameters are discarded and execution continues ahead.

At compile time, **+LOOP** compiles the runtime word **(+LOOP)** and the branch offset computed from **HERE** to the address left on the stack by **DO** . The value n_2 is used for compile-time error checking.

, RESIDENT

(*n* ---)

Store *n* into the next available dictionary memory cell, advancing the dictionary pointer. Pronounced “comma”.

- RESIDENT

(n_1 n_2 --- n_3)

Leave the difference n_3 of $n_1 - n_2$.

--> RESIDENT

(---)

Continue interpretation with the next Forth screen on disk. Pronounced “next screen”.

-DUP RESIDENT

(n_1 --- n_1 | n_1 n_1)

Duplicate n_1 only if it is non-zero. This is usually used to copy a value just before **IF** , to eliminate the need for an **ELSE** clause to drop it.

-FIND RESIDENT

(--- *false* | *pfa len true*)

Accepts the next text word (delimited by blanks) in the input stream to **HERE** , searches the **CONTEXT** and then **CURRENT** vocabularies for a matching entry. If found, the dictionary entry’s parameter field address *pfa*, its length byte *len* and *true* are left. Otherwise, only *false* is left. [*Note:* See Chapter 12 about the length byte.]

-TRAILING RESIDENT

(*addr* n_1 --- *addr* n_2)

Adjusts the character count n_1 of a text string beginning at *addr* to suppress the output of trailing blanks, *i.e.*, the characters at *addr* + n_2 to *addr* + n_1 are blanks.

. RESIDENT

(*n* ---)

Print a number from a signed 16-bit two’s complement value *n*, converted according to the numeric base stored in **BASE** . A trailing blank follows. Pronounced “dot”.

." RESIDENT

(---)

Used in the form:

```
." cccc"
```

Compiles an in-line string **cccc** (delimited by the trailing **"**) with an execution procedure to transmit the text to the selected output device. If executed outside a definition, **."** will immediately print the text until the final **"** . See (**."**) .

.LINE RESIDENT

(*n scr* ---)

Print on the terminal device, a line of text from the disk by its line number *n* and Forth screen number *scr*. Trailing blanks are suppressed.

.R RESIDENT

(n_1 n_2 ---)

Print the number n_1 right aligned in a field whose width is n_2. No following blank is printed.

.S SCR 43 -DUMP

(---)

Prints the entire contents of the parameter stack as unsigned numbers in the current **BASE** .

/ RESIDENT

(n_1 n_2 --- n_3)

Leave the quotient n_3 of n_1/n_2.

/MOD RESIDENT

(n_1 n_2 --- *rem* n_3)

Leave the remainder *rem* and signed quotient n_3 of n_1/n_2. The remainder has the sign of the dividend.

0 1 2 3 RESIDENT

(--- *n*)

These small numbers are used so often that it is attractive to define them by name in the dictionary as constants.

0< RESIDENT

(*n* --- *flag*)

Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

0= RESIDENT

(*n* --- *flag*)

Leave a true flag if the number is equal to zero, otherwise leave a false flag.

0BRANCH RESIDENT

(*flag* ---)

The runtime procedure to conditionally branch. If *flag* is *false* (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by **IF** , **UNTIL** and **WHILE** .

1+ RESIDENT

(n_1 --- n_2)

Increment n_1 by 1.

1– RESIDENT

(n_1 --- n_2)

Decrement n_1 by 1.

2+ RESIDENT

(n_1 --- n_2)

Leave n_1 incremented by 2 as n_2.

2– RESIDENT

(n_1 --- n_2)

Leave n_1 decremented by 2 as n_2.

: RESIDENT

(---)

Used in the form called a colon definition:

```
: cccc … ;
```

Creates a dictionary entry defining **cccc** as equivalent to the following sequence of Forth word definitions '...' until the next **;** or **;CODE** . The compiling process is done by the text interpreter as long as **STATE** is non-zero. Other details are that the **CONTEXT** vocabulary is set to the **CURRENT** vocabulary and that words with the precedence bit set are executed rather than being compiled.

: (*traceable*) SCR 44 -TRACE

(---)

This is an alternate definition of **:** that adds the capability to colon definitions of being traced when they are executed. When a colon definition is compiled under the **TRACE** option, tracing output may be turned on with **TRON** and off with **TROFF** prior to executing the word so defined. After **TRON** is executed, each time the word is executed its name will be output along with the contents of the stack. See **TRACE** , **UNTRACE** , **TRON** and **TROFF** .

; RESIDENT

(---)

Terminates a colon definition and stops further compilation. Compiles the runtime **;S** .

;CODE SCR 74 -CODE

(---)

Used with **<BUILDS** in the form:

```
: cccc <BUILDS … ;CODE <assembly mnemonics>
```

Stop compilation and terminate a new defining word **cccc** by compiling **(;CODE)** . Set the **CONTEXT** vocabulary to **ASSEMBLER** , assembling to machine code the assembly mnemonics following **;CODE** .

When **cccc** later executes in the form:

```
cccc  nnnn
```

the word **nnnn** will be created with its execution procedure given by the machine code following **(;CODE)** in the definition of **cccc** , *i.e.*, when **nnnn** is executed, it does so by jumping to that code in **cccc** . An existing defining word (**<BUILDS** in this case) must exist in **cccc** prior to **;CODE** .

;S RESIDENT

(---)

Stop interpretation of a Forth screen. **;S** is also the runtime word compiled at the end of a colon definition, which returns execution to the calling procedure.

< RESIDENT

(n_1 n_2 --- *flag*)

Leave a true flag if n_1 is less than n_2, otherwise, leave a false flag.

<# RESIDENT

(---)

Setup for pictured numeric output formatting using the words:

```
<# # #S SIGN #>
```

The conversion is done on a double number producing text at **PAD** (working downward toward **HERE**), eventually suitable for output by **TYPE** . The picture template between **<#** and **#>** represents the output picture from right to left, *i.e.*, the rightmost digit is processed first. See **#** , **#S** , **SIGN** , **#>** and **HOLD** .

<BUILDS RESIDENT

(---)

Used within a colon-definition:

```
: cccc <BUILDS … DOES> … ;     or
: cccc <BUILDS … ;CODE … ;
```

Each time **cccc** is executed, **<BUILDS** defines a new word with a high-level (**DOES>**) or machine-code (**;CODE**) execution procedure. Executing **cccc** in the form:

```
cccc  nnnn
```

uses **<BUILDS** to create a dictionary entry for **nnnn** . For the definition with **DOES>** , when **nnnn** is later executed, it has the address of its parameter area on the stack and executes the words after **DOES>** in **cccc** . For the definition with **;CODE** , when **nnnn** is later executed, it executes the words after **;CODE** in **cccc** . **<BUILDS** with **DOES>** or **;CODE** allows runtime procedures to be written in high-level code with **DOES>** or in assembler code with **;CODE** .

<CLOAD> SCR 21 BOOT SCR

(---)

The runtime procedure compiled by **CLOAD** .

= RESIDENT

(n_1 n_2 --- *flag*)

Leave a true flag if $n_1 = n_2$, otherwise leave a false flag.

=CELLS RESIDENT

($addr_1$ --- $addr_1$ | $addr_2$)

This instruction expects an address or an offset to be on the stack. If this number is odd, it is incremented by 1 to put it on the next even word boundary. Otherwise, it remains unchanged.

> RESIDENT

(n_1 n_2 --- *flag*)

Leave a true flag if n_1 is greater than n_2, otherwise leave a false flag.

>ARG SCR 45 -FLOAT

(*f* ---)

Moves a floating point number *f* from the stack into the **ARG** register.

>F SCR 48 -FLOAT

(--- *f*)

This instruction expects to be followed by a string representing a legitimate floating point number terminated by a space. This string is converted into floating point and placed on the stack. This instruction can be used in colon definitions or directly from the keyboard.

>FAC SCR 45 -FLOAT

(*f* ---)

Moves a floating point number from the stack into the **FAC** register.

>R RESIDENT

(*n* ---) (R: --- *n*)

Remove a number from the parameter stack and place as the most accessible number on the return stack. Use should be balanced with **R>** in the same definition.

? RESIDENT

(*addr* ---)

Print the value contained at address *addr* in free format according to the current **BASE** . This word is short for the two words, **@ .** .

?COMP RESIDENT

(---)

Issue error message if not compiling.

?CSP RESIDENT

(---)

Issue error message if stack position differs from value saved in **CSP** .

?ERROR RESIDENT

(*flag n* ---)

Issue an error message number *n* if the Boolean flag is true.

?EXEC RESIDENT

(---)

Issue an error message if not executing.

?FLERR SCR 49 -FLOAT

(---)

Determines if the previous floating point operation resulted in an error. An appropriate error message is printed upon finding an error.

?KEY RESIDENT

(--- *char*)

Scans the keyboard for input. If no key is pressed, a 0 is left on the stack. Else, the ASCII code of the key pressed is left on the stack.

?KEY8 RESIDENT

(--- *n*)

Scans the keyboard for input. If no key is pressed, a 0 is left on the stack. Else, the 8-bit code of the key pressed is left on the stack.

?LOADING RESIDENT

(---)

Issue an error message if not loading.

?PAIRS RESIDENT

(n_1 n_2 ---)

Issue an error message if n_1 does not equal n_2. The message indicates that compiled conditionals do not match.

?STACK RESIDENT

(---)

Issue an error message if the stack is out of bounds.

?TERMINAL RESIDENT

(--- *flag*)

Perform a test on the terminal keyboard for actuation of the break key (***<BREAK>***). A true flag indicates actuation. On the TI-99/4A, ***<FCTN+4>***, ***<BREAK>*** and ***<CLEAR>*** are all the same key.

@ RESIDENT

(*addr* --- *n*)

Leave the 16-bit contents *n* of *addr*.

A$$M SCR 82 -ASSEMBLER

(---)

This word is compiled into the **FORTH** vocabulary and marks the end of the **ASSEMBLER** vocabulary. It is used by **CLOAD** .

ABORT RESIDENT

(---)

Clear the stacks and enter the execution state. Return control to the operator's terminal, printing an appropriate message.

ABS RESIDENT

(n_1 --- n_2)

Leave the absolute value of n_1 as n_2.

AGAIN RESIDENT

Compilation: (*addr n* ---)

Used in a colon definition in the form:

BEGIN … AGAIN

At run time, **AGAIN** forces execution to return to corresponding **BEGIN** . There is no effect on the stack. Execution cannot leave the loop unless **R> DROP** is executed one level below.

At compile time, **AGAIN** compiles **BRANCH** with an offset from **HERE** to *addr*. The vale *n* is used for compile time error checking.

ALLOT RESIDENT

(*n* ---)

Add the signed number *n* to the dictionary pointer **DP** . May be used to reserve dictionary space or re-origin memory.

ALTIN RESIDENT

(--- *addr*)

A user variable whose value is 0 if input is coming from the keyboard else its value is a pointer to the VDP address where the PAB (Peripheral Access Block) for the alternate input device is located.

ALTOUT RESIDENT

(--- *addr*)

A user variable whose value is 0 if output is going to the monitor else its value is a pointer to the VDP address where the PAB (Peripheral Access Block) for the alternate output device is located.

AND RESIDENT

(n_1 n_2 --- n_3)

Leave the bitwise logical AND of n_1 and n_2 as n_3.

APPND SCR 69 -FILE

(---)

Assigns the APPEND attribute to the file whose PAB (Peripheral Access Block) is pointed to by **PAB-ADDR** .

ARG SCR 45 -FLOAT

(--- *addr*)

A constant which contains the address of the **ARG** register.

ASSEMBLER SCR 74 -ASSEMBLER

(---)

The name of the TI Forth Assembler vocabulary. Execution makes **ASSEMBLER** the **CONTEXT** vocabulary. **ASSEMBLER** is immediate, so it will execute during the creation of a colon definition to select this vocabulary at compile time. See **VOCABULARY** .

ATN SCR 50 -FLOAT

(f_1 --- f_2)

Calculates the arctangent in radians of f_1 leaving the floating point result f_2 on the stack.

B/BUF RESIDENT

(--- *n*)

This constant leaves the number of bytes *n* per disk buffer, the byte count read from disk by **BLOCK** .

B/BUF$ RESIDENT

(--- *addr*)

A user variable which contains the number of bytes per buffer.

B/SCR RESIDENT

(--- *n*)

This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.

B/SCR$ RESIDENT

(--- *addr*)

A user variable which contains the number of blocks per Forth screen.

BACK RESIDENT

(*addr* ---)

Calculate the backward branch offset from **HERE** to *addr* and compile into the next available dictionary memory address.

BASE RESIDENT

(--- *addr*)

A user variable containing the current number base used for input and output conversion.

BASE->R RESIDENT

(---)

Place the current number base on the return stack. Caution must be exercised when using **BASE->R** and **R->BASE** with **CLOAD** as these will cause the return stack to be polluted if a **LOAD** is aborted and the **BASE->R** is not balanced by a **R->BASE** at execution time. See **R->BASE** .

BEEP SCR 60 -GRAPH

(---)

Produces the sound associated with correct input or prompting.

BEGIN RESIDENT

Compilation: (--- *addr n*)

Occurs in a colon-definition in the form:

BEGIN … UNTIL

BEGIN … AGAIN

BEGIN … WHILE … REPEAT

At runtime, **BEGIN** marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding **UNTIL** , **AGAIN** or **REPEAT** . When executing **UNTIL** , a return to **BEGIN** will occur if the top of the stack is false; for **AGAIN** and **REPEAT** a return to **BEGIN** always occurs.

At compile time, **BEGIN** leaves its return address *addr* and *n* for compiler error checking.

BL RESIDENT

(--- *char*)

A constant that leaves the ASCII value for “blank”.

BLANKS RESIDENT

(*addr count* ---)

Fill an area of memory beginning at *addr* with *count* blanks.

BLK RESIDENT

(--- *addr*)

A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

BLOAD RESIDENT

(*scr* --- *flag*)

Loads the binary image at *scr* which was created by **BSAVE** . **BLOAD** returns a true flag (1) if the load was not successful and a false flag (0) if the load was successful.

BLOCK RESIDENT

(*n* --- *addr*)

Leave the memory address of the block buffer containing block *n*. If the block is not already in memory, it is transferred from disk to whichever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is written to disk before block *n* is read into the buffer. See also **BUFFER** , **R/W** , **UPDATE** and **FLUSH** .

BOOT RESIDENT

(---)

Examines the Forth screen designated as the boot screen (screen #3). If it contains only displayable characters (ASCII 32 – 127), it performs a **LOAD** on that screen.

BRANCH RESIDENT

(---)

The runtime procedure to unconditionally branch. An in-line offset is added to the interpretive pointer (IP) to branch ahead or back. **BRANCH** is compiled by **ELSE** , **AGAIN** , **REPEAT** , and **ENDOF** .

BSAVE SCR 83 -BSAVE

(*addr* scr_1 --- scr_2)

Places a binary image (starting at scr_1 and going as far as necessary) of all dictionary contents between *addr* and **HERE** . The next available Forth screen number scr_2 is returned on the stack. See **BLOAD** .

BUFFER RESIDENT

(*n* --- *addr*)

Obtain the next memory buffer, assigning it to block *n*. If the contents of the buffer is marked as updated, it is written to the disk. The block is not read from the disk. The address left is the first cell within the buffer for data storage.

C! RESIDENT

(*b addr* ---)

Store the low-order byte (8 bits) of *b* (16-bit number on the stack) at *addr*.

C, RESIDENT

(*b* ---)

Store the low-order byte (8 bits) of *b* (16-bit number on the stack) into the next available dictionary byte, advancing the dictionary pointer. This instruction should be used with caution on byte addressing, word oriented computers such as the TI 9900.

C/L RESIDENT

(--- *n*)

Returns on the stack the number of characters per line.

C/L$ RESIDENT

(--- *addr*)

A user variable whose value is the number of characters per line.

C@ RESIDENT

(*addr* --- *b*)

Leave the 8-bit contents of the memory address on the stack.

CASE RESIDENT

Compilation: (---) Runtime: (*n* --- *n*)

Used in a colon definition to initiate the construct:

```
CASE
    n1 OF … ENDOF
    n2 OF … ENDOF
    …
ENDCASE
```

At runtime, **CASE** itself does nothing with the number *n* on the stack; but, it must be there for **OF** or **ENDCASE** to consume. If $n = n_1$, the code between the immediately following **OF … ENDOF** is executed. Execution then continues after **ENDCASE** . If *n* does not match any of the values preceding any **OF** , the code between the last **ENDOF** and **ENDCASE** is executed and may consume *n*; but, one cell *must* be left for **ENDCASE** to consume. Execution then continues after **ENDCASE** .

CFA RESIDENT

(*pfa* --- *cfa*)

Convert the parameter field address *pfa* of a definition to its code field address *cfa* .

CHAR SCR 57 -GRAPH

(n_1 n_2 n_3 n_4 *char* ---)

Defines character # *char* to have the pattern specified by the 4 numbers (n_1, n_2, n_3, n_4) on the stack. The definition for character #0 by default resides at **800h**. Each character definition is 8 bytes long with each number on the stack representing two bytes.

CHAR-CNT! SCR 69 -FILE

(*n* ---)

Used in file I/O to store in the current PAB the character count of a record to be transmitted by **WRT** .

CHAR-CNT@ SCR 69 -FILE

(--- *n*)

Used in file I/O to retrieve from the current PAB the character count of a record that has been read. Used by **RD** .

CHARPAT SCR 57 -GRAPH

(*char* --- n_1 n_2 n_3 n_4)

Places the 4-number (8-byte) pattern of a specified character *char* on the stack. By default, the definition for character #0 resides at **800h**.

CHK-STAT SCR 68 -FILE

(---)

Checks for errors following an I/O operation. If an error has occurred, an appropriate message is printed.

CLEAR RESIDENT

(*scr* ---)

Fills the designated Forth screen with blanks.

CLINE SCR 66 -64SUPPORT

(*addr count n* ---)

Prints one line of tiny characters on the display screen. **CLINE** expects on the stack the address *addr* of the line to be written in memory, the number of characters *count* in that line, and the line number *n* on which it is to be written on the display screen. **CLINE** calls **SMASH** to do the actual work. See **SMASH** and **CLIST** .

CLIST SCR 66 -64SUPPORT

(*scr* ---)

Lists the specified Forth screen in tiny characters to the monitor. **CLIST** executes a multiple call to **CLINE** . See **CLINE** and **TCHAR** .

CLOAD SCR 21 BOOT SCR

(*scr* ---)

Used in the form:

***scr* CLOAD nnnn**

CLOAD will load Forth screen *scr* only if the word **nnnn** (the last word loaded by *scr*) is not in the **CONTEXT** vocabulary. A screen number of 0 will suppress loading of the current Forth screen if the specified word has already been compiled.

CLR-STAT SCR 68 -FILE

(---)

Clears (zeroes) the error code in bits 0–2 of the flag/status byte of the PAB (Peripheral Access Block) pointed to by **PAB-ADDR** .

CLS SCR 33 -SYNONYMS

(---)

Clears display screen by filling the screen image table with blanks. The screen image table runs from **SCRN_START** to **SCRN_END** .

CLSE SCR 71 -FILE

(---)

Closes the file whose PAB (Peripheral Access Block) is pointed to by **PAB-ADDR** .

CMOVE RESIDENT

($addr_1$ $addr_2$ *count* ---)

Move *count* number of bytes from $addr_1$ to $addr_2$. The contents of $addr_1$ is moved first proceeding toward high memory.

CODE SCR 74 -CODE

(---)

A defining word initializing the definition of a code (assembly) word. It sets the context vocabulary to Assembler. See Chapter 9 for details.

COINC SCR 61 -GRAPH

(spr_1 spr_2 *tol* --- *flag*)

Detects a coincidence between two given sprites within a specified tolerance limit *tol*. A true flag indicates a coincidence.

COINCALL SCR 61 -GRAPH

(--- *flag*)

Detects a coincidence between the visible portions of any two sprites on the display screen. A true flag indicates a coincidence.

COINCXY SCR 61 -GRAPH

(*dotcol dotrow spr tol --- flag*)

Detects a coincidence between a specified sprite and a given point (*dotcol*,*dotrow*) within a given tolerance limit *tol*. A true flag indicates a coincidence.

COLD RESIDENT

(---)

The **COLD** start procedure to adjust the dictionary pointer to the minimum standard and restart via **ABORT** . May be called from the terminal to remove application programs and restart. **COLD** calls **BOOT** prior to calling **ABORT** .

COLOR SCR 58 -GRAPH

(n_1 n_2 n_3 ---)

Causes a specified character set n_3 to have the given foreground n_1 and background n_2 colors.

COLTAB SCR 57 -GRAPH

(--- *vaddr*)

A constant whose value is the beginning VDP address of the color table. The default value is **380h**.

COMPILE RESIDENT

(---)

When the word containing **COMPILE** executes, the execution address of the word following **COMPILE** is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter already does).

CONSTANT RESIDENT

(*n* ---)

A defining word used in the form:

```
n CONSTANT cccc
```

to create word **cccc** , with its parameter field containing *n*. When **cccc** is later executed, it will push the value of *n* to the stack.

CONTEXT RESIDENT

(--- *addr*)

A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

COS SCR 50 -FLOAT

(f_1 --- f_2)

Calculates the cosine of f_1 radians and leaves the floating point result f_2 on the stack.

COUNT RESIDENT

($addr_1$ --- $addr_2$ n)

Leave the byte address $addr_2$ and byte count n of a message text beginning at $addr_1$. It is presumed that the first byte at $addr_1$ contains the text byte count and the actual text starts with the second byte. Typically, **COUNT** is followed by **TYPE** .

CR RESIDENT

(---)

Transmit a carriage return and a line feed to the selected output device.

CREATE RESIDENT

(---)

A defining word used in the form:

```
CREATE cccc
```

by such words as **CODE** and **CONSTANT** to create a dictionary header for a Forth definition. The code field contains the address of the word's parameter field. The new word is created in the **CURRENT** vocabulary.

CSP RESIDENT

(--- *addr*)

A user variable temporarily storing the stack pointer position for compilation error checking.

CURPOS RESIDENT

(--- *addr*)

A user variable that stores the current VDP (Visual Display Processor) cursor position.

CURRENT RESIDENT

(--- *addr*)

A user variable pointing to the vocabulary into which new definitions will be compiled.

D+ RESIDENT

(d_1 d_2 --- d_3)

Leave the double number sum of two double numbers ($d_3 = d_1 + d_2$).

D+- RESIDENT

(d_1 n --- d_2)

Apply the sign of n to the double number d_1, leaving it as d_2.

D. RESIDENT

(d ---)

Print a signed double number from a 32-bit two's complement value d. The high-order 16 bits are most accessible on the stack. Conversion is performed according to the current **BASE** . A blank follows. Pronounced "d dot".

D.R RESIDENT

(d n ---)

Print a signed double number d right-aligned in a field n characters wide.

DABS RESIDENT

(d_1 --- d_2)

Leave the absolute value d_2 of a double number d_1.

DCOLOR RESIDENT

(--- *addr*)

A variable which contains the dot-color information used by **DOT** . Its value may be a two-digit HEX number which defines the foreground and background color, or it may be -1 which means no color information is changed in the VDP (Visual Display Processor).

DDOT SCR 63 -GRAPH

(*dotcol dotrow* --- *b vaddr*)

The assembly code routine called by **DOT** . It expects a dot column and a dot row on the stack and returns a byte *b* with only one bit set and a VDP address *vaddr*. The dot referenced by (*dotcol*,*dotrow*) is translated by ddot to the address *vaddr* of the byte containing it and a mask *b* that locates the dot within the byte. [*Editor's Note:* The original glossary entry was missing *b* and its description.]

DECIMAL RESIDENT

(---)

Set the numeric conversion **BASE** for decimal input/output.

DEFINITIONS RESIDENT

(---)

Used in the form:

```
cccc  DEFINITIONS
```

Set the **CURRENT** vocabulary to the **CONTEXT** vocabulary. In the example, executing vocabulary name **cccc** makes it the **CONTEXT** vocabulary and executing **DEFINITIONS** makes both specify vocabulary **cccc** .

DELALL SCR 61 -GRAPH

(---)

Delete all sprites.

DELSPR SCR 61 -GRAPH

(*spr* ---)

Delete the specified sprite.

DIGIT RESIDENT

(*char* n_1 --- *false* | n_2 *true*)

Convert the ASCII character *char* (using number base n_1) to its binary equivalent n_2, accompanied by a true flag. If the conversion is invalid, leave only a false flag. For example, **DECIMAL 53 10 DIGIT** will leave **5 1** on the stack because 53 is the ASCII code for '5' and is a legitimate digit in base 10. On the other hand, **DECIMAL 74 16 DIGIT** will leave only **0** on the stack because 74 is the ASCII code for 'J' and is *not* a legitimate digit in base 16. However, **DECIMAL 74 20 DIGIT** will leave **19 1** on the stack because 'J' *is* a legitimate digit in base 20.

DISK-HEAD SCR 40 -COPY

(---)

Writes a disk header on Forth screen 0 that makes the disk compatible with the TI 99/4A Disk Manager and with TI BASIC.

DISK_BUF RESIDENT

(--- *addr*)

A user variable that points to the first byte in VDP RAM of the 1K disk buffer.

DISK_HI RESIDENT

(--- *addr*)

A user variable which contains the Forth screen number immediately above the Forth screen range wherein screen writes are permitted.

DISK_LO RESIDENT

(--- *addr*)

A user variable which contains the first Forth screen number of the range wherein disk writes are permitted.

DISK_SIZE RESIDENT

(--- *addr*)

A user variable whose value is the number of Forth screens logically assigned to a diskette.

DLITERAL RESIDENT

Compilation: (*d* ---) Runtime: (--- *d*) Interpretation: (---)

Same behavior as **LITERAL**, *q.v.,* except for a double number *d*

DLT SCR 71 -FILE

(---)

The file I/O routine that deletes the file whose PAB (Peripheral Access Block) is pointed to by **PAB-ADDR** .

DMINUS RESIDENT

(d_1 --- d_2)

Convert d_1 to its double number two's complement d_2.

DMODE SCR 63 -GRAPH

(--- *addr*)

A variable that determines which dot mode is currently in effect. A **DMODE** value of 0 indicates DRAW mode, a value of 1 indicates UNDRAW mode, and a value of 2 indicates DOT-TOGGLE mode. This variable is set by the **DRAW** , **UNDRAW** and **DTOG** words.

DO RESIDENT

Compilation: (*addr n* ---) Runtime: (n_1 n_2 ---)

Occurs in a colon-definition in the form:

```
DO … LOOP
DO … +LOOP
```

When compiling within the colon-definition, **DO** compiles **(DO)** , leaving the following address *addr* and *n* for later error checking.

At run time, **DO** begins a sequence with repetitive execution controlled by a loop limit n_1 and an index with initial value n_2. **DO** removes these from the stack. Upon reaching **LOOP** , the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after **DO** , otherwise the loop parameters are discarded and execution continues ahead. Both n_1 and n_2 are determined at runtime and may be the result of other operations. Within a loop, **I** will copy the current value of the index to the stack. See **I** , **LOOP** , **+LOOP** and **LEAVE** .

DOES> RESIDENT

(---)

A word which defines the runtime action within a high-level defining word. **DOES>** alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following **DOES>** . It is always used in combination with **<BUILDS** . When the **DOES>** part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multidimensional arrays and compiler generation.

DOT SCR 63 -GRAPH

(*dotcol dotrow* ---)

Plots a dot at (*dotcol*,*dotrow*) in whatever mode is selected by **DMODE** and in whatever color is selected by **DCOLOR** .

DP RESIDENT

(--- *addr*)

A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by **HERE** and altered by **ALLOT** .

DPL RESIDENT

(--- *addr*)

A user variable containing the number of digits to the right of the decimal point on double integer input. It may also be used to hold output column location of a decimal point in user-generated formatting. The default value on single number input is -1.

DR0 DR1 DR2 RESIDENT

(---)

Command to select disk drives by presetting **OFFSET** . The contents of **OFFSET** is added to the block number in **BLOCK** to allow for this selection. **OFFSET** is suppressed for error text so that it may always originate from drive 0.

DRAW SCR 63 -GRAPH

(---)

Sets **DMODE** equal to 0. This means that dots are plotted in the ‘on’ state.

DRIVE RESIDENT

(*n* ---)

Adjusts **OFFSET** so that the drive number on the stack becomes the first drive in the system.

DROP RESIDENT

(*n* ---)

Drop the top number from the stack.

DSPLY SCR 69 -FILE

(---)

Assigns the attribute DISPLAY to the file pointed to by **PAB-ADDR** .

DSRLNK SCR 33 -SYNONYMS

(---)

Links a Forth program to any Device Service Routine (DSR) in ROM. Before this instruction may be used, a PAB must be set up in VDP RAM.

DTEST SCR 39 -COPY

(---)

Performs a non-destructive test of the disk in DR0 by attempting to read each Forth screen.

DTOG SCR 63 -GRAPH

(---)

Sets **DMODE** equal to 2. This means that each dot plotted takes on the opposite state as the dot currently at that location.

DUMP SCR 43 -DUMP

(*addr n* ---)

Print the contents of *n* memory locations beginning at *addr*. Both addresses and contents are shown in hexadecimal notation. See **PAUSE** .

DUP RESIDENT

(*n* --- *n n*)

Duplicates the value on the stack.

DXY SCR 59 -GRAPH

($dotcol_1$ $dotrow_1$ $dotcol_2$ $dotrow_2$ --- n_1 n_2)

Places on the stack the square of the ***x*** distance n_1 and the square of the ***y*** distance n_2 between the points ($dotcol_1$,$dotrow_1$) and ($dotcol_2$,$dotrow_2$).

ECOUNT RESIDENT

(--- *addr*)

A user variable that contains an error count. This is used to prevent error recursion.

ED@ (*EDITOR1 Vocabulary*) SCR 38 -EDITOR

(---)

Brings you back into the 40-column editor on the last Forth screen you edited. This screen is pointed to by **SCR** . Must be in Text mode.

ED@ (*EDITOR2 Vocabulary*) SCR 29 -64 SUPPORT

(---)

Brings you back into the 64-column editor on the last Forth screen you edited. This screen is pointed to by **SCR** .

EDIT (*EDITOR1 Vocabulary*) SCR 38 -EDITOR

(*scr* ---)

Brings you into the 40-column editor on the specified Forth screen. Must be in Text mode.

EDIT (*EDITOR2 Vocabulary*) SCR 29 -64SUPPORT

(*scr* ---)

Brings you into the 64-column editor on the specified Forth screen.

ELSE RESIDENT

Compilation: ($addr_1$ n_1 --- $addr_2$ n_2) Runtime: (---)

Occurs within a colon-definition in the form:

```
IF … ELSE … ENDIF
```

At compile-time, **ELSE** emplaces **BRANCH** , reserving a branch offset and leaves the address $addr_2$ and n_2 for error testing. **ELSE** also resolves the pending forward from **IF** by calculating the offset from $addr_1$ to **HERE** and storing it at $addr_1$.

At runtime, **ELSE** executes after the true part following **IF** . **ELSE** forces execution to skip over the following false part and resume execution after **ENDIF** . It has no stack effect.

EMIT RESIDENT

(*char* ---)

Transmit ASCII character *char* to the selected output device. **OUT** is incremented for each character output.

EMIT8 RESIDENT

(*char* ---)

Transmit an 8-bit character *char* to the selected output device. **OUT** is incremented for each character output.

EMPTY-BUFFERS RESIDENT

(---)

Mark all block buffers as empty, not necessarily affecting the contents. Updated blocks are not written to the disk. This is also an initialization procedure before first use of the disk.

ENCLOSE RESIDENT

($addr_1$ *char* --- $addr_1$ n_1 n_2 n_3)

The text scanning primitive used by **WORD** . From the text address $addr_1$ and an ASCII-delimiting character *char*, is determined the byte offset n_1 to the first non-delimiter character, the offset n_2 to the delimiter after the text and the offset n_3 to the first character not included. This procedure will not process past an ASCII 'null' (0), treating it as an unconditional delimiter.

END RESIDENT

(*flag* ---)

This is an alias or duplicate definition for **UNTIL** .

ENDCASE RESIDENT

(*n* ---)

Terminates the **CASE** construct and, if actually executed at runtime because all intervening **OF … ENDOF** clauses failed, removes the number *n* left on the stack. See **CASE** .

ENDIF RESIDENT

Compilation: (*addr n* ---)

Occurs in a colon-definition in the form:

```
IF … ENDIF
IF … ELSE … ENDIF
```

AT runtime, **ENDIF** serves only as the destination of a forward branch from **IF** or **ELSE** . It marks the conclusion of the conditional structure. **THEN** is another name for **ENDIF** . Both names are supported in fig-Forth. See also **IF** and **ELSE** .

At compile-time, **ENDIF** computes the forward branch offset from addr to **HERE** and stores it at addr. *n* is used for error tests.

ENDOF RESIDENT

(---)

Terminates the **OF** construct within the **CASE** construct. If executed at runtime, causes execution to proceed just beyond **ENDCASE** . See **OF** .

ERASE RESIDENT

(*addr n* ---)

Clear *n* bytes of memory to zero starting at *addr*.

ERROR RESIDENT

(n_1 --- n_2 n_3)

ERROR processes error notification and restarts the interpreter. **WARNING** is first examined. If **WARNING** < 1, **(ABORT)** is executed. The sole action of **(ABORT)** is to execute **ABORT** . This allows the user to (cautiously!) modify this behavior by redefining **(ABORT)** . **ABORT** clears the stacks and executes **QUIT** , which stops compilation and restarts the interpreter. If **WARNING** ≥ 0, **ERROR** leaves the contents of **IN** n_2 and **BLK** n_3 on the stack to assist in determining the location of the error. If **WARNING** > 0, **ERROR** prints the text of line n_1, relative to Forth screen 4 of drive 0. If **WARNING** = 0, **ERROR** prints n_1 as an error number (as in a non-disk installation). The last thing **ERROR** does is to execute **QUIT** , which, as above, stops compilation and restarts the interpreter.

EXECUTE RESIDENT

(*cfa* ---)

Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXP SCR 50 -FLOAT

(f_1 --- f_2)

Raises ***e*** to the power specified by the floating point number f_1 on the stack and leaves the result f_2 on the stack.

EXPECT RESIDENT

(*addr count* ---)

Transfer characters from the terminal to *addr* until ***<ENTER>*** or *count* characters have been received. One or more nulls are added at the end of the text.

F! SCR 45 -FLOAT

(*f addr* ---)

Stores a floating point number *f* into the 4 words (cells) beginning with the specified address.

F* SCR 46 -FLOAT

($f_1 f_2$ --- f_3)

Multiplies the top two floating point numbers on the stack and leaves the result on the stack. $f_1 * f_2 = f_3$.

F+ SCR 46 -FLOAT

($f_1 f_2$ --- f_3)

Adds the top two floating point numbers on the stack and places the result on the stack. $f_1 * f_2 = f_3$.

F– SCR 46 -FLOAT

($f_1 f_2$ --- f_3)

Subtracts f_2 from f_1 and places the result on the stack ($f_1 - f_2 = f_3$).

F->S SCR 46 -FLOAT

(f --- n)

Converts a floating point number f on the parameter stack into a single precision number n.

F-D" SCR 70 -FILE

(---)

Expects a file descriptor ending with a **"** to follow. This instruction places the file descriptor in the PAB (Peripheral Access Block) pointed to by **PAB-ADDR** .

F. SCR 48 -FLOAT

(f ---)

Prints a floating point number in BASIC format to the output device.

F.R SCR 48 -FLOAT

(f n ---)

Prints the floating point number f in BASIC format right justified in a field of width n.

F/ SCR 46 -FLOAT

($f_1 f_2$ --- f_3)

Divides f_1 by f_2 and leaves the floating point quotient f_3 on the stack. $f_1 / f_2 = f_3$.

F0< SCR 49 -FLOAT

(f --- *flag*)

Compares the floating point number f on the stack to 0. If it is less than 0, a true flag is left on the stack, else a false flag is left.

F0= SCR 49 -FLOAT

(f --- *flag*)

Compares the floating point number f on the stack to 0. If it is equal to 0, a true flag is left on the stack, else a false flag is left.

F< SCR 49 -FLOAT

($f_1 f_2$ --- *flag*)

Leaves a true flag if $f_1 < f_2$, else leaves a false flag.

F= SCR 49 -FLOAT

($f_1 f_2$ --- *flag*)

Leaves a true flag if $f_1 = f_2$, else leaves a false flag.

F> SCR 49 -FLOAT

($f_1 f_2$ --- *flag*)

Leaves a flag if $f_1 > f_2$, else leaves a false flag.

F@ SCR 45 -FLOAT

(*addr* --- *f*)

Retrieves the floating point contents *f* of the given address (4 words) and places it on the stack.

FAC SCR 45 -FLOAT

(--- *addr*)

A constant which contains the address of the **FAC** register.

FAC->S SCR 46 -FLOAT

(--- *n*)

Converts a floating point number in **FAC** to a single precision number and places it on the parameter stack.

FAC> SCR 45 -FLOAT

(--- *f*)

Brings a floating point number *f* from **FAC** to the stack.

FAC>ARG SCR 46 -FLOAT

(---)

Moves a floating point number from **FAC** into **ARG** .

FADD SCR 45 -FLOAT

(---)

Adds the floating point number in **FAC** to the floating point number in **ARG** and leaves the result in **FAC** .

FDIV SCR 45 -FLOAT

(---)

Divides the floating point number in **FAC** by the floating point number in **ARG** leaving the quotient in **FAC** .

FDROP SCR 45 -FLOAT

(*f* ---)

Drops the top floating point number *f* from the stack.

FDUP SCR 45 -FLOAT

(*f* --- *f f*)

Duplicates the top floating point number *f* on the stack.

FENCE RESIDENT

(--- *addr*)

A user variable containing an address (usually the NFA of a Forth word) below which **FORGET**ting is trapped. To **FORGET** below this point the user must alter the contents of **FENCE** . It *is* possible to set the value of **FENCE** to a value that is actually less than the address of the end of the last word in the core dictionary (**TASK**) such that **UNFORGETABLE** [*sic*] will report false; however, **FORGET** will still trap that error.

FF. SCR 48 -FLOAT

(*f* n_1 n_2 ---)

Prints the floating point number *f* with n_2 digits following the decimal point and a maximum of n_1 digits.

FF.R SCR 48 -FLOAT

(*f* n_1 n_2 n_3 ---)

Prints the floating point number *f*, with n_2 digits following the decimal point, right justified in a field of width n_3 with a maximum of n_1 digits.

FILE SCR 68 -FILE

($vaddr_1$ *addr* $vaddr_2$ ---)

A defining word which permits you to create a word by which a file will be known. You must place on the stack the **PAB-ADDR** , **PAB-BUF** and **PAB-VBUF** addresses you wish to be associated with the file.

Used in the form:

***vaddr*$_1$ *addr* *vaddr*$_2$ FILE cccc**

When **cccc** executes, **PAB-ADDR** , **PAB-BUF** and **PAB-VBUF** are set to $vaddr_1$, *addr* and $vaddr_2$, respectively.

FILL RESIDENT

(*addr count b* ---)

Fill memory beginning at *addr* with *count* bytes of byte *b*.

FIRST RESIDENT

(--- *addr*)

A constant that leaves the address of the first (lowest) block buffer.

FIRST$ RESIDENT

(--- *addr*)

A user variable which contains the first byte of the disk buffer area.

FLD RESIDENT

(--- *addr*)

A user variable for control of number output field width. Presently unused in fig-Forth and TI Forth.

FLERR SCR 49 -FLOAT

(--- *n*)

Returns on the stack the contents *n* of the floating point status register (**8354h**).

FLUSH RESIDENT

(---)

Writes to disk all disk buffers that have been marked as updated.

FMUL SCR 45 -FLOAT

(---)

Multiplies the floating point number in **FAC** with the floating point number in **ARG** leaving the product in **FAC** .

FORGET RESIDENT

(---)

Executed in the form:

```
FORGET  cccc
```

Deletes the definition named **cccc** from the dictionary along with all dictionary entries physically following it.

FORGET first checks the LFA of **cccc** to see if it is lower than the address in **FENCE** . If it is not, **FORGET** then checks whether it is lower than the address of the last byte of the core dictionary. If it is not lower than either of these addresses, **FORGET** updates **HERE** to the LFA of **cccc** , effectively deleting the desired part of the dictionary. Otherwise, an appropriate error message is displayed.

FORMAT-DISK SCR 33 -SYNONYMS

(*n* ---)

Initializes the disk in DR0 (*n* = 0), DR1 (*n* = 1) or DR2 (*n* = 2) for use with the Forth system. ***Caution:*** All data on the disk will be destroyed. Also, disks initialized by the Disk Manager may be used without any changes. Drive number *n* must be 0, 1 or 2.

FORTH RESIDENT

(---)

The name of the primary vocabulary. Execution makes **FORTH** the **CONTEXT** vocabulary. Until additional user vocabularies are defined, new user definitions become a part of **FORTH** because it is at that point also the **CURRENT** vocabulary. **FORTH** is immediate, so it will execute during the creation of a colon definition to select this vocabulary at compile time.

FORTH-COPY SCR 39 -COPY

(---)

Copies the entire disk in DR1 onto the disk in DR0.

FORTH_LINK RESIDENT

(--- *addr*)

A user variable used for vocabulary linkage.

FOVER SCR 45 -FLOAT

(f_1 f_2 --- f_1 f_2 f_1)

Copies the second floating point number on the stack to the top of the stack.

FRND SCR 46 FLOAT

(--- f)

Generates a pseudo-random floating point number greater than or equal to 0 and less than 1.

FSUB SCR 45 -FLOAT

(---)

Subtracts the floating point number in **ARG** from the number in **FAC** and leaves the result in **FAC** .

FSWAP SCR 45 -FLOAT

(f_1 f_2 --- f_2 f_1)

Swaps the top two floating point numbers on the stack.

FXD SCR 68 -FILE

(---)

Assigns the attribute FIXED to the file whose PAB (Peripheral Access Block) is pointed to by **PAB-ADDR** .

GCHAR SCR 58 -GRAPH

(*col row* --- *char*)

Returns on the stack the ASCII code *char* of the character currently at (*col*,*row*). *Note:* Rows and columns are numbered from 0.

GET-FLAG SCR 68 -FILE

(--- *b*)

Retrieves the flag byte *b* from the current PAB and places it on the stack.

GOTOXY RESIDENT

(*col row* ---)

Places the cursor at the designated column *col* and row *row* position. *Note:* Rows and columns are numbered from 0.

GPLLNK SCR 33 -SYNONYMS

(*addr* ---)

Links a Forth program to the Graphics Programming Language (GPL) routine located at the given address.

GRAPHICS SCR 52 -GRAPH1

(---)

Converts from present display screen mode into standard Graphics mode configurations.

GRAPHICS2 SCR 54 -GRAPH2

(---)

Converts from present Forth screen mode into standard Graphics2 mode configuration.

HCHAR SCR 57 -GRAPH

(*col row count char* ---)

Prints a horizontal stream of a specified character *char* beginning at (*col*,*row*) and having a length *char*. *Note:* Rows and columns are numbered from 0.

HERE RESIDENT

(--- *addr*)

Leave the address of the next available dictionary location.

HEX RESIDENT

(---)

Set the numeric conversion base to sixteen (hexadecimal).

HLD RESIDENT

(--- *addr*)

A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD RESIDENT

(*char* ---)

Used between **<#** and **#>** to insert an ASCII character into a pictured numeric output string, *e.g.*, **2E HOLD** will place a decimal point.

HONK SCR 60 -GRAPH

(---)

Produces the sound associated with incorrect input.

I RESIDENT

(--- *n*)

Used within a **DO** loop to copy the loop index to the stack. Other use is implementation dependent. **I** is a synonym for **R** .

ID. RESIDENT

(*nfa* ---)

Print a definition's name from its name field address *nfa*.

IF RESIDENT

Compilation: (--- *addr n*) Runtime: (*flag* ---)

Occurs in a colon definition in form:

IF *(true part)* … ENDIF

IF *(true part)* … ELSE *(false part)* … ENDIF

At compile time, **IF** compiles **0BRANCH** and reserves space for an offset at *addr*; *addr* and *n* are used later for resolution of the offset and error testing.

At runtime, **IF** selects execution based on a Boolean flag. If *flag* is *true* (non-zero), execution continues ahead through the true part. If *flag* is *false* (zero), execution skips to just after **ELSE** to execute the false part. After either part, execution resumes after **ENDIF** . **ELSE** and its false part are optional. If missing, false execution skips to just after **ENDIF** .

IMMEDIATE RESIDENT

(---)

Mark the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled. *i.e.*, the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceding it with **[COMPILE]** .

IN RESIDENT

(--- *addr*)

A user variable containing the byte offset within the current input text buffer (terminal or disk) from which the next text will be accepted. **WORD** uses and moves the value of **IN** .

INDEX SCR 73 -PRINT

(n_1 n_2 ---)

Prints to the terminal a list of the line #0 comments from Forth screen n_1 through Forth screen n_2. See **PAUSE** .

INPT SCR 69 -FILE

(---)

Assigns the attribute INPUT to the file whose PAB is pointed to by **PAB-ADDR** .

INT SCR 50 -FLOAT

(f_1 --- f_2)

Leaves the integer portion of a floating point number on the stack.

INTERPRET RESIDENT

(---)

The outer text interpreter, which sequentially executes or compiles text from the input stream (terminal or disk) depending on **STATE** . If the word name cannot be found after a search of **CONTEXT** and then **CURRENT** , it is converted into a number according to the current base. That also failing, an error message echoing the name with a "?" will be given. Text input will be taken according to the convention for **WORD** . If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose than to force this action. See **NUMBER** .

INTLNK RESIDENT

(--- *addr*)

A user variable which is a pointer to the Interrupt Service linkage.

INTRNL SCR 69 -FILE

(---)

Assigns the attribute INTERNAL to the file whose PAB is pointed to by **PAB-ADDR** .

ISR RESIDENT

(--- *addr*)

A user variable that initially contains the address of the interrupt service linkage code to install an Interrupt Service Routine. The user must modify **ISR** to contain the CFA of the routine to be executed each 1/60 second. Next, the contents of **83C4h** must be modified to point to this address. Note that the interrupt service linkage code address is also available in **INTLNK** .

J RESIDENT

(--- *n*)

Copies the loop index of the next outer loop to the stack.

JOYST SCR 60 -GRAPH

(n_1 --- *char* n_2 n_3)

Allows you to accept input from joystick #1 and the left side of the keyboard (n_1 = 1) or from joystick #2 and the right side of the keyboard (n_1 = 2). Values returned are the character code *char* of the key pressed, the ***x*** status n_2 and the ***y*** status n_3.

KEY RESIDENT

(--- *char*)

Leave the ASCII value of the next terminal key struck.

KEY8 RESIDENT

(--- *char*)

Leave the 8-bit value of the next terminal key struck.

L/SCR RESIDENT

(--- *n*)

Returns on the stack the number of lines per Forth screen.

LATEST RESIDENT

(--- *nfa*)

Leave the name field address *nfa* of the most recently defined word in the **CURRENT** vocabulary. At compile time, this "latest" word will be the most recently compiled word.

LD SCR 71 -FILE

(*n* ---)

The file I/0 process to load a program file from a disk into VDP RAM. The parameter *n* specifies the maximum number of bytes to be loaded and is usually the size of the file on disk. The file's PAB must be set up and be the current PAB, to which **PAB-ADDR** points, before executing this word.

LDCR SCR 88

(n_1 n_2 *addr* ---)

Performs a TMS9900 LDCR instruction. The CRU base address *addr* will be shifted left one bit and stored in workspace register R12 prior to executing the TMS9900 LDCR instruction. The value n_1 is transferred to the CRU with a field width of n_2 bits.

LEAVE RESIDENT

(---)

Force termination of a **DO** loop at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and the execution proceeds normally until **LOOP** or **+LOOP** is encountered.

LFA RESIDENT

(*pfa* --- *lfa*)

Convert the parameter field address *pfa* of a dictionary definition to its link field address *lfa*.

LIMIT RESIDENT

(--- *addr*)

A constant which leaves the address *addr* just above the highest memory available for a disk buffer.

LIMIT$ RESIDENT

(--- *addr*)

A user variable that contains the address just above the highest memory available for a disk buffer. The address of **LIMIT$** is left on the stack.

LINE SCR 64 -GRAPH

($dotcol_1$ $dotrow_1$ $dotcol_2$ $dotrow_2$ ---)

The high resolution graphics routine which plots a line from ($dotcol_1$,$dotrow_1$) to ($dotcol_2$,$dotrow_2$). **DCOLOR** and **DMODE** must be set before this instruction is used.

LIST RESIDENT

(*scr* ---)

Lists the specified Forth screen to the output device. See **PAUSE** .

LIT RESIDENT

(--- *n*)

Within a colon-definition, **LIT** is automatically compiled before each 16-bit literal number encountered in input text. Later execution of **LIT** causes the contents of the next dictionary address to be pushed to the stack.

LITERAL RESIDENT

Compilation: (*n* ---) Runtime: (--- *n*) Interpretation: (---)

During compilation, compiles the stack value *n* as a 16-bit literal. This will execute during a colon definition. The intended use is:

: **xxx [** ***calculation*** **] LITERAL ;**

Compilation is suspended for the compile-time calculation of a value. Compilation is resumed and **LITERAL** compiles this value.

At runtime, *n* is pushed to the stack. Interpretation of **LITERAL** does nothing, unlike other compiling words.

LOAD RESIDENT

(*n* ---)

Begin interpretation of Forth screen *n*. Loading will terminate at the end of the Forth screen or at **;S** . See **;S** and **-->** .

LOG SCR 50 -FLOAT

(f_1 --- f_2 | f_1)

The floating point operation that returns the natural logarithm f_2 of the floating point number f_1. If f_1 is 0 or negative, the original number f_1 is returned instead.

LOOP RESIDENT

Compilation: (*addr n* ---)

Occurs in a colon definition in the form:

DO … LOOP

At runtime, **LOOP** selectively controls branching back to the corresponding **DO** based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to **DO** occurs until the index equals or exceeds the limit. At that time, the parameters are discarded and execution continues ahead.

At compile time, **LOOP** compiles **(LOOP)** and uses *addr* to calculate an offset to **DO** . *n* is used for error testing.

M* RESIDENT

(n_1 n_2 --- d)

A mixed magnitude math operation that leaves the double number signed product d of two signed numbers, n_1 and n_2.

M/ RESIDENT

(d n_1 --- n_2 n_3)

A mixed magnitude math operator that leaves the signed remainder n_2 and signed quotient n_3, from a double number dividend d and divisor n_1. The remainder takes its sign from the dividend.

M/MOD RESIDENT

(ud_1 u_2 --- u_3 ud_4)

An unsigned mixed magnitude math operation that leaves an unsigned double quotient ud_4 and a single remainder u_3, from a double dividend ud_1 and a single divisor u_2.

MAGNIFY SCR 60 -GRAPH

(n_1 ---)

Alters the sprite magnification factor to be n_1. The value of n_1 must be 0, 1, 2 or 3.

MAX RESIDENT

(n_1 n_2 --- n_3)

Leave the greater n_3 of the two numbers, n_1 and n_2.

MCHAR SCR 62 -GRAPH

(*n col row* ---)

Places a square of color *n* at (*col*,*row*). Used in multicolor mode.

MENU SCR 20 BOOT SCR

(---)

Displays the available Load Options.

MESSAGE RESIDENT

(*n* ---)

Print on the selected output device the text of line *n* relative to screen 4 of drive 0. The value of *n* may be positive or negative. **MESSAGE** may be used to print incidental text such as report headers. If **WARNING** = 0, the message will simply be printed as a number (disk unavailable).

MIN RESIDENT

(n_1 n_2 --- n_3)

Leave the smaller n_3 of the two numbers (n_1 and n_2).

MINIT SCR 62 -GRAPH

(---)

Initializes the monitor screen for use with **MCHAR** .

MINUS RESIDENT

(n_1 --- n_2)

Leave the two's complement n_2 of a number n_1.

MOD RESIDENT

(n_1 n_2 --- *rem*)

Leave the remainder *rem* of n_1/n_2, with the same sign as n_1.

MON SCR 33 -SYNONYMS

(---)

Exit to the TI 99/4A color bar display screen.

MOTION SCR 59 -GRAPH

(n_1 n_2 *spr* ---)

Assigns a horizontal n_1 and vertical n_2 velocity to the specified sprite *spr*.

MOVE RESIDENT

($addr_1$ $addr_2$ *n* ---)

Move the contents of *n* memory cells (16-bit contents) beginning at $addr_1$ into *n* cells beginning at $addr_2$. The contents of $addr_1$ is moved first.

MULTI SCR 53 -GRAPH

(---)

Converts from present display screen mode into standard Multicolor mode configuration.

MYSELF RESIDENT

(---)

Used in a colon definition. Places the code field address (CFA) of a word into its own definition. This permits recursion.

NFA RESIDENT

(*pfa* --- *nfa*)

Convert the parameter field address *pfa* of a definition to its name field address *nfa*.

NOP RESIDENT

(---)

A do-nothing instruction. **NOP** is useful for patching as in assembly code.

NUMBER RESIDENT

(*addr* --- *d*)

Convert a character string left at *addr* with the character count in the first byte, to a signed double number *d* , using the current numeric base. If a decimal point is encountered in the text, its position will be given in **DPL** , but no other effect occurs. If numeric conversion is not possible, an error message will be given.

OF RESIDENT

(*n* --- *n* |)

Initiates the **OF … ENDOF** construct inside of the **CASE** construct. *n* is compared to the value which was on top of the stack when **CASE** was executed. If the numbers are identical, the words between **OF** and **ENDOF** will be executed. Otherwise, *n* is put back on the stack. See **CASE** .

OFFSET RESIDENT

(--- *addr*)

A user variable which may contain a block offset to disk drives. The contents of **OFFSET** is added to the stack number by **BLOCK** . Messages issued by **MESSAGE** are independent of **OFFSET** . See **BLOCK** , **DR0** and **MESSAGE** .

OPN SCR 71 -FILE

(---)

Opens the file whose PAB is pointed to by **PAB-ADDR** .

OR RESIDENT

(n_1 n_2 --- n_3)

Leave the bit-wise logical OR n_3 of two 16-bit values, n_1 and n_2.

OUT RESIDENT

(--- *addr*)

A user variable that contains a value incremented by **EMIT** and **EMIT8** . The user may alter and examine **OUT** to control display formatting.

OUTPT SCR 69 -FILE

(---)

Assigns the attribute OUTPUT to the file whose PAB is pointed to by **PAB-ADDR** .

OVER RESIDENT

(n_1 n_2 --- n_1 n_2 n_1)

Copy the second stack value n_1 to the top of the stack.

PAB-ADDR SCR 68 -FILE

(--- *addr*)

A variable containing the VDP address of the first byte of the current PAB (Peripheral Access Block).

PAB-BUF SCR 68 -FILE

(--- *addr*)

A variable which holds the address of the area in CPU RAM used as the source or destination of the data to be transferred to/from a file. This is a file I/O word.

PAB-VBUF SCR 68 -FILE

(--- *addr*)

A variable pointing to a VDP RAM buffer which serves as a temporary buffer when transferring data to/from a file. The VDP address stored in **PAB-VBUFF** is also stored in the file's PAB.

PABS RESIDENT

(--- *addr*)

A user variable which points to a region in VDP RAM, which has been set aside for creating PABs.

PAD RESIDENT

(--- *addr*)

Leave the address of the text output buffer, which is a fixed offset (68 bytes in TI Forth) above **HERE** . Every time **HERE** changes, **PAD** is updated.

PAUSE RESIDENT

(--- *flag*)

Checks for a keystroke and issues *false* if none, *true* if ***<BREAK>*** (***<CLEAR>*** or ***<FCTN+4>***) or idles until a second keystroke before issuing *false* (or *true* if second keystroke is ***<BREAK>***). The words **LIST** , **INDEX** , **DUMP** and **VLIST** all call the word **PAUSE** . These routines exit when *flag* = *true*. **PAUSE** allows the user to temporarily halt the output by pressing any key. Pressing another key will allow continuation. To exit one of these routines prematurely, press ***<BREAK>*** .

PDT SCR 57 -GRAPH

(--- *vaddr*)

A constant which contains the VDP address of the Pattern Descriptor Table. Default value is **800h**.

PFA RESIDENT

(*nfa* --- *pfa*)

Convert the name field address *nfa* of a compiled definition to its parameter field address *pfa*.

PI SCR 50 -FLOAT

(--- *f*)

A floating point approximation of π to 13 significant figures. (3.141592653590)

PREV RESIDENT

(--- *addr*)

A user variable containing the address of the disk buffer most recently referenced. The **UPDATE** command marks this buffer to be later written to disk.

PUT-FLAG SCR 68 -FILE

(*b* ---)

Writes the flag byte *b* into the appropriate PAB referenced by **PAB-ADDR** .

QUERY RESIDENT

(---)

Input 80 characters of text (or until ***<ENTER>*** is pressed) from the operator's terminal. Text is positioned at the address contained in **TIB** with **IN** set to 0.

QUIT RESIDENT

(---)

Clear the return stack, stop compilation and return control to the operator's terminal. No message is given, including the usual "ok".

R RESIDENT

(--- *n*) (R: *n* --- *n*)

Copy the top of the return stack to the parameter stack.

R# RESIDENT

(--- *addr*)

A user variable which may contain the location of an editing cursor or other file-related function.

R->BASE RESIDENT

(---) (R: *n* ---)

Restore the current base from the return stack. See **BASE->R** .

R/W RESIDENT

(*addr* n_1 *flag* ---)

The fig-Forth standard disk read/write linkage. The source or destination block buffer address is *addr*, n_1 is the sequential number of the referenced block and *flag* indicates whether the operation is write (*flag* = 0) or read (*flag* = 1). **R/W** determines the location on mass storage, performs the read/write and error checking.

R0 RESIDENT

(--- *addr*)

A user variable containing the initial location of the return stack. Pronounced "r zero". See **RP!** .

R> RESIDENT

(--- *n*) (R: *n* ---)

Remove the top value from the return stack and leave it on the parameter stack. See **>R** and **R** .

RANDOMIZE SCR 33 -SYNONYMS

(---)

Creates an unpredictable seed for the random number generator.

RD SCR 71 -FILE

(--- *count*)

The file I/O instruction that reads from the current PAB. This instruction uses **PAB-BUF** and **PAB-VBUF** .

RDISK RESIDENT

(*addr* n_1 n_2 --- n_3)

The primitive routine that performs disk reads. The address where the block is to be written in CPU RAM is *addr*. The block number n_1, the number of bytes per block is n_2 and n_3 is the returned error code.

REC-LEN SCR 69 -FILE

(*b* ---)

Stores the length *b* of the record for the upcoming write into the appropriate byte in the current PAB.

REC-NO SCR 69 -FILE

(*n* ---)

Writes a zero-based record number *n* into the appropriate location in the current PAB.

REPEAT RESIDENT

Compilation: (*addr n* ---)

Used within a colon-definition in the form:

BEGIN … WHILE … REPEAT

At runtime, **REPEAT** forces an unconditional branch back to just after the corresponding **BEGIN** .

At compile-time, **REPEAT** compiles **BRANCH** and the offset from **HERE** to *addr*. *n* is used for error testing.

RLTV SCR 69 -FILE

(---)

Assigns the attribute RELATIVE to the file whose PAB is pointed to by **PAB-ADDR** .

RND SCR 33 -SYNONYMS

(n_1 --- n_2)

Generates a positive random integer n_2 greater than or equal to 0 and less than n_1.

RNDW SCR 33 -SYNONYMS

(--- *n*)

Generates a random word. The value of the word may be positive or negative depending on whether the sign bit is set.

ROT RESIDENT

(n_1 n_2 n_3 --- n_2 n_3 n_1)

Rotate the top three values on the stack, bringing the third n_1 to the top.

RP! RESIDENT

(---)

A procedure to initialize the return stack pointer from user variable **R0** .

RSTR SCR 71 -FILE

(*n* ---)

Restores the file whose PAB is pointed to by the current PAB to the specified record number *n*.

S->D RESIDENT

(*n* --- *d*)

Sign-extend a single number *n* to form a double number *d*.

S->F SCR 46 -FLOAT

(*n* --- *f*)

Converts a single-precision number *n* on the stack to a floating point number *f*.

S->FAC SCR 46 -FLOAT

(*n* ---)

Takes a single-precision number *n* from the stack, converts it to floating point, and leaves it in FAC.

S0 RESIDENT

(--- *addr*)

User variable that points to the base of the parameter stack. Pronounced "s zero". See **SP!** .

SATR SCR 57 -GRAPH

(--- *vaddr*)

A constant whose value *vaddr* is the VDP address of the Sprite Attribute List. Default value is **300h**.

SBO SCR 88 -CRU

(*addr* ---)

This word expects to find on the stack the CRU address *addr* of the bit to be set to 1. **SBO** will put this address into workspace register R12, shift it left (double it) and execute **0 SBO**, to effect setting the bit. See CRU documentation in the *Editor/Assembler Manual* for more information.

SBZ SCR 88 -CRU

(*addr* ---)

This word expects to find on the stack the CRU address *addr* of the bit to be reset to 0. **SBZ** will put this address into workspace register R12, shift it left (double it) and execute **0 SBZ**, to effect resetting the bit. See CRU documentation in the *Editor/Assembler Manual* for more information.

SCOPY SCR 39 -COPY

(scr_1 scr_2 ---)

Copies the source Forth screen scr_1 to the destination Forth screen scr_2 on disk. Does not destroy the source screen.

SCR RESIDENT

(--- *addr*)

A user variable containing the Forth screen number most recently referenced by **LIST** or **EDIT** .

SCREEN SCR 58 -GRAPH

(*n* ---)

Changes the display screen color to the color specified *n*. The foreground (FG) and background (BG) screen colors must be placed in the low-order byte of *n*, with FG the high-order 4 bits and BG the low-order 4 bits, *e.g.*, *n* = 27 (**1Bh**) for black on light yellow.

SCRN_END RESIDENT

(--- *addr*)

A user variable containing the address *addr* of the byte immediately following the last byte of the display screen image table to be used as the logical display screen.

SCRN_START RESIDENT

(--- *addr*)

A user variable containing the address *addr* of the first byte of the display screen image table to be used as the logical display screen.

SCRN_WIDTH RESIDENT

(--- *addr*)

A user variable which contains the number of characters that will fit across the display screen. (32 or 40) Used by the display screen scroller.

SCRTCH SCR 71 -FILE

(*n* ---)

Removes the specified record *n* from the RELATIVE file whose PAB is pointed to by **PAB-ADDR** . [*Editor's Note:* This word should *never* be used. TI never implemented this operation for files. It will *always* result in a file I/O error message.]

SEED SCR 33 -SYNONYMS

(*n* ---)

Places a new seed *n* into the random number generator.

SET-PAB SCR 68 -FILE

(---)

This instruction assumes that **PAB-ADDR** is set. It then zeroes out the PAB (Peripheral Access Block) pointed to by **PAB-ADDR** and places the contents of **PAB-VBUF** into the appropriate word of the PAB. This initializes the PAB.

SETFL SCR 45 -FLOAT

(f_1 f_2 ---)

Performs **>FAC** on f_2 and **>ARG** on f_1.

SIGN RESIDENT

(*n d* --- *d*)

Stores a minus sign (ASCII 45 or **2Dh**) at the current location in a converted numeric output string in the text output buffer if *n* is negative. At the time *n* is evaluated, it is discarded; but, double number *d* is maintained for continued conversion until **#>** removes it from the stack. Must be used between **<#** and **#>** . Using **SIGN** implies that *d* can be negative, which means that *d* should be used to produce *n*. You should then replace *d* with its absolute value (|*d*|) on the stack by using **DABS** . This can be done by pushing *d* to the stack and executing **SWAP OVER DABS** : (*d* --- *n* |*d*|) prior to **<# … SIGN … #>** .

SIN SCR 50 -FLOAT

(f_1 --- f_2)

Finds the SIN f_2 of the floating point number f_1 on the stack and leaves the result f_2 on the stack.

SLA RESIDENT

(n_1 *count* --- n_2)

Arithmetically shifts the number n_1 on the stack *count* bits to the left, leaving the result n_2 on the stack. Shifting by count will be modulo 16 except when *count* = 0, which causes 16 bits to be shifted. To create a word which does not perform a 16-bit shift when *count* is zero, use the following definition for the same stack contents:

```
: SLA0 -DUP IF SLA ENDIF ;
```

SLIT SCR 20 BOOT SCR

(--- *addr*)

SLIT is similar to **LIT** but acts on strings instead of numbers. **SLIT** places the address *addr* of the string following it on the stack. It modifies the top of the return stack to point to just after the string.

SMASH SCR 65 -64SUPPORT

($addr_1$ $count_1$ *n* --- $addr_2$ *vaddr* $count_2$)

The assembly code routine that formats a line of tiny characters. It expects the address $addr_1$ of the line in memory, the number $count_1$ of characters per line, and the line number *n* to which it is to be written. It returns on the stack the line buffer address $addr_2$, a VDP address *vaddr*, and a character count $count_2$. See **CLIST** and **CLINE** .

SMOVE SCR 39 -COPY

(scr_1 scr_2 *count* ---)

Copies *count* Forth screens beginning with the source Forth screen scr_1 to the destination Forth screen scr_2. Overlapping screen ranges may be specified without detrimental effects.

SMTN SCR 57 -GRAPH

(--- *vaddr*)

A constant whose value is the VDP address of the Sprite Motion Table. Default value is **780h**.

SMUDGE RESIDENT

(---)

Used during word definition to toggle the smudge bit in a definition's name field. This prevents an uncompleted definition from being found during dictionary searches until compilation is completed without error.

SP! RESIDENT

(---)

A procedure to initialize the parameter stack pointer from **S0** , the user variable that points to the base of the parameter stack.

SP@ RESIDENT

(--- *addr*)

This word returns the address of the top of the stack as it was before **SP@** was executed, *e.g.*, **1 2 SP@ @ . . .** would type 2 2 1.

SPACE RESIDENT

(---)

Transmit a blank character (ASCII 32|**20h**) to the output device.

SPACES RESIDENT

(*n* ---)

Transmit *n* blank characters (ASCII 32|**20h**) to the output device.

SPCHAR SCR 58 -GRAPH

(n_1 n_2 n_3 n_4 *char* ---)

Defines a character *char* in the Sprite Descriptor Table to have the pattern composed of the 4 words (cells) on the stack.

SPDTAB SCR 57 -GRAPH

(--- *vaddr*)

A constant whose value is the VDP address of the Sprite Descriptor Table. Default value is **800h**. Notice that this coincides with the Pattern Descriptor Table.

SPLIT SCR 55 -SPLIT

(---)

Converts from present display screen mode into standard Split mode configuration.

SPLIT2 SCR 55 -SPLIT

(---)

Converts from present display screen mode into standard Split2 mode configuration.

SPRCOL SCR 58 -GRAPH

(*n spr* ---)

Changes color of the given sprite number *spr* to the color *n* specified.

SPRDIST SCR 60 -GRAPH

(spr_1 spr_2 --- *n*)

Returns on the stack the square of the distance *n* between two specified sprites, spr_1 and spr_2. Distance is measured in pixels and the maximum distance that can be detected accurately is 181 pixels.

SPRDISTXY SCR 60 -GRAPH

(*dotcol dotrow spr* --- *n*)

Places on the stack *n*, the square of the distance between the point (*dotcol*,*dotrow*) and a given sprite *spr*. Distance is measured in pixels and the maximum distance that can be detected accurately is 181 pixels.

SPRGET SCR 59 -GRAPH

(*spr* --- *dotcol dotrow*)

Returns the dot column *dotcol* and dot row *dotrow* position of sprite *spr*.

SPRITE SCR 59 -GRAPH

(*dotcol dotrow n char spr* ---)

Defines sprite number *spr* to have the specified location (*dotcol*,*dotrow*), color *n*, and character pattern *char*. The size of the sprite will depend on the magnification factor.

SPRPAT SCR 59 -GRAPH

(*char spr* ---)

Changes the character pattern of a given sprite *spr* to *char*.

SPRPUT SCR 59 -GRAPH

(*dotcol dotrow spr* ---)

Places a given sprite *spr* at location (*dotcol*,*dotrow*).

SQNTL SCR 69 -FILE

(---)

Assigns the attribute SEQUENTIAL to the file whose PAB is pointed to by **PAB-ADDR** .

SQR SCR 50 -FLOAT

(f_1 --- f_2)

Finds the square root of a floating point number f_1 and leaves the result f_2 on the stack.

SRA RESIDENT

(n_1 *count* --- n_2)

Arithmetically shifts n_1 count bits to the right and leaves the result n_2 on the stack. Shifting by *count* will be modulo 16 except when *count* = 0, which causes 16 bits to be shifted. To create a word which does not perform a 16-bit shift when count is zero, use the following definition for the same stack contents:

```
: SRA0 -DUP IF SRA ENDIF ;
```

SRC RESIDENT

(n_1 *count* --- n_2)

Performs a circular right shift of count bits on n_1 leaving the result n_2 on the stack. If *count* is 0, 16 bits are shifted. To create a word which does not perform a 16-bit shift when *count* is zero, use the following definition for the same stack contents:

```
: SRC0 -DUP IF SRC ENDIF ;
```

SRL RESIDENT

(n_1 *count* --- n_2)

Performs a logical right shift of *count* bits on n_1 and leaves the result n_2 on the stack. If *count* is 0, 16 bits are shifted. To create a word which does not perform a 16-bit shift when count is zero, use the following definition for the same stack contents:

```
: SRL0 -DUP IF SRL ENDIF ;
```

SSDT SCR 58 -GRAPH

(*vaddr* ---)

Places the Sprite Descriptor Table at the specified VDP address *vaddr* and initializes all sprite tables. The address given must be on an even 2K boundary. This instruction must be executed before sprites can be used.

STAT SCR 71 -FILE

(--- *b*)

Reads the status of the current PAB and returns the status byte *b* to the stack. See the table in § 8.4 following the explanation of **STAT** for the meaning of each bit of the status byte.

STATE RESIDENT

(--- *addr*)

A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

STCR SCR 88

(n_1 *addr* --- n_2)

Performs the TMS9900 STCR instruction. The field width is n_1, the CRU base address is *addr*, and the returned value is n_2. The CRU base address will be shifted left 1 bit and stored in workspace register R12 prior to executing the TMS9900 STCR instruction.

STR SCR 47 -FLOAT

(---)

Converts the number in the FAC to a string, which is placed in PAD. The string is in BASIC format. Used by **F.** and **F.R** .

STR. SCR 47 -FLOAT

(n_1 n_2 n_3 ---)

Converts the number in the FAC to a string which is placed in PAD. The maximum number of output digits is n_1 (**STR.** places n_1 in the byte at FAC+11). Calling **STR.** with $n_1 = 0$ is identical to calling **STR** . The number of significant digits of output is n_2 (**STR.** places n_2 in the byte at FAC+12). The number of digits to be output after the decimal point is n_3 (**STR.** places n_3 in the byte at FAC+13). See the GPL STR routine on page 254 in the *Editor/Assembler Manual* for more detail.

SV SCR 71 -FILE

(*count* ---)

Performs the file I/O save operation. The number of bytes *count* to be saved will be the size of the file on disk. The file's PAB must be set up and be the current PAB, to which **PAB-ADDR** points, before executing this word.

SWAP RESIDENT

(n_1 n_2 --- n_2 n_1)

Exchange the top two values on the stack.

SWCH SCR 72 -PRINT

(---)

A special purpose word which permits **EMIT** to output characters to an RS232 device rather than to the screen. See **UNSWCH** .

SWPB RESIDENT

(n_1 --- n_2)

Reverses the order of the two bytes in n_1 and leaves the new number as n_2.

SYS$ RESIDENT

(--- *addr*)

A user variable that contains the address of the system support entry point.

SYSTEM RESIDENT

(*n* ---)

Calls the system synonyms. You must specify an offset *n* into a jump table for the routine you wish to call. The offset *n* must be one of the predefined even numbers. See system Forth screen 33 for offsets 0 – 26.

TAN SCR 50 -FLOAT

(f_1 --- f_2)

Finds the tangent of the floating point number (f_1 = angle in radians) on the stack and leaves the result f_2.

TASK RESIDENT

(---)

A no-operation word or null definition, **TASK** is the last word defined in the resident Forth vocabulary of TI Forth and the last word that cannot be forgotten using **FORGET** . Its definition is simply **: TASK ;** . Its address can be used to **BSAVE** a personalized TI Forth system disk (see Chapter 11): **' TASK 21 BSAVE** (*Be sure to back up the original disk before trying this!*). By redefining **TASK** at the beginning of an application, you can mark the boundary between applications. By forgetting **TASK**

and re-compiling, an application can be discarded in its entirety. You will be able to **FORGET** each instance of the definition of **TASK** except the first one described above.

TB SCR 88 -CRU

(*addr* --- *flag*)

TB performs the TMS9900 TB instruction. The bit at CRU address *addr* is tested by this instruction. Its value (*flag* = 1|0) is returned to the stack. The CRU base address *addr* will be shifted left one bit and stored in workspace register R12 prior to executing the TMS9900 TB instruction.

TCHAR SCR 65 & 67 -64SUPPORT

(--- *addr*)

Points to the array that holds the tiny character definitions for the 64-column editor. See **CLIST** .

TEXT SCR 51 -TEXT

(---)

Converts from present display screen mode into standard Text mode configuration.

THEN RESIDENT

(---)

An alias for **ENDIF** .

TIB RESIDENT

(--- *addr*)

A user variable containing the address of the terminal input buffer.

TOGGLE RESIDENT

(*addr b* ---)

Complement (XOR) the contents of the byte at *addr* by the bit pattern of byte *b*.

TRACE SCR 44 -TRACE

(---)

Forces colon definitions that follow it to be compiled in such a way that their execution can be traced. Once a routine has been compiled with the **TRACE** option, it may be executed with or without a trace. To implement a trace, type **TRON** before execution. To execute without a trace, type **TROFF** . Colon definitions that have been compiled under the **TRACE** option must be recompiled under the **UNTRACE** option to remove the tracing capability. **TRACE** and **UNTRACE** can be used alternately to select words to be traced. See **TRON** , **TROFF** , **UNTRACE** and § 5.4 .

TRAVERSE RESIDENT

($addr_1$ n --- $addr_2$)

Traverse the name field of a fig-Forth variable-length name field. The starting point $addr_1$ is the address of either the length byte or the last letter. If n = 1, the direction is toward high memory; if n = -1, the direction is toward low memory. The resulting address $addr_2$ points to the other end of the name.

TRIAD SCR 72 -PRINT

(*scr* ---)

Display on the RS232 device the three Forth screens that include screen number *scr*, beginning with a Forth screen evenly divisible by three. Output is suitable for source text records and includes a reference line at the bottom taken from line 15 of screen 4: "TI FORTH --- a fig-FORTH extension".

TRIADS SCR 73 -PRINT

(scr_1 scr_2 ---)

May be thought of as a multiple **TRIAD** , *q.v.* You must specify a Forth screen range. **TRIADS** will execute **TRIAD** as many times as necessary to cover that range.

TROFF SCR 44 -TRACE

(---)

Turn off tracing of words compiled with the **TRACE** option. See **TRON** , **TRACE** , **UNTRACE** and § 5.4 .

TRON SCR 44 -TRACE

(---)

Turn on tracing of words compiled with the **TRACE** option. See **TROFF** , **TRACE** , **UNTRACE** and § 5.4 .

TYPE RESIDENT

(*addr count* ---)

Transmit *count* characters from *addr* to the selected output device.

U RESIDENT

(--- *n*)

Places the contents *n* of workspace register UP (R8) on the stack. Register U contains the base address of the user variable area. This is quicker than executing **U0 @** , which accomplishes the same thing.

U* RESIDENT

(u_1 u_2 --- *ud*)

Leave the unsigned double number product *ud* of two unsigned numbers, u_1 and u_2.

U. RESIDENT

(*u* ---)

Prints an unsigned number *u* to the output device.

U.R RESIDENT

(*u n* ---)

Prints an unsigned number *u* right justified in a field of width *n*.

U/ RESIDENT

(*ud* u_1 --- *rem quot*)

Leave the unsigned remainder *rem* and unsigned quotient *quot* from the unsigned double dividend *ud* and unsigned divisor u_1.

U0 RESIDENT

(--- *addr*)

A user variable that points to the base of the user variable area.

U< RESIDENT

(u_1 u_2 --- *flag*)

Leaves a true flag if u_1 is less than u_2, else leaves a false flag.

UCONS$ RESIDENT

(--- *addr*)

A user variable which contains the base address of the user variable initial value table, which is used to initialize the user variables at a **COLD** start.

UD. RESIDENT

(*ud* ---)

Prints an unsigned double number *ud* to the output device.

UD.R RESIDENT

(*ud n* ---)

Prints an unsigned double number *ud* right justified in a field of length *n*.

UNDRAW SCR 62 -GRAPH

(---)

Sets **DMODE** to 1. This means that dots are plotted in the off mode.

UNFORGETABLE [*sic*] RESIDENT

(*addr* --- *flag*)

Decides whether or not a word can be forgotten. A true flag is returned if the address is not located between **FENCE** and **HERE** . Otherwise, a false flag is left. See **FORGET** . It *is* possible to set the value of **FENCE** to a value that is actually less than the address of the end of the last word in the core dictionary (**TASK**) such that **UNFORGETABLE** [sic] will report false; however, **FORGET** will still trap that error.

UNSWCH SCR 72 -PRINT

(---)

Causes the computer to send output to the display screen instead of an RS232 device. See **SWCH** .

UNTIL RESIDENT

Compilation: (*addr n* ---) Runtime: (*flag* ---)

Occurs within a colon-definition in the form:

BEGIN … UNTIL

At compile-time, **UNTIL** compiles **(0BRANCH)** and an offset from **HERE** to *addr*. Number *n* is used for error tests.

At runtime, **UNTIL** controls the conditional branch back to the corresponding **BEGIN** . If flag is *false*, execution returns to just after **BEGIN** ; if *true*, execution continues ahead.

UNTRACE SCR 44 -TRACE

(---)

Colon definitions that have been compiled under the **TRACE** option must be recompiled under the **UNTRACE** option to remove the tracing capability. **TRACE** and **UNTRACE** can be used alternately to select words to be traced.

UPDATE RESIDENT

(---)

Marks the most recently referenced block pointed to by **PREV** as altered. The block will subsequently be transferred automatically to disk should its buffer be required for storage of a different block. See **FLUSH** .

UPDT SCR 69 -FILE

(---)

Assigns the attribute UPDATE to the file whose PAB is pointed to by **PAB-ADDR** .

USE RESIDENT

(--- *addr*)

A user variable containing the address of the block buffer to use next as the least recently written.

USER RESIDENT

(*n* ---)

A defining word used in the form:

```
n  USER  cccc
```

which creates a user variable **cccc** . The parameter field of **cccc** contains *n* as a fixed offset relative to the user variable base address pointed to by workspace register UP (R8) for this user variable. When **cccc** is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable. You should only use the even numbers **68h** – **7Eh** for *n*. You should actually avoid **68h** as well because the TI Forth boot screen (screen 3) uses that offset for defining user variable **VDPMDE** , leaving **6Ah** – **7Eh** as the available offsets—enough for 11 user variables.

Even if you use odd offsets, storage/retrieval is always on even-address boundaries one byte less. However, **USER** does not check that the definition is within the **80h** size allotted to the user variable table.

VAL SCR 47 -FLOAT

(---)

Causes the string at PAD to be converted into a floating point number and put into the FAC. The string must have a leading length byte with no embedded blanks.

VAND SCR 33 -SYNONYMS

(*b vaddr* ---)

Performs a logical AND on the byte at the specified VDP location *vaddr* and the given byte *b*. The result byte is stored back into the VDP address.

VARIABLE RESIDENT

(*n* ---)

A defining word used in the form:

```
n  VARIABLE  cccc
```

When **VARIABLE** is executed, it creates the definition **cccc** with its parameter field initialized to *n*. When **cccc** is later executed, the address of its parameter field (containing *n*) is left on the stack, so that a fetch or store may access this location.

VCHAR SCR 57 -GRAPH

(*col row count char* ---)

Prints on the display screen a vertical stream of length *count* of the specified character *char*. The first character of the stream is located at (*col*,*row*). Rows and columns are numbered from 0 beginning at the upper left of the display screen.

VFILL SCR 33 -SYNONYMS

(*vaddr count b* ---)

Fills *count* locations beginning at the given VDP address *vaddr* with the specified byte *b*.

VLIST SCR 43 -DUMP

(---)

Prints the names of all words defined in the **CONTEXT** vocabulary. See **PAUSE** .

VMBR SCR 33 -SYNONYMS

(*vaddr addr count* ---)

Reads *count* bytes beginning at the given VDP address *vaddr* and places them at *addr*.

VMBW SCR 33 -SYNONYMS

(*addr vaddr count* ---)

Writes *count* bytes from *addr* into VDP beginning at the given VDP address *vaddr*.

VOC-LINK RESIDENT

(--- *addr*)

A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for forgetting with **FORGET** through multiple vocabularies.

VOCABULARY RESIDENT

(---)

A defining word used in the form:

```
VOCABULARY  cccc
```

to create a vocabulary definition **cccc** . Subsequent use of **cccc** will make it the **CONTEXT** vocabulary which is searched first by **INTERPRET** . The sequence **cccc DEFINITIONS** will also make **cccc** the **CURRENT** vocabulary into which new definitions are placed.

cccc will be so chained as to include all definitions of the vocabulary in which **cccc** is itself defined. All vocabularies ultimately chain to Forth. By convention, vocabulary names are to be declared **IMMEDIATE** . See **VOC-LINK** .

VOR SCR 33 -SYNONYMS

(*b vaddr* ---)

Performs a logical OR on the byte at the specified VDP address and the given byte *b*. The result byte is stored back into the VDP address.

VRBL SCR 68 -FILE

(---)

Assigns the attribute VARIABLE to the file whose PAB is pointed to by **PAB-ADDR** .

VSBR SCR 33 -SYNONYMS

(*vaddr* --- *b*)

Reads a single byte from the given VDP address *vaddr* and places its value *b* on the stack.

VSBW SCR 33 -SYNONYMS

(*b vaddr* ---)

Writes a single byte *b* into the given VDP address *vaddr*.

VWTR SCR 33 -SYNONYMS

(*b n* ---)

Writes the given byte *b* into the specified VDP write-only register *n*.

VXOR SCR 33 -SYNONYMS

(*b vaddr* ---)

Performs a logical XOR on the byte at the specified VDP address *vaddr* and the given byte *b*. The result byte is stored back into the VDP address *vaddr*.

WARNING RESIDENT

(--- *addr*)

A user variable initialized by **COLD** at system startup containing a value controlling messages. If **WARNING** > 0, a disk is present and Forth screen 4 of drive 0 is the base location for messages. If **WARNING** = 0, no disk is present and messages will be presented by number. If **WARNING** < 0 when **ERROR** executes, **ERROR** will execute **(ABORT)** , which can be redefined to execute a user-specified procedure instead of the default **ABORT** . See **MESSAGE** , **ERROR** .

WDISK RESIDENT

(*addr* n_1 n_2 --- n_3)

The primitive routine which performs a disk write. The CPU RAM location of the block to be written is *addr*. The block number is n_1, the number of bytes per block is n_2 and the returned error code is n_3.

WHERE (*EDITOR1 Vocabulary*) SCR 38 -EDITOR

(n_1 n_2 ---)

When an error occurs on a **LOAD** instruction, typing **WHERE** will bring you into the 40-column editor and place the cursor at the exact location of the error. **WHERE** consumes the two numbers, n_1 and n_2, left on the stack by the **LOAD** error.

WHERE (*EDITOR2 Vocabulary*) SCR 29 -64SUPPORT

(n_1 n_2 ---)

When an error occurs on a **LOAD** instruction, typing **WHERE** will bring you into the 64-column editor and place the cursor at the exact location of the error. **WHERE** consumes the two numbers, n_1 and n_2, left on the stack by the **LOAD** error.

WHILE RESIDENT

Compilation: ($addr_1$ n_1 --- $addr_1$ n_1 $addr_2$ n_2) Runtime: (*flag* ---)

Occurs in a colon-definition in the form:

BEGIN … WHILE (true part) **… REPEAT**

At compile time, **WHILE** emplaces **(0BRANCH)** and leaves $addr_2$ of the reserved offset. The stack values will be resolved by **REPEAT** .

At runtime, **WHILE** selects conditional execution based on *flag*. If *flag* is *true* (non-zero), **WHILE** continues execution of the true part through to **REPEAT** , which then branches back to **BEGIN** . If *flag* is *false* (zero), execution skips to just after **REPEAT** , exiting the structure.

WIDTH RESIDENT

(--- *addr*)

A user variable containing the maximum number of letters saved in the compilation of a definition's name. It must be 1 – 31, with a default value of 31. The name character count and its natural characters are saved up to the value in **WIDTH** . The value may be changed at any time within the above limits.

WLITERAL SCR 20 BOOT SCR

(---)

A compiling word which compiles **SLIT** and the string which follows **WLITERAL** into the dictionary.

Used in the form: **WLITERAL cccc**

WORD RESIDENT

(*char* ---)

Read the text characters from the input stream being interpreted until a delimiter *char* is found, storing the packed character string beginning at the dictionary buffer **HERE** .

WORD leaves the character count in the first byte followed by the input characters and ends with two or more blanks. Leading occurrences of *char* are ignored. If **BLK** is zero, text is taken from the terminal input buffer, otherwise from the disk block stored in **BLK** . See **BLK** , **IN** .

WRT SCR 71 -FILE

(*count* ---)

Performs the file I/O write operation. You must specify the number of bytes *count* to be written.

XMLLNK SCR 33 -SYNONYMS

(*addr* ---)

Links a Forth program to a routine in ROM or to a routine located in the memory expansion unit. A ROM address *addr* or XML vector must be specified as in the *Editor/Assembler Manual*.

XOR RESIDENT

(n_1 n_2 --- n_3)

Leave n_3, the bitwise logical exclusive OR (XOR) of n_1 and n_2.

[RESIDENT

(---)

Used in a colon-definition in the form:

```
: xxxx [ words ] more ;
```

Suspend compilation. The words after **[** are executed, not compiled. This allows calculation or compilation exceptions before resuming compilation with **]** . See **LITERAL** and **]** .

[COMPILE] RESIDENT

(---)

Used in a colon definition in the form:

```
: xxxx [COMPILE] FORTH ;
```

[COMPILE] will force the compilation of an immediate definition that would otherwise execute during compilation. The above example will select the Forth vocabulary when **xxxx** executes rather than at compile time.

] RESIDENT

(---)

Resume compilation to the completion of a colon definition. See **[** .

^ SCR 50 -FLOAT

(f_1 f_2 --- f_3)

Returns f_3 on the stack f_1 raised to the f_2 power. The operands must be floating point numbers.

message SCR 84

(---)

A replacement for **MESSAGE** that contains the error messages in memory instead of on the disk. When Forth screen #84 is loaded, the error messages are compiled into the space formerly occupied by the fifth disk buffer leaving only four working disk buffers. **MESSAGE** is patched so that it now points to message.

Appendix E User Variables in TI Forth

The purpose of this appendix is to detail the User Variables in TI Forth to assist in their use and to provide the necessary information to change or add to this list as necessary. A more comprehensive description of each of these variables is provided in Appendix D. The table follows these comments in two layouts. The first is in address offset order and the second is in alphabetical order by variable name.

The user may use even numbers **6Ah** through **7Eh** to create his/her own user variables. See the definition of **USER** in Appendix D.

E.1 TI Forth User Variables (Address Offset Order)

Name	Offset	Initial Value	Description
UCONS$	6h	3944h	Base of User Var initial value table
S0	8h	FFA0h	Base of Stack
R0	Ah	3FFEh	Base of Return Stack
U0	Ch	3980h	Base of User Variables
TIB	Eh	FFA0h	Terminal Input Buffer address
WIDTH	10h	31	Name length in dictionary
DP	12h	BC80h	Dictionary Pointer
SYS$	14h	348Eh	Address of System Support
CURPOS	16h	0	Cursor location in VDP RAM
INTLNK	18h	3424h	Pointer to Interrupt Service Linkage
WARNING	1Ah	1	Message Control
C/L$	1Ch	64	Characters per Line
FIRST$	1Eh	2010h	Beginning of Disk Buffers
LIMIT$	20h	3424h	End of Disk Buffers
B/BUF$	22h	1024	Bytes per Buffer
B/SCR$	24h	1	Blocks per Forth Screen
DISK_LO	26h	1	Low end Disk Fence
DISK_HI	28h	90	High end Disk Fence
DISK_SIZE	2Ah	90	Logical Disk Size in Forth Screens
DISK_BUF	2Ch	1000h	VDP location of 1K Forth Buffer
PABS	2Eh	460h	VDP location for PABs
SCRN_WIDTH	30h	40	Display Screen Width in Characters
SCRN_START	32h	0	Display Screen Image Start in VDP
SCRN_END	34h	960	Display Screen Image End in VDP
ISR	36h	3424h	Interrupt Service Pointer
ALTIN	38h	0	Alternate Input Pointer
ALTOUT	3Ah	0	Alternate Output Pointer
FENCE	3Ch		Dictionary Fence
BLK	3Eh		Block being interpreted
IN	40h		Byte offset in text buffer
OUT	42h		Incremented by **EMIT**
SCR	44h		Last Forth Screen referenced
OFFSET	46h		Block offset to disks

Name	Offset	Initial Value	Description
CONTEXT	48h		Pointer to Context Vocabulary
CURRENT	4Ah		Pointer to Current Vocabulary
STATE	4Ch		Compilation State
BASE	4Eh		Number Base for Conversions
DPL	50h		Decimal Point Location
FLD	52h		Field Width (unused)
CSP	54h		Stack Pointer for error checking
R#	56h		Editing Cursor location
HLD	58h		Holds address during numeric conversion
USE	5Ah		Next Block Buffer to Use
PREV	5Ch		Most recently accessed disk buffer
[*unavailable*]	5Eh		*—Do Not Use—*
[*unavailable*]	60h		*—Do Not Use—*
FORTH_LINK	62h		Forth Vocabulary base
ECOUNT	64h		Error control
VOC-LINK	66h		Vocabulary linkage
VDPMDE	68h	0	VDP Mode (defined in TI Forth Screen #3)
[*user to define*]	6Ah		*—available to user—*
[*user to define*]	6Ch		*—available to user—*
[*user to define*]	6Eh		*—available to user—*
[*user to define*]	70h		*—available to user—*
[*user to define*]	72h		*—available to user—*
[*user to define*]	74h		*—available to user—*
[*user to define*]	76h		*—available to user—*
[*user to define*]	78h		*—available to user—*
[*user to define*]	7Ah		*—available to user—*
[*user to define*]	7Ch		*—available to user—*
[*user to define*]	7Eh		*—available to user—*

E.2 TI Forth User Variables (Variable Name Order)

Name	Offset	Initial Value	Description
ALTIN	38h	0	Alternate Input Pointer
ALTOUT	3Ah	0	Alternate Output Pointer
B/BUF$	22h	1024	Bytes per Buffer
B/SCR$	24h	1	Blocks per Forth Screen
BASE	4Eh		Number Base for Conversions
BLK	3Eh		Block being interpreted
C/L$	1Ch	64	Characters per Line
CONTEXT	48h		Pointer to Context Vocabulary
CSP	54h		Stack Pointer for error checking
CURPOS	16h	0	Cursor location in VDP RAM
CURRENT	4Ah		Pointer to Current Vocabulary
DISK_BUF	2Ch	1000h	VDP location of 1K Forth Buffer
DISK_HI	28h	90	High end Disk Fence
DISK_LO	26h	1	Low end Disk Fence
DISK_SIZE	2Ah	90	Logical Disk Size in Forth Screens
DP	12h	BC80h	Dictionary Pointer
DPL	50h		Decimal Point Location
ECOUNT	64h		Error control
FENCE	3Ch		Dictionary Fence
FIRST$	1Eh	2010h	Beginning of Disk Buffers
FLD	52h		Field Width (unused)
FORTH_LINK	62h		Forth Vocabulary base
HLD	58h		Holds address during numeric conversion
IN	40h		Byte offset in text buffer
INTLNK	18h	3424h	Pointer to Interrupt Service Linkage
ISR	36h	3424h	Interrupt Service Pointer
LIMIT$	20h	3424h	End of Disk Buffers
OFFSET	46h		Block offset to disks
OUT	42h		Incremented by **EMIT**
PABS	2Eh	460h	VDP location for PABs
PREV	5Ch		Most recently accessed disk buffer
R#	56h		Editing Cursor location
R0	Ah	3FFEh	Base of Return Stack
S0	8h	FFA0h	Base of Stack
SCR	44h		Last Forth Screen referenced
SCRN_END	34h	960	Display Screen Image End in VDP
SCRN_START	32h	0	Display Screen Image Start in VDP
SCRN_WIDTH	30h	40	Display Screen Width in Characters
STATE	4Ch		Compilation State
SYS$	14h	348Eh	Address of System Support
TIB	Eh	FFA0h	Terminal Input Buffer address
U0	Ch	3980h	Base of User Variables
UCONS$	6h	3944h	Base of User Var initial value table
USE	5Ah		Next Block Buffer to Use

Name	Offset	Initial Value	Description
VDPMDE	68h	0	VDP Mode (defined in TI Forth Screen #3)
VOC-LINK	66h		Vocabulary linkage
WARNING	1Ah	1	Message Control
WIDTH	10h	31	Name length in dictionary
[*unavailable*]	5Eh		*—Do Not Use—*
[*unavailable*]	60h		*—Do Not Use—*
[*user to define*]	6Ah		*—available to user—*
[*user to define*]	6Ch		*—available to user—*
[*user to define*]	6Eh		*—available to user—*
[*user to define*]	70h		*—available to user—*
[*user to define*]	72h		*—available to user—*
[*user to define*]	74h		*—available to user—*
[*user to define*]	76h		*—available to user—*
[*user to define*]	78h		*—available to user—*
[*user to define*]	7Ah		*—available to user—*
[*user to define*]	7Ch		*—available to user—*
[*user to define*]	7Eh		*—available to user—*

Appendix F TI Forth Load Option Directory

The load options are displayed on the TI Forth welcome screen and may subsequently be displayed by typing **MENU** . The load options allow you to load only the Forth extensions you wish to use.

You will notice, for example, that the **-EDITOR** option also loads the Forth screens that **-SYNONYMS** loads. The words loaded by **-SYNONYMS** are prerequisites for the words loaded by **-EDITOR** . If, by chance, the **-SYNONYMS** words were already in the dictionary at the time you type **-EDITOR** , they would not be loaded again. This is called a conditional load.

F.1 Option: -SYNONYMS

Starting screen: 33

Prerequisite options loaded: **-CODE**

Words loaded:

VSBW	**VMBW**	**VSBR**
VMBR	**VWTR**	**GPLLNK**
XMLLNK	**DSRLNK**	**CLS**
FORMAT-DISK	**VFILL**	**VAND**
VOR	**VXOR**	**MON**
RNDW	**RND**	**SEED**
RANDOMIZE		

F.2 Option: -EDITOR (40-Column Editor)

Starting screen: 34

Prerequisite options loaded: **-SYNONYMS**

Words loaded:

EDIT	**ED@**	**WHERE**

F.3 Option: -COPY

Starting screen: 39

Words loaded:

!"	**DTEST**	**SCOPY**
SMOVE	**FORTH-COPY**	**DISK-HEAD**

F.4 Option: -DUMP

Starting screen: 42

Words loaded:

DUMP	**.S**	**VLIST**

F.5 Option: -TRACE

Starting screen: 44

Prerequisite options loaded: **-DUMP**

Words loaded: **TRACE** **UNTRACE** **TRON**
TROFF **:** *(alternate)*

F.6 Option: -FLOAT

Starting screen: 45

Prerequisite options loaded: **-SYNONYMS**

Words loaded:

FDUP	**FDROP**	**FOVER**
FSWAP	**F!**	**F@**
>FAC	**SETFL**	**FADD**
FMUL	**F+**	**F-**
F*	**F/**	**S->FAC**
FAC->S	**FAC>ARG**	**F->S**
S->F	**FRND**	**STR**
STR.	**VAL**	**F$**
>F	**F.R**	**F.**
FF.R.	**FF.**	**F0<**
F0=	**F>**	**F=**
F<	**FLERR**	**?FLERR**
INT	**^**	**SQR**
EXP	**LOG**	**COS**
SIN	**TAN**	**ATN**
PI		

F.7 Option: -TEXT

Starting screen: 51

Prerequisite options loaded: **-SYNONYMS**

Words loaded: **TEXT**

F.8 Option: -GRAPH1

Starting screen: 52

Prerequisite options loaded: **-SYNONYMS**

Words loaded: **GRAPHICS**

F.9 Option: -MULTI

Starting screen: 53

Prerequisite options loaded: **-SYNONYMS**

Words loaded: **MULTI**

F.10 Option: -GRAPH2

Starting screen: 54

Prerequisite options loaded: **-SYNONYMS**

Words loaded: **GRAPHICS2**

F.11 Option: -SPLIT

Starting screen: 55

Prerequisite options loaded: **-SYNONYMS** **-GRAPH2**

Words loaded: **SPLIT** **SPLIT2**

F.12 Option: -VDPMODES

Starting screen: 51

Prerequisite options loaded: **-SYNONYMS** **-TEXT** **-GRAPH1**
-MULTI **-GRAPH2** **-SPLIT**

F.13 Option: -GRAPH

Starting screen: 57

Prerequisite options loaded: **-SYNONYMS** **-CODE**

Words loaded:

CHAR	**CHARPAT**	**VCHAR**
HCHAR	**COLOR**	**SCREEN**
GCHAR	**SSDT**	**SPCHAR**
SPRCOL	**SPRPAT**	**SPRPUT**
SPRITE	**MOTION**	**#MOTION**
SPRGET	**DXY**	**SPRDIST**
SPRDISTXY	**MAGNIFY**	**JOYST**
COINC	**COINCXY**	**COINCALL**
DELSPR	**DELALL**	**MINIT**
MCHAR	**DRAW**	**UNDRAW**
DTOG	**DOT**	**LINE**

F.14 Option: -FILE

Starting screen: 68

Prerequisite options loaded: **-SYNONYMS**

Words loaded:

FILE	**GET-FLAG**	**PUT-FLAG**
SET-PAB	**CLR-STAT**	**CHK-STAT**
FXD	**VRBL**	**DSPLY**
INTRNL	**I/OMD**	**INPT**
OUTPT	**UPDT**	**APPND**
SQNTL	**RLTV**	**REC-LEN**
CHAR-CNT!	**CHAR-CNT@**	**REC-NO**
N-LEN!	**F-D"**	**DOI/O**
OPN	**CLSE**	**RD**
WRT	**RSTR**	**LD**
SV	**DLT**	**SCRTCH**[18]
STAT		

F.15 Option: -PRINT

Starting screen: 72

Prerequisite options loaded: **-SYNONYMS** **-FILE**

Words loaded:

SWCH	**UNSWCH**	**?ASCII**
TRIAD	**TRIADS**	**INDEX**

F.16 Option: -CODE

Starting screen: 74

Words loaded: **CODE** **;CODE**

F.17 Option: -ASSEMBLER

Starting screen: 75

Prerequisite options loaded: **-CODE**

Words loaded: Entire Assembler vocabulary. See Chapter 9.

18 See footnote 12, page 52.

F.18 Option: -64SUPPORT (64-Column Editor)

Starting screen: 22

Prerequisite options loaded: **-SYNONYMS** **-GRAPH** **-TEXT**
-GRAPH2 **-SPLIT**

Words loaded: **EDIT** **ED@** **WHERE**
CLIST **CLINE**

F.19 Option: -BSAVE

Starting screen: 83

Words loaded: **BSAVE**

F.20 Option: -CRU

Starting screen: 88

Prerequisite options loaded: **-CODE**

Words loaded: **SBO** **SBZ** **TB**
LDCR **STCR**

Appendix G Assembly Source for CODEd Words

Several words on the Forth System Disk have been written in TMS9900 code to increase their execution speeds and/or decrease their size. They include the words:

SBO — a CRU instruction
SBZ — a CRU instruction
TB — a CRU instruction
LDCR — a CRU instruction
STCR — a CRU instruction
DDOT — used by the dot plotting routine
SMASH — used by **CLINE** and **CLIST**
TCHAR — definitions for the tiny characters
MON — returns to 99/4A color bar screen

These words have been coded in hexadecimal on your System disk, thus they do not require that the TI Forth Assembler be in memory before they can be loaded. Their Assembly source code (written in Forth Assembler) is listed on the following pages.

Editor's Notes: I detected a few errors and items in need of clarification in the TI Forth Assembler source code listed in this section. The errors are corrected in red text on the TI Forth screens in this section. The corrected lines are also highlighted in gray. The errors are as follows:

1. Screen 40, line 5: In the code for **SBZ**, the first ***SP** should be ***SP+**. The TMS9900-coded word on screen 88 of the TI Forth system diskette is correct.
2. Screen 43: There are several errors on this screen:
   a. **DTAB** is supposed to be an initialized table of 12 cells (24 bytes), not just the one cell defined on this screen in the original (see screen 62 of the TI Forth system diskette to verify)—though, to be fair, it may have been done that way to justify the presence of **DTAB** in the assembly code.
   b. **DDOT** is missing **1 *SP MOV,** and **NEXT,** from the end of the definition of **DDOT**, which can be verified by examining the code compiled into the dictionary from the source code here with screen 63 of the TI Forth system diskette.
3. Screen 44: This screen is missing the definitions of two variables (tables), *viz.*, **TCHAR** and **LB**.

4. Screen 45 clarifications:

   a. It should be noted that the definition of **TCHAR** in screen 45 is not actually Assembly source code. It is high-level Forth source code. If you wanted to change the character definitions and copy your new table to screen 67 of the system disk, you would need to first load the new character definitions. Let's say you have screens 45 – 47 on a non-system disk set up with your new character definitions for **TCHAR** . For a system with two 90KB-disks and the foregoing disk in the second drive, this would require loading screen 135, obtained by adding the number (90) of screens on the system disk to the beginning screen number (45) for the definition of **TCHAR** . The following code will do the trick:

```
135 LOAD                          <== Load TCHAR
TCHAR 67 BLOCK 194 MOVE FLUSH     <== Copy TCHAR to screen 67
FORGET TCHAR                      <== Recover space in dictionary used by
                                      TCHAR
```

   b. The comment, **(^0)** (Shift+0), on line 5 is a substitute for **( ))** , a syntax error.

```
SCR #40
  0 ( SOURCE FOR CRU WORDS )   BASE->R  HEX
  1 CODE SBO
  2      *SP+ 0C MOV,  0C 0C A,
  3           0 SBO,  NEXT,
  4 CODE SBZ
  5      *SP+  0C MOV,  0C 0C A,
  6           0 SBZ,  NEXT,
  7 CODE TB
  8      *SP  0C MOV,  0C 0C  A,
  9          *SP CLR,      0 TB,
 10      EQ IF,
 11          *SP INC,
 12         ENDIF,
 13      NEXT,
 14
 15 R->BASE -->

SCR #41
  0 ( SOURCE FOR CRU WORDS ) BASE->R HEX
  1 0C CONSTANT CRU
  2 CODE LDCR
  3      *SP+ CRU MOV,  CRU CRU A, *SP+ 1 MOV,
  4      *SP+   0 MOV,   01  OF ANDI,
  5      NE  IF,
  6           01 08 CI,
  7           LTE IF,
  8                0 SWPB,
  9               ENDIF,
 10          ENDIF,
 11      01 06 SLA,      01 3000 ORI,   01 X,
 12 NEXT,
 13
 14
 15 R->BASE -->

SCR #42
  0 (SOURCE FOR CRU WORDS )  BASE->R  HEX
  1 CODE STCR
  2      *SP+ CRU MOV,  CRU CRU A,  *SP 01 MOV,
  3      0 CLR,  01 000F ANDI,  01 02 MOV,
  4      01 06 SLA,  01 3400 ORI,  01 X,
  5      02 02 MOV,
  6      NE IF,
  7           02 08 CI,
  8           LTE IF,
  9                0 SWPB,
 10              ENDIF,
 11        ENDIF,
 12      0 *SP MOV,
 13 NEXT,
 14
 15 R->BASE
```

```
SCR #43
  0 ( SOURCE FOR DDOT )
  1 BASE->R HEX  8040 VARIABLE DTAB 2010 , 804 , 201 , 7FBF , DFEF ,
  2         F7FB , FDFE , 8040 , 2010 , 804 , 201 ,
  3 CODE DDOT
  4          *SP+ 1 MOV,   *SP 3 MOV,     1 2 MOV,
  5          3 4 MOV,      1 7 ANDI,      3 7 ANDI,
  6          2 F8 ANDI,    4 F8 ANDI,     2 5 SLA,
  7          2 1 A,        4 1 A,         1 2000 AI,
  8          4 CLR,        DTAB 3  @(?) 4 MOVB,
  9          4 SWPB,       4 *SP MOV,     SP DECT,
 10          1 *SP MOV,
 11    NEXT,
 12
 13
 14
 15 R->BASE

SCR #44
  0 ( SOURCE FOR SMASH ) BASE->R HEX
  1 0 VARIABLE TCHAR 17E ALLOT 43 BLOCK TCHAR 180 CMOVE
  2 TCHAR 7C - CONSTANT TC 0 VARIABLE LB FE ALLOT
  3 CODE SMASH ( ADDR #CHAR LINE# --- LB VADDR CNT )
  4   *SP+ 1 MOV,  *SP+ 2 MOV,  *SP 3 MOV,  4 LB   LI,
  5   4 *SP  MOV,  SP    DECT,  1    SWPB,  1 2000 AI,
  6   1 *SP  MOV,  2 1   MOV, 1 INC, 1 FFFE ANDI,  SP   DECT,
  7   1 2 SLA,  1 *SP  MOV,
  8    3 2    A,  BEGIN,  2 3 C,  GT  WHILE,  5 CLR,  6 CLR,
  9              3 *?+ 5 MOVB,  3 *?+ 6 MOVB,   5 6 SRL,  6 6 SRL,
 10   BEGIN,  TC 5 @(?) 0 MOV,  TC 6 @(?) 1 MOV,  1 4 SRC,
 11   C 4 LI, BEGIN, 0 B MOV, B F000 ANDI,  1 7 MOV,  7 F00 ANDI,
 12   B 7 SOC,  7 4 *?+ MOVB,  0 C SRC,  1 C SRC, C DEC, EQ UNTIL,
 13   5 INCT,  6 INCT,  5 C MOV,  C 2 ANDI,  EQ UNTIL,  REPEAT,
 14   NEXT,
 15 R->BASE
```

```
SCR #45
  0 ( DEFINITIONS FOR TINY CHARACTERS ) BASE->R HEX
  1 0EEE VARIABLE TCHAR EEEE ,
  2 0000 , 0000 , (  )  0444 , 4404 , ( !)  0AA0 , 0000 , ( ")
  3 08AE , AEA2 , ( #)  04EC , 46E4 , ( $)  0A24 , 448A , ( %)
  4 06AC , 4A86 , ( &)  0480 , 0000 , ( ')  0248 , 8842 , ( ()
  5 0842 , 2248 , ( ^0) 04EE , 0400 , ( *)  0044 , E440 , ( +)
  6 0000 , 0048 , ( ,)  0000 , E000 , ( -)  0000 , 0004 , ( .)
  7 0224 , 4488 , ( /)  04AA , AAA4 , ( 0)  04C4 , 4444 , ( 1)
  8 04A2 , 488E , ( 2)  0C22 , C22C , ( 3)  02AA , AE22 , ( 4)
  9 0E8C , 222C , ( 5)  0688 , CAA4 , ( 6)  0E22 , 4488 , ( 7)
 10 04AA , 4AA4 , ( 8)  04AA , 622C , ( 9)  0004 , 0040 , ( :)
 11 0004 , 0048 , ( ;)  0024 , 8420 , ( <)  000E , 0E00 , ( =)
 12 0084 , 2480 , ( >)  04A2 , 4404 , ( ?)  04AE , AEA4 , ( @)
 13 04AA , EAAA , ( A)  0CAA , CAAC , ( B)  0688 , 8886 , ( C)
 14 0CAA , AAAC , ( D)  0E88 , C88E , ( E)  0E88 , C888 , ( F)
 15 -->

SCR #46
  0 ( TINY CHARACTERS CONTINUED )
  1 04A8 , 8AA6 , ( G)  0AAA , EAAA , ( H)  0E44 , 444E , ( I)
  2 0222 , 22A4 , ( J)  0AAC , CAAA , ( K)  0888 , 888E , ( E)
  3 0AEE , AAAA , ( M)  0AAE , EEAA , ( N)  0EAA , AAAE , ( 0)
  4 0CAA , C888 , ( P)  0EAA , AAEC , ( Q)  0CAA , CAAA , ( R)
  5 0688 , 422C , ( S)  0E44 , 4444 , ( T)  0AAA , AAAE , ( U)
  6 0AAA , AA44 , ( V)  0AAA , AEEA , ( W)  0AA4 , 44AA , ( X)
  7 0AAA , E444 , ( Y)  0E24 , 488E , ( Z)  0644 , 4446 , ( [)
  8 0884 , 4422 , ( \)  0C44 , 444C , ( ])  044A , A000 , ( $)
  9 0000 , 000F , ( _)  0420 , 0000 , ( `)  0004 , AEAA , ( a)
 10 000C , ACAC , ( b)  0006 , 8886 , ( c)  000C , AAAC , ( d)
 11 000E , 8C8E , ( e)  000E , 8C88 , ( f)  0004 , A8A6 , ( g)
 12 000A , AEAA , ( h)  000E , 444E , ( i)  0002 , 22A4 , ( j)
 13 000A , CCAA , ( k)  0008 , 888E , ( l)  000A , EEAA , ( m)
 14 000A , EEEA , ( n)  000E , AAAE , ( o)  000C , AC88 , ( p)
 15 -->

SCR #47
  0 ( TINY CHARACTERS CONCLUDED )
  1 000E , AAEC , ( q)  000C , ACAA , ( r)  0006 , 842C , ( s)
  2 000E , 4444 , ( t)  000A , AAAE , ( u)  OOOA , AA44 , ( v)
  3 000A , AEEA , ( w)  000A , A4AA , ( x)  000A , AE44 , ( y)
  4 000E , 248E , ( z)  0644 , 8446 , ( {)  0444 , 0444 , ( |)
  5 0C44 , 244C , ( })  02E8 , 0000 , ( ~)  0EEE , EEEE , ( DEL)
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15 R->BASE
```

```
SCR #48
  0 ( SOURCE FOR MON )  BASE->R HEX
  1
  2 CODE MON
  3      0 4E4F LI,  1 2000 LI,
  4      BEGIN,
  5           0 1 *?+ MOV,
  6           1 4000 CI,
  7      EQ UNTIL,
  8      0 @() BLWP,
  9
 10
 11
 12
 13
 14
 15 R->BASE
```

Appendix H Error Messages

Error#	Message	Probable Causes
1	empty stack	Procedure being executed attempts to pop a number off the parameter stack when there is no number on the parameter stack. The error may have occurred long before it is detected as Forth checks for this condition only when control returns to the outer interpreter.
2	dictionary full	The user dictionary space is full. Too many definitions have been compiled.
3	has incorrect address mode	Not used by TI Forth. Some fig-Forth assemblers use this message.
4	isn't unique	This message is more a warning than an error. It informs the user that a word with the same name as the one just compiled is already in the **CURRENT** or **CONTEXT** vocabulary.
6	disk error	This has several possible causes: No disk in disk drive, disk not initialized, disk drive or controller not connected properly, disk drive or controller not plugged in. The diskette may be damaged with some sector having a hard error.
7	full stack	The procedure being executed is leaving extra unwanted numbers on the parameter stack resulting in a stack overflow.
9	file I/O error	Any file I/O operation which results in an error will return this message. The **GET-FLAG** instruction will fetch the status byte. An error code of 0 indicates no error only if the COND bit (bit 2) of the STATUS byte located at **837Ch** is ***not*** set.

code	meaning
00	Bad device name
01	Device is write protected
02	Bad open attribute
03	Illegal operation
04	Out of table or buffer space on the device
05	Attempt to read past EOF
06	Device error
07	File error. Attempt to open nonexistent file, *etc*.

Error#	Message	Probable Causes
10	floating point error	This error message will be issued only when **?FLERR** is executed and a true flag is returned. **FLERR** may be executed to fetch the floating point status byte. **code** **meaning** 01 Overflow 02 Syntax 03 Integer overflow on conversion 04 Square root of negative 05 Negative number to non-integer power 06 Logarithm of a non-positive number 07 Invalid argument in a trigonometric function
11	disk fence violation	An attempt has been made to write to a screen outside the disk fence area. The values of **DISK_LO** and **DISK_HI** must be changed to include this screen before it may be written to.
12	can't load from screen 0	Self explanatory. Loading from screen 0 is Forth's indication for loading from the keyboard.
17	compilation only, use in definition	Occurs when conditional constructs such as **DO … LOOP** or **IF … THEN** are executed outside a colon definition.
18	execution only	Occurs when you attempt to compile a compiling word into a colon definition.
19	conditionals not paired	A **DO** has been left with a **LOOP** , an IF has no corresponding **THEN** , etc.
20	definition not finished	A **;** was encountered and the parameter stack was not at the same height as when the preceding **:** was encountered. For example, an incomplete conditional construct: **: xx IF ;**
23	off current editing screen	Not used in TI Forth.
24	declare vocabulary	Not used in TI Forth due to the way TI Forth's **FORGET** is configured.
25	bad jump token	Improper use of jump tokens or conditionals in the TI Forth Assembler.

Appendix I Contents of the TI Forth Diskette

The Forth screens that follow have been modified from the original to fix known bugs as documented in Appendix J. The changed lines are highlighted in gray and the actual changes are marked by red text.

```
SCR #2
  0                       T I   F O R T H
  1
  2               THIS VERSION OF THE FORTH LANGUAGE
  3                 IS BASED ON THE fig-FORTH MODEL
  4
  5           THE ADDRESS OF THE FORTH INTEREST GROUP IS:
  6
  7                       FORTH INTEREST GROUP
  8                       P.O. BOX 1105
  9                       SAN CARLOS, CA 94070
 10
 11          TEXAS INSTRUMENTS PERSONNEL WITH SIGNIFICANT
 12                 INPUT TO THIS VERSION INCLUDE:
 13                           LEON TIETZ
 14                         LESLIE O'HAGAN
 15                        EDWARD E. FERGUSON
```

```
SCR #3
  0 ( WELCOME SCREEN )  0 0 GOTOXY  ." BOOTING..." CR
  1 BASE->R HEX 10 83C2 C! ( QUIT OFF! )
  2 DECIMAL (  84 LOAD ) 20 LOAD  16 SYSTEM   MENU
  3 HEX 68 USER VDPMDE  1 VDPMDE ! DECIMAL
  4 : -SYNONYMS  33 LOAD ;  : -EDITOR 34 LOAD ;  : -COPY  39 LOAD ;
  5 : -DUMP      42 LOAD ;  : -TRACE  44 LOAD ;  : -FLOAT 45 LOAD ;
  6 : -TEXT      51 LOAD ;  : -GRAPH1 52 LOAD ;  : -MULTI 53 LOAD ;
  7 : -GRAPH2    54 LOAD ;  : -SPLIT  55 LOAD ;  : -GRAPH 57 LOAD ;
  8 : -FILE      68 LOAD ;  : -PRINT  72 LOAD ;  : -CODE  74 LOAD ;
  9 : -ASSEMBLER 75 LOAD ;  : -64SUPPORT 22 LOAD ;
 10 : -VDPMODES  -TEXT -GRAPH1 -MULTI -GRAPH2 -SPLIT ;
 11 : -BSAVE     83 LOAD ;  : -CRU    88 LOAD ;
 12
 13
 14
 15 R->BASE

SCR #4
  0 ( ERROR MESSAGES )
  1 empty stack
  2 dictionary full
  3 has incorrect address mode
  4 isn't unique.
  5
  6 disk error
  7 full stack
  8
  9 file i/o error
 10 floating point error
 11 disk fence violation
 12 can't load from screen zero
 13
 14
 15 TI FORTH --- a fig-FORTH extension

SCR #5
  0 ( ERROR MESSAGES )
  1 compilation only, use in definition
  2 execution only
  3 conditionals not paired
  4 definition not finished
  5 in protected dictionary
  6 use only when loading
  7 off current editing screen
  8 declare vocabulary
  9 bad jump token
 10
 11
 12
 13
 14
 15
```

```
SCR #20
  0 ( CONDITIONAL LOAD  )
  1 : MENU CR 272 265 DO I MESSAGE CR LOOP CR CR CR ;
  2 : SLIT ( --- ADDR OF STRING LITERAL )
  3     R> DUP C@ 1+ =CELLS OVER + >R ;
  4
  5 : WLITERAL ( WLITERAL word )
  6     BL STATE @
  7     IF COMPILE SLIT WORD HERE C@ 1+ =CELLS ALLOT
  8     ELSE WORD HERE ENDIF ; IMMEDIATE  -->
  9 -SYNONYMS   -EDITOR     -COPY
 10 -DUMP       -TRACE      -FLOAT
 11 -TEXT       -GRAPH1     -MULTI
 12 -GRAPH2     -SPLIT      -VDPMODES
 13 -GRAPH      -FILE       -PRINT
 14 -CODE       -ASSEMBLER  -64SUPPORT
 15 -BSAVE      -CRU
```

```
SCR #21
  0 ( CONDITIONAL LOAD )
  1 : <CLOAD>  ( SCREEN  STRING_ADDR --- )
  2     CONTEXT @ @ (FIND)
  3     IF DROP DROP 0=
  4        IF BLK @
  5           IF R> DROP R> DROP
  6           ENDIF
  7        ENDIF
  8     ELSE -DUP
  9        IF LOAD
 10        ENDIF
 11     ENDIF ;
 12 : CLOAD  ( scr_no CLOAD name )
 13     [COMPILE] WLITERAL STATE @
 14     IF COMPILE <CLOAD> ELSE <CLOAD> ENDIF
 15 ; IMMEDIATE

SCR #22
  0 ( 64 COLUMN EDITOR ) 0 CLOAD ED@
  1 BASE->R DECIMAL 57 R->BASE CLOAD LINE BASE->R DECIMAL 51 R->BASE
  2  CLOAD TEXT BASE->R DECIMAL 54 R->BASE CLOAD GRAPHICS2 BASE->R
  3 DECIMAL 55 R->BASE CLOAD SPLIT
  4 BASE->R DECIMAL 65 R->BASE CLOAD CLIST
  5 BASE->R HEX  ( 3800 ' SATR ! )
  6 VOCABULARY EDITOR2 IMMEDIATE EDITOR2 DEFINITIONS
  7   0 VARIABLE CUR
  8   : !CUR 0 MAX B/SCR B/BUF * 1- MIN CUR ! ;
  9   : +CUR CUR @ + !CUR ;
 10   : +LIN CUR @ C/L / + C/L * !CUR ;            DECIMAL
 11 : LINE. DO I SCR @ (LINE) I CLINE LOOP ;
 12 : BCK 0 0 GOTOXY QUIT ;  ( <--This line can be removed)
 13 : PTR SCR @ B/SCR * CUR @ B/BUF /MOD ROT + BLOCK + ;
 14 : R/C CUR @ C/L /MOD ; ( --- COL ROW )    R->BASE  -->
 15

SCR #23
  0 ( 64 COLUMN EDITOR )  BASE->R HEX
  1
  2 : CINIT 3800 DUP ' SPDTAB ! 800 / 6 VWTR 3800 ‘ SATR !
  3   SATR 2 0 DO DUP >R D000 SP@ R> 2 VMBW DROP 4 + LOOP DROP
  4   0000 0000 0000 0000 5 SPCHAR  0 CUR !
  5   F090 9090 9090 90F0 6 SPCHAR 0 1 F 5 0 SPRITE ; DECIMAL
  6
  7 : PLACE CUR @ 64 /MOD 8 * 1+ SWAP 4 * 1- DUP 0< IF DROP 0 ENDIF
  8   SWAP 0 SPRPUT ;
  9 : UP  -64 +CUR PLACE  ;
 10 : DOWN  64 +CUR PLACE   ;
 11 : LEFT  -1 +CUR PLACE  ;
 12 : RIGHT  1 +CUR PLACE ;
 13 : CGOTOXY ( COL ROW --- ) 64 * + !CUR PLACE ;
 14
 15 R->BASE -->
```

```
SCR #24
  0 ( 64 COLUMN EDITOR ) BASE->R
  1
  2 DECIMAL
  3
  4   : .CUR CUR @ C/L /MOD CGOTOXY ;
  5   : DELHALF PAD 64 BLANKS PTR PAD C/L R/C DROP - CMOVE ;
  6
  7   : DELLIN R/C SWAP MINUS +CUR PTR PAD C/L CMOVE DUP L/SCR SWAP
  8     DO PTR 1 +LIN PTR SWAP C/L CMOVE LOOP
  9     0 +LIN PTR C/L 32 FILL C/L * !CUR ;
 10   : INSLIN R/C SWAP MINUS +CUR L/SCR +LIN DUP 1+ L/SCR 0 +LIN
 11     DO PTR -1 +LIN PTR SWAP C/L CMOVE -1 +LOOP
 12     PAD PTR C/L CMOVE C/L * !CUR ;
 13   : RELINE R/C SWAP DROP DUP  LINE. UPDATE .CUR ;
 14   : +.CUR +CUR .CUR ;
 15 R->BASE -->

SCR #25
  0 ( 64 COLUMN EDITOR )  BASE->R  DECIMAL
  1 : -TAB PTR DUP C@ BL >
  2   IF BEGIN 1- DUP -1 +CUR C@ BL =
  3      UNTIL
  4   ENDIF
  5   BEGIN CUR @ IF 1- DUP -1 +CUR C@ BL > ELSE .CUR 1 ENDIF UNTIL
  6   BEGIN CUR @ IF 1- DUP -1 +CUR C@ BL = DUP IF 1 +.CUR ENDIF
  7               ELSE .CUR 1 ENDIF
  8   UNTIL DROP ;
  9 : TAB PTR DUP C@ BL = 0=
 10   IF BEGIN 1+ DUP 1 +CUR C@ BL =
 11      UNTIL
 12   ENDIF
 13   CUR @ 1023 = IF .CUR 1
 14                ELSE BEGIN 1+ DUP 1 +CUR C@ BL > UNTIL .CUR
 15                ENDIF DROP ;  R->BASE -->

SCR #26
  0 ( 64 COLUMN EDITOR )  BASE->R
  1 DECIMAL
  2 : !BLK PTR  C! UPDATE  ;
  3 : BLNKS PTR R/C DROP C/L SWAP - 32 FILL ;
  4 : HOME  0 0 CGOTOXY ;
  5 : REDRAW SCR @ CLIST UPDATE .CUR ;
  6 : SCRNO CLS 0 0 GOTOXY ." SCR #" SCR @ BASE->R DECIMAL U.
  7   R->BASE CR ;
  8 : +SCR SCR @ 1+ DUP SCR ! SCRNO CLIST  ;
  9 : -SCR SCR @ 1- 0 MAX DUP SCR ! SCRNO  CLIST ;
 10 : DEL PTR DUP 1+ SWAP R/C DROP C/L SWAP - CMOVE 32
 11   PTR R/C DROP - C/L + 1- C! ;
 12 : INS 32 PTR DUP R/C DROP C/L SWAP - + SWAP DO
 13   I C@ LOOP DROP PTR DUP R/C DROP C/L SWAP - + 1- SWAP 1- SWAP
 14   DO I C! -1 +LOOP ;   R->BASE  -->
 15
```

```
SCR #27
  0 ( 64 COLUMN EDITOR  15JUL82 LAO )              BASE->R DECIMAL
  1 0 VARIABLE BLINK  0 VARIABLE OKEY
  2 10 CONSTANT RL  150 CONSTANT RH  0 VARIABLE KC RH VARIABLE RLOG
  3 : RKEY BEGIN ?KEY -DUP 1 BLINK +! BLINK @ DUP 60 < IF 6 0 SPRPAT
  4  ELSE 5 0 SPRPAT ENDIF    120 = IF 0 BLINK ! ENDIF
  5         IF ( SOME KEY IS PRESSED )    KC @   1 KC +!  0 BLINK !
  6           IF ( WAITING TO REPEAT )   RLOG @  KC @  <
  7              IF ( LONG ENOUGH ) RL RLOG ! 1 KC ! 1 ( FORCE EXT)
  8              ELSE OKEY @ OVER =
  9                IF DROP 0   ( NEED TO WAIT MORE )
 10                ELSE 1 ( FORCE EXIT )    DUP KC !   ENDIF
 11              ENDIF
 12           ELSE ( NEW KEY ) 1 ( FORCE LOOP EXIT )  ENDIF
 13         ELSE ( NO KEY PRESSED) RH RLOG ! 0 KC !   0
 14         ENDIF
 15 UNTIL DUP OKEY !      ;                 R->BASE -->

SCR #28
  0 ( 64 COLUMN EDITOR )  BASE->R HEX
  1 : EDT  VDPMDE @ 5 = 0= IF SPLIT ENDIF CINIT !CUR R/C CGOTOXY
  2   DUP DUP  SCR ! SCRNO  CLIST  BEGIN RKEY
  3   CASE 08 OF LEFT  ENDOF             0C OF -SCR          ENDOF
  4        0A OF DOWN  ENDOF             03 OF DEL RELINE    ENDOF
  5        0B OF UP    ENDOF             04 OF INS RELINE    ENDOF
  6        09 OF RIGHT ENDOF             07 OF DELLIN REDRAW ENDOF
  7        0E OF HOME  ENDOF             06 OF INSLIN REDRAW ENDOF
  8        02 OF +SCR  ENDOF             16 OF TAB ENDOF
  9        0D OF 1 +LIN .CUR PLACE ENDOF 7F OF -TAB          ENDOF
 10        01 OF DELHALF BLNKS RELINE ENDOF
 11        0F OF 5 0 SPRPAT CLS SCRNO DROP 300 ' SATR ! QUIT ENDOF
 12        1E OF INSLIN BLNKS REDRAW ENDOF
 13  DUP 1F > OVER 7F < AND IF DUP !BLK R/C SWAP DROP DUP SCR @
 14  (LINE) ROT CLINE 1 +.CUR ELSE  7 EMIT  ENDIF ENDCASE AGAIN ;
 15 R->BASE -->

SCR #29
  0 ( 64 COLUMN EDITOR ) BASE->R  HEX
  1 FORTH DEFINITIONS
  2 : EDIT EDITOR2 0 EDT ;
  3 : WHERE EDITOR2 B/SCR /MOD SWAP B/BUF * ROT + 2- EDT ;
  4
  5 : ED@ EDITOR2 SCR @ SCRNO EDIT ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15 R->BASE
```

```
SCR #33
  0 ( SYSTEM CALLS 09JUL82 LCT)  0 CLOAD RANDOMIZE
  1 BASE->R DECIMAL 74 R->BASE CLOAD ;CODE
  2 BASE->R  DECIMAL
  3 : VSBW 0 SYSTEM ; : VMBW 2 SYSTEM ;
  4 : VSBR 4 SYSTEM ; : VMBR 6 SYSTEM ;
  5 : VWTR 8 SYSTEM ; : GPLLNK 0 33660 C! 10 SYSTEM ;
  6 : XMLLNK 12 SYSTEM ; : DSRLNK 8 14 SYSTEM ;
  7 : CLS 16 SYSTEM ; : FORMAT-DISK 1+ 18 SYSTEM ;
  8 : VFILL 20 SYSTEM ; : VAND 22 SYSTEM ; : VOR 24 SYSTEM ;
  9 : VXOR 26 SYSTEM ;      HEX
 10 CODE MON 0200 , 4E4F , 0201 , 2000 , CC40 , 0281 , 4000 , 16FC ,
 11          0420 , 0000 ,
 12 : RNDW 83C0 DUP @ 6FE5 * 7AB9 + 5 SRC DUP ROT ! ;
 13 : RND RNDW ABS SWAP MOD ;  : SEED 83C0 ! ;
 14 : RANDOMIZE 8802 C@ DROP 0 BEGIN 1+ 8802 C@ 80 AND UNTIL SEED ;
 15 R->BASE

SCR #34
  0 ( SCREEN EDITOR 09JUL82 LCT) 0 CLOAD ED@
  1 BASE->R DECIMAL 33 R->BASE CLOAD RANDOMIZE
  2 BASE->R   HEX VOCABULARY EDITOR1 IMMEDIATE EDITOR1 DEFINITIONS
  3   0 VARIABLE OLDCUR 6 ALLOT
  4   : GETCUR 8F0 OLDCUR 8 VMBR ;  : PUTCUR OLDCUR 8F0 8 VMBW ;
  5   : BOX 8F7 8F1 DO 84 I VSBW LOOP 0FC 8F0 VSBW 0FC 8F7 VSBW ;
  6   : CUR R# ;  : !CUR 0 MAX B/SCR B/BUF * 1- MIN CUR ! ;
  7   : +CUR CUR @ + !CUR ;  : +LIN CUR @ C/L / + C/L * !CUR ;
  8   0 VARIABLE S_H     DECIMAL
  9   : FTYPE 40 * 124 + SWAP VMBW ;
 10   : LISTA DECIMAL 0 0 GOTOXY DUP SCR !
 11     ."  SCR # " . CR CR CR 16 0 DO I 3 .R CR LOOP ;
 12   : ROWCAL S_H @ IF 29 + ENDIF ;
 13   : LINE. DO I SCR @ (LINE) DROP ROWCAL 35 I FTYPE LOOP ;
 14   : LISTB L/SCR 0 LINE. ;
 15 R->BASE -->

SCR #35
  0 ( SCREEN EDITOR 09JUL82 LCT)
  1
  2 : LISTL BASE->R LISTA 4 1  GOTOXY
  3 ."          1         2         3     " 4 2 GOTOXY
  4 ." ....+....0....+....0....+....0....+"
  5 0 S_H ! LISTB R->BASE ;
  6 : LISTR BASE->R DROP 4 1 GOTOXY
  7 ." 3         4         5         6    " 4 2 GOTOXY
  8 ." 0....+....0....+....0....+....0...."
  9 1 S_H ! LISTB R->BASE ;
 10 : BCK  0 L/SCR 2+ GOTOXY PUTCUR QUIT ;
 11 : PTR SCR @ B/SCR * CUR @ B/BUF /MOD ROT + BLOCK + ;
 12 : R/C CUR @ C/L /MOD ;  ( --- COL ROW )
 13 : DELHALF PAD 64 BLANKS PTR PAD C/L R/C DROP - CMOVE ;
 14
 15 -->
```

```
SCR #36
  0 ( SCREEN EDITOR 12JUL82 LCT) BASE->R  DECIMAL
  1   : .CUR CUR @ C/L /MOD 3 + SWAP 4 + DUP S_H @
  2     IF 32 > IF 29 - ELSE SCR @ LISTL ENDIF
  3     ELSE 39 < 0= IF SCR @ LISTR 29 - ENDIF
  4     ENDIF SWAP GOTOXY ;
  5   : DELLIN R/C SWAP MINUS +CUR PTR PAD C/L CMOVE DUP L/SCR SWAP
  6     DO PTR 1 +LIN PTR SWAP C/L CMOVE LOOP
  7     0 +LIN PTR C/L 32 FILL C/L * !CUR ;
  8   : INSLIN R/C SWAP MINUS +CUR L/SCR +LIN DUP 1+ L/SCR 0 +LIN
  9     DO PTR -1 +LIN PTR SWAP C/L CMOVE -1 +LOOP
 10     PAD PTR C/L CMOVE C/L * !CUR ;
 11   : RELINE R/C SWAP DROP DUP 13 EMIT LINE. UPDATE .CUR ;
 12 : +.CUR +CUR .CUR ;
 13 : TAB PTR DUP @ 32 = 0= IF BEGIN 1+ DUP 1 +CUR C@ 32 = UNTIL
 14   ENDIF CUR @ 1023 = IF .CUR 1 ELSE  BEGIN 1+ DUP 1 +CUR C@ 32 >
 15   UNTIL .CUR ENDIF ; R->BASE -->

SCR #37
  0 ( SCREEN EDITOR 12JUL82 LCT)  BASE->R  DECIMAL
  1 : -TAB PTR DUP C@ 32 > IF BEGIN 1- DUP -1 +CUR C@ 32 = UNTIL
  2   ENDIF BEGIN CUR @ IF 1- DUP -1 +CUR C@ 32 > ELSE .CUR 1 ENDIF
  3   UNTIL BEGIN CUR @ IF 1- DUP -1 +CUR C@ 32 = DUP IF 1 +.CUR
  4   ENDIF ELSE .CUR 1 ENDIF UNTIL ; : !BLK PTR C! UPDATE 1 +.CUR ;
  5  : BLNKS PTR R/C DROP C/L SWAP - 32 FILL ;
  6 : FLIP S_H @ IF -29 ELSE 29 ENDIF +.CUR ;
  7 : REDRAW SCR @ S_H @ IF LISTR ELSE LISTL ENDIF UPDATE .CUR ;
  8 : NEWSCR 0 SWAP LISTL !CUR .CUR ;
  9 : +SCR SCR @ 1+ NEWSCR ;
 10 : -SCR SCR @ 1- 0 MAX NEWSCR ;
 11 : DEL PTR DUP 1+ SWAP R/C DROP C/L SWAP - CMOVE 32
 12   PTR R/C DROP - C/L + 1- C! ;
 13 : INS 32 PTR DUP R/C DROP C/L SWAP - + SWAP DO
 14   I C@ LOOP DROP PTR DUP R/C DROP C/L SWAP - + 1- SWAP 1- SWAP
 15   DO I C! -1 +LOOP ; R->BASE -->

SCR #38
  0 ( SCREEN EDITOR 12JUL82 LCT) BASE->R HEX
  1 : VED GETCUR BOX SWAP CLS LISTL !CUR .CUR BEGIN KEY CASE
  2   0F OF BCK             ENDOF  01 OF DELHALF BLNKS RELINE ENDOF
  3   08 OF -1 +.CUR        ENDOF  02 OF +SCR                 ENDOF
  4   0A OF C/L +.CUR       ENDOF  0C OF -SCR                 ENDOF
  5   0B OF C/L MINUS +.CUR ENDOF  03 OF DEL RELINE           ENDOF
  6   09 OF 1 +.CUR         ENDOF  04 OF INS RELINE           ENDOF
  7   0D OF 1 +LIN .CUR     ENDOF  07 OF DELLIN REDRAW        ENDOF
  8   0E OF FLIP            ENDOF  06 OF INSLIN REDRAW        ENDOF
  9   1E OF INSLIN BLNKS REDRAW ENDOF 16 OF TAB             ENDOF
 10   7F OF -TAB ENDOF
 11    DUP 1F > OVER 7F < AND IF DUP EMIT DUP !BLK ELSE 7 EMIT ENDIF
 12    ENDCASE AGAIN ; FORTH DEFINITIONS
 13 : WHERE EDITOR1 B/SCR /MOD SWAP B/BUF * ROT + 2- VED ;
 14 : EDIT EDITOR1 0 VED ;  : ED@ EDITOR1 SCR @ EDIT ;
 15 R->BASE
```

```
SCR #39
  0 ( STRING STORE AND SCREEN COPY WORDS 12JUL82 LCT)  0 CONSTANT AD
  1  0 CLOAD DISK-HEAD  ( ADDR ---   ) BASE->R  HEX
  2 : (!") R COUNT DUP 1+ =CELLS R> + >R >R SWAP R> CMOVE ;
  3 : !" 22 STATE @  ( STORE STRING AT ADDR )
  4     IF COMPILE (!") WORD HERE C@
  5        1+ =CELLS ALLOT
  6     ELSE WORD HERE COUNT >R SWAP R> CMOVE
  7     ENDIF ; IMMEDIATE  DECIMAL ( SCREEN COPYING WORDS )
  8 : DTEST 90 0 DO I DUP . BLOCK DROP LOOP ;
  9 : SCOPY OFFSET @ + SWAP BLOCK 2- ! UPDATE FLUSH ; ( 1K BLOCKS )
 10 : SMOVE >R OVER OVER - DUP 0< SWAP R MINUS > + 2 = IF
 11   OVER OVER SWAP R + 1- SWAP R + 1-  -1 ' AD ! ELSE 1 ' AD !
 12   ENDIF R> 0 DO OVER OVER SCOPY AD + SWAP AD + SWAP LOOP DROP
 13   DROP ;
 14 : FORTH-COPY 90 0 DO I DUP . 90 + I SCOPY LOOP ;
 15 R->BASE  -->

SCR #40
  0 ( WRITE A HEAD COMPATABLE WITH THE DISK MANAGER 12JUL82 LCT)
  1 BASE->R HEX
  2 : DISK-HEAD 0 CLEAR 0 BLOCK ( START SECTOR 0)
  3   DUP !" FORTH     " DUP A + 168 SWAP !
  4   DUP C + 944 SWAP ! DUP E + 534B SWAP !
  5   DUP 10 + 2000 SWAP ! DUP 12 + 26 0 FILL
  6   DUP 38 + C8 FF FILL 100 + ( START SECTOR 1)
  7   DUP 2 SWAP ! DUP 2+ FE 00 FILL
  8   100 + ( START SECTOR 2)
  9   DUP !" SCREENS  " DUP A + 0 SWAP !
 10   DUP C + 2 SWAP ! DUP E + 165 SWAP !
 11   DUP 10 + 80 SWAP ! DUP 12 + CA02 SWAP !
 12   DUP 14 + 8 0 FILL DUP 1C + 2250 SWAP !
 13   DUP 1E + 1403 SWAP ! DUP 20 + 4016 SWAP ! 22 + 0DE 0 FILL
 14   FLUSH
 15 ; R->BASE
```

```
SCR #42
  0 ( DUMP ROUTINES 12JUL82 LCT)
  1 0 CLOAD VLIST   BASE->R HEX
  2 : DUMP8 -DUP
  3   IF
  4     BASE->R HEX 0 OUT ! SPACE OVER 4 U.R
  5     OVER OVER 0 DO
  6       DUP @ 0 <# # # # # BL HOLD BL HOLD #> TYPE 2+ 2
  7     +LOOP DROP 1F OUT @ - SPACES
  8   0 DO
  9     DUP C@ DUP 20 < OVER 7E > OR
 10     IF DROP 2E ENDIF
 11     EMIT 1+
 12   LOOP
 13   CR R->BASE
 14   ENDIF ;
 15 -->

SCR #43
  0 ( DUMP ROUTINES 12JUL82 LCT)
  1 : DUMP CR 00 8 U/ >R SWAP R> -DUP
  2   IF 0
  3     DO 8 DUMP8 PAUSE IF SWAP DROP 0 SWAP LEAVE ENDIF LOOP
  4   ENDIF SWAP DUMP8 DROP ;
  5 : .S CR SP@ 2- S0 @ 2- ." | " OVER OVER = 0= IF
  6      DO I @ U. -2 +LOOP ELSE DROP DROP ENDIF ;
  7 : VLIST 80 OUT ! CONTEXT @ @
  8     BEGIN DUP C@ 3F AND OUT @ + 25 >
  9       IF CR 0 OUT ! ENDIF
 10       DUP ID. PFA LFA @ SPACE DUP 0= PAUSE OR
 11     UNTIL DROP ;  R->BASE
 12
 13
 14
 15

SCR #44
  0 ( TRACE COLON WORDS-FORTH DIMENSIONS III/2 P.58 26OCT82 LCT)
  1 0 CLOAD (TRACE)  BASE->R DECIMAL 42 R->BASE CLOAD VLIST
  2 FORTH DEFINITIONS
  3 0 VARIABLE TRACF   ( CONTROLS INSERTION OF TRACE ROUTINE )
  4 0 VARIABLE TFLAG   ( CONTROLS TRACE OUTPUT )
  5 : TRACE 1 TRACF ! ;
  6 : UNTRACE 0 TRACF ! ;
  7 : TRON 1 TFLAG ! ;
  8 : TROFF 0 TFLAG ! ;
  9 : (TRACE) TFLAG @       ( GIVE TRACE OUTPUT? )
 10   IF  CR R 2- NFA ID.  ( BACK TO PFA NFA FOR NAME )
 11       .S ENDIF ;        ( PRINT STACK CONTENTS )
 12 : :  ( REDEFINED TO INSERT TRACE WORD AFTER COLON )
 13   ?EXEC !CSP CURRENT @ CONTEXT ! CREATE [ ' : CFA @ ] LITERAL
 14  HERE 2- !   TRACF @ IF ' (TRACE) CFA DUP @ HERE 2- ! , ENDIF ]
 15   ; IMMEDIATE
```

```
SCR #45
  0 ( FLOATING POINT <4 WORD> STACK ROUTINES 12JUL82 LCT)
  1 0 CLOAD PI  BASE->R DECIMAL 33 R->BASE CLOAD RANDOMIZE
  2 BASE->R HEX
  3 : FDUP SP@ DUP 2- SWAP 6 + DO I @ -2 +LOOP ;
  4 : FDROP DROP DROP DROP DROP ;
  5 : FOVER SP@ DUP 6 + SWAP E + DO I @ -2 +LOOP ;
  6 : FSWAP FOVER >R >R >R >R >R >R >R >R
  7         FDROP R> R> R> R> R> R> R> R> ;
  8 : F! 4 0 DO DUP >R ! R> 2+ LOOP DROP ;
  9 : F@ 6 + 4 0 DO DUP >R @ R> 2- LOOP DROP ;
 10 834A CONSTANT FAC  835C CONSTANT ARG
 11 : >FAC FAC F! ;   : >ARG ARG F! ;  : FAC> FAC F@ ;
 12 : SETFL >FAC >ARG ;
 13 : FADD 0600 C SYSTEM ;  : FSUB 0700 C SYSTEM ;
 14 : FMUL 0800 C SYSTEM ;  : FDIV 0900 C SYSTEM ;
 15 R->BASE  -->

SCR #46
  0 ( FLOATING POINT ARITHMETIC ROUTINES 12JUL82 LCT)
  1 BASE->R HEX
  2 : F+ SETFL FADD FAC> ;
  3 : F- SETFL FSUB FAC> ;
  4 : F* SETFL FMUL FAC> ;
  5 : F/ SETFL FDIV FAC> ;
  6 : S->FAC FAC ! 2300 C SYSTEM ;
  7 : FAC->S 1200 C SYSTEM FAC @ ;
  8 : FAC>ARG FAC ARG 8 CMOVE ;
  9 : F->S >FAC FAC->S ;
 10 : S->F S->FAC FAC> ;
 11 DECIMAL
 12 : FRND 3 0 DO 100 RND 100 RND 256 * + LOOP
 13   100 RND 16128 + ;
 14
 15 R->BASE -->

SCR #47
  0 ( FLOATING POINT CONVERSION ROUTINES CONTINUED 12JUL82 LCT)
  1 BASE->R HEX
  2 : DOSTR FAC B + C! 14 GPLLNK
  3   FAC B + C@ 8300 + FAC C + C@ DUP PAD C!
  4   PAD 1+ SWAP CMOVE ;
  5
  6 ( NUMBER IN FAC CONVERTED TO BASIC STRING AND PLACED AT PAD)
  7 : STR 0 DOSTR ;
  8
  9 ( NUMBER IN FAC CONVERTED TO FIXED STRING AND PLACED AT PAD)
 10 : STR. FAC D + C! FAC C + C! DOSTR ;
 11
 12 ( STRING AT PAD CONVERTED TO NUMBER IN FAC)
 13 : VAL PAD 1+ DISK_BUF @ DUP FAC C + ! PAD C@ OVER OVER + 20
 14   SWAP VSBW VMBW 1000 XMLLNK ;
 15 R->BASE -->
```

```
SCR #48
  0 ( FLOATING POINT - COMPILE NO TO STACK 12JUL82 LCT) BASE->R HEX
  1 : F$ PAD 1+ SWAP >R R CMOVE R> PAD C! VAL FAC> ;
  2 : (>F) R COUNT DUP 1+ =CELLS R> + >R F$ ;
  3 : >F 20 STATE @
  4     IF   COMPILE (>F) WORD HERE C@
  5          1+ =CELLS ALLOT
  6     ELSE WORD HERE COUNT F$
  7     ENDIF ; IMMEDIATE
  8
  9 ( FLOATING POINT OUTPUT ROUTINES )
 10 : JST PAD C@ - SPACES PAD COUNT TYPE ;
 11 : F.R >R >FAC STR R> JST ;
 12 : F. 0 F.R ;
 13 : FF.R >R >R >R >FAC R> 0 R> STR. R> JST ;
 14 : FF. 0 FF.R ;
 15 R->BASE -->

SCR #49
  0 ( FLOATING POINT COMPARE ROUTINES 12JUL82 LCT)
  1 BASE->R HEX
  2 : FCLEAN >R DROP DROP DROP R> ;
  3
  4 : F0< 0< FCLEAN ;
  5
  6 : F0= 0= FCLEAN ;
  7
  8 : FCOM SETFL 0A00 C SYSTEM 837C C@ ;
  9 : F> FCOM 40 AND MINUS 0< ;
 10 : F= FCOM 20 AND MINUS 0< ;
 11 : F< FCOM 60 AND 0= ;
 12 : FLERR 8354 C@ ;
 13 : ?FLERR FLERR A ?ERROR ;
 14
 15 R->BASE -->

SCR #50
  0 ( FLOATING POINT TRANSCENDENTAL FUNCTIONS 12JUL82 LCT)
  1 BASE->R HEX
  2 0 VARIABLE LNKSAV
  3 : GLNK 83C4 @ LNKSAV ! GPLLNK LNKSAV @ 83C4 ! ;
  4 : INT >FAC 22 GLNK FAC> ;
  5 : ^   SETFL ARG 836E @ 8 VMBW 24 GLNK FAC> 8 836E +! ;
  6 : SQR >FAC 26 GLNK FAC> ;
  7 : EXP >FAC 28 GLNK FAC> ;
  8 : LOG >FAC 2A GLNK FAC> ;
  9 : COS >FAC 2C GLNK FAC> ;
 10 : SIN >FAC 2E GLNK FAC> ;
 11 : TAN >FAC 30 GLNK FAC> ;
 12 : ATN >FAC 32 GLNK FAC> ;
 13 : PI  >F 3.141592653590 ;
 14
 15 R->BASE
```

```
SCR #51
  0 ( CONVERT TO TEXT MODE CONFIGURATION 14SEP82 LAO)
  1 0 CLOAD TEXT BASE->R DECIMAL 56 R->BASE CLOAD SETVDP2
  2 BASE->R  HEX
  3
  4 : TEXT
  5 0 3C0 20 VFILL ( BLANKS TO SCREEN IMAGE AREA )
  6 28 SCRN_WIDTH ! 0 SCRN_START ! 3C0 SCRN_END ! 460 PABS !
  7 SETVDP1     1 VDPMDE !
  8 ( NOW SET VDP REGISTERS )
  9   1 6 VWTR   0F4 7 VWTR
 10 0F0 SETVDP2 ;
 11
 12
 13
 14
 15 R->BASE

SCR #52
  0 ( CONVERT TO GRAPHICS MODE CONFIG 14SEP82 LAO)
  1 0 CLOAD GRAPHICS  BASE->R DECIMAL 56 R->BASE CLOAD SETVDP2
  2 BASE->R  HEX
  3
  4 : GRAPHICS
  5 0 300 20 VFILL ( BLANKS TO SCREEN IMAGE AREA ) 300 80 0 VFILL
  6 380 20 F4 VFILL
  7 20 SCRN_WIDTH ! 0 SCRN_START ! 300 SCRN_END !
  8 SETVDP1   2 VDPMDE !
  9 ( NOW SET VDP REGISTERS )
 10    1 6 VWTR   0F4 7 VWTR
 11 E0 SETVDP2 ;
 12
 13
 14
 15 R->BASE

SCR #53
  0 ( CONVERT TO MULTI-COLOR MODE CONFIG 14SEP82 LAO)
  1 0 CLOAD MULTI  BASE->R DECIMAL 56 R->BASE CLOAD SETVDP2
  2 BASE->R  HEX
  3
  4 : MULTI      0B0 1 VWTR ( BLANK THE SCREEN )
  5 -1 18 0 DO I 4 / 0FF SWAP DO 1+ I OVER VSBW 8 +LOOP LOOP DROP
  6 800 800 0 VFILL         ( INIT 256 CHAR PATTERNS TO 0 )
  7  300 80 0 VFILL 380 20 0F4 VFILL
  8 20 SCRN_WIDTH ! 0 SCRN_START ! 300 SCRN_END ! 460 PABS !
  9 1000 DISK_BUF !         ( RESTORE USER VARIABLES )
 10 3 VDPMDE !
 11 ( NOW SET VDP REGISTERS )
 12    4 6 VWTR   11 7 VWTR
 13 0EB SETVDP2 ;
 14
 15 R->BASE
```

```
SCR #54
  0 ( CONVERT TO GRAPHICS2 MODE CONFIG 14SEP82 LAO)
  1 0 CLOAD GRAPHICS2 BASE->R DECIMAL 56 R->BASE CLOAD SETVDP2
  2 BASE->R HEX : GRAPHICS2  0A0 1 VWTR
  3 -1 1B00 1800 DO 1+ DUP 0FF AND I VSBW LOOP DROP
  4 1 PABS @ VSBW 16 PABS @ 1+ VSBW 1 ( #FILE) 834C C! PABS @ 8356 !
  5  0A 0E SYSTEM ( SUBROUTINE TYPE DSRLNK TO SET 2 DISK BUFFERS )
  6 0 1800 0F0 VFILL ( INIT COLOR TABLE )
  7 2000 1800 0 VFILL ( INIT BIT MAP )
  8 20 SCRN_WIDTH ! 1800 SCRN_START ! 1B00 SCRN_END ! 1B00 PABS !
  9 1C00 DISK_BUF ! ( USER VARIABLES NOW SET UP )
 10  2 0 VWTR      6 2 VWTR  ( SET VDP REGISTERS )
 11 07F 3 VWTR     0FF 4 VWTR
 12 70 5 VWTR      7 6 VWTR
 13 0F1 7 VWTR     0E0 DUP 83D4 C! 1 VWTR    1BC0 836E ! ( VSPTR )
 14 0 0 GOTOXY 4 VDPMDE !  0 837A C!  ;
 15 R->BASE

SCR #55
  0 ( CONVERT TO SPLIT MODE CONFIG 14SEP82 LAO)
  1 0 CLOAD SPLIT  BASE->R DECIMAL 56 R->BASE CLOAD SETVDP2
  2 BASE->R DECIMAL 54 R->BASE CLOAD GRAPHICS2
  3 BASE->R  HEX
  4 : SPLIT GRAPHICS2 1A00 SCRN_START ! 0A0 1 VWTR 3000 800 0FF
  5   VFILL 3100 834A ! 18 GPLLNK 3300 834A ! 4A GPLLNK
  6   1A00 100 20 VFILL 1000 800 0F4 VFILL 0 0 GOTOXY 0E0 1 VWTR
  7   5 VDPMDE !   0 837A C!  ;
  8
  9 : SPLIT2 GRAPHICS2 1880 SCRN_END ! 2000 400 0FF VFILL
 10   2100 834A ! 18 GPLLNK 2300 834A ! 4A GPLLNK
 11   1800 80 20 VFILL 0 400 0F4 VFILL 0 0 GOTOXY 6 VDPMDE !
 12   0 837A C! ;
 13
 14
 15 R->BASE

SCR #56
  0 ( VDPMODES 14SEP82 LAO ) 0 CLOAD SETVDP2  BASE->R DECIMAL 33
  1 R->BASE CLOAD RANDOMIZE  BASE->R  HEX
  2  : SETVDP1 0B0 1 VWTR     ( BLANK THE SCREEN )
  3     800 800 0FF VFILL     ( INIT 256 CHAR PATTERNS TO FF )
  4     900 834A ! 18 GPLLNK  ( LOAD CAPITAL LETTERS )
  5     B00 834A ! 4A GPLLNK  ( LOAD LOWER CASE -ON 99/4A ONLY ) ;
  6 : SETVDP2 ( n --- )  460 PABS !
  7     1000 DISK_BUF !       ( RESTORE USER VARIABLES )
  8     ( SET VDP REGISTERS )
  9     0 0 VWTR  0 2 VWTR  0E 3 VWTR
 10     1 4 VWTR  6 5 VWTR
 11     3E0 836E !            ( VSPTR )
 12 1 PABS @ VSBW 16 PABS @ 1+ VSBW 3 ( #FILE) 834C C! PABS @ 8356 !
 13     0A 0E SYSTEM          ( SUB TYPE DSRLNK TO SET 3 DISK BUF )
 14     0 0 GOTOXY  0 837A C!
 15     DUP 83D4 C! 1 VWTR ;  R->BASE
```

```
SCR #57
  0 ( GRAPHICS PRIMITIVES 12JUL82 LCT)  0 CLOAD LINE BASE->R DECIMAL
  1  33 R->BASE CLOAD RANDOMIZE  BASE->R DECIMAL 74 R->BASE CLOAD
  2 ;CODE   BASE->R HEX
  3 380 CONSTANT COLTAB 300 CONSTANT SATR 780 CONSTANT SMTN
  4 800 CONSTANT PDT  800 CONSTANT SPDTAB
  5 : CHAR ( W1 W2 W3 W4 CH ---  )
  6   8 * PDT + >R -2 6 DO PAD I + ! -2 +LOOP PAD R> 8 VMBW ;
  7 : CHARPAT ( CH --- W1 W2 W3 W4 )
  8   8 * PDT + PAD 8 VMBR 8 0 DO PAD I + @ 2 +LOOP ;
  9 : VCHAR ( X Y CNT CH ---  )
 10   >R >R SCRN_WIDTH @ * + SCRN_END @ SCRN_START @ - SWAP
 11   R> R> SWAP 0 DO SWAP OVER OVER SCRN_START @ + VSBW SCRN_WIDTH
 12   @ + ROT OVER OVER /MOD IF 1+ SCRN_WIDTH @ OVER OVER = IF -
 13   ELSE DROP ENDIF ENDIF ROT DROP ROT LOOP DROP DROP DROP ;
 14 R->BASE -->
 15

SCR #58
  0 ( GRAPHICS PRIMITIVES 20OCT83 LAO)   BASE->R HEX
  1 : HCHAR ( X Y CNT CH ---  )
  2   >R >R SCRN_WIDTH @ * + SCRN_START @ + R> R> VFILL ;
  3 : COLOR ( FG BG CHSET  ---  ) >R SWAP 10 * + R> COLTAB + VSBW ;
  4 : SCREEN ( COLOR  --- )  7 VWTR ;
  5 : GCHAR ( X Y --- ASCII ) ( COLUMNS AND ROWS NUMBERED FROM 0 )
  6   SCRN_WIDTH @ * + SCRN_START @ + VSBR ;
  7 : SSDT  ( ADDR --- ) ( SET SPRITE DESCRIPTOR TABLE ADDRESS )
  8   DUP ' SPDTAB ! 800 / 6 VWTR ( RESET VDP REG 6 )
  9   VDPMDE @ 4 < IF SMTN 80 0 VFILL 300 ' SATR ! ENDIF
 10   SATR 20 0 DO DUP >R D000 SP@ R> 2 VMBW DROP 4 + LOOP DROP
 11   ( INIT ALL SPRITES ) ;
 12 : SPCHAR  ( W1 W2 W3 W4 CH#  ---  )
 13   8 * SPDTAB + >R -2 6 DO PAD I + ! -2 +LOOP PAD R> 8 VMBW ;
 14 : SPRCOL ( COL #  ---  )  4 * SATR 3 + + DUP >R VSBR 0F0 AND OR
 15   R> VSBW ;             R->BASE -->

SCR #59
  0 ( GRAPHICS PRIMITIVES 20OCT83 LCT)
  1 BASE->R  HEX
  2 : SPRPAT ( CH # --- )  4 * SATR 2+ + VSBW ;
  3 : SPRPUT ( DX DY # --- )
  4   4 * SATR + >R 1- 100 U* DROP + SP@ R> 2 VMBW DROP ;
  5 : SPRITE ( DX DY COL CH # ---  ) ( SPRITES NUMBERED 0 - 31 )
  6   DUP 4 * SATR + >R DUP >R SPRPAT R SPRCOL R> SPRPUT R> 4 +
  7   SATR DO I VSBR D0 = IF C001 SP@ I 2 VMBW DROP ENDIF 4 +LOOP ;
  8 : MOTION ( SPX SPY # --- )
  9   4 * SMTN + >R 8 SLA SWAP 00FE AND OR SP@ R> 2 VMBW DROP ;
 10 : #MOTION ( NO ---  )   837A C! ;
 11 : SPRGET ( # --- DX DY )
 12   4 * SATR + DUP VSBR 1+ 0FF AND SWAP 1+ VSBR SWAP ;
 13 : DXY ( X2 Y2 X1 Y1 --- X^2 Y^2 )
 14   ROT - ABS ROT ROT - ABS DUP * SWAP DUP * ;
 15 R->BASE -->
```

```
SCR #60
  0 ( GRAPHICS PRIMITIVES 12JUL82 LCT)
  1 BASE->R HEX : BEEP 34 GPLLNK ;   : HONK 36 GPLLNK ;
  2 : SPRDIST ( #1 #2 --- DIST^2 ) ( DISTANCE BETWEEN 2 SPRITES )
  3   SPRGET ROT SPRGET DXY OVER OVER
  4   + DUP >R OR OR 8000 AND IF R> DROP 7FFF ELSE R> ENDIF ;
  5 : SPRDISTXY ( X Y # --- DIST^2 ) SPRGET DXY OVER OVER
  6   + DUP >R OR OR 8000 AND IF R> DROP 7FFF ELSE R> ENDIF ;
  7 : MAGNIFY ( MAG-FACTOR ---  )
  8   83D4 C@ 0FC AND + DUP 83D4 C! 1 VWTR ;
  9 : JOYST ( KEYBDNO --- ASCII XSTAT YSTAT )  8374 C!
 10   ?KEY DROP 8375 C@ DUP DUP 12 = IF DROP 0 0 ELSE 0FF =
 11   IF 8377 C@ 8376 C@ ELSE 8375 C@
 12   CASE 4 OF 0FC 4  ENDOF  5 OF 0 4 ENDOF  6 OF 4   4 ENDOF
 13        2 OF 0FC 0  ENDOF  3 OF 4 0 ENDOF  0 OF 0 0FC ENDOF
 14       0F OF 0FC 0FC ENDOF 0E OF 4 0FC ENDOF DROP DROP 0 0 0 0
 15   ENDCASE ENDIF ENDIF  4 8374 C! ; R->BASE -->

SCR #61
  0 ( GRAPHICS PRIMITIVES 12JUL82 LCT)  BASE->R HEX
  1 : COINC ( #1 #2 TOL  --- F ) ( 0= NO COINC  1= COINC )
  2   DUP * DUP + >R SPRDIST R> > 0= ;
  3 : COINCXY ( DX DY # TOL --- F )
  4   DUP * DUP + >R SPRDISTXY R>  > 0= ;
  5 : COINCALL  ( --- F ) ( BIT SET IF ANY TWO SPRITES OVERLAP )
  6   8802 C@ 20 AND 20 = ;
  7 : DELSPR ( # --- )
  8   4 * DUP SATR + >R 0 C001 SP@ R> 4 VMBW DROP DROP
  9   SMTN + >R 0 0 SP@ R> 4 VMBW DROP DROP ;
 10 : DELALL ( --- )
 11   0 #MOTION SATR 20 0 DO DUP D0 SWAP VSBW 4 + LOOP DROP
 12   SMTN 80 0 VFILL ;
 13
 14
 15   R->BASE -->

SCR #62
  0 ( GRAPHICS PRIMITIVES 24NOV82 LAO) BASE->R  HEX  0 VARIABLE ADR
  1 : MINIT 18 0 DO 0 I 4 / 20 * DUP 20 + SWAP
  2              DO DUP J 1 I HCHAR 1+ LOOP  DROP LOOP ;
  3 : MCHAR ( COLOR C R --- ) DUP >R 2 / SWAP DUP >R 2 / SWAP
  4   DUP >R GCHAR  DUP 20 /  100 U* DROP 800 + >R 20 MOD
  5   8 * R> + R>  4 MOD 2 * + ADR ! R> 2 MOD R> 2 MOD SWAP
  6   IF  IF 3 ELSE 1 ENDIF  ELSE  IF 2 ELSE 0 ENDIF   ENDIF
  7   DUP 2 MOD 0= IF SWAP 10 * SWAP ENDIF
  8   CASE 0 OF ADR @ VSBR 0F ENDOF  1 OF ADR @ VSBR F0 ENDOF
  9   2 OF 1 ADR +! ADR @ VSBR 0F ENDOF
 10   3 OF 1 ADR +! ADR @ VSBR F0 ENDOF
 11   ENDCASE AND +   ADR @  VSBW ;
 12 0 VARIABLE DMODE   -1 VARIABLE DCOLOR
 13 : DRAW 0 DMODE ! ;  : UNDRAW 1 DMODE ! ;   : DTOG 2 DMODE ! ;
 14 8040 VARIABLE DTAB 2010 , 804 , 201 , 7FBF , DFEF , F7FB ,
 15 FDFE , 8040 , 2010 , 804 , 201 ,  R->BASE -->
```

```
SCR #63
  0 ( GRAPHICS PRIMITIVES ) BASE->R  HEX
  1 CODE DDOT   C079 ,
  2      C0D9 , C081 , C103 , 0241 ,
  3      0007 , 0243 , 0007 , 0242 ,
  4      00F8 , 0244 , 00F8 , 0A52 ,
  5      A042 , A044 , 0221 , 2000 ,
  6      04C4 , D123 , DTAB , 06C4 ,
  7      C644 , 0649 , C641 , 045F ,
  8 : DOT ( X Y --- )
  9   DDOT DUP 2000 - >R DMODE @
 10   CASE  0 OF VOR  ENDOF  ( DRAW )
 11         1 OF SWAP FF XOR SWAP VAND ENDOF  ( UNDRAW )
 12         2 OF VXOR ENDOF  ( TOGGLE )
 13   DROP DROP ENDCASE R>
 14   DCOLOR @ 0 < IF DROP ELSE DCOLOR @ SWAP VSBW ENDIF ;
 15  R->BASE  -->

SCR #64
  0 ( GRAPHICS PRIMITIVES 12JUL82 LCT) BASE->R HEX
  1 : SGN DUP IF DUP 0< IF -1 ELSE 1 ENDIF ELSE 0 ENDIF + ;
  2 : LINE >R R ROT >R R - SGN SWAP >R R ROT >R R - SGN OVER ABS
  3   OVER ABS < >R R 0= IF SWAP ENDIF 100 ROT ROT */ R>
  4   IF ( X AXIS ) R> R> OVER OVER >
  5      IF ( MAKE L TO R ) SWAP R> DROP R>
  6      ELSE R> R> DROP
  7      ENDIF 100 * ROT ROT 1+ SWAP
  8      DO I OVER 0 100 M/ SWAP DROP DOT OVER + LOOP
  9   ELSE ( Y AXIS ) R> R> R> R> ROT >R ROT >R OVER OVER >
 10      IF ( MAKE T TO B ) SWAP R> DROP R>
 11      ELSE R> R> DROP
 12      ENDIF 100 * ROT ROT 1+ SWAP
 13      DO DUP 0 100 M/ SWAP DROP I DOT OVER + LOOP
 14   ENDIF DROP DROP ;
 15 R->BASE

SCR #65
  0 ( COMPACT LIST )
  1 0 CLOAD SMASH  BASE->R DECIMAL 74 R->BASE CLOAD ;CODE
  2 BASE->R DECIMAL 33 R->BASE CLOAD RANDOMIZE  BASE->R DECIMAL
  3 0 VARIABLE TCHAR 382 ALLOT 67 BLOCK TCHAR 384 CMOVE HEX
  4 TCHAR 7C - CONSTANT TC  0 VARIABLE BADDR  0 VARIABLE INDX
  5 ( SMASH EXPECTS ADDR #CHAR LINE# --- LB VADDR CNT )
  6 0 VARIABLE LB FE ALLOT
  7 CODE SMASH
  8   C079 , C0B9 , C0D9 , 0204 , LB   , C644 , 0649 , 06C1 ,
  9   0221 , 2000 , C641 , C042 , 0581 , 0241 , FFFE , 0649 ,
 10   0A21 , C641 , A083 , 80C2 , 1501 , 1020 , 04C5 , 04C6 ,
 11   D173 , D1B3 , 0965 , 0966 , C025 , TC ,   C066 , TC ,
 12   0B41 , 020C , 0004 , C2C0 , 024B , F000 , C1C1 , 0247 ,
 13   0F00 , E1CB , DD07 , 0BC0 , 0BC1 , 060C , 16F4 , 05C5 ,
 14   05C6 , C305 , 024C , 0002 , 16E7 , 10DD , 045F ,
 15 R->BASE -->
```

```
SCR #66
  0 ( COMPACT LIST ) BASE->R DECIMAL
  1 : CLINE LB 100 ERASE  SMASH   VMBW    ;
  2 : CLOOP  DO I 64 * OVER + 64 I CLINE LOOP DROP ;
  3
  4 : CLIST BLOCK 16 0 CLOOP ;
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14 R->BASE
 15

SCR #68
  0 ( FILE I/O ROUTINES 12JUL82 LCT)
  1 0 CLOAD STAT  BASE->R DECIMAL 33 R->BASE CLOAD RANDOMIZE
  2 BASE->R HEX
  3 0 VARIABLE PAB-ADDR
  4 0 VARIABLE PAB-BUF
  5 0 VARIABLE PAB-VBUF
  6 : FILE <BUILDS , , , DOES> DUP @ PAB-VBUF ! 2+ DUP @ PAB-BUF !
  7     2+ @ PAB-ADDR ! ;
  8 : GET-FLAG PAB-ADDR @ 1+ VSBR ;
  9 : PUT-FLAG PAB-ADDR @ 1+ VSBW ;
 10 : SET-PAB PAB-ADDR @ DUP 0A 0 VFILL 2+ PAB-VBUF SWAP 2 VMBW ;
 11 : CLR-STAT GET-FLAG 1F AND PUT-FLAG ;
 12 : CHK-STAT GET-FLAG 0E0 AND
 13      837C C@ 20 AND OR 9 ?ERROR ;
 14 : FXD GET-FLAG 0EF AND PUT-FLAG ;
 15 : VRBL GET-FLAG 10 OR PUT-FLAG ;   R->BASE -->
```

```
SCR #69
  0 ( FILE I/O ROUTINES 12JUL82 LCT) BASE->R HEX
  1 : DSPLY GET-FLAG 0F7 AND PUT-FLAG ;
  2 : INTRNL GET-FLAG 8 OR PUT-FLAG ;
  3 : I/OMD GET-FLAG 0F9 AND ;
  4 : INPT I/OMD 4 OR PUT-FLAG ;
  5 : OUTPT I/OMD 2 OR PUT-FLAG ;
  6 : UPDT I/OMD PUT-FLAG ;
  7 : APPND I/OMD 6 OR PUT-FLAG ;
  8 : SQNTL GET-FLAG 0FE AND PUT-FLAG ;
  9 : RLTV GET-FLAG 1 OR PUT-FLAG ;
 10 : REC-LEN PAB-ADDR @ 4 + VSBW ;
 11 : CHAR-CNT! PAB-ADDR @ 5 + VSBW ;
 12 : CHAR-CNT@ PAB-ADDR @ 5 + VSBR ;
 13 : REC-NO DUP SWPB PAB-ADDR @ 6 + VSBW PAB-ADDR @ 7 + VSBW ;
 14 : N-LEN! PAB-ADDR @ 9 + VSBW ;
 15 R->BASE  -->

SCR #70
  0 ( FILE I/O ROUTINES 12JUL82 LCT) BASE->R HEX
  1 ( COMPILE A STRING WHICH IS MOVED TO VDP-ADDR AT EXECUTION)
  2
  3 : (F-D")
  4     PAB-ADDR @ 0A + R COUNT DUP 1+ =CELLS R> +
  5     >R >R SWAP R VMBW R> N-LEN! ;
  6 : F-D" 22 STATE @
  7     IF
  8       COMPILE (F-D") WORD HERE C@
  9       1+ =CELLS ALLOT
 10     ELSE
 11       PAB-ADDR @ 0A + SWAP WORD HERE COUNT >R SWAP R
 12       VMBW R> N-LEN!
 13     ENDIF ; IMMEDIATE
 14
 15 R->BASE  -->

SCR #71
  0 ( FILE I/O ROUTINES 12JUL82 LCT)
  1 BASE->R HEX
  2 : DOI/O CLR-STAT PAB-ADDR @ VSBW PAB-ADDR @ 9 + 8356 !
  3   0 837C C! DSRLNK CHK-STAT ;
  4 : OPN 0 DOI/O ;
  5 : CLSE 1 DOI/O ;
  6 : RD 2 DOI/O PAB-VBUF @ PAB-BUF @ CHAR-CNT@ VMBR CHAR-CNT@ ;
  7 : WRT >R PAB-BUF @ PAB-VBUF @ R VMBW R> CHAR-CNT! 3 DOI/O ;
  8 : RSTR REC-NO 4 DOI/O ;
  9 : LD REC-NO 5 DOI/O ;
 10 : SV REC-NO 6 DOI/O ;
 11 : DLT 7 DOI/O ;
 12 : SCRTCH REC-NO 8 DOI/O ;
 13 : STAT 9 DOI/O PAB-ADDR @ 8 + VSBR ;
 14
 15 R->BASE
```

```
SCR #72
  0 ( ALTERNATE I/O SUPPORT FOR RS232 PNTR 12JUL82 LCT)
  1 0 CLOAD INDEX     BASE->R DECIMAL 68 R->BASE CLOAD STAT
  2   0  0  0  FILE >RS232  BASE->R HEX
  3 : SWCH >RS232 PABS @ 10 + DUP PAB-ADDR ! 1- PAB-VBUF !
  4   SET-PAB OUTPT F-D" RS232.BA=9600"              OPN 3
  5   PAB-ADDR @ VSBW 1 PAB-ADDR @ 5 + VSBW  PAB-ADDR @ ALTOUT ! ;
  6 : UNSWCH 0 ALTOUT ! CLSE ;
  7 : ?ASCII ( BLOCK# --- FLAG )
  8        BLOCK 0 SWAP DUP 400 + SWAP
  9        DO I C@ 20 > + I C@ DUP 20 < SWAP 7F > OR
 10           IF DROP 0 LEAVE ENDIF LOOP  ;
 11 : TRIAD 0 SWAP SWCH 3 / 3 * DUP 3 + SWAP
 12   DO I ?ASCII IF 1+ I LIST CR ENDIF LOOP
 13   -DUP IF 3 SWAP - 14 * 0 DO CR LOOP
 14   0F MESSAGE 0C EMIT  ENDIF UNSWCH  ;
 15 R->BASE   -->

SCR #73
  0 ( SMART TRIADS AND INDEX 15SEP82 LAO ) BASE->R DECIMAL
  1 : TRIADS ( FROM TO --- )
  2   3 / 3 * 1 + SWAP 3 / 3 * DO I TRIAD 3 +LOOP ;
  3 : INDEX  ( FROM TO --- )  1+ SWAP
  4   DO  I DUP ?ASCII IF CR 4 .R 2 SPACES I BLOCK 64 TYPE ELSE DROP
  5      ENDIF PAUSE IF LEAVE ENDIF LOOP  ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15 R->BASE

SCR #74
  0 ( ASSEMBLER 12JUL82 LCT)
  1 FORTH DEFINITIONS
  2 0 CLOAD CODE
  3
  4 VOCABULARY ASSEMBLER IMMEDIATE
  5
  6 : CODE
  7     ?EXEC CREATE SMUDGE LATEST PFA DUP CFA !
  8     [COMPILE] ASSEMBLER    ;
  9
 10 : ;CODE
 11     ?CSP COMPILE (;CODE) SMUDGE
 12     [COMPILE] [  [COMPILE] ASSEMBLER     ; IMMEDIATE
 13
 14
 15
```

```
SCR #75
  0 ( ASSEMBLER 12JUL82 LCT)  0 CLOAD A$$M
  1 BASE->R DECIMAL 74 R->BASE CLOAD ;CODE
  2 BASE->R HEX
  3 ASSEMBLER DEFINITIONS
  4 : GOP' OVER DUP 1F > SWAP 30 < AND
  5        IF + , , ELSE + , ENDIF ;
  6 : GOP <BUILDS , DOES> @ GOP' ;
  7 0440 GOP B,     0680 GOP BL,     0400 GOP BLWP,
  8 04C0 GOP CLR,   0700 GOP SETO,   0540 GOP INV,
  9 0500 GOP NEG,   0740 GOP ABS,    06C0 GOP SWPB,
 10 0580 GOP INC,   05C0 GOP INCT,   0600 GOP DEC,
 11 0640 GOP DECT,  0480 GOP X,
 12 : GROP <BUILDS , DOES> @ SWAP 40 * + GOP' ;
 13 2000 GROP COC,  2400 GROP CZC,  2800 GROP XOR,
 14 3800 GROP MPY,  3C00 GROP DIV,  2C00 GROP XOP,
 15 -->

SCR #76
  0 ( ASSEMBLER 12JUL82 LCT)
  1 : GGOP <BUILDS ,
  2      DOES> @ SWAP DUP DUP 1F > SWAP 30 < AND
  3            IF 40 * + SWAP >R GOP' R> ,
  4            ELSE 40 * + GOP' ENDIF ;
  5 A000 GGOP A,    B000 GGOP AB,
  6 8000 GGOP C,    9000 GGOP CB,
  7 6000 GGOP S,    7000 GGOP SB,
  8 E000 GGOP SOC,  F000 GGOP SOCB,
  9 4000 GGOP SZC,  5000 GGOP SZCB,
 10 C000 GGOP MOV,  D000 GGOP MOVB,
 11
 12 : 0OP <BUILDS , DOES> @ , ;
 13 0340 0OP IDLE,  0360 0OP RSET, 03C0 0OP CKOF,
 14 03A0 0OP CKON,  03E0 0OP LREX, 0380 0OP RTWP,
 15 -->

SCR #77
  0 ( ASSEMBLER 12JUL82 LCT)
  1
  2 : ROP <BUILDS , DOES> @ + , ;
  3
  4 02C0 ROP STST,  02A0 ROP STWP,
  5
  6 : IOP <BUILDS , DOES> @ , , ;
  7
  8 02E0 IOP LWPI,  0300 IOP LIMI,
  9
 10 : RIOP <BUILDS , DOES> @ ROT + , , ;
 11
 12 0220 RIOP AI,   0240 RIOP ANDI,
 13 0280 RIOP CI,   0200 RIOP LI,
 14 0260 RIOP ORI,
 15 -->
```

```
SCR #78
  0 ( ASSEMBLER 12JUL82 LCT)
  1 : RCOP <BUILDS , DOES> @ SWAP 10 * + + , ;
  2 0A00 RCOP SLA,  0800 RCOP SRA,
  3 0B00 RCOP SRC,  0900 RCOP SRL,
  4 : DOP <BUILDS , DOES> @ SWAP 00FF AND OR , ;
  5 1300 DOP JEQ,   1500 DOP JGT,
  6 1B00 DOP JH,    1400 DOP JHE,
  7 1A00 DOP JL,    1200 DOP JLE,
  8 1100 DOP JLT,   1000 DOP JMP,
  9 1700 DOP JNC,   1600 DOP JNE,
 10 1900 DOP JNO,   1800 DOP JOC,
 11 1C00 DOP JOP,   1D00 DOP SBO,
 12 1E00 DOP SBZ,   1F00 DOP TB,
 13 : GCOP <BUILDS , DOES> @ SWAP 000F AND 040 * + GOP' ;
 14 3000 GCOP LDCR,  3400 GCOP STCR,
 15 -->

SCR #79
  0 ( ASSEMBLER 12JUL82 LCT)
  1 : @()    020   ;   : *?      010 +   ;
  2 : *?+    030 + ;   : @(?)    020 +   ;
  3 : W      0A    ;   : @(W)    W @(?)  ;
  4 : *W     W *?  ;   : *W+     W *?+   ;
  5 : RP     0E    ;   : @(RP)   RP @(?) ;
  6 : *RP    RP *? ;   : *RP+    RP *?+  ;
  7 : IP     0D    ;   : @(IP)   IP @(?) ;
  8 : *IP    IP *? ;   : *IP+    IP *?+  ;
  9 : SP     09    ;   : @(SP)   SP @(?) ;
 10 : *SP    SP *? ;   : *SP+    SP *?+  ;
 11 : UP     08    ;   : @(UP)   UP @(?) ;
 12 : *UP    UP *? ;   : *UP+    UP *?+  ;
 13 : NEXT   0F    ;   : *NEXT+  NEXT *?+ ;
 14 : *NEXT  NEXT *? ; : @(NEXT) NEXT @(?) ;
 15 -->

SCR #80
  0 ( ASSEMBLER 12JUL82 LCT)
  1 ( DEFINE JUMP TOKENS )
  2 : GTE  1 ;  : H   2 ;  : NE 3  ;
  3 : L    4 ;  : LTE 5 ;  : EQ 6  ;
  4 : OC   7 ;  : NC  8 ;  : OO 9  ;
  5 : HE  0A ;  : LE 0B ;  : NP 0C ;
  6 : LT  0D ;  : GT 0E ;  : NO 0F ;
  7 : OP  10 ;
  8 : CJMP    ?EXEC
  9      CASE LT OF 1101 , 0 ENDOF
 10           GT OF 1501 , 0 ENDOF
 11           NO OF 1901 , 0 ENDOF
 12           OP OF 1C01 , 0 ENDOF
 13           DUP 0< OVER 10 > OR IF 19 ERROR ENDIF DUP
 14      ENDCASE 100 * 1000 + , ;
 15 -->
```

```
SCR #81
  0 ( ASSEMBLER 12JUL82 LCT)
  1 : IF,      ?EXEC
  2     [COMPILE] CJMP HERE 2- 42 ; IMMEDIATE
  3 : ENDIF,   ?EXEC
  4     42 ?PAIRS HERE OVER - 2- 2 / SWAP 1+ C! ; IMMEDIATE
  5 : ELSE,    ?EXEC
  6     42 ?PAIRS 0 [COMPILE] CJMP HERE 2- SWAP 42 [COMPILE]
  7     ENDIF, 42 ; IMMEDIATE
  8 : BEGIN,   ?EXEC
  9     HERE 41 ; IMMEDIATE
 10 : UNTIL,   ?EXEC
 11     SWAP 41 ?PAIRS [COMPILE] CJMP HERE - 2 / 00FF AND
 12     HERE 1- C! ; IMMEDIATE
 13 : AGAIN,   ?EXEC
 14     0 [COMPILE] UNTIL, ; IMMEDIATE
 15 -->

SCR #82
  0 ( ASSEMBLER 12JUL82 LCT)
  1 : REPEAT,   ?EXEC
  2   >R >R [COMPILE] AGAIN, R> R> 2- [COMPILE] ENDIF,
  3   ; IMMEDIATE
  4 : WHILE,    ?EXEC
  5   [COMPILE] IF, 2+ ; IMMEDIATE
  6
  7
  8
  9
 10 : NEXT, *NEXT B, ;
 11
 12 FORTH DEFINITIONS
 13
 14 : A$$M ;             R->BASE
 15

SCR #83
  0 ( BSAVE -- BINARY SAVER FOR FORTH OVERLAYS     LCT 14SEP82 )
  1 0 CLOAD BSAVE    BASE->R DECIMAL
  2 : BSAVE ( from  scrn-no ---  )  FLUSH
  3     BEGIN
  4       SWAP >R DUP 1+ SWAP
  5       OFFSET @ + BUFFER UPDATE DUP B/BUF ERASE
  6       R OVER ! 2+ HERE OVER ! 2+
  7       CURRENT @ OVER ! 2+             LATEST      OVER ! 2+
  8       CONTEXT @ OVER ! 2+             CONTEXT @ @ OVER ! 2+
  9       VOC-LINK @ OVER ! 2 +   29801 OVER ! 10 +
 10       HERE R -
 11       R> DUP 1000 + >R SWAP >R SWAP R>
 12       1000 MIN CMOVE
 13       R SWAP HERE R> <
 14     UNTIL
 15     SWAP DROP FLUSH   ;  R->BASE
```

```
SCR #84
  0 ( NEW MESSAGE ROUTINE  13SEP82   LCT )    BASE->R DECIMAL
  1
  2 ( THIS VERSION OF MESSAGE HAS THE SCREEN 4 AND 5 MESSAGES
  3   INCLUDED IN THIS ROUTINE. )
  4
  5 FLUSH EMPTY-BUFFERS HERE LIMIT$ @ B/BUF 4 + - DUP LIMIT$ !
  6 DP ! ( PLACES message WHERE 5TH DISK BUF IS.  NOW HAVE 4 BUFS )
  7 : message
  8      WARNING @
  9      IF
 10        -DUP
 11        IF  ( NON-ZERO MESSAGE NUMBER )
 12          DUP 26 <
 13          IF  ( MESSAGE NEED NOT BE RETRIEVED FROM DISK )
 14            CASE  ( FOLLOWING CASES FOR MESSAGE NUMBERS )
 15 -->

SCR #85
  0 ( NEW MESSAGE CONTINUED )
  1 01 OF ." empty stack"                          ENDOF
  2 02 OF ." dictionary full"                      ENDOF
  3 03 OF ." has incorrect address mode"           ENDOF
  4 04 OF ." isn't unique."                        ENDOF
  5
  6 06 OF ." disk error"                           ENDOF
  7 07 OF ." full stack"                           ENDOF
  8
  9 09 OF ." file i/o error"                       ENDOF
 10 10 OF ." floating point error"                 ENDOF
 11 11 OF ." disk fence violation"                 ENDOF
 12 12 OF ." can't load from screen zero"          ENDOF
 13
 14
 15 15 OF ." TI FORTH --- a fig-FORTH extension"  ENDOF  -->

SCR #86
  0 ( NEW MESSAGE CONTINUED )
  1 17 OF ." compilation only, use in definition" ENDOF
  2 18 OF ." execution only"                       ENDOF
  3 19 OF ." conditionals not paired"              ENDOF
  4 20 OF ." definition not finished"              ENDOF
  5 21 OF ." in protected dictionary"              ENDOF
  6 22 OF ." use only when loading"                ENDOF
  7
  8 24 OF ." declare vocabulary"                   ENDOF
  9 25 OF ." bad jump token"                       ENDOF
 10
 11 ENDCASE
 12
 13 -->
 14
 15
```

```
SCR #87
  0 ( NEW MESSAGE CONTINUED )
  1
  2          ELSE
  3            4 OFFSET @ B/SCR / - .LINE
  4          ENDIF
  5        ENDIF
  6      ELSE
  7        ." MSG # " .
  8      ENDIF
  9  ;
 10
 11 DP !  ( RESTORE DP TO POSITION PRIOR TO message )
 12 ( INSTALL NEW MESSAGE )
 13 ' BRANCH CFA     ' MESSAGE
 14 ' message OVER - 2- OVER 2+ ! !
 15 R->BASE

SCR #88
  0 ( CRU WORDS   12OCT82 LAO )   0 CLOAD STCR
  1 BASE->R DECIMAL 74 R->BASE CLOAD ;CODE
  2 BASE->R  HEX
  3 CODE SBO  C339 , A30C , 1D00 , 045F ,
  4 CODE SBZ  C339 , A30C , 1E00 , 045F ,
  5 CODE TB   C319 , A30C , 04D9 , 1F00 , 1601 , 0599 , 045F ,
  6
  7 CODE LDCR C339 , A30C , C079 , C039 , 0241 , 000F , 1304 ,
  8           0281 , 0008 , 1501 , 06C0 , 0A61 , 0261 , 3000 ,
  9           0481 , 045F ,
 10
 11 CODE STCR C339 , A30C , C059 , 04C0 , 0241 , 000F , C081 ,
 12           0A61 , 0261 , 3400 , 0481 , C082 , 1304 , 0282 ,
 13           0008 , 1501 , 06C0 , C640 , 045F ,
 14
 15 R->BASE
```

Appendix J TI Forth Bugs

TI Forth Bugs found as of the May, 1985 issue of HOCUS (Milwaukee TI Users Group):

- Jeff Stanford—

  Screen 22, Line 5:

  ```
  BASE->R HEX ( 3800 ' SATR ! )
  ```

  Screen 23, Line 2:

  ```
  : CINIT 3800 DUP ' SPDTAB ! 800 / 6 VWTR 3800 ' SATR !
  ```

  Screen 28, Line 1

  ```
  : EDT VDPMDE @ 5 =  0= IF SPLIT ENDIF CINIT !CUR R/C CGOTOXY
  ```

  Screen 28, Line 11:

  ```
  0F OF 5 0 SPRPAT CLS SCRNO DROP 300 ' SATR ! QUIT ENDOF
  ```

- Tom Freeman—

  Screens 53 – 55, Line 1 on each screen:

  Change **VDPSET2** to **SETVDP2**

  Screen 58:

  Switch Lines 9 & 10

  In new Line 9, change **300 ! SATR** to **300 ' SATR**

  Screen 59, Line 9, between **SWAP** and **AND**

  Change **00FF** to **00FE**

- Everybody & his brother—

  Screen 72 Line 5:

  PAB_ADDR to **PAB-ADDR**

- Jim Vincent—

  Original Manual, Chapter 6, Page 10, Line 1 (see this manual, footnote 10, page 37):

  ```
  HEX 3800 ' SATR !
  ```

  Original Manual, Chapter 10, Page 3, Line 20 (see this manual, footnote 17, page 73):

  ```
  : DOWN -100 ALLOT DROP ;
  ```

TI Forth Bugs found more recently by the Editor:

- 64-column editor—

  Screen 22, Line 12:

  The definition of **BCK** is not used for this editor, so it can safely be removed. This is not really a bug, but it takes up unnecessary space in the dictionary.

- In the 40-column editor, the cursor is changed to a box shape upon entry, but is not restored upon exit. The following changes will save and restore the old cursor—

  Screen 34, Lines 3 – 7:

Consolidate Lines 6 & 7 to Line 7, Lines 4 & 5 to Line 6 and move Line 3 to Line 5 to make room for **OLDCUR** , **GETCUR** and **PUTCUR** on Lines 3 & 4:

```
0 VARIABLE OLDCUR 6 ALLOT
: GETCUR 8F0 OLDCUR 8 VMBR ;  : PUTCUR OLDCUR 8F0 8 VMBW ;
```

Add to the definition of **BOX** after **LOOP** in Line 3:

```
0FC 8F0 VSBW 0FC 8F7 VSBW
```

Screen 35, Line 10:

Insert **PUTCUR** between **GOTOXY** and **QUIT** .

Screen 38, Line 1:

Insert **GETCUR** between **VED** and **BOX** .

- In the Floating Point Routines, use of the definition of **VAL** in bitmap mode corrupts the screen because **VAL** uses the TI Forth disk buffer explicitly (**1000h**) instead of via the user variable **DISK_BUF** , which moves the buffer out of the way in bitmap mode—

  Screen 47, Line 13:

  Move the last two words, **SWAP VSBW** , to the beginning of the next line to make room. Change **1000** to the two words, **DISK_BUF @** .

- In the Graphics Primitives load screen (57), the load should be conditional on finding **LINE** , the last word defined, rather than **CHAR** , the first word defined, to maintain consistency with the instructions in § 11.2 —

  Screen 57, Line 0:

  Change **CHAR** to **LINE** .

- Compact List—

  Screen 65, Line 4:

  BADDR and **INDX** are defined but never used. They can probably be safely be removed; but, because the editor has not analyzed what exactly **SMASH** is doing and because it is just possible they are there to act as a buffer ahead of **LB** , we'll leave them alone for now.

- Bad definition of **;CODE** —

  Screen 74, Line 12:

  The definition of **;CODE** needs to be made immediate by adding the word **IMMEDIATE** after the definition-ending **;** . Be sure to leave an intervening space.

See also the corrections and clarifications in the Editor's Notes in Appendix G .

Appendix K Diskette Format Details

The information in this section is based on TI's *Software Specifications for the 99/4 Disk Peripheral (March 28, 1983)*.

The original disk drives supplied by TI supported only single-sided, single-density (SSSD), 90-KB diskettes. The TI Forth system was designed around and supplied in this disk format. Though different formats are possible, we will consider the usual format of 256 bytes per sector and 40 tracks per side. The following table shows possible formats with 256 bytes/sector and 40 tracks/side:

Disk Type	Sides	Density	Sectors/ Track	Total Sectors	Capacity
SSSD	1	single	9	360	90 KB
DSSD	2	single	9	720	180 KB
SSDD	1	double	18	720	180 KB
DSDD	2	double	18	1440	360 KB
Compact Flash[19]	2	double	20	1600[20]	400 KB

The information in the following sections accrues to all the above formats:

K.1 Volume Information Block (VIB)

Byte #	1st Byte	2nd Byte	Byte #
0	Disk Volume Name (10 characters padded on the right with blanks)		1
8			9
10	Total Number of Sectors		11
12	Sectors/Track	"D"	13
14	"S"	"K"	15
16	Protection ("P" or " ")	Tracks/Side	17
18	# of Sides	Density	19
20	Reserved		21
54			55
56	Allocation Bit Map (room for 1600 sectors)		57
254			255

Sector 0 contains the volume information block (VIB). The layout is shown in the above table.

19 This is a third-party peripheral expansion device with 400 KB virtual disks using Compact Flash memory on devices named nanoPEB and CF7+ (see website: *http://webpages.charter.net/nanopeb/*)

20 1600 sectors is the maximum possible number of sectors that can be managed by the current specification.

K.2 File Descriptor Index Record (FDIR)

Sector 1 contains the file descriptor index record (FDIR). It can hold up to 127 2-byte entries, each pointing to a file descriptor record (FDR—see next section). These pointers are alphabetically sorted by the file names to which they point. This list of pointers starts at the beginning of sector 1 and ends with a pointer value of 0.

K.3 File Descriptor Record (FDR)

Byte #	1st Byte	2nd Byte	Byte #
0	File Name (10 characters padded on the right with blanks)		1
8			9
10	Reserved		11
12	File Status Flags	# of Records/Sector (0 for program)	13
14	# of Sectors currently allocated (not counting this FDR)		15
16	EOF Offset (bytes in last Sector)	Bytes/Record	17
18	# of Records (Fixed) or # of Sectors (Variable)—bytes are in reverse order		19
20	Reserved		21
26			27
28	Data Chain Pointer Blocks (3 bytes/block encoding two 12-bit numbers that indicate cluster start and highest, cumulative sector offset)		29
254			255

There can be as many as 127 file descriptor records (FDRs) laid out as in the above table. There are no subdirectories. FDRs will start in sector 2 and continue, at least, until sector 33, unless a file allocation requires more space than is available in sectors 34 – end-of-disk, in which case the system will begin allocating space for the file in the first available sector in sectors 3 – 33. This is done "to obtain faster directory search response times"[21]. Each FDR beyond 32 files will be placed in the first available sector.

Byte 12 contains file status flags defined as follows, with bit 0 as the least significant bit:

Bit #	Description
0	Program or Data file (0 = Data; 1 = Program)
1	Binary or ASCII data (0 = ASCII, DISPLAY file; 1 = Binary, INTERNAL or program file)
2	Reserved
3	PROTECT flag (0 = not protected; 1 = protected)
4–6	Reserved
7	FIXED/VARIABLE flag (0 = fixed-length records; 1 = variable-length records)

21 *Software Specifications for the 99/4 Disk Peripheral (March 28, 1983)*, p. 19.

The cluster blocks listed in bytes 28 – 255 of the FDR each contain 2 12-bit (3-nybble[22]) numbers. The first points to the beginning sector of that cluster of contiguous sectors and the second is the sector offset reached by that cluster. If we label the 3 nybbles of the cluster pointer as $n_1 - n_3$ and the 3 nybbles of the cumulative sector offset as $m_1 - m_3$, with the subscripts indicating the significance of the nybble, then the 3 bytes are laid out as follows:

Byte 1: n_2n_1 Byte 2: m_1n_3 Byte 3: m_3m_2

The actual 12-bit numbers, then, are

Cluster Pointer: $n_3n_2n_1$ Sector Offset: $m_3m_2m_1$

For example, the following represents 2 blocks in the FDR for a file with 2 clusters allocated:

Actual layout in the FDR: **`4D20h 5F05h F060h`**

1st Cluster Pointer: **`04Dh`** (77_{10})[23] Record Offset: **`5F2h`** (1522_{10})

2nd Cluster Pointer: **`005h`** (5_{10}) Record Offset: **`60Fh`** (1551_{10})

The above example represents a file, the data for which occupies 1552 sectors on the disk. If we assume that no files have been deleted in this case, you should also be able to deduce that there are only 3 files on the disk because the second cluster starts in sector 5 and occupies all sectors from 5 – 33, which should tell you there are 3 FDRs before this cluster was allocated: Sector 0 (VIB), sector 1 (FDIR), sector 2 (FDR of first file), sector 3 (FDR of second file), sector 4 (FDR of third file and sector 5 (second cluster start of the third file, the first two occupying sectors 34 – 76 by inference). Furthermore, the disk contains 1600 sectors because that is the maximum and the first cluster ended in the 1600th sector of the disk (1st cluster starts in sector 77 and ends 1522 sectors later in sector 1599).[24]

K.4 Comparison of TI Forth and TI File System Layouts on the Same Disk

The TI file system layout has been detailed earlier in this appendix. The TI Forth system is based on 1-KB blocks or screens that each consist of 16 lines of 64 characters. TI Forth screens start in the first sector of the disk and contiguously occupy the entire disk with each screen consuming 4 contiguous sectors ($4 \cdot 256 = 1024$ bytes/screen). The TI Forth system reads and writes screens using direct sector access, thus making it possible to easily destroy the normal file-system layout of the system disk and, less so, work disks that have been set up by **`DISK-HEAD`** or the method of § L.2 if you are not careful. The sections that follow show the two layouts side by side to make it easier to understand the relationship of the 128-byte file records with where they appear on TI Forth screens.

22 A nybble (also nibble) is half of one byte (8 bits) and is equal to 4 bits. The editor prefers "nybble" to "nibble" because of its obvious relationship to "byte". 2 nybbles = 1 byte.

23 The subscript, 10, indicates base 10 (decimal).

24 This example is taken from one of my (Lee Stewart's) Compact Flash volumes.

K.4.1 TI Forth System Disk

The TI Forth system disk of the original 90-KB disk is shown in the following table *[Note: In this table and the next, a row with white on black is the beginning of a file.]*:

Sector	Record Bytes	Record	File Name	Contents (Characters as DOS/IBM ASCII)	Forth Screen	Line
0	256	0	{VIB}	TI/FORTH ☺h○DSK	0	0
:	:	:	:	:	:	:
1	256	0	{FDIR}	.☻.♥.♦.....	0	4
:	:	:	:	:	:	:
2	256	0	{FDR} FORTH	FORTH ...♥.♣.P♪....."@.....	0	8
:	:	:	:	:	:	:
3	256	0	{FDR} FORTHSAVE	FORTHSAVE ..☺..&(.....'P☻.....	0	12
:	:	:	:	:	:	:
4	256	0	{FDR} SYS-SCRNS	SYS-SCRNS ...☻☺8.Çp☻.....Má◄♣p!!.....	1	0
:	:	:	:	:	:	:
5	128	566	SYS-SCRNS	.....	1	4
:	:	:	:	:	:	:
8	128	572	SYS-SCRNS	T I F O R T H	2	0
			SYS-SCRNS		2	1
	128	573	SYS-SCRNS	THIS VERSION OF THE FORTH LANGUAGE	2	2
			SYS-SCRNS	IS BASED ON THE fig-FORTH MODEL	2	3
9	128	574	SYS-SCRNS		2	4
			SYS-SCRNS	THE ADDRESS OF THE FORTH INTEREST GROUP IS:	2	5
	128	575	SYS-SCRNS		2	6
			SYS-SCRNS	FORTH INTEREST GROUP	2	7
10	128	576	SYS-SCRNS	P.O. BOX 1105	2	8
			SYS-SCRNS	SAN CARLOS, CA 94070	2	9
	128	577	SYS-SCRNS		2	10
			SYS-SCRNS	TEXAS INSTRUMENTS PERSONNEL WITH SIGNIFICANT	2	11
11	128	578	SYS-SCRNS	INPUT TO THIS VERSION INCLUDE:	2	12
			SYS-SCRNS	LEON TIETZ	2	13
	128	579	SYS-SCRNS	LESLIE O'HAGAN	2	14
			SYS-SCRNS	EDWARD E. FERGUSON	2	15
12	128	580	SYS-SCRNS	(WELCOME SCREEN) 0 0 GOTOXY ." BOOTING..." CR	3	0
			SYS-SCRNS	BASE->R HEX 10 83C2 C! (QUIT OFF!)	3	1
	128	581	SYS-SCRNS	DECIMAL (84 LOAD) 20 LOAD 16 SYSTEM MENU	3	2
			SYS-SCRNS	HEX 68 USER VDPMDE 1 VDPMDE ! DECIMAL	3	3
13	128	582	SYS-SCRNS	: -SYNONYMS 33 LOAD ; : -EDITOR 34 LOAD ; : -COPY 39 LOAD ;	3	4
			SYS-SCRNS	: -DUMP 42 LOAD ; : -TRACE 44 LOAD ; : -FLOAT 45 LOAD ;	3	5
	128	583	SYS-SCRNS	: -TEXT 51 LOAD ; : -GRAPH1 52 LOAD ; : -MULTI 53 LOAD ;	3	6
			SYS-SCRNS	: -GRAPH2 54 LOAD ; : -SPLIT 55 LOAD ; : -GRAPH 57 LOAD ;	3	7
14	128	584	SYS-SCRNS	: -FILE 68 LOAD ; : -PRINT 72 LOAD ; : -CODE 74 LOAD ;	3	8

Sector	Record Bytes	Record	File Name	Contents (Characters as DOS/IBM ASCII)	Forth Screen	Line
			SYS-SCRNS	`: -ASSEMBLER 75 LOAD ;  : -64SUPPORT 22 LOAD ;`	3	9
	128	585	SYS-SCRNS	`: -VDPMODES  -TEXT -GRAPH1 -MULTI -GRAPH2 -SPLIT ;`	3	10
			SYS-SCRNS	`: -BSAVE     83 LOAD ;  : -CRU    88 LOAD ;`	3	11
15	128	586	SYS-SCRNS		3	12
			SYS-SCRNS		3	13
	128	587	SYS-SCRNS		3	14
			SYS-SCRNS	`R->BASE`	3	15
:	:	:	:	:	:	:
33	128	622	SYS-SCRNS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	4
				σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	5
	128	623		σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	6
				σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	7
34			**FORTH**	**`...BOOT    A..B..B..B..B%(B..BDSBK1B.FBORBTHBSABVE9..B.IB.JB..B.`**	**8**	**8**
:	:	:	:	:	:	:
39			**FORTHSAVE**	**`..42............... .....`...@... :"...:. <..N....4`...6.`.<4b4d`**	**9**	**12**
:	:	:	:	:	:	:
77	**128**	**0**	**SYS-SCRNS**	**σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ**	**19**	**4**
			SYS-SCRNS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	19	5
:	:	:	:	:	:	:
81	128	8	SYS-SCRNS	`( CONDITIONAL LOAD  )`	20	0
			SYS-SCRNS	`: MENU CR 272 265 DO I MESSAGE CR LOOP CR CR CR ;`	20	1
	128	9	SYS-SCRNS	`: SLIT ( --- ADDR OF STRING LITERAL )`	20	2
			SYS-SCRNS	`    R> DUP C@ 1+ =CELLS OVER + >R ;`	20	3
82	128	10	SYS-SCRNS		20	4
			SYS-SCRNS	`: WLITERAL ( WLITERAL word )`	20	5
	128	11	SYS-SCRNS	`    BL STATE @`	20	6
			SYS-SCRNS	`    IF COMPILE SLIT WORD HERE C@ 1+ =CELLS ALLOT`	20	7
83	128	12	SYS-SCRNS	`    ELSE WORD HERE ENDIF ; IMMEDIATE  -->`	20	8
			SYS-SCRNS	`-SYNONYMS   -EDITOR     -COPY`	20	9
	128	13	SYS-SCRNS	`-DUMP       -TRACE      -FLOAT`	20	10
			SYS-SCRNS	`-TEXT       -GRAPH1     -MULTI`	20	11
84	128	14	SYS-SCRNS	`-GRAPH2     -SPLIT      -VDPMODES`	20	12
			SYS-SCRNS	`-GRAPH      -FILE       -PRINT`	20	13
	128	15	SYS-SCRNS	`-CODE       -ASSEMBLER  -64SUPPORT`	20	14
			SYS-SCRNS	`-BSAVE      -CRU`	20	15
:	:	:	:	:	:	:
359	128	564	SYS-SCRNS		89	12
			SYS-SCRNS		89	13
	128	565	SYS-SCRNS		89	14
			SYS-SCRNS		89	15

K.4.2 TI Forth Work Disk

The TI Forth original, 90-KB format disk written by **DISK-HEAD** is shown in the following table:

Sector	Record Bytes	Record	File Name	Contents (Characters as DOS/IBM ASCII)	Forth Screen	Line
0	256	0	{VIB}	`FORTH     ☺h○DSK .......................................`	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	256	0	{FDIR}	`.☻..............................................................`	0	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2	256	0	{FDR} SCREENS	`SCREENS   ...☻☺e.P╩☻........"á(♥Ç,..............................`	0	8
⋮	⋮	⋮	⋮	⋮	⋮	⋮
3	128	652	SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	0	12
			SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	0	13
⋮	⋮	⋮	⋮	⋮	⋮	⋮
33	128	712	SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	4
			SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	5
	128	713	SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	6
			SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	7
34	**128**	**0**	**SCREENS**	**σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ**	**8**	**8**
			SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	9
	128	1	SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	10
			SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	8	11
⋮	⋮	⋮	⋮	⋮	⋮	⋮
359	128	650	SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	89	12
	128		SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	89	13
	128	651	SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	89	14
	128		SCREENS	σσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσσ	89	15

Appendix L TI Forth System for Larger Disks

Most users of TI Forth these days are using disk sizes that are larger than the original 90 KB disks on which TI supplied TI Forth to TI-99/4A users groups at the end of 1983. This appendix will show you how to put TI Forth on a larger system disk and how to create larger, non-system TI Forth work disks.

L.1 Larger System Disk

With the following procedure, you can make a TI Forth system disk in a larger disk format:

1. Make a backup copy of the original system disk and use that below where "original system disk" is indicated.
2. Format a disk with Disk Manager or a third-party disk manager to the desired size.
3. With the same disk manager program, copy "FORTH" from the original system disk to the newly formatted disk.
4. Repeat (3) for "FORTHSAVE".
5. Repeat (3) for "SYS-SCRNS".
6. Put the following screen on an available blank screen on the original system disk (screens 30 – 32 should be available) and load it:

```
( Swell TI FORTH SYS-SCRNS file to fill disk       07SEP11 LES)
BASE->R : LSYS ; DECIMAL 68 CLOAD STAT 33 CLOAD RANDOMIZE
HEX 0 VARIABLE LESBUF 7E ALLOT 0 VARIABLE LASTREC
PABS @ A + LESBUF 1700 FILE SCRFIL
: FORTHSYS ( size_KB drive_no --- )
    SCRFIL SET-PAB RLTV DSPLY 80 REC-LEN
    F-D" DSK .SYS-SCRNS" ( filename to PAB--space for drive#)
    31 + PAB-ADDR @ D + VSBW ( drive# + 1 to ASCII & put in PAB)
    OPN LESBUF 80 BLANKS ( open file & blank fill buffer)
    4 * 30 - 2 * 1- LASTREC ! LASTREC @ REC-NO ( Set last rec#)
    80 WRT                    ( Write last record)
    25C 23C DO                ( Restore screens 2-5)
      I REC-NO RD LASTREC @ I + 26F - REC-NO WRT LOOP
    CLSE ;                    ( Close file) R->BASE
```

7. Type the size in KB of the new system disk, the zero-based drive number and the word **FORTHSYS** . If your new system disk is 360 KB and the drive number is 1 (DSK2), type the following on the keyboard:

```
360 1 FORTHSYS
```

To accommodate your larger disk, you now need to add to line 12 of Forth screen 3:

```
360 DISK_SIZE !
```

Depending on whether you use 2 or 3 disk drives, you might also want to follow that with:

720 DISK_HI ! or **1080 DISK_HI !**

When you are done using **FORTHSYS** , you can get rid of its part of the dictionary by executing

```
FORGET LSYS
```

L.2 Larger Work Disk

With the procedure delineated below (an alternative to **DISK-HEAD**), you can make a TI Forth work disk in a larger disk format. If you study it, you will see at a higher level what it is that **DISK-HEAD** is actually doing at a lower level. An updated, more general-purpose **DISK-HEAD** (as **DSK-HD**) follows in § L.3 .

1. Format a disk with Disk Manager or a third-party disk manager to the desired size.
2. Put the following screen on an available blank screen on your new system disk (there are now plenty of empty screens beyond screen 89) and load it:

```
( Create TI FORTH work disk larger than 90 KB      07SEP11 LES)
BASE->R DECIMAL : LWRK ;  68 CLOAD STAT 33 CLOAD RANDOMIZE
39 CLOAD DISK-HEAD
HEX 0 VARIABLE LESBUF 7E ALLOT
PABS @ A + LESBUF 1700 FILE SCRFIL
: FORTHWRK ( size_KB drive_no --- ) OVER OVER
    SCRFIL SET-PAB RLTV DSPLY 80 REC-LEN
    F-D" DSK .SCREENS" ( filename to PAB--space for drive#)
    31 + PAB-ADDR @ D + VSBW ( drive# + 1 to ASCII & put in PAB)
    OPN LESBUF 80 BLANKS ( open file & blank fill buffer)
    4 * 3 - 2 * 1- REC-NO ( Calculate last record # & set it)
    80 WRT CLSE           ( Write last record & close file)
    * BLOCK !" FORTH     " UPDATE FLUSH ( write disk name->VIB)
;  R->BASE
```

3. Ensure that **DISK_SIZE** and **DISK_HI** are properly set before executing **FORTHWRK** .
4. If the work disk is in drive 0, be sure to set **DISK_LO** to 0 before executing **FORTHWRK** .
5. Type the size in KB of the new work disk, the zero-based drive number and the word **FORTHWRK** . If your new work disk is 360 KB and the drive number is 1 (DSK2), type the following on the keyboard:

```
360 1 FORTHWRK
```

When you are done using **FORTHWRK** , you can get rid of its part of the dictionary by executing

```
FORGET  LWRK
```

L.3 Updating Disk Utilities for Larger Disks

With disks larger than 90 KB, you will need either to update several disk utility words on the system disk or avoid using them with any but 90 KB disks. These words are

Word	Screen	Lines
DTEST	39	8
FORTH-COPY	39	14
DISK-HEAD	40	0 – 15
FORMAT-DISK	33	7

The above words are redefined in this section to remove hardwired disk sizes from the definitions. The words may then be used for any size disk. You can replace the original definitions of these words on the Forth system screens indicated above except for **FORMAT-DISK** , which will require a bit more work. *Always remember to keep backups (more than one!) of the original disks!!* Please note, also, that the stack effects are the same for only two of these redefined words (**DTEST** , **FORTH-COPY**) as for the originals. The other two words (**DSK-HD** , **FMT-DSK**) are actually renamed from the originals because their stack effects are different. If you wish to also use the original names with the new stack effects, simply uncomment the definitions that follow each new definition. If you do uncomment the definition of **FORMAT-DISK** , be sure to remove the original definition on screen 33 and, perhaps, add a conditional load for the screen where you put **FMT-DSK** on the bottom line of screen 33. With **FMT-DSK** on screen 100, the new bottom line for screen 33 would be

```
DECIMAL 100 CLOAD FMT-DSK     R->BASE
```

You should probably have a definition for **DISK-HEAD** on screen 40 because other system screens use it for conditional loads. Either uncomment the definition after **DSK-HD** , make it a null definition (**: DISK-HEAD ;**) or just bite the bullet by changing the name of **DSK-HD** to **DISK-HEAD** in the new definition and try your best to remember the new stack effects.

Before executing any of these words, be sure that **DISK_SIZE** , **DISK_LO** and **DISK_HI** are properly set.

DTEST (---)

```
: DTEST DISK_SIZE @ 0 DO I DUP . BLOCK DROP LOOP ;
```

FORTH-COPY (---)

```
: FORTH-COPY DISK_SIZE @ 0 DO I DUP . DISK_SIZE @ + I
     SCOPY LOOP ;
```

```
DSK-HD      ( drive sides density --- )

      ( WRITE A HEAD COMPATABLE WITH THE DISK MANAGER    07SEP11 LES)
      BASE->R HEX : DSK-HD SWAP SWPB + >R DISK_SIZE * DUP
                    CLEAR BLOCK ( START SECTOR 0)
        DUP !" FORTH     " DUP A +  DISK_SIZE @ 4 * SWAP !
        DUP C + 944 SWAP ! DUP E + 534B SWAP ! DUP 10 + 2028 SWAP !
        DUP 12 + R> SWAP ! DUP 14 + 24 0 FILL DUP 38 + C8 FF FILL
        100 + ( START SECTOR 1) DUP 2 SWAP ! DUP 2+ FE 00 FILL
        100 + ( START SECTOR 2) DUP !" SCREENS   " DUP A + 0 SWAP !
        DUP C + 2 SWAP ! DUP E + DISK_SIZE @ 4 * 3 - DUP >R SWAP !
        DUP 10 + 80 SWAP ! DUP 12 + R> 2 * SWPB SWAP !
        DUP 14 + 8 0 FILL >R 22 R 1C + C! DISK_SIZE @ 4 * 1- DUP 34 -
        DUP F AND 4 SLA R 1D + C! 4 SRA R 1E + C!
        03 R 1F + C! DUP 3 - F AND 4 SLA R 20 + C! 4 SRA R 21 + C!
        R> 22 + 0DE 0 FILL UPDATE FLUSH
      ; ( : DISK-HEAD DSK-HD ; )             R->BASE
```

Be advised that **DSK-HD** *does* **CLEAR** sectors 0 – 3 (screen 0 if disk is in drive 0) of the disk as does the original **DISK-HEAD** . Also, note that unlike **DISK-HEAD** , **DSK-HD** requires three numbers on the stack, *viz*., the drive number, the number of sides (1 or 2) and the density (1 or 2). To invoke it for a DSDD disk in drive 1 (DSK2), you would type

```
1 2 2 DSK-HD
```

```
FMT-DSK          ( drive sides density --- sectors )

      ( Format Disk, given drive #, sides & density  07SEP11 LES)
      BASE->R DECIMAL 33 CLOAD RANDOMIZE 0 CLOAD FMT-DSK HEX
      : FMT-DSK ( drive sides density --- sectors )
          1 PABS @ VSBW 11 PABS @ 1+ VSBW ( subroutine 11h)
          8350 C! ( density)
          8351 C! ( sides)
          1+ 834C C! ( drive)
          28 834D C! ( 40 tracks)
          DISK_BUF @ 834E ! ( VDP buffer)
          PABS @ 8356 ! 0A 0E SYSTEM ( call DSRLNK subroutine)
          834A @ ( leave sectors formatted)
      ;          ( : FORMAT-DISK FMT-DSK ; )        R->BASE
```

This disk-formatting word requires on the stack the drive number, number of sides and density of the disk to be formatted. To format a 360-KB DSDD diskette in drive 2 (DSK3), you would type

```
2 2 2 FMT-DSK
```

The following two lines will each format a 90-KB SSSD diskette:

```
2 1 1 FMT-DSK
```

```
2 FORMAT-DISK
```
(if you keep the original definition)

If you were to store the above definition of **FMT-DSK** on screen 100 and would like it to load when the system-synonyms screen loads (the screen with **FORMAT-DISK** on it), then replace line 15 on screen 33 with

```
DECIMAL 100 CLOAD FMT-DSK R->BASE
```

You might also want to consider deleting the definition of **FORMAT-DISK** from screen 33 because you will not need it with the above word. It is probably not a good idea to rename the new definition **FORMAT-DISK** because its stack effects are so different from the old definition and could be confusing.

As with **FORMAT-DISK** , **FMT-DSK** creates a disk that can only be used by TI Forth. There is no information written to sectors 0 and 1 that will allow file-access words to work with the disk. If you run **DSK-HD** after **FMT-DSK** , you can then use the disk for file access from TI Forth, TI BASIC, etc. A clunky way to create a blank disk would be to delete the file "SCREENS" from the disk after running **DSK-HD** by using file-access words described in Chapter 8. You can *(carefully!)* change the name of the disk by editing the first 10 bytes of Forth screen 0 for a disk in drive 0 and after setting **DISK_LO** to 0. Just remember to use names of no more than 10 characters that contain no spaces or periods. Spaces *should* be used *after* the name to fill out the 10 characters in this field.

Appendix M Notes on Radix-100 Notation

TI Forth floating-point math routines use radix-100 format for floating-point numbers. The term "radix" is used in mathematics to mean "number base". We will use "radix 100" to describe the base-100 or centimal number system and "radix 10" to describe the base-10 or decimal number system. Radix-100 format is the same format used by the XML and GPL routines in the TI-99/4A console. Each floating-point number is stored in 8 bytes (4 cells) with a sign bit, a 7-bit, excess-64 (64-biased) integer exponent of the radix (100) and a normalized, 7-digit (1 radix-100 digit/byte) significand for a total of 8 bytes per floating point number. The signed, radix-100 exponent can be -64 to +63. (Keep in mind that the exponent is for radix-100 notation. Those same exponents radix 10 would be -128 to +126.) The exponent is stored in the most significant byte (MSB) biased by 64, *i.e.*, 64 is added to the actual exponent prior to storing, *i.e.*, -64 to +63 is stored as 0 to 127.

The significand (significant digits of the number) must be normalized, *i.e.*, if the number being represented is not zero, the MSB of the significand must always contain the first non-zero (significant) radix-100 digit, with the radix exponent of such a value that the radix point immediately follows the first digit. This is essentially scientific notation for radix 100. Each byte contains one radix-100 digit of the number, which, of course, means that each byte can have a value from 0 to 99 (`0` to `63h`) except for the first byte of a non-zero number, which must be 1 to 99. It is easy to view a radix-100 number as a radix-10 number by representing the radix-100 digits as pairs of radix-10 digits because radix 100 is the square of radix 10. In the following list of largest and smallest possible 8-byte floating point numbers, the radix-100 representation is on the left with spaces between pairs of radix-100 digits. The radix-16 (hexadecimal) internal representation of each byte of the number is also shown:

- Largest positive floating point number [hexadecimal: `7F 63 63 63 63 63 63 63`]:

  $$99\ .\ 99\ 99\ 99\ 99\ 99\ 99 \times 100^{63} = 99.999999999999 \times 10^{126}$$
  $$= 9.9999999999999 \times 10^{127}$$

- Largest negative floating point number [hexadecimal: `80 9D 63 63 63 63 63 63`]:

  $$-99\ .\ 99\ 99\ 99\ 99\ 99\ 99 \times 100^{63} = -99.999999999999 \times 10^{126}$$
  $$= -9.9999999999999 \times 10^{127}$$

- Smallest positive floating point number [hexadecimal: `00 01 00 00 00 00 00 00`]:

  $$01\ .\ 00\ 00\ 00\ 00\ 00\ 00 \times 100^{-64} = 1.000000000000 \times 10^{-128}$$

- Smallest negative floating point number [hexadecimal: `FF FF 00 00 00 00 00 00`]:

  $$-01\ .\ 00\ 00\ 00\ 00\ 00\ 00 \times 100^{-64} = -1.000000000000 \times 10^{-128}$$

The only difference in the internal storage of positive and negative floating point numbers is that only the first word (2 bytes) of negative numbers is negated or complemented (two's complement).

A floating point zero is represented by zeroing only the first word. The remainder of the floating point number does not need to be zeroed for the number to be treated as zero for all floating point calculations.

Appendix N Adding True Lowercase Character Sets

This appendix explains how to add true lowercase character sets to text, graphics and bitmap modes of TI Forth.

N.1 True Lowercase for Text and Graphics Modes

The following graphic shows the true lowercase character set the editor designed for text and graphics modes:

We will store the character codes on system screen 41 (**29h**) to be loaded by code we will modify on system screens 55 and 56. The screens that follow should be loaded to store the character codes on screen 41 and **FORGET** the definition of **TRUE_LC** , which is no longer needed:

```
SCR #100
  0 ( True lowercase characters for TEXT mode)
  1 BASE->R HEX   0000 VARIABLE TRUE_LC
  2        2010 , 0800 , 0000 , ( `)
  3 0000 , 0038 , 043C , 443C , ( a)
  4 0040 , 4078 , 4444 , 4478 , ( b)
  5 0000 , 003C , 4040 , 403C , ( c)
  6 0004 , 043C , 4444 , 443C , ( d)
  7 0000 , 0038 , 447C , 4038 , ( e)
  8 0018 , 2420 , 7020 , 2020 , ( f)
  9 0000 , 003C , 443C , 0438 , ( g)
 10 0040 , 4058 , 6444 , 4444 , ( h)
 11 0010 , 0030 , 1010 , 107C , ( i)
 12 0004 , 0004 , 0404 , 4438 , ( j)
 13 0040 , 4044 , 4870 , 4844 , ( k)
 14 0030 , 1010 , 1010 , 107C , ( l)
 15 0000 , 0068 , 5454 , 5454 , ( m)   -->
```

```
SCR #101
  0 ( True lowercase characters for TEXT mode continued)
  1 0000 , 0058 , 6444 , 4444 , ( n)
  2 0000 , 0038 , 4444 , 4438 , ( o)
  3 0000 , 0078 , 4478 , 4040 , ( p)
  4 0000 , 0038 , 443C , 0404 , ( q)
  5 0000 , 0058 , 6440 , 4040 , ( r)
  6 0000 , 003C , 4038 , 0478 , ( s)
  7 0010 , 107C , 1010 , 1408 , ( t)
  8 0000 , 0044 , 4444 , 4C34 , ( u)
  9 0000 , 0044 , 4444 , 2810 , ( v)
 10 0000 , 0044 , 4454 , 5428 , ( w)
 11 0000 , 0044 , 2810 , 2844 , ( x)
 12 0000 , 0044 , 4C34 , 0438 , ( y)
 13 0000 , 007C , 0810 , 207C , ( z)
 14 0018 , 2020 , 4020 , 2018 , ( {)
 15 0010 , 1010 , 0010 , 1010 , ( |)   -->

SCR #102
  0 ( True lowercase characters for TEXT mode concluded)
  1 0030 , 0808 , 0408 , 0830 , ( })
  2 0000 , 2054 , 0800 , 0000 , ( ~)
  3 TRUE_LC 29 BLOCK 7C MOVE FLUSH FORGET TRUE_LC   R->BASE
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

We now need to replace the code on screens 55 and 56 that uses GPL routine 4A to load "lowercase" letters. In screen 55, replace

```
3300 834A ! 4A GPLLNK
```

with

```
29 BLOCK 3300 0F8 VMBW
```

and

```
2300 834A ! 4A GPLLNK
```

with

```
29 BLOCK 2300 0F8 VMBW
```

In screen 56, replace

```
B00 834A ! 4A GPLLNK
```

with

```
29 BLOCK B00 0F8 VMBW
```

N.2 True Lowercase for Bitmap mode

The following graphic shows the complete character set, with the true lowercase letters and the '@' designed by the editor, for bitmap mode:

This character set is used principally by the 64-column editor via the word **SMASH** defined on system screen 65. Designing the characters for a 3×7 matrix was quite a challenge. The '&' should probably be re-designed.

The only change necessary here is to overwrite the character codes for the tiny character set on system screen 67. Loading the following screens will accomplish this:

```
SCR #103
  0 ( DEFINITIONS FOR true lowercase TINY CHARACTERS) BASE->R HEX
  1 0EEE  VARIABLE TCHAR EEEE ,
  2 0000 , 0000 , (  )  0444 , 4404 , ( !)  0AA0 , 0000 , ( ")
  3 08AE , AEA2 , ( #)  04EC , 46E4 , ( $)  0A24 , 448A , ( %)
  4 06AC , 4A86 , ( &)  0480 , 0000 , ( ')  0248 , 8842 , ( ()
  5 0842 , 2248 , ( ^0) 04EE , 4000 , ( *)  0044 , E440 , ( +)
  6 0000 , 0048 , ( ,)  0000 , E000 , ( -)  0000 , 0004 , ( .)
  7 0224 , 4488 , ( /)  04AA , AAA4 , ( 0)  04C4 , 4444 , ( 1)
  8 04A2 , 488E , ( 2)  0C22 , C22C , ( 3)  02AA , AE22 , ( 4)
  9 0E8C , 222C , ( 5)  0688 , CAA4 , ( 6)  0E22 , 4488 , ( 7)
 10 04AA , 4AA4 , ( 8)  04AA , 622C , ( 9)  0004 , 0040 , ( :)
 11 0004 , 0048 , ( ;)  0024 , 8420 , ( <)  000E , 0E00 , ( =)
 12 0084 , 2480 , ( >)  04A2 , 4404 , ( ?)  04AE , AE86 , ( @)
 13 04AA , EAAA , ( A)  0CAA , CAAC , ( B)  0688 , 8886 , ( C)
 14 0CAA , AAAC , ( D)  0E88 , C88E , ( E)  0E88 , C888 , ( F)
 15                                        -->
```

```
 SCR #104
  0 ( DEFINITIONS FOR true lowercase TINY CHARACTERS continued)
  1 04A8 , 8AA6 , ( G)  0AAA , EAAA , ( H)  0E44 , 444E , ( I)
  2 0222 , 22A4 , ( J)  0AAC , CAAA , ( K)  0888 , 888E , ( E)
  3 0AEE , AAAA , ( M)  0AAE , EEAA , ( N)  0EAA , AAAE , ( 0)
  4 0CAA , C888 , ( P)  0EAA , AAEC , ( Q)  0CAA , CAAA , ( R)
  5 0688 , 422C , ( S)  0E44 , 4444 , ( T)  0AAA , AAAE , ( U)
  6 0AAA , AA44 , ( V)  0AAA , AEEA , ( W)  0AA4 , 44AA , ( X)
  7 0AAA , E444 , ( Y)  0E24 , 488E , ( Z)  0644 , 4446 , ( [)
  8 0884 , 4422 , ( \)  0C44 , 444C , ( ])  044A , A000 , ( $)
  9 0000 , 000F , ( _)  0420 , 0000 , ( `)  000E , 2EAE , ( a)
 10 088C , AAAC , ( b)  0006 , 8886 , ( c)  0226 , AAA6 , ( d)
 11 0004 , AE86 , ( e)  0688 , E888 , ( f)  0006 , A62C , ( g)
 12 088C , AAAA , ( h)  0404 , 4442 , ( i)  0202 , 22A4 , ( j)
 13 088A , ACAA , ( k)  0444 , 4444 , ( l)  000A , EEAA , ( m)
 14 0008 , EAAA , ( n)  0004 , AAA4 , ( o)  000C , AC88 , ( p)
 15                                      -->


SCR #105
  0 ( DEFINITIONS FOR true lowercase TINY CHARACTERS concluded)
  1 0006 , A622 , ( q)  0008 , E888 , ( r)  0006 , 842C , ( s)
  2 044E , 4442 , ( t)  000A , AAA6 , ( u)  000A , AAA4 , ( v)
  3 000A , AEEA , ( w)  000A , A4AA , ( x)  000A , A62C , ( y)
  4 000E , 248E , ( z)  0644 , 8446 , ( {)  0444 , 0444 , ( |)
  5 0C44 , 244C , ( })  02E8 , 0000 , ( ~)  0EEE , EEEE , ( DEL)
  6 TCHAR 43 BLOCK C2 MOVE FLUSH FORGET TCHAR    R->BASE
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

Appendix O TMS9900 Assembly Source Code for TI Forth

This appendix includes the original TMS9900 Assembly source code from the two 90-KB diskettes made available to user groups when TI Forth was released into the Public Domain. The editor has extensively annotated the code for BOOT (FORTH) and, less so, DRIVER. All such annotations are preceded with "*++" in this *hand-printing style font* to distinguish them from the original, which is in this **BOLD, MONO-SPACED FONT**.

The distributed diskettes were labeled "03NOV82" and "08DEC82". Disk "03NOV82" contained the following files:

ASMSRC
ASMSRC1
ASMSRC2
ASMSRC3

File 08DEC82 contained the following files:

BOOT
DRIVER
UTILEQU
UTILROM
UTILRAM

TI Forth can be generated from source code by following the procedure described in § O.4 Generating TI Forth from Source Code below.

O.1 DRIVER—Part 1 of FORTHSAVE

DRIVER contains the system support for TI Forth, virtually all the functionality that the Editor/Assembler cartridge loads in low memory expansion RAM, except that it is unique to TI Forth's needs. It appears to the editor that DRIVER was initially intended to be the start-up program because (1) there is quite a bit of extraneous code that is very similar to BOOT and (2) loading ASMSRC followed by DRIVER followed by ***<ENTER> <ENTER>***, actually appears to properly start TI Forth with the exception that CIF is not properly initialized. The source code that follows has been expanded into a single file:

```
       TITL 'FORTH DRIVER WITH UTIL'
       IDT  'FORTH'
*************************************************************
*++ The TI Forth workspace registers
TEMP0  EQU  0
TEMP1  EQU  1
TEMP2  EQU  2
TEMP3  EQU  3
TEMP4  EQU  4
TEMP5  EQU  5
TEMP6  EQU  6
TEMP7  EQU  7
U      EQU  8
```

```
SP     EQU  9
W      EQU  10
LINK   EQU  11
CRU    EQU  12
IP     EQU  13
R      EQU  14
NEXT   EQU  15
*************************************************************
*++ TI Forth's workspace is 32 bytes at start of PAD (>8300->831F)
MAINWS EQU  >8300        IN CONSOLE CPU RAM
KYSTAT EQU  >837C        *++ GPL status byte
KYCHAR EQU  >8375        *++ keycode detected by keyboard scan routine
FAC    EQU  >834A        FLOATING POINT ACCUMULATOR
SUBPTR EQU  >8356        POINTS TO SUBROUTINE NAMES IN VDP
VSPTR  EQU  >836E        *++ pointer to value stack in VDP RAM
DSKERR EQU  >8350        DISK DSR ERROR CODES HERE
*************************************************************
       DEF  FORTH  *++ not really necessary because we're not starting TI Forth from this program
       REF  KEY,SEMIS
*
FF9900 EQU  >A000        *++ pointer to TI Forth COLD start code
VDPSTA EQU  >8802        *++ VDPram read status register
GRMWA  EQU  >9C02        *++  GROM write address register
GRMRA  EQU  >9802        *++  GROM read address register
*************************************************************
*++ TI Forth's inner interpreter uses 26 bytes in PAD
       DORG >832E
DODOES DECT SP           DUMMY COPY TO GET ADRESSES
       MOV  W,*SP
       MOV  LINK,W
DOCOL  DECT R
       MOV  IP,*R
       MOV  W,IP
$NEXT  MOV  *IP+,W
DOEXEC MOV  *W+,TEMP1
       B    *TEMP1
$SEMIS MOV  *R+,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
       DORG 0
UBASE  BSS  6             BASE OF USER VARIABLES
$UCONS BSS  2             06 USER UCONS
$S0    BSS  2             08 USER S0
$R0    BSS  2             0A USER R0 { R0$
$U0    BSS  2             0C USER U0
       BSS  2             0E USER TIB
       BSS  2             10 USER WIDTH
       BSS  2             12 USER DP
$SYS   BSS  2             14 USER SYS$
CURPO$ BSS  2             16 USER CURPOS
       BSS  2             18 USER INTLNK
       BSS  2             1A USER WARNING
       BSS  2             1C USER C/L$ { CL$
       BSS  2             1E USER FIRST$
       BSS  2             20 USER LIMIT$
       BSS  2             22 USER B/BUF$ { BBUF$
       BSS  2             24 USER B/SCR$ { BSCR$
$DKFLO BSS  2             26 USER DISK_LO  (LOW DISK FENCE)
$DKFHI BSS  2             28 USER DISK_HI  (HIGH DISK FENCE)
$DKSIZ BSS  2             2A USER DISK_SIZE (IN SCREENS)
```

```
$DKBUF BSS  2            2C USER DISK_BUF  (BUFFER LOC IN VDP. SIZE=1K)  1K)
$PABS  BSS  2            2E USER PABS  (AREA FOR PABS ETC.)
$SWDTH BSS  2            30 USER SCRN_WIDTH
$SSTRT BSS  2            32 USER SCRN_START
$SEND  BSS  2            34 USER SCRN_END
$ISR   BSS  2            36 USER ISR
$ALTI  BSS  2            38 USER ALTIN
$ALTO  BSS  2            3A USER ALTOUT
       BSS  2            3C USER FENCE
       BSS  2            3E USER BLK
       BSS  2            40 USER IN
$OUT   BSS  2            42 USER OUT
       BSS  2            44 USER SCR
$OFFST BSS  2            46 USER OFFSET
       BSS  2            48 USER CONTEXT
       BSS  2            4A USER CURRENT
       BSS  2            4C USER STATE
       BSS  2            4E USER BASE
       BSS  2            50 USER DPL
       BSS  2            52 USER FLD
       BSS  2            54 USER CSP
       BSS  2            56 USER R# { RNUM
       BSS  2            58 USER HLD
       BSS  2            5A USER USE
       BSS  2            5C USER PREV
       BSS  2            5E
FORLNK BSS  2            60 USER FORTH_LINK
       BSS  2            62
       BSS  2            64 USER ECOUNT
UMAX   BSS  0
*
*
*++ End of resident dictionary and start of optional/removable section of dictionary
       DORG >BC80
DPBASE BSS  >4320        START OF RAM DICTIONARY
SPBASE BSS  0            BASE OF PARAMETER STACK  *++ grows down toward HERE
       BSS  82           TEXT INPUT BUFFER  *++ same address as base of stack
RBASE  EQU  >3FFE        BASE OF RETURN STACK  *++ grows down toward system support
*
*** UTILITY EQUATES ******
**
**       COPY "DSK2.UTILEQU"
**
** vvvvvvvvvvvv UTILEQU vvvvv below vvvvvvvvvvvvvvvvvvvvvvvv
**
SCNKEY EQU  >000E
XMLTAB EQU  >0CFA        XML TABLES (BASE)
FLAG2  EQU  >8349
SCLEN  EQU  >8355
SCNAME EQU  >8356
SUBSTK EQU  >8373
CRULST EQU  >83D0
SADDR  EQU  >83D2
GPLWS  EQU  >83E0        GPL/EXTENDED BASIC WORKSPACE
PAD    EQU  >8300
VDPRD  EQU  >8800        VDP read data address
VDPWD  EQU  >8C00        VDP write data address
VDPWA  EQU  >8C02        VDP write address address
R0LB   EQU  >83E1
R1LB   EQU  >83E3
R3LB   EQU  >83E7
**
** ^^^^^^^^^^^^ UTILEQU ^^^^^ above ^^^^^^^^^^^^^^^^^^^^^^^^
**
*
```

```
*++ This section of the program up to DORG >2010 is not necessary because we don't start TI Forth
*++ with this program.  Once TI decided to develop the BOOT program (FORTH) to start TI Forth,
*++ this code was essentially abandoned.  You can actually start TI Forth by loading (E/A option 3)
*++ the ASMSRC object file followed by this object file and then tapping <ENTER> to the "FILE NAME?"
*++ prompt and the subsequent "PROGRAM NAME?" prompt.  There is, however, at least one problem
*++ with starting up this way, viz., a missing reference at >2006 to CIF, the integer-to-floating-point
*++ conversion routine, which could be remedied by adding the following code after "DATA ENTLNK":
*++     AORG >2006
*++     DATA CIF
*++ In addition to the above fix, you would need to have a disk in drive 0 (DSK1) with all the system
*++ screens/blocks in place and load the above object files from any drive except the first one.
*
       AORG >2002
       DATA ENTLNK       PATCH TO MY ENTRY
*
       AORG >C000
FMOVE  DECT SP           COPY TO MOVE TO CONSOLE RAM
       MOV  W,*SP
       MOV  LINK,W
       DECT R
       MOV  IP,*R
       MOV  W,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
       MOV  *R+,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
*
FORTH  LIMI 0
       LWPI MAINWS
       LI   TEMP1,FORTH-FMOVE
       LI   TEMP2,FMOVE
       LI   TEMP3,DODOES
MLOOP  MOV  *TEMP2+,*TEMP3+
       DECT TEMP1
       JNE  MLOOP
*
*** INITIALIZE VDP STUFF
*
       LI   TEMP0,>01B0  BLANK SCREEN
       BLWP @VWTR
       LI   TEMP0,>030E  SET COLOR TABLE AT >0380
       BLWP @VWTR
       LI   TEMP0,>0401  SET PATTERN DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>0506  SET SPRITE ATTRIBUTE TABLE >0300
       BLWP @VWTR
       LI   TEMP0,>0601  SET SPRITE DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>07F4  SET TEXTMODE COLORS
       BLWP @VWTR
       LI   TEMP0,>2000  BLANK
       LI   TEMP1,>960   TEXT-MODE SCREEN SIZE   *++ should be 960 or >3C0
       LI   TEMP2,>0     SCREEN STARTS AT 0
       BL   @FILLER      CLEAR SCREEN
       LI   TEMP0,>FF00  CHAR FF
       LI   TEMP1,>2048  BLOCK SIZE              *++  should be 2048 or >800
       LI   TEMP2,>800   STARTING LOCATION IN VDP
       BL   @FILLER      FILL AREA WITH FF'S
       LI   TEMP0,>81F0
```

```
       SWPB TEMP0
       MOVB TEMP0,@>83D4 USED TO UPDATE VDP REG EACH KEYSTROKE
       MOVB TEMP0,@VDPWA FORCE TEXT MODE
       SWPB TEMP0
       MOVB TEMP0,@VDPWA
       LI   TEMP0,>900   VDP LOCATION
       MOV  TEMP0,@FAC
       CLR  TEMP1        CLEAR GPL STATUS
       MOVB TEMP1,@KYSTAT
       LI   TEMP7,>3E0
       MOV  TEMP7,@VSPTR
       BLWP @GPLLNK      LOAD CAPITAL LETTER SHAPES
       DATA >0018
       LI   TEMP2,>1200
       CB   TEMP2,@3     IF BYTE @3 IN THE CONSOLE IS >12 IT'S A 99/4
       JEQ  FOUR         DON'T LOAD LOWER CASE IN A 99/4
       LI   TEMP0,>0B00
       MOV  TEMP0,@FAC
       MOVB TEMP1,@KYSTAT
       BLWP @GPLLNK      LOAD LOWER CASE IN 99/4A
       DATA >004A
*
FOUR   LI   TEMP1,UBASE0
       B    @FF9900      BRANCH TO ACMSRC
*
FILLER ORI  TEMP2,>4000  SET BIT FOR VDP WRITE
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
FLLOOP MOVB TEMP0,@VDPWD WRITE A BYTE
       DEC  TEMP1
       JNE  FLLOOP       NOT DONE, FILL ANOTHER
       B    *LINK
*
       DORG >2010
*
$BUFF  BSS  5*>404        I/O BUFFERS
$LO    BSS  0
*
*++ beginning of BOOT (FORTH) load of DRIVER (1st part of FORTHSAVE)
       AORG $LO
*
*
*** INTERRUPT SERVICE
*
INT1   LI   TEMP1,INT2   FIX 'NEXT' SO THAT INTERRUPT IS
       MOV  TEMP1,@2*NEXT+MAINWS  PROCESSED AT END OF
       LWPI >83C0        NEXT 'CODE' WORD
       RTWP
*
INT2   LIMI 0
       MOVB @>83D4,TEMP0
       SRL  TEMP0,8
       ORI  TEMP0,>100
       ANDI TEMP0,>FFDF
       BLWP @VWTR        TURN OFF VDP INTERRUPTS
       LI   NEXT,$NEXT   RESTORE 'NEXT'
       SETO @INTACT
       DECT R            SET UP RETURN LINKAGE
       MOV  IP,*R
       LI   IP,INT3
       MOV  @$ISR(U),W   DO THE FORTH ROUTINE
       B    @DOEXEC
INT3   DATA $+2
```

```
       DATA $+2
       MOV  *R+,IP
       CLR  @INTACT
       MOVB @>83D4,TEMP0
       SRL  TEMP0,8
       AI   TEMP0,>100
       MOVB @VDPSTA,TEMP1 REMOVE PENDING INTERRUPT
       BLWP @VWTR
       LIMI 2
       B    *NEXT        CONTINUE NORMAL TASK
*=============================================================
BKLINK MOV  @INTACT,TEMP7
       JNE  BKLIN1
       LIMI 2
BKLIN1 B    *LINK
*=============================================================
*
$SYS$  LIMI 0
       MOV  @SYSTAB(TEMP1),TEMP0
       B    *TEMP0
       DATA DSK          CODE=-18 DRIVE SELECTION
       DATA DSK          CODE=-16 READ DISK
       DATA DSK          CODE=-14 WRITE DISK
       DATA GXY          CODE=-12 GOTOXY
       DATA QKY          CODE=-10 ?KEY
       DATA QTM          CODE=-8  ?TERMINAL
       DATA CLF          CODE=-6  CRLF
       DATA EMT          CODE=-4  EMIT
       DATA KY           CODE=-2  KEY
SYSTAB DATA SBW          CODE=0   VSBW
       DATA MBW          CODE=2   VMBW
       DATA SBR          CODE=4   VSBR
       DATA MBR          CODE=6   VMBR
       DATA WTR          CODE=8   VWTR
       DATA GPL          CODE=10  GPLLNK
       DATA XML          CODE=12  XMLLNK
       DATA DSR          CODE=14  DSRLNK
       DATA CLS          CODE=16  CLS
       DATA FMT          CODE=18  FORMAT-DISK
       DATA FILL         CODE=20  VFILL
       DATA AOX          CODE=22  VAND
       DATA AOX          CODE=24  VOR
       DATA AOX          CODE=26  VXOR
*
*== THIS IS A VDP SINGLE BYTE WRITE.  CODE=0  ==============
*
SBW    MOV  *SP+,TEMP0   VDP ADDRESS (DESTINATION)
       MOV  *SP+,TEMP1   CHARACTER TO WRITE
       SWPB TEMP1        GET IN LEFT BYTE
       BLWP @VSBW
       B    @BKLINK
*
*== THIS IS A VDP MULTI BYTE WRITE.  CODE=2   ==============
*
MBW    MOV  *SP+,TEMP2   NUMBER OF BYTE TO MOVE
       MOV  *SP+,TEMP0   VDP ADDRESS (DESTINATION)
       MOV  *SP+,TEMP1   RAM ADDRESS (SOURCE)
       BLWP @VMBW
       B    @BKLINK
*
*== THIS IS A VDP SINGLE BYTE READ.  CODE=4   ==============
*
SBR    MOV  *SP,TEMP0    VDP ADDRESS (SOURCE)
       BLWP @VSBR
       SRL  TEMP1,8      CHARACTER TO RIGHT HALF FOR FORTH
       MOV  TEMP1,*SP    STACK IT
```

```
       B    @BKLINK
*
*== THIS IS A VDP MULTI BYTE READ.  CODE=6    ==============
*
MBR    MOV  *SP+,TEMP2   NUMBER OF BYTES TO READ
       MOV  *SP+,TEMP1   RAM ADDRESS (DESTINATION)
       MOV  *SP+,TEMP0   VDP ADDRESS (SOURCE)
       BLWP @VMBR
       B    @BKLINK
*
*== VDP REGISTER WRITE.  CODE=8               ==============
*
WTR    MOV  *SP+,TEMP1   VDP REGISTER NUMBER
       MOV  *SP+,TEMP0   DATA FOR REGISTER
       SWPB TEMP1        GET REGISTER TO LEFT BYTE
       MOVB TEMP1,TEMP0  PLACE WITH DATA
       BLWP @VWTR
       B    @BKLINK
*
*== THIS IS THE GPL LINK UTILITY.  CODE=10    ==============
*
GPL    CLR  TEMP0
       MOVB TEMP0,@KYSTAT
       LI   TEMP0,>0420       CONSTRUCT THE BLWP INSTRUCTION
       LI   TEMP1,GPLLNK      TO THE GPLLNK UTILITY
       MOV  *SP+,TEMP2        WITH THIS DATA IDENTIFYING THE ROUTINE
       LI   TEMP3,>045B       CONSTRUCT THE B *LINK INSTRUCTION
       MOV  LINK,TEMP4        SAVE LINK ADDRESS
       BL   @2*TEMP0+MAINWS   EXECUTE THE ABOVE INSTRUCTIONS
       MOV  TEMP4,LINK        AND RECONSTRUCT LINK
       B    @BKLINK
*
*== THIS IS THE XML LINK UTILITY.  CODE=12    ==============
*
XML    LI   TEMP0,>0420       CONSTRUCT THE BLWP INSTRUCTION
       LI   TEMP1,XMLLNK      TO THE XMLLNK UTILITY
       MOV  *SP+,TEMP2        WITH THIS DATA IDENTIFYING THE ROUTINE
       LI   TEMP3,>045B       CONSTRUCT THE B *LINK INSTRUCTION
       MOV  LINK,TEMP4        SAVE LINK ADDRESS
       BL   @2*TEMP0+MAINWS   EXECUTE THE ABOVE INSTRUCTIONS
       MOV  TEMP4,LINK        AND RECONSTRUCT LINK
       B    @BKLINK
*
*== THIS IS THE DSR LINK UTILITY.  CODE=14    ==============
*
DSR    LI   TEMP0,>0420       CONSTRUCT THE BLWP INSTRUCTION
       LI   TEMP1,DSRLNK      TO THE DSRLNK UTILITY
       MOV  *SP+,TEMP2        THIS DATUM SELECTS DSR OR SUBROUTINE
       LI   TEMP3,>045B       CONSTRUCT THE B *LINK INSTRUCTION
       MOV  LINK,TEMP4        SAVE LINK ADDRESS
       BL   @2*TEMP0+MAINWS   EXECUTE THE ABOVE INSTRUCTIONS
       MOV  TEMP4,LINK        AND RECONSTRUCT LINK
       B    @BKLINK
*
*== THIS IS THE SCREEN CLEARING UTILITY.  CODE=16 ==========
*
CLS    MOV  @$SSTRT(U),TEMP2     BEGINNING OF SCREEN IN VDP
       MOV  @$SEND(U),TEMP1      END OF SCREEN IN VDP
       S    TEMP2,TEMP1       SCREEN SIZE
       LI   TEMP0,>2000       BLANK CHARACTER
       MOV  LINK,TEMP7
       BL   @FILL1
       MOV  TEMP7,LINK
       B    @BKLINK
*
*== THIS IS THE DISK FORMATTER.  CODE=18   =================
```

```
*
FMT    MOV  @$DKBUF(U),TEMP0 VDP BUFFER START ADDRESS (3300 BYTES)
       MOV  TEMP0,@FAC+4
       MOV  *SP+,TEMP0
       SLA  TEMP0,8      DRIVE # TO LEFT BYTE
       AI   TEMP0,40     40 TRACKS
       MOV  TEMP0,@FAC+2
       MOV  @$PABS(U),TEMP0 SETUP SUBR NAME (>11)
       MOV  TEMP0,@SUBPTR
       LI   TEMP1,DFMT
       LI   TEMP2,2
       BLWP @VMBW
       BLWP @DSRLNK
       DATA >A
       B    @BKLINK
DFMT   DATA >0111        USED BY VMBW ABOVE
*
*== THIS IS THE VDP FILL ROUTINE.  CODE=20
*
FILL   MOV  *SP+,TEMP0   FILL CHARACTER
       SWPB TEMP0        TO LEFT BYTE
       MOV  *SP+,TEMP1   FILL COUNT
       MOV  *SP+,TEMP2   ADDRESS TO START VDP FILL
       MOV  LINK,TEMP7
       BL   @FILL1
       MOV  TEMP7,LINK
       B    @BKLINK
*=========================================================
FILL1  ORI  TEMP2,>4000  SET BIT FOR VDP WRITE
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
FLOOP  MOVB TEMP0,@VDPWD WRITE A BYTE
       DEC  TEMP1
       JNE  FLOOP        NOT DONE, FILL ANOTHER
       B    *LINK
*=========================================================
*
*== VDP BYTE 'AND' 'OR' 'XOR' ROUTINES.  CODE=22,24,26   ===
*
AOX    MOV  *SP+,TEMP2   VDP ADDRESS
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
       MOVB @VDPRD,TEMP3 READ BYTE
       MOV  *SP+,TEMP0   GET DATA TO OPERATE WITH
       SWPB TEMP0        TO LEFT BYTE
*** NOW DO REQUESTED OPERATION *****************
       CI   TEMP1,24
       JEQ  DOOR
       JGT  DOXOR
       INV  TEMP3        THESE TWO INSTRUCTIONS
       SZC  TEMP3,TEMP0  PERFORM AN 'AND'
       JMP  FINAOX
DOOR   SOC  TEMP3,TEMP0  PERFORM OR
       JMP  FINAOX
DOXOR  XOR  TEMP3,TEMP0  PERFORM XOR
FINAOX LI   TEMP1,1
       MOV  LINK,TEMP7
       BL   @FILL1
       MOV  TEMP7,LINK
       B    @BKLINK
```

```
*
*=========================================================
*
*== KEY ROUTINE  CODE= -2  ================================
*
KY     MOV  @$ALTI(U),TEMP0
       JEQ  KEY0
       CLR  TEMP7
       MOVB TEMP7,@KYSTAT
       INC  TEMP0
       BLWP @VSBR
       ANDI TEMP1,>1F00
       BLWP @VSBW
       MOV  TEMP0,TEMP1
       AI   TEMP1,8
       MOV  TEMP1,@SUBPTR
       BLWP @DSRLNK
       DATA >8
       DECT TEMP0
       BLWP @VSBR
       SRL  TEMP1,8
       MOV  TEMP1,TEMP0
       B    @BKLINK
KEY0   MOV  @KEYCNT,TEMP7
       INC  TEMP7
       JNE  KEY1
       MOV  @CURPO$(U),TEMP0
       BLWP @VSBR             READ CHARACTER AT CURSOR POSITION
       MOVB TEMP1,@CURCHR     AND SAVE IT
       LI   TEMP1,>1E00       PLACE CURSOR CHARACTER ON SCREEN
       BLWP @VSBW
*
KEY1   LI   TEMP4,>2000       MASK TO CHECK STATUS
       BLWP @KSCAN
       MOVB @KYSTAT,TEMP0
       COC  TEMP4,TEMP0
       JEQ  KEY2              JMP IF KEY WAS PRESSED
*
       CI   TEMP7,100         NO KEY PRESSED
       JNE  KEY3
       MOVB @CURCHR,TEMP1
       JMP  KEY5
*
KEY3   CI   TEMP7,200
       JNE  KEY4
       CLR  TEMP7
       LI   TEMP1,>1E00       CURSOR CHAR
KEY5   MOV  @CURPO$(U),TEMP0
       BLWP @VSBW
KEY4   MOV  TEMP7,@KEYCNT
       MOV  @INTACT,TEMP7
       JNE  KEY6
       LIMI 2
KEY6   DECT IP                THIS WILL RE-EXECUTE KEY
       B    *NEXT
*KE     DATA KEY,SEMIS
*
*
KEY2   SETO @KEYCNT           KEY WAS PRESSED
       MOV  @CURPO$(U),TEMP0  RESTORE CHARACTER AT CURSOR LOCATION
       MOVB @CURCHR,TEMP1
       BLWP @VSBW
       MOVB @KYCHAR,TEMP0     PUT CHAR IN RIGHT HALF OF TEMP0
       SRL  TEMP0,8
       B    @BKLINK
*
```

```
*== EMIT ROUTINE  CODE= -4  ================================
*
EMT    MOV  TEMP2,TEMP1
       MOV  @$ALTO(U),TEMP0
       JEQ  EMIT0
       CLR  TEMP7             ALTOUT ACTIVE
       MOVB TEMP7,@KYSTAT
       DEC  TEMP0
       SWPB TEMP1
       BLWP @VSBW
       INCT TEMP0
       BLWP @VSBR
       ANDI TEMP1,>1F00
       BLWP @VSBW
       AI   TEMP0,8
       MOV  TEMP0,@SUBPTR
       BLWP @DSRLNK
       DATA >8
       B    @BKLINK
*
EMIT0  CI   TEMP1,7      IS IT A BELL?
       JNE  NOTBEL
       CLR  TEMP2
       MOVB TEMP2,@KYSTAT
       MOVB @GRMSAV,@GRMWA    RESTORE GROM ADDRESS
       NOP
       MOVB @GRMSAV+1,@GRMWA
       BLWP @GPLLNK
       DATA >0036        EMIT ERROR TONE
       JMP  EMEXIT
*
NOTBEL CI   TEMP1,8      IS IT A BACKSPACE?
       JNE  NOTBS
       LI   TEMP1,>2000
       MOV  @CURPO$(U),TEMP0
       BLWP @VSBW
       JGT  DECCUR
       JMP  EMEXIT
DECCUR DEC  @CURPO$(U)
       JMP  EMEXIT
*
NOTBS  CI   TEMP1,>A     IS IT A LINE FEED?
       JNE  NOTLF
       MOV  @$SEND(U),TEMP7
       S    @$SWDTH(U),TEMP7
       C    @CURPO$(U),TEMP7
       JHE  SCRLL
       A    @$SWDTH(U),@CURPO$(U)
       JMP  EMEXIT
SCRLL  MOV  LINK,TEMP7
       BL   @SCROLL
       MOV  TEMP7,LINK
       JMP  EMEXIT
*
*** SCROLLING ROUTINE
*
SCROLL MOV  @$SSTRT(U),TEMP0     VDP ADDR
       LI   TEMP1,LINBUF      BUFFER
       MOV  @$SWDTH(U),TEMP2     COUNT
       A    TEMP2,TEMP0       START AT LINE 2
SCROL1 BLWP @VMBR
       S    TEMP2,TEMP0       ONE LINE BACK TO WRITE
       BLWP @VMBW
       A    TEMP2,TEMP0       TWO LINES AHEAD FOR NEXT READ
       A    TEMP2,TEMP0
       C    TEMP0,@$SEND(U)      END OF SCREEN?
```

```
       JL   SCROL1
       MOV  TEMP2,TEMP1       BLANK BOTTOM ROW OF SCREEN
       LI   TEMP0,>2000       BLANK
       S    @$SEND(U),TEMP2
       NEG  TEMP2             NOW CONTAINS ADDRESS OF START OF LAST LINE
       MOV  LINK,TEMP6
       BL   @FILL1            WRITE THE BLANKS
       B    *TEMP6
*
NOTLF  CI   TEMP1,>D     IS IT A CARRIAGE RETURN?
       JNE  NOTCR
       CLR  TEMP0
       MOV  @CURPO$(U),TEMP1
       MOV  TEMP1,TEMP3
       S    @$SSTRT(U),TEMP1  ADJUSTED FOR SCREEN NOT AT 0
       MOV  @$SWDTH(U),TEMP2
       DIV  TEMP2,TEMP0
       S    TEMP1,TEMP3
       MOV  TEMP3,@CURPO$(U)
       JMP  EMEXIT
*
NOTCR  SWPB TEMP1        ASSUME IT IS A PRINTABLE CHARACTER
       MOV  @CURPO$(U),TEMP0
       BLWP @VSBW
       MOV  @$SEND(U),TEMP2
       DEC  TEMP2
       C    TEMP0,TEMP2
       JNE  NOTCR1
       MOV  @$SEND(U),TEMP0
       S    @$SWDTH(U),TEMP0 WAS LAST CHAR ON SCREEN. SCROLL
       MOV  TEMP0,@CURPO$(U)
       JMP  SCRLL
NOTCR1 INC  TEMP0        NO SCROLL NECESSARY
       MOV  TEMP0,@CURPO$(U)
*
EMEXIT B    @BKLINK
*
*== CRLF ROUTINE  CODE= -6  ================================
*
CLF    MOV  LINK,TEMP5
       LI   TEMP2,>000D
       BL   @EMT
       LI   TEMP2,>000A
       LIMI 0            PREVIOUS CALL TO EMT ALTERED INT MASK
       BL   @EMT
       MOV  TEMP5,LINK
       B    @BKLINK
*
*== ?TERMINAL ROUTINE CODE= -8  ============================
*
QTM    BLWP @KSCAN
       MOVB @KYCHAR,TEMP0
       SRL  TEMP0,8
       CI   TEMP0,>2
       JEQ  QTERM1
       CLR  TEMP0
QTERM1 B    @BKLINK
*
*== ?KEY ROUTINE CODE= -10  ================================
*
QKY    BLWP @KSCAN
       MOVB @KYCHAR,TEMP0
       SRL  TEMP0,8
       CI   TEMP0,>00FF
       JNE  QKEY1
       CLR  TEMP0
```

```
QKEY1  B    @BKLINK
*
*== GOTOXY ROUTINE CODE= -12  ============================
*
GXY    MPY  @$SWDTH(U),TEMP3
       A    TEMP2,TEMP4       POSITION WITHIN SCREEN
       A    @$SSTRT(U),TEMP4     ADD VDP OFFSET TO SCREEN TOP
       MOV  TEMP4,@CURPO$(U)
       B    @BKLINK
*
*** ENTRY POINT FOR DISK HANDLING ROUTINES
*
*++ only used for TI Forth screen/block reads and writes.
DSK    MOV  TEMP1,TEMP7  SAVE CODE OF DISK ROUTINE
       MOV  @$PABS(U),TEMP0
       LI   TEMP1,>0100  SET UP VDP FOR DISK DSR
       BLWP @VSBW        LEVEL ONE R/W ACCESS
       INC  TEMP0
       LI   TEMP1,>1000
       BLWP @VSBW
       MOV  TEMP4,TEMP1  ADDRESS TO TEMP1
       CI   TEMP7,-14
       JNE  NOTWD
*
*** DISK WRITE ROUTINE  CODE=-14
*
*++ writes one block of B/BUF bytes to 4 sectors of the disk
       C    TEMP2,@$DKFLO(U)  TEST DISK FENCES
       JHE  DKWT1
DKWT0  LI   TEMP0,>B
       JMP  DKEXIT
DKWT1  C    TEMP2,@$DKFHI(U)
       JHE  DKWT0
       MOV  @$DKBUF(U),TEMP0 VDP BUFFER
       MOV  TEMP2,TEMP5  SAVE BLOCK #
       MOV  TEMP3,TEMP2  BYTES/BLOCK
       BLWP @VMBW
       CLR  TEMP4
       DIV  @$DKSIZ(U),TEMP4  BLOCK # IN TEMP5
       INC  TEMP4
       SWPB TEMP4
       MOV  TEMP4,@FAC+2
       SLA  TEMP5,2      CONV BLOCK # TO SECT #
       MOV  TEMP5,@FAC+6
       LI   TEMP6,4      SET COUNTER
       MOV  @$DKBUF(U),@FAC+4
WTLOOP MOV  @$PABS(U),@SUBPTR
       BLWP @DSRLNK
       DATA >A
       MOVB @DSKERR,TEMP7
       JNE  DKERR
       DEC  TEMP6        CHECK LOOP COUNT
       JEQ  DSKDON
       MOV  @FAC,@FAC+6
       INC  @FAC+6
       LI   TEMP0,256
       A    TEMP0,@FAC+4
       JMP  WTLOOP
DKERR  LI   TEMP0,6
       JMP  DKEXIT
DSKDON CLR  TEMP0
*
DKEXIT B    @BKLINK
*
*
```

```
NOTWD  CI   TEMP7,-16
       JNE  NOTRD
*
*** DISK READ ROUTINES  CODE=-16
*
*++ reads 4 sectors from disk and copies B/BUF bytes to the requested block buffer
       MOV  TEMP2,TEMP5  SAVE BLK#
       CLR  TEMP4
       DIV  @$DKSIZ(U),TEMP4  BLOCK # IN TEMP5
       INC  TEMP4
       SWPB TEMP4
       INC  TEMP4        SET BIT FOR READ
       MOV  TEMP4,@FAC+2
       SLA  TEMP5,2
       MOV  TEMP5,@FAC+6
       MOV  @$PABS(U),TEMP4
       LI   TEMP6,4      INIT COUNTER
       MOV  @$DKBUF(U),@FAC+4
RDLOOP MOV  TEMP4,@SUBPTR
       BLWP @DSRLNK
       DATA >A
       MOVB @DSKERR,TEMP7
       JNE  DKERR
       DEC  TEMP6
       JEQ  RDDONE
       MOV  @FAC,@FAC+6
       INC  @FAC+6
       LI   TEMP0,256
       A    TEMP0,@FAC+4
       JMP  RDLOOP
RDDONE MOV  @$DKBUF(U),TEMP0 VDP BUFFER
       MOV  TEMP3,TEMP2  #BYTES/BLOCK
       BLWP @VMBR
       JMP  DSKDON
*
*
NOTRD  EQU  $
*
*** DRIVE SELECTION ROUTINE   CODE =-18
*
       CLR  TEMP0
DRV1   DEC  TEMP2
       JLT  DRV2
       A    @$DKSIZ(U),TEMP0
       JMP  DRV1
DRV2   B    @BKLINK
*
*=============================================================
*
*++ initial values for first 27 user variables—COLD resets user variable table to these
UBASE0 BSS  6             BASE OF USER VARIABLES
       DATA UBASE0        06 USER UCONS
       DATA SPBASE        08 USER S0
       DATA RBASE         0A USER R0 { R0$
       DATA $UVAR         0C USER U0
       DATA SPBASE        0E USER TIB
       DATA 31            10 USER WIDTH
       DATA DPBASE        12 USER DP
       DATA $SYS$         14 USER SYS$
       DATA 0             16 USER CURPOS
       DATA INT1          18 USER INTLNK
       DATA 1             1A USER WARNING
       DATA 64            1C USER C/L$ { CL$
       DATA $BUFF         1E USER FIRST$
       DATA $LO           20 USER LIMIT$
```

```
       DATA 1024          22 USER B/BUF$ { BBUF$
       DATA 1             24 USER B/SCR$ { BSCR$
       DATA 1             26 USER DISK_LO  (LOW DISK FENCE)
       DATA 90            28 USER DISK_HI  (HIGH DISK FENCE)
       DATA 90            2A USER DISK_SIZE (IN SCREENS)
       DATA >1000         2C USER DISK_BUF  (BUFFER LOC IN VDP. SIZE=1K)  1K)
       DATA >460          2E USER PABS  (AREA FOR PABS ETC.)
       DATA 40            30 USER SCRN_WIDTH
       DATA 0             32 USER SCRN_START
       DATA 960           34 USER SCRN_END
       DATA INT1          36 USER ISR
       DATA 0             38 USER ALTIN
       DATA 0             3A USER ALTOUT
*
$UVAR  BSS  >80           USER VARIABLE AREA
*
**
**       COPY "DSK2.UTILROM"
**
** vvvvvvvvvvvv UTILROM vvvvv below vvvvvvvvvvvvvvvvvvvvvvvv
**
C100   DATA 100
H20    EVEN
H2000  DATA >2000
DECMAL TEXT '.'
HAA    BYTE >AA
       EVEN
*
* Utility Vectors
*
GPLLNK DATA UTILWS,GLENTR     Link to GROM routines
XMLLNK DATA UTILWS,XMLENT     Link to ROM routines
KSCAN  DATA UTILWS,KSENTR     Keyboard scan
VSBW   DATA UTILWS,VSBWEN     VDP single byte write
VMBW   DATA UTILWS,VMBWEN     VDP multiple byte write
VSBR   DATA UTILWS,VSBREN     VDP single byte read
VMBR   DATA UTILWS,VMBREN     VDP multiple byte read
VWTR   DATA UTILWS,VWTREN     VDP write to register
DSRLNK DATA DLNKWS,DLENTR     Link to device service routine
*
*=========================================================
ENTLNK MOV  R11,@SVGPRT  Save return to GPL interpreter
       MOVB @GRMRA,R1    Save GROM address of XML to
       SWPB R1           this routine.
       MOVB @GRMRA,R1
       SWPB R1
       AI   R1,-3
       MOV  R1,@GRMSAV
       LI   R1,RTFGPL    Change XML vectors
       MOV  R1,@>2000    for Extended BASIC
       MOV  R1,@>2002    for Editor/Assembler
       B    @FORTH       Go to FORTH
*
*=========================================================
*      LINK TO SYSTEM XML UTILITIES
*
XMLENT MOV  *R14+,@GPLWS+2    Get argument
       LWPI GPLWS             Select GPL workspace
       MOV  R11,@UTILWS+22    Save GPL return address
       MOV  R1,R2             Make a copy of argument
       CI   R1,>8000          Direct address in ALC?
       JH   XML30             We have the address
       SRL  R1,12
       SLA  R1,1
       SLA  R2,4
       SRL  R2,11
```

```
       A    @XMLTAB(R1),R2
       MOV  *R2,R2
XML30  BL   *R2
       LWPI UTILWS            GET BACK TO RIGHT WS
       MOV  R11,@GPLWS+22     Restore GPL return address
       RTWP
*
*=============================================================
*** Link to GPL utilities
*
GLENTR MOVB @SUBSTK,R2        Fetch GPL subroutine stack ptr
       SRL  R2,8              Make it an index
       AI   R2,PAD
       INCT R2
       MOV  @GRMSAV,R1        Push XML address for return
       MOVB R1,*R2
       SWPB R1
       MOVB R1,@1(R2)
       SWPB R2                Adjust stack pointer
       MOVB R2,@SUBSTK
       MOVB *R14+,@GRMWA      Set up address to call
       MOVB *R14+,@GRMWA      and second byte, adjusting return
       LWPI GPLWS
       MOV  @SVGPRT,R11
       RT                     Return to GPL
*
*** Return to assembly language from GPL
*
RTFGPL LWPI UTILWS       Select utility workspace
       RTWP              Return to calling AL routine
*
*=============================================================
*      KEYBOARD SCAN
*
KSENTR LWPI GPLWS
       MOV  R11,@UTILWS+22   Save GPL return address
       BL   @SCNKEY
       LWPI UTILWS
       MOV  R11,@GPLWS+22    Restore GPL return address
       RTWP
*
*=============================================================
*      VDP UTILITIES
*
** VDP single byte write
*
VSBWEN BL   @WVDPWA          Write out address
       MOVB @2(R13),@VDPWD   Write data
       RTWP                  Return to calling program
*
** VDP multiple byte write
*
VMBWEN BL   @WVDPWA          Write out address
VWTMOR MOVB *R1+,@VDPWD      Write a byte
       DEC  R2               Decrement byte count
       JNE  VWTMOR           More to write?
       RTWP                  Return to calling Program
*
** VDP single byte read
*
VSBREN BL   @WVDPRA          Write out address
       MOVB @VDPRD,@2(R13)   Read data
       RTWP                  Return to calling program
*
** VDP multiple byte read
*
```

```
VMBREN BL   @WVDPRA          Write out address
VRDMOR MOVB @VDPRD,*R1+      Read a byte
       DEC  R2               Decrement byte count
       JNE  VRDMOR           More to read?
       RTWP                  Return to calling program
*
** VDP write to register
*
VWTREN MOV  *R13,R1          Get register number and value
       MOVB @1(R13),@VDPWA   Write out value
       ORI  R1,>8000         Set for register write
       MOVB R1,@VDPWA        Write out register number
       RTWP                  Return to calling program
*
** Set up to write to VDP
*
WVDPWA LI   R1,>4000
       JMP  WVDPAD
*
** Set up to read VDP
*
WVDPRA CLR  R1
*
** Write VDP address
*
WVDPAD MOV  *R13,R2          Get VDP address
       MOVB @R2LB,@VDPWA     Write low byte of address
       SOC  R1,R2            Properly adjust VDP write bit
       MOVB R2,@VDPWA        Write high byte of address
       MOV  @2(R13),R1       Get CPU RAM address
       MOV  @4(R13),R2       Get byte count
       RT                    Return to calling routine
*
*=============================================================
*      CIF - Convert integer to floating                    *
*
CIF    LI   R4,FAC            Will convert into the FAC
       MOV  *R4,R0            Get integer into register
       MOV  R4,R6             Copy ptr to FAC to clear it
       CLR  *R6+              Clear FAC,FAC+1
       CLR  *R6+              IN CASE HAD A STRING IN FAC
       MOV  R0,R5             IS INTEGER EQUAL TO ZERO?
       JEQ  CIFRT             YES - ZERO RESULT AND RETURN
       ABS  R0                GET ABS VALUE OF ARG
       LI   R3,>40            GET EXPONENT BIAS
       CLR  *R6+              CLEAR WORDS IN RESULT THAT
       CLR  *R6                 MIGHT NOT GET A VALUE
       CI   R0,100            IS INTEGER < 100?
       JL   CIF02             YES-JUST PUT IN 1ST FRACTION
*                                PART
       CI   R0,10000          NO-IS ARG < 100,2?
       JL   CIF01             YES-JUST 1 DIVISION NECESSARY
*                             NO - 2 DIVISIONS ARE NECESSARY
       INC  R3                ADD 1 TO EXPONENT FOR 1ST DIV
       MOV  R0,R1             PUT # IN LOW ORDER WORD FOR
*                               THE DIVIDE
       CLR  R0                CLEAR HIGH ORDER WORD FOR THE
*                                DIVIDE
       DIV  @C100,R0          DIVIDE BY THE RADIX
       MOVB @R1LB,@3(R4)      MOVE THE RADIX DIGIT IN
CIF01
       INC  R3                ADD 1 TO EXPONENT FOR DIVIDE
       MOV  R0,R1             PUT IN LOW ORDER FOR DIVIDE
       CLR  R0                CLEAR HIGH ORDER FOR DIVIDE
       DIV  @C100,R0          DIVIDE BY THE RADIX
       MOVB @R1LB,@2(R4)      PUT NEXT RADIX DIGIT IN
```

```
CIF02
       MOVB @R0LB,@1(R4)      PUT HIGHEST ORDER RADIX DIGIT
*                             IN
       MOVB @R3LB,*R4         PUT EXPONENT IN
       INV  R5                IS RESULT POSITIVE?
       JLT  CIFRT             YES - SIGN IS CORRECT
       NEG  *R4               NO - MAKE IT NEGATIVE
CIFRT  RT
*
*=======================================================
*** Link to device service routine
*
DLENTR MOV  *R14+,R5          Fetch program type for link
       SZCB @H20,R15          Reset equal bit
       MOV  @SCNAME,R0        Fetch pointer into PAB
       MOV  R0,R9             Save pointer
       AI   R9,-8             Adjust pointer to flag byte
       BLWP @VSBR             Read device name length
       MOVB R1,R3             Store it elsewhere
       SRL  R3,8              Make it a word value
       SETO R4                Initialize a counter
       LI   R2,NAMBUF         Point to NAMBUF
LNK$LP INC  R0                Point to next char of name
       INC  R4                Increment character counter
       C    R4,R3             End of name?
       JEQ  LNK$LN            Yes
       BLWP @VSBR             Read current character
       MOVB R1,*R2+           Move it to NAMBUF
       CB   R1,@DECMAL        Is it a decimal point?
       JNE  LNK$LP            No
LNK$LN MOV  R4,R4             Is name length zero?
       JEQ  LNKERR            Yes, error
       CI   R4,7              Is name length > 7?
       JGT  LNKERR            Yes, error
       CLR  @CRULST
       MOV  R4,@SCLEN-1       Store name length for search
       MOV  R4,@SAVLEN        Save device name length
       INC  R4                Adjust it
       A    R4,@SCNAME        Point to position after name
       MOV  @SCNAME,@SAVPAB   Save pointer into device name
*
*** Search ROM CROM GROM for DSR
*
SROM   LWPI GPLWS             Use GPL workspace to search
       CLR  R1                Version found of DSR etc.
       LI   R12,>0F00         Start over again
NOROM  MOV  R12,R12           Anything to turn off
       JEQ  NOOFF             No
       SBZ  0                 Yes, turn it off
NOOFF  AI   R12,>0100         Next ROM'S turn on
       CLR  @CRULST           Clear in case we're finished
       CI   R12,>2000         At the end
       JEQ  NODSR             No more ROMs to turn on
       MOV  R12,@CRULST       Save address of next CRU
       SBO  0                 Turn on ROM
       LI   R2,>4000          Start at beginning
       CB   *R2,@HAA          Is it a valid ROM?
       JNE  NOROM             No
       A    @TYPE,R2          Go to first pointer
       JMP SGO2
SGO    MOV  @SADDR,R2         Continue where we left off
       SBO  0                 Turn ROM back on
SGO2   MOV  *R2,R2            Is address a zero
       JEQ  NOROM             Yes, no program to look at
       MOV  R2,@SADDR         Remember where we go next
       INCT R2                Go to entry point
```

```
       MOV  *R2+,R9           Get entry address
*
*** See if name matches
*
       MOVB @SCLEN,R5         Get length as counter
       JEQ  NAME2             Zero length, don't do match
       CB   R5,*R2+           Does length match?
       JNE  SGO               No
       SRL  R5,8              Move to right place
       LI   R6,NAMBUF         Point to NAMBUF
NAME1  CB   *R6+,*R2+         Is character correct?
       JNE  SGO               No
       DEC  R5                More to look at?
       JNE  NAME1             Yes
NAME2  INC  R1                Next version found
       MOV  R1,@SAVVER        Save version number
       MOV  R9,@SAVENT        Save entry address
       MOV  R12,@SAVCRU       Save CRU address
       BL   *R9               Match, call subroutine
       JMP  SGO               Not right version
       SBZ  0                 Turn off ROM
       LWPI DLNKWS            Select DSRLNK workspace
       MOV  R9,R0             Point to flag byte in PAB
       BLWP @VSBR             Read flag byte
       SRL  R1,13             Just want the error flags
       JNE  IOERR             Error!
       RTWP
*
*** Error handling
*
NODSR  LWPI DLNKWS       Select DSRLNK workspace
LNKERR CLR  R1           Clear the error flags
IOERR  SWPB R1
       MOVB R1,*R13      Store error flags in calling R0
       SOCB @H20,R15     Indicate an error occured
       RTWP              Return to caller
**
** ^^^^^^^^^^^^ UTILROM ^^^^^ above ^^^^^^^^^^^^^^^^^^^^^^^^
**
**       COPY "DSK2.UTILRAM"
**
** vvvvvvvvvvvv UTILRAM vvvvv below vvvvvvvvvvvvvvvvvvvvvvvv
**
SVGPRT DATA 0            Save GPL return address
SAVCRU DATA 0            CRU address of peripheral
SAVENT DATA 0            Entry address of DSR
SAVLEN DATA 0            Save device name length
SAVPAB DATA 0            Ptr into device name in PAB
SAVVER DATA 0            Version number of DSR
NAMBUF DATA 0,0,0,0
*
*** General utility workspace registers (Overlaps next WS)
UTILWS DATA 0,0
       BYTE 0
R2LB   BYTE 0
*
*** DSR link routine workspace registers (Overlaps prev. WS)
DLNKWS DATA 0,0,0,0,0
TYPE   DATA 0,0,0,0,0,0,0,0,0,0,0
*
** ^^^^^^^^^^^^ UTILRAM ^^^^^ above ^^^^^^^^^^^^^^^^^^^^^^^^
**
*=========================================================
LINBUF BSS  40           BUFFER FOR SCROLLING
KEYCNT DATA -1           USED IN CURSOR FLASH LOGIC
CURCHR BSS  2            CHAR AT CURSOR POSITION
```

```
GRMSAV BSS  2           SAVE GROM ADDRESS DURING DSRLNK
INTACT DATA 0           NON-ZERO DURING INTERRUPT SERVICE
*
*=========================================================
*
       END
```

O.2 ASMSRC—Part 2 of FORTHSAVE

ASMSRC contains the resident portion of the TI Forth dictionary, which includes the routine that initializes the system. The source code that follows has been expanded into a single file:

```
**       COPY "DSK2.ASMSRC1"
**
TEMP0  EQU  0
TEMP1  EQU  1
TEMP2  EQU  2
TEMP3  EQU  3
TEMP4  EQU  4
TEMP5  EQU  5
TEMP6  EQU  6
TEMP7  EQU  7
U      EQU  8
SP     EQU  9
W      EQU  10
LINK   EQU  11
CRU    EQU  12
IP     EQU  13
R      EQU  14
NEXT   EQU  15
*
*****************************************************************
DODOES EQU  >832E
DOCOL  EQU  >8334
$NEXT  EQU  >833A
DOEXEC EQU  >833C
$SEMIS EQU  >8340
*****************************************************************
*
FF9900 LI   IP,COLD+2
       MOV  @$U0(TEMP1),U
       MOV  TEMP1,@$UCONS(U)
       MOV  @$UCONS(U),TEMP1
       MOV  @$S0(TEMP1),SP
       MOV  @$R0(TEMP1),R
       MOV  @$U0(TEMP1),U
       MOV  U,@$U0(U)
       LI   NEXT,$NEXT
       B    *NEXT
*
*
*** EXECUTE ***
       DATA >0
L1000  DATA >8745,>5845,>4355,>54C5
EXECUT DATA $+2
       MOV  *SP+,W
       B    @DOEXEC
*
*
*** LIT ***
       DATA L1000
```

```
L1001  DATA >834C,>49D4
LIT    DATA $+2
       DECT SP
       MOV  *IP+,*SP
       B    *NEXT
*
*
*** BRANCH ***
       DATA L1001
L1002  DATA >8642,>5241,>4E43,>48A0
BRANCH DATA $+2
BRAN2  A    *IP,IP
       B    *NEXT
*
*
*** 0BRANCH ***
       DATA L1002
L1003  DATA >8730,>4252,>414E,>43C8
ZBRAN  DATA $+2
       MOV  *SP+,TEMP1
       JEQ  ZBRAN1
       INCT IP
       B    *NEXT
ZBRAN1 A    *IP,IP
       B    *NEXT
*
*
*** (OF) ***
       DATA L1003
L1004  DATA >8428,>4F46,>29A0
POF    DATA $+2
       C    *SP+,*SP
       JNE  POF1
       INCT SP
       INCT IP
       B    *NEXT
POF1   A    *IP,IP
       B    *NEXT
*
*
*** (LOOP) ***
       DATA L1004
L1005  DATA >8628,>4C4F,>4F50,>29A0
PLOOP  DATA $+2
       INC  *R
       C    *R,@2(R)
       JLT  PLOOPA
       AI   R,4
       INCT IP
       B    *NEXT
PLOOPA A    *IP,IP
       B    *NEXT
*
*
*** (+LOOP) ***
       DATA L1005
L1006  DATA >8728,>2B4C,>4F4F,>50A9
PPLOOP DATA $+2
       MOV  *SP+,TEMP1
       A    TEMP1,*R
       MOV  TEMP1,TEMP1
       JLT  PLOOP2
PLOOP1 C    *R,@2(R)
       JLT  PLOOP3
       AI   R,4
       INCT IP
```

```
       B    *NEXT
PLOOP2 C    *R,@2(R)
       JGT  PLOOP3
       AI   R,4
       INCT IP
       B    *NEXT
PLOOP3 A    *IP,IP
       B    *NEXT
*
*
*** (DO) ***
       DATA L1006
L1007  DATA >8428,>444F,>29A0
PDO    DATA $+2
       AI   R,-4
       MOV  *SP+,*R
       MOV  *SP+,@2(R)
       B    *NEXT
*
*
*** I ***
       DATA L1007
L1008  DATA >81C9
I      DATA $+2
       DECT SP
       MOV  *R,*SP
       B    *NEXT
*
*
*** J ***
       DATA L1008
J1008  DATA >81CA
J      DATA $+2
       DECT SP
       MOV  @4(R),*SP
       B    *NEXT
*
*
*** DIGIT ***
       DATA J1008
L1009  DATA >8544,>4947,>49D4
DIGIT  DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP,TEMP2
       AI   TEMP2,->0030
       CI   TEMP2,10
       JL   DIGIT1
       AI   TEMP2,-7
       CI   TEMP2,10
       JHE  DIGIT1
DIGIT2 CLR  *SP
       B    *NEXT
DIGIT1 C    TEMP2,TEMP1
       JHE  DIGIT2
       MOV  TEMP2,*SP
       DECT  SP
       SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** (FIND) ***
       DATA L1009
L100A  DATA >8628,>4649,>4E44,>29A0
PFIND  DATA $+2
       MOV  *SP,TEMP1
```

```
       JEQ  PFIND4
PFIND1 MOV  TEMP1,TEMP2
       MOV  @2(SP),TEMP3
       MOVB *TEMP2+,W
       ANDI W,>3F00
       CB   W,*TEMP3+
       JNE  PFIND3
PFIND2 MOVB *TEMP2+,W
       JLT  PFIND5
       CB   W,*TEMP3+
       JEQ  PFIND2
PFIND3 MOV  @-2(TEMP1),TEMP1
       JNE  PFIND1
PFIND4 INCT SP
       CLR  *SP
       B    *NEXT
PFIND5 ANDI W,>7F00
       CB   W,*TEMP3
       JNE  PFIND3
       INCT TEMP2
       MOV  TEMP2,@2(SP)
       CLR  *SP
       MOVB *TEMP1,@1(SP)
       DECT SP
       SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** ENCLOSE ***
       DATA L100A
L100B  DATA >8745,>4E43,>4C4F,>53C5
ENCLOS DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP,TEMP2
       SWPB TEMP1
       SETO TEMP3
ENCL1  INC  TEMP3
       CB   TEMP1,*TEMP2+
       JEQ  ENCL1
       DEC  TEMP2
       AI   SP,-6
       MOV  TEMP3,@4(SP)
       MOV  TEMP3,*SP
       INC  TEMP3
       MOV  TEMP3,@2(SP)
       MOVB *TEMP2,W
       JNE  ENCL4
       B    *NEXT
ENCL4  INC  TEMP2
ENCL2  MOV  TEMP3,@2(SP)
       MOVB *TEMP2,W
       JEQ  ENCL3
       INC  TEMP3
       CB   TEMP1,*TEMP2+
       JNE  ENCL2
ENCL3  MOV  TEMP3,*SP
       B    *NEXT
*
*
*** kEY ***
       DATA L100B
L100C  DATA >836B,>45D9
KE     DATA $+2
       LI   TEMP1,-2
       MOV  @$SYS(U),LINK
```

```
       BL   *LINK
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** KEY ***
       DATA L100C
L100CX DATA >834B,>45D9
KEY    DATA DOCOL,KE,LIT,>7F,AND,SEMIS
*
*
*** KEY8 ***
       DATA L100CX
L100CY DATA >844B,>4559,>38A0
KEY8   DATA DOCOL,KE,SEMIS
*
*
*** EMIT ***
       DATA L100CY
L100D  DATA >8445,>4D49,>54A0
EMIT   DATA $+2
       MOV  *SP+,TEMP2
       ANDI TEMP2,>007F
       LI   TEMP1,-4
       MOV  @$SYS(U),LINK
       BL   *LINK
       INC  @$OUT(U)
       B    *NEXT
*
*
*** EMIT8 ***
       DATA L100D
L100DX DATA >8545,>4D49,>54B8
EMIT8  DATA $+2
       MOV  *SP+,TEMP2
       ANDI TEMP2,>00FF
       LI   TEMP1,-4
       MOV  @$SYS(U),LINK
       BL   *LINK
       INC  @$OUT(U)
       B    *NEXT
*
*
*** CR ***
       DATA L100DX
L100E  DATA >8243,>52A0
CR     DATA $+2
       LI   TEMP1,-6
       MOV  @$SYS(U),LINK
       BL   *LINK
       B    *NEXT
*
*
*** ?TERMINAL ***
       DATA L100E
L100F  DATA >893F,>5445,>524D,>494E,>41CC
QTERM  DATA $+2
       LI   TEMP1,-8
       MOV  @$SYS(U),LINK
       BL   *LINK
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
*
```

```
*** ?KEY ***
       DATA L100F
L1010  DATA >843F,>4B45,>59A0
QKEY   DATA $+2
       LI   TEMP1,-10
       MOV  @$SYS(U),LINK
       BL   *LINK
       ANDI TEMP0,>007F
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** ?KEY8 ***
       DATA L1010
L1010X DATA >853F,>4B45,>59B8
QKEY8  DATA $+2
       LI   TEMP1,-10
       MOV  @$SYS(U),LINK
       BL   *LINK
       ANDI TEMP0,>00FF
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** GOTOXY ***
       DATA L1010X
L1011  DATA >8647,>4F54,>4F58,>59A0
GOTOXY DATA $+2
       MOV  *SP+,TEMP3
       MOV  *SP+,TEMP2
       LI   TEMP1,-12
       MOV  @$SYS(U),LINK
       BL   *LINK
       B    *NEXT
*
*
*** WDISK ***
       DATA L1011
L1012  DATA >8557,>4449,>53CB
WDISK  DATA $+2
       LI   TEMP1,-14
       MOV  *SP+,TEMP3
       MOV  *SP+,TEMP2
       MOV  *SP,TEMP4
       MOV  @$SYS(U),LINK
       BL   *LINK
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** RDISK ***
       DATA L1012
L1013  DATA >8552,>4449,>53CB
RDISK  DATA $+2
       LI   TEMP1,-16
       MOV  *SP+,TEMP3
       MOV  *SP+,TEMP2
       MOV  *SP,TEMP4
       MOV  @$SYS(U),LINK
       BL   *LINK
       MOV  TEMP0,*SP
       B    *NEXT
*
*
```

```
*** DRIVE ***
       DATA L1013
L1014  DATA >8544,>5249,>56C5
DRIVE  DATA $+2
       MOV  *SP+,TEMP2
       LI   TEMP1,-18
       MOV  @$SYS(U),LINK
       BL   *LINK
       MOV  TEMP0,@$OFFST(U)
       B    *NEXT
*
*
*** CMOVE ***
       DATA L1014
L1015  DATA >8543,>4D4F,>56C5
CMOVE  DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,TEMP2
       MOV  *SP+,TEMP3
       MOV  TEMP1,TEMP1
       JEQ  CMOVE2
CMOVE1 MOVB *TEMP3+,*TEMP2+
       DEC  TEMP1
       JNE  CMOVE1
CMOVE2 B    *NEXT
*
*
*** MOVE ***
       DATA L1015
A1000  DATA >844D,>4F56,>45A0
MOVE   DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,TEMP2
       MOV  *SP+,TEMP3
       MOV  TEMP1,TEMP1
       JEQ  MOVE2
MOVE1  MOV  *TEMP3+,*TEMP2+
       DEC  TEMP1
       JNE  MOVE1
MOVE2  B    *NEXT
*
*
*** SWPB ***
       DATA A1000
A1001  DATA >8453,>5750,>42A0
SWPB   DATA $+2
       SWPB *SP
       B    *NEXT
*
*
*** SRL ***
       DATA A1001
A1002  DATA >8353,>52CC
SRL    DATA $+2
       MOV  *SP+,TEMP0
       MOV  *SP,TEMP1
       SRL  TEMP1,0
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** SLA ***
       DATA A1002
A1003  DATA >8353,>4CC1
SLA    DATA $+2
       MOV  *SP+,TEMP0
```

```
       MOV  *SP,TEMP1
       SLA  TEMP1,0
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** SRA ***
       DATA A1003
A1004  DATA >8353,>52C1
SRA    DATA $+2
       MOV  *SP+,TEMP0
       MOV  *SP,TEMP1
       SRA  TEMP1,0
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** SRC ***
       DATA A1004
A1005  DATA >8353,>52C3
SRC    DATA $+2
       MOV  *SP+,TEMP0
       MOV  *SP,TEMP1
       SRC  TEMP1,0
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** U* ***
       DATA A1005
L1016  DATA >8255,>2AA0
MULT   DATA $+2
       MOV  *SP+,TEMP2
       MPY  *SP,TEMP2
       MOV  TEMP3,*SP
       DECT SP
       MOV  TEMP2,*SP
       B    *NEXT
*
*
*** U/ ***
       DATA L1016
L1017  DATA >8255,>2FA0
DIV    DATA $+2
       MOV  @2(SP),TEMP2
       MOV  @4(SP),TEMP3
       DIV  *SP+,TEMP2
       MOV  TEMP2,*SP
       MOV  TEMP3,@2(SP)
       B    *NEXT
*
*
*** AND ***
       DATA L1017
L1018  DATA >8341,>4EC4
AND    DATA $+2
       INV  *SP
       SZC  *SP+,*SP
       B    *NEXT
*
*
*** OR ***
       DATA L1018
L1019  DATA >824F,>52A0
OR     DATA $+2
       SOC  *SP+,*SP
```

```
       B    *NEXT
*
*
*** XOR ***
       DATA L1019
L101A  DATA >8358,>4FD2
XOR    DATA $+2
       MOV  *SP+,TEMP1
       XOR  *SP,TEMP1
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** SP@ ***
       DATA L101A
L101B  DATA >8353,>50C0
SPAT   DATA $+2
       DECT SP
       MOV  SP,*SP
       INCT *SP
       B    *NEXT
*
*
*** SP! ***
       DATA L101B
L101C  DATA >8353,>50A1
SPSTOR DATA $+2
       MOV  @$S0(U),SP
       B    *NEXT
*
*
*** RP! ***
       DATA L101C
L101D  DATA >8352,>50A1
RSTOR  DATA $+2
       MOV  @$R0(U),R
       B    *NEXT
*
*
*** ;S ***
       DATA L101D
L101E  DATA >823B,>53A0
SEMIS  DATA $SEMIS
*
*
*** LEAVE ***
       DATA L101E
L101F  DATA >854C,>4541,>56C5
LEAVE  DATA $+2
       MOV  *R,@2(R)
       B    *NEXT
*
*
*** >R ***
       DATA L101F
L1020  DATA >823E,>52A0
TOR    DATA $+2
       DECT R
       MOV  *SP+,*R
       B    *NEXT
*
*
*** R> ***
       DATA L1020
L1021  DATA >8252,>3EA0
FROMR  DATA $+2
```

```
       DECT SP
       MOV  *R+,*SP
       B    *NEXT
*
*
*** R ***
       DATA L1021
L1022  DATA >81D2
RR     DATA $+2
       DECT SP
       MOV  *R,*SP
       B    *NEXT
*
*
*** U ***
       DATA L1022
L1023  DATA >81D5
UU     DATA $+2
       DECT SP
       MOV  U,*SP
       B    *NEXT
*
*
*** 0= ***
       DATA L1023
L1024  DATA >8230,>3DA0
ZEQU   DATA $+2
       MOV  *SP,TEMP1
       JEQ  ZEQUTR
       CLR  *SP
       B    *NEXT
ZEQUTR SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** 0< ***
       DATA L1024
L1025  DATA >8230,>3CA0
ZLESS  DATA $+2
       MOV  *SP,TEMP1
       JLT  PUSHTR
PUSHFL CLR  *SP
       B    *NEXT
PUSHTR SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** + ***
       DATA L1025
L1026  DATA >81AB
PLUS   DATA $+2
       A    *SP+,*SP
       B    *NEXT
*
*
*** D+ ***
       DATA L1026
L1027  DATA >8244,>2BA0
DPLUS  DATA $+2
       A    *SP+,@2(SP)
       A    *SP+,@2(SP)
       JNC  DPLUS1
       INC  *SP
DPLUS1 B    *NEXT
```

```
*
*
*** MINUS ***
       DATA L1027
L1028  DATA >854D,>494E,>55D3
MINUS  DATA $+2
       NEG  *SP
       B    *NEXT
*
*
*** DMINUS ***
       DATA L1028
L1029  DATA >8644,>4D49,>4E55,>53A0
DMINUS DATA $+2
       INV  @2(SP)
       INV  *SP
       INC  @2(SP)
       JNC  DM1
       INC  *SP
DM1    B    *NEXT
*
*
*** OVER ***
       DATA L1029
L102A  DATA >844F,>5645,>52A0
OVER   DATA $+2
       DECT SP
       MOV  @4(SP),*SP
       B    *NEXT
*
*
*** DROP ***
       DATA L102A
L102B  DATA >8444,>524F,>50A0
DROP   DATA $+2
       INCT SP
       B    *NEXT
*
*
*** SWAP ***
       DATA L102B
L102C  DATA >8453,>5741,>50A0
SWAP   DATA $+2
       MOV  *SP,TEMP1
       MOV  @2(SP),*SP
       MOV  TEMP1,@2(SP)
       B    *NEXT
*
*
*** DUP ***
       DATA L102C
L102D  DATA >8344,>55D0
DUP    DATA $+2
       DECT SP
       MOV  @2(SP),*SP
       B    *NEXT
*
*
*** +! ***
       DATA L102D
L102E  DATA >822B,>21A0
PSTORE DATA $+2
       MOV  *SP+,TEMP1
       A    *SP+,*TEMP1
       B    *NEXT
*
```

```
*
*** TOGGLE ***
       DATA L102E
L102F  DATA >8654,>4F47,>474C,>45A0
TOGGLE DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,TEMP2
       MOVB *TEMP2,TEMP3
       SWPB TEMP1
       XOR  TEMP1,TEMP3
       MOVB TEMP3,*TEMP2
       B    *NEXT
*
*
*** @ ***
       DATA L102F
L1030  DATA >81C0
AT     DATA $+2
       MOV  *SP,TEMP1
       MOV  *TEMP1,*SP
       B    *NEXT
*
*
*** C@ ***
       DATA L1030
L1031  DATA >8243,>40A0
CAT    DATA $+2
       MOV  *SP,TEMP1
       MOVB *TEMP1,TEMP1
       SRL  TEMP1,8
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** ! ***
       DATA L1031
L1032  DATA >81A1
STORE  DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,*TEMP1
       B    *NEXT
*
*
*** C! ***
       DATA L1032
L1033  DATA >8243,>21A0
CSTORE DATA $+2
       MOV  *SP+,TEMP1
       MOVB @1(SP),*TEMP1
       INCT SP
       B    *NEXT
*
*
*** 1+ ***
       DATA L1033
L1034  DATA >8231,>2BA0
ONEP   DATA $+2
       INC  *SP
       B    *NEXT
*
*
*** 2+ ***
       DATA L1034
L1035  DATA >8232,>2BA0
TWOP   DATA $+2
       INCT *SP
```

```
       B    *NEXT
*
*
*** 1- ***
       DATA L1035
L1035A DATA >8231,>2DA0
ONEM   DATA $+2
       DEC  *SP
       B    *NEXT
*
*
*** 2- ***
       DATA L1035A
L1035B DATA >8232,>2DA0
TWOM   DATA $+2
       DECT *SP
       B    *NEXT
*
*
*** - ***
       DATA L1035B
L1036  DATA >81AD
SUB    DATA $+2
       S    *SP+,*SP
       B    *NEXT
*
*
*** =CELLS ***
       DATA L1036
L1037  DATA >863D,>4345,>4C4C,>53A0
ECELLS DATA $+2
       MOV  *SP,TEMP1
       INC  TEMP1
       ANDI TEMP1,>FFFE
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** S->D ***
       DATA L1037
L1038  DATA >8453,>2D3E,>44A0
STOD   DATA $+2
       SETO TEMP1
       MOV  *SP,TEMP2
       JLT STOD1
       CLR  TEMP1
STOD1  DECT SP
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** ABS ***
       DATA L1038
L1039  DATA >8341,>42D3
ABS    DATA $+2
       ABS  *SP
       B    *NEXT
*
*
*** MIN ***
       DATA L1039
L103A  DATA >834D,>49CE
MIN    DATA $+2
       C    @2(SP),*SP
       JLT  MIN1
       MOV  *SP,@2(SP)
```

```
MIN1   INCT SP
       B    *NEXT
*
*
*** MAX ***
       DATA L103A
L103B  DATA >834D,>41D8
MAX    DATA $+2
       C    *SP,@2(SP)
       JLT  MAX1
       MOV  *SP,@2(SP)
MAX1   INCT SP
**
**       COPY "DSK2.ASMSRC2"
**
       B    *NEXT
*
*
*** U< ***
       DATA L103B
L103C  DATA >8255,>3CA0
ULESS  DATA $+2
       MOV  *SP+,TEMP2
       MOV  *SP,TEMP1
       CLR  *SP
       C    TEMP1,TEMP2
       JHE  ULESS1
       INC  *SP
ULESS1 B    *NEXT
*
*
*** 0 ***
       DATA L103C
L103F  DATA >81B0
ZERO   DATA DOCON,>0
*
*** 1 ***
       DATA L103F
L1040  DATA >81B1
ONE    DATA DOCON,>1
*
*** 2 ***
       DATA L1040
L1041  DATA >81B2
TWO    DATA DOCON,>2
*
*** 3 ***
       DATA L1041
L1042  DATA >81B3
THREE  DATA DOCON,>3
*
*** BL ***
       DATA L1042
L1043  DATA >8242,>4CA0
BL     DATA DOCON,>20
*
*** UCONS$ ***
       DATA L1043
L1044  DATA >8655,>434F,>4E53,>24A0
UCONS$ DATA DOUSER,>6
*
*** S0 ***
       DATA L1044
L1045  DATA >8253,>30A0
S0     DATA DOUSER,>8
*
```

```
*** R0 ***
       DATA L1045
L1046  DATA >8252,>30A0
RR0    DATA DOUSER,>A
*
*** U0 ***
       DATA L1046
L1047  DATA >8255,>30A0
U0     DATA DOUSER,>C
*
*** TIB ***
       DATA L1047
L1048  DATA >8354,>49C2
TIB    DATA DOUSER,>E
*
*** WIDTH ***
       DATA L1048
L1049  DATA >8557,>4944,>54C8
WIDTH  DATA DOUSER,>10
*
*** DP ***
       DATA L1049
L104A  DATA >8244,>50A0
DP     DATA DOUSER,>12
*
*** SYS$ ***
       DATA L104A
L104B  DATA >8453,>5953,>24A0
SYS$   DATA DOUSER,>14
*
*** CURPOS ***
       DATA L104B
L104C  DATA >8643,>5552,>504F,>53A0
TERM$  DATA DOUSER,>16
*
*** INTLNK ***
       DATA L104C
L104D  DATA >8649,>4E54,>4C4E,>4BA0
DISK$  DATA DOUSER,>18
*
*** WARNING ***
       DATA L104D
L104E  DATA >8757,>4152,>4E49,>4EC7
WARNIN DATA DOUSER,>1A
*
*** C/L$ ***
       DATA L104E
L104F  DATA >8443,>2F4C,>24A0
CL$    DATA DOUSER,>1C
*
*** FIRST$ ***
       DATA L104F
L1050  DATA >8646,>4952,>5354,>24A0
FIRST$ DATA DOUSER,>1E
*
*** LIMIT$ ***
       DATA L1050
L1051  DATA >864C,>494D,>4954,>24A0
LIMIT$ DATA DOUSER,>20
*
*** B/BUF$ ***
       DATA L1051
L1052  DATA >8642,>2F42,>5546,>24A0
BBUF$  DATA DOUSER,>22
*
*** B/SCR$ ***
```

```
       DATA L1052
L1053  DATA >8642,>2F53,>4352,>24A0
BSCR$  DATA DOUSER,>24
*
*** DISK_LO ***
       DATA L1053
X0001  DATA >8744,>4953,>4B5F,>4CCF
       DATA DOUSER,>26
*
*** DISK_HI ***
       DATA X0001
X0002  DATA >8744,>4953,>4B5F,>48C9
       DATA DOUSER,>28
*
*** DISK_SIZE ***
       DATA X0002
X0003  DATA >8944,>4953,>4B5F,>5349,>5AC5
       DATA DOUSER,>2A
*
*** DISK_BUF ***
       DATA X0003
X0004  DATA >8844,>4953,>4B5F,>4255,>46A0
       DATA DOUSER,>2C
*
*** PABS ***
       DATA X0004
X0005  DATA >8450,>4142,>53A0
       DATA DOUSER,>2E
*
*** SCRN_WIDTH ***
       DATA X0005
X0006  DATA >8A53,>4352,>4E5F,>5749,>4454,>48A0
       DATA DOUSER,>30
*
*** SCRN_START ***
       DATA X0006
X0007  DATA >8A53,>4352,>4E5F,>5354,>4152,>54A0
       DATA DOUSER,>32
*
*** SCRN_END ***
       DATA X0007
X0008  DATA >8853,>4352,>4E5F,>454E,>44A0
       DATA DOUSER,>34
*
*** ISR ***
       DATA X0008
X0009  DATA >8349,>53D2
       DATA DOUSER,>36
*
*** ALTIN ***
       DATA X0009
X000A  DATA >8541,>4C54,>49CE
       DATA DOUSER,>38
*
*** ALTOUT ***
       DATA X000A
X000B  DATA >8641,>4C54,>4F55,>54A0
       DATA DOUSER,>3A
*
*** FENCE ***
       DATA X000B
L1054  DATA >8546,>454E,>43C5
FENCE  DATA DOUSER,>3C
*
*** BLK ***
       DATA L1054
```

```
L1055  DATA >8342,>4CCB
BLK    DATA DOUSER,>3E
*
*** IN ***
       DATA L1055
L1056  DATA >8249,>4EA0
IN     DATA DOUSER,>40
*
*** OUT ***
       DATA L1056
L1057  DATA >834F,>55D4
OUT    DATA DOUSER,>42
*
*** SCR ***
       DATA L1057
L1058  DATA >8353,>43D2
SCR    DATA DOUSER,>44
*
*** OFFSET ***
       DATA L1058
L1059  DATA >864F,>4646,>5345,>54A0
OFFSET DATA DOUSER,>46
*
*** CONTEXT ***
       DATA L1059
L105A  DATA >8743,>4F4E,>5445,>58D4
CONTEX DATA DOUSER,>48
*
*** CURRENT ***
       DATA L105A
L105B  DATA >8743,>5552,>5245,>4ED4
CURREN DATA DOUSER,>4A
*
*** STATE ***
       DATA L105B
L105C  DATA >8553,>5441,>54C5
STATE  DATA DOUSER,>4C
*
*** BASE ***
       DATA L105C
L105D  DATA >8442,>4153,>45A0
BASE   DATA DOUSER,>4E
*
*** DPL ***
       DATA L105D
L105E  DATA >8344,>50CC
DPL    DATA DOUSER,>50
*
*** FLD ***
       DATA L105E
L105F  DATA >8346,>4CC4
FLD    DATA DOUSER,>52
*
*** CSP ***
       DATA L105F
L1060  DATA >8343,>53D0
CSP    DATA DOUSER,>54
*
*** R# ***
       DATA L1060
L1061  DATA >8252,>23A0
RNUM   DATA DOUSER,>56
*
*** HLD ***
       DATA L1061
L1062  DATA >8348,>4CC4
```

```
HLD    DATA DOUSER,>58
*
*** USE ***
       DATA L1062
L1063  DATA >8355,>53C5
USE    DATA DOUSER,>5A
*
*** PREV ***
       DATA L1063
L1064  DATA >8450,>5245,>56A0
PREV   DATA DOUSER,>5C
*
*** FORTH_LINK ***
       DATA L1064
L1065  DATA >8A46,>4F52,>5448,>5F4C,>494E,>4BA0
FORTHL DATA DOUSER,>62
*
*** ECOUNT ***
       DATA L1065
L1066  DATA >8645,>434F,>554E,>54A0
ECOUNT DATA DOUSER,>64
*
*** VOC-LINK ***
       DATA L1066
L1066X DATA >8856,>4F43,>2D4C,>494E,>4BA0
VLINK  DATA DOUSER,>66
*
*
       DORG 0
UBASE  BSS  6             BASE OF USER VARIABLES
$UCONS BSS  2             06 USER UCONS
$S0    BSS  2             08 USER S0
$R0    BSS  2             0A USER R0 { R0$
$U0    BSS  2             0C USER U0
       BSS  2             0E USER TIB
       BSS  2             10 USER WIDTH
       BSS  2             12 USER DP
$SYS   BSS  2             14 USER SYS$
CURPO$ BSS  2             16 USER CURPOS
$INTLK BSS  2             18 USER INTLNK
       BSS  2             1A USER WARNING
       BSS  2             1C USER C/L$ { CL$
       BSS  2             1E USER FIRST$
       BSS  2             20 USER LIMIT$
       BSS  2             22 USER B/BUF$ { BBUF$
       BSS  2             24 USER B/SCR$ { BSCR$
       BSS  2             26 USER DISK_LO
       BSS  2             28 USER DISK_HI
       BSS  2             2A USER DISK_SIZE
       BSS  2             2C USER DISK_BUF
       BSS  2             2E USER PABS
       BSS  2             30 USER SCRN_WIDTH
       BSS  2             32 USER SCRN_START
       BSS  2             34 USER SCRN_END
       BSS  2             36 USER ISR
       BSS  2             38 USER ALTIN
       BSS  2             3A USER ALTOUT
ULNGTH EQU  $
       BSS  2             3C USER FENCE
       BSS  2             3E USER BLK
       BSS  2             40 USER IN
$OUT   BSS  2             42 USER OUT
       BSS  2             44 USER SCR
$OFFST BSS  2             46 USER OFFSET
       BSS  2             48 USER CONTEXT
       BSS  2             4A USER CURRENT
```

```
       BSS  2             4C USER STATE
       BSS  2             4E USER BASE
       BSS  2             50 USER DPL
       BSS  2             52 USER FLD
       BSS  2             54 USER CSP
       BSS  2             56 USER R# { RNUM
       BSS  2             58 USER HLD
       BSS  2             5A USER USE
       BSS  2             5C USER PREV
       BSS  2             5E
       BSS  2             60
       BSS  2             62 USER FORTH_LINK
       BSS  2             64 USER ECOUNT
UMAX   BSS  0
       RORG
*
*** C/L ***
       DATA L1066X
L1067  DATA >8343,>2FCC
CSL    DATA DOCOL,CL$,AT,SEMIS
*
*** B/BUF ***
       DATA L1067
L1068  DATA >8542,>2F42,>55C6
BSLBUF DATA DOCOL,BBUF$,AT,SEMIS
*
*** B/SCR ***
       DATA L1068
L1069  DATA >8542,>2F53,>43D2
BSLSCR DATA DOCOL,BSCR$,AT,SEMIS
*
*** FIRST ***
       DATA L1069
L106A  DATA >8546,>4952,>53D4
FIRST  DATA DOCOL,FIRST$,AT,SEMIS
*
*** LIMIT ***
       DATA L106A
L106B  DATA >854C,>494D,>49D4
LIMIT  DATA DOCOL,LIMIT$,AT,SEMIS
*
*** DR0 ***
       DATA L106B
L106C  DATA >8344,>52B0
DR0    DATA DOCOL,ZERO,DRIVE,SEMIS
*
*** DR1 ***
       DATA L106C
L106X  DATA >8344,>52B1
DR1    DATA DOCOL,ONE,DRIVE,SEMIS
*
*** DR2 ***
       DATA L106X
L106Y  DATA >8344,>52B2
DR2    DATA DOCOL,TWO,DRIVE,SEMIS
*
*** HERE ***
       DATA L106Y
L106D  DATA >8448,>4552,>45A0
HERE   DATA DOCOL,DP,AT,SEMIS
*
*** ALLOT ***
       DATA L106D
L106E  DATA >8541,>4C4C,>4FD4
ALLOT  DATA DOCOL,SPAT,OVER,HERE,PLUS,LIT,>80
       DATA PLUS,ULESS,TWO,QERROR,DP,PSTORE
```

```
       DATA SEMIS
*
*** , ***
       DATA L106E
L106F  DATA >81AC
COMMA  DATA DOCOL,HERE,STORE,TWO,ALLOT,SEMIS
*
*** C, ***
       DATA L106F
L1070  DATA >8243,>2CA0
CCOMMA DATA DOCOL,HERE,CSTORE,ONE,ALLOT,SEMIS
*
*** = ***
       DATA L1070
L1071  DATA >81BD
EQUAL  DATA DOCOL,SUB,ZEQU,SEMIS
*
*** < ***
       DATA L1071
L1072  DATA >81BC
LESS   DATA $+2
       CLR  TEMP1
       C    *SP+,*SP
       JLT  LESS1
       JEQ  LESS1
       INC  TEMP1
LESS1  MOV  TEMP1,*SP
       B    *NEXT
*
*** > ***
       DATA L1072
L1073  DATA >81BE
GREAT  DATA DOCOL,SWAP,LESS,SEMIS
*
*** ROT ***
       DATA L1073
L1074  DATA >8352,>4FD4
ROT    DATA DOCOL,TOR,SWAP,FROMR,SWAP,SEMIS
*
*** SPACE ***
       DATA L1074
L1075  DATA >8553,>5041,>43C5
SPACE  DATA DOCOL,BL,EMIT,SEMIS
*
*** -DUP ***
       DATA L1075
L1076  DATA >842D,>4455,>50A0
DDUP   DATA DOCOL,DUP,ZBRAN,L1077-$,DUP
L1077  DATA SEMIS
*
*** TRAVERSE ***
       DATA L1076
L1078  DATA >8854,>5241,>5645,>5253,>45A0
TRAVER DATA DOCOL,SWAP
L1079  DATA OVER,PLUS,LIT,>7F,OVER,CAT,LESS,ZBRAN
       DATA L1079-$,SWAP,DROP,SEMIS
*
*** CFA ***
       DATA L1078
L107A  DATA >8343,>46C1
CFA    DATA DOCOL,TWOM,SEMIS
*
*** NFA ***
       DATA L107A
L107B  DATA >834E,>46C1
NFA    DATA DOCOL,THREE,SUB,LIT,>FFFF,TRAVER,SEMIS
```

```
*
*** PFA ***
       DATA L107B
L107C  DATA >8350,>46C1
PFA    DATA DOCOL,ONE,TRAVER,THREE,PLUS,SEMIS
*
*** LFA ***
       DATA L107C
L107D  DATA >834C,>46C1
LFA    DATA DOCOL,NFA,TWOM,SEMIS
*
*** LATEST ***
       DATA L107D
L107E  DATA >864C,>4154,>4553,>54A0
LATEST DATA DOCOL,CURREN,AT,AT,SEMIS
*
*** !CSP ***
       DATA L107E
L107F  DATA >8421,>4353,>50A0
STRCSP DATA DOCOL,SPAT,CSP,STORE,SEMIS
*
*** ?ERROR ***
       DATA L107F
L1080  DATA >863F,>4552,>524F,>52A0
QERROR DATA DOCOL,SWAP,ZBRAN,L1081-$,ERROR,BRANCH
       DATA L1082-$
L1081  DATA DROP
L1082  DATA SEMIS
*
*** ?COMP ***
       DATA L1080
L1083  DATA >853F,>434F,>4DD0
QCOMP  DATA DOCOL,STATE,AT,ZEQU,LIT,>11,QERROR
       DATA SEMIS
*
*** ?EXEC ***
       DATA L1083
L1084  DATA >853F,>4558,>45C3
QEXEC  DATA DOCOL,STATE,AT,LIT,>12,QERROR,SEMIS
*
*** ?PAIRS ***
       DATA L1084
L1085  DATA >863F,>5041,>4952,>53A0
QPAIRS DATA DOCOL,SUB,LIT,>13,QERROR,SEMIS
*
*** ?CSP ***
       DATA L1085
L1086  DATA >843F,>4353,>50A0
QCSP   DATA DOCOL,SPAT,CSP,AT,SUB,LIT,>14,QERROR
       DATA SEMIS
*
*** ?LOADING ***
       DATA L1086
L1087  DATA >883F,>4C4F,>4144,>494E,>47A0
QLOADI DATA DOCOL,BLK,AT,ZEQU,LIT,>16,QERROR,SEMIS
*
*** COMPILE ***
       DATA L1087
L1088  DATA >8743,>4F4D,>5049,>4CC5
COMPIL DATA DOCOL,QCOMP,FROMR,DUP,TWOP,TOR,AT,COMMA
       DATA SEMIS
*
*** [ ***
       DATA L1088
L1089  DATA >C1DB
LBRCKT DATA DOCOL,ZERO,STATE,STORE,SEMIS
```

```
*
*** ] ***
       DATA L1089
L108A  DATA >81DD
RBRCKT DATA DOCOL,LIT,>C0,STATE,STORE,SEMIS
*
*** SMUDGE ***
       DATA L108A
L108B  DATA >8653,>4D55,>4447,>45A0
SMUDGE DATA DOCOL,LATEST,LIT,>20,TOGGLE,SEMIS
*
*** HEX ***
       DATA L108B
L108C  DATA >8348,>45D8
HEX    DATA DOCOL,LIT,>10,BASE,STORE,SEMIS
*
*** DECIMAL ***
       DATA L108C
L108D  DATA >8744,>4543,>494D,>41CC
DECIMA DATA DOCOL,LIT,>A,BASE,STORE,SEMIS
*
*** COUNT ***
       DATA L108D
L108E  DATA >8543,>4F55,>4ED4
COUNT  DATA DOCOL,DUP,ONEP,SWAP,CAT,SEMIS
*
*** TYPE ***
       DATA L108E
L108F  DATA >8454,>5950,>45A0
TYPE   DATA DOCOL,DDUP,ZBRAN,L1090-$,ZERO,PDO
L1091  DATA DUP,CAT,EMIT,ONEP,PLOOP,L1091-$
L1090  DATA DROP,SEMIS
*
*** -TRAILING ***
       DATA L108F
L1092  DATA >892D,>5452,>4149,>4C49,>4EC7
DTRAIL DATA DOCOL,DUP,ZERO,PDO
L1093  DATA OVER,OVER,PLUS,ONEM,CAT,BL,SUB,ZBRAN
       DATA L1094-$,LEAVE,BRANCH,L1095-$
L1094  DATA ONEM
L1095  DATA PLOOP,L1093-$,SEMIS
*
*** ?STACK ***
       DATA L1092
L1096  DATA >863F,>5354,>4143,>4BA0
QSTACK DATA DOCOL,SPAT,S0,AT,SWAP,ULESS,ONE,QERROR
       DATA SPAT,HERE,LIT,>80,PLUS,ULESS
       DATA LIT,>7
       DATA QERROR,SEMIS
*
*** EXPECT ***
       DATA L1096
L1097  DATA >8645,>5850,>4543,>54A0
EXPECT DATA DOCOL,ZERO,PDO
L1098  DATA KEY,DUP,LIT,>D,EQUAL,ZBRAN,L1099-$
       DATA DROP,SPACE,LEAVE,ZERO,BRANCH,L109A-$
L1099  DATA DUP,LIT,>8,EQUAL,ZBRAN,L109B-$,DROP
       DATA I,ZEQU,ZBRAN,L109C-$,LIT,>7,EMIT,ZERO
       DATA BRANCH,L109D-$
L109C  DATA LIT,>8,EMIT,FROMR,ONEM,TOR,ONEM
       DATA ZERO
L109D  DATA BRANCH,L109E-$
L109B  DATA DUP,EMIT,OVER,CSTORE,ONEP,ONE
L109E
L109A  DATA PPLOOP,L1098-$,ZERO,SWAP,OVER,OVER
       DATA CSTORE,ONEP,CSTORE,SEMIS
```

```
*
*** QUERY ***
       DATA L1097
L109F  DATA >8551,>5545,>52D9
QUERY  DATA DOCOL,TIB,AT,LIT,>50,EXPECT,ZERO,IN
       DATA STORE,SEMIS
*
*** FILL ***
       DATA L109F
L10A0  DATA >8446,>494C,>4CA0
FILL   DATA DOCOL,SWAP,TOR,OVER,CSTORE,DUP,ONEP
       DATA FROMR,ONEM,CMOVE,SEMIS
*
*** ERASE ***
       DATA L10A0
L10A1  DATA >8545,>5241,>53C5
ERASE  DATA DOCOL,ZERO,FILL,SEMIS
*
*** BLANKS ***
       DATA L10A1
L10A2  DATA >8642,>4C41,>4E4B,>53A0
BLANKS DATA DOCOL,BL,FILL,SEMIS
*
*** HOLD ***
       DATA L10A2
L10A3  DATA >8448,>4F4C,>44A0
HOLD   DATA DOCOL,LIT,>FFFF,HLD,PSTORE,HLD,AT,CSTORE
       DATA SEMIS
*
*** PAD ***
       DATA L10A3
L10A4  DATA >8350,>41C4
PAD    DATA DOCOL,HERE,LIT,>44,PLUS,SEMIS
*
*** WORD ***
       DATA L10A4
L10A5  DATA >8457,>4F52,>44A0
WORD   DATA DOCOL,BLK,AT,ZBRAN,L10A6-$,BLK,AT,BLOCK
       DATA BRANCH,L10A7-$
L10A6  DATA TIB,AT
L10A7  DATA IN,AT,PLUS,SWAP,ENCLOS,HERE,LIT,>22
       DATA BLANKS,IN,PSTORE,OVER,SUB,DUP,TOR,HERE
       DATA CSTORE,PLUS,HERE,ONEP,FROMR,CMOVE,SEMIS
*
*** (.") ***
       DATA L10A5
L10A8  DATA >8428,>2E22,>29A0
PTYPE  DATA DOCOL,RR,COUNT,DUP,ONEP,ECELLS,FROMR
       DATA PLUS,TOR,TYPE,SEMIS
*
*** ." ***
       DATA L10A8
L10A9  DATA >C22E,>22A0
STRNG  DATA DOCOL,LIT,>22,STATE,AT,ZBRAN,L10AA-$
       DATA COMPIL,PTYPE,WORD,HERE,CAT,ONEP,ECELLS
       DATA ALLOT,BRANCH,L10AB-$
L10AA  DATA WORD,HERE,COUNT,TYPE
L10AB  DATA SEMIS
*
*** (NUMBER) ***
       DATA L10A9
L10AC  DATA >8828,>4E55,>4D42,>4552,>29A0
PNUMBR DATA DOCOL
L10AD  DATA ONEP,DUP,TOR,CAT,BASE,AT,DIGIT,ZBRAN
       DATA L10AE-$,SWAP,BASE,AT,MULT,DROP,ROT
       DATA BASE,AT,MULT,DPLUS,DPL,AT,ONEP,ZBRAN
```

```
       DATA L10AF-$,ONE,DPL,PSTORE
L10AF  DATA FROMR,BRANCH,L10AD-$
L10AE  DATA FROMR,SEMIS
*
*** NUMBER ***
       DATA L10AC
L10B0  DATA >864E,>554D,>4245,>52A0
NUMBER DATA DOCOL,ZERO,ZERO,ROT,DUP,ONEP,CAT,LIT
       DATA >2D,EQUAL,DUP,TOR,PLUS,LIT,>FFFF
L10B1  DATA DPL,STORE,PNUMBR,DUP,CAT,BL,SUB,ZBRAN
       DATA L10B2-$,DUP,CAT,LIT,>2E,SUB,ZERO,QERROR
       DATA ZERO,BRANCH,L10B1-$
L10B2  DATA DROP,FROMR,ZBRAN,L10B3-$,DMINUS
L10B3  DATA SEMIS
*
*** -FIND ***
       DATA L10B0
L10B4  DATA >852D,>4649,>4EC4
DFIND  DATA DOCOL,BL,WORD,HERE,CONTEX,AT,AT,PFIND
       DATA DUP,ZEQU,ZBRAN,L10B5-$,DROP,HERE,LATEST
       DATA PFIND
L10B5  DATA SEMIS
*
*** (ABORT) ***
       DATA L10B4
L10B6  DATA >8728,>4142,>4F52,>54A9
PABORT DATA DOCOL,ABORT,SEMIS
*
*** ERROR ***
       DATA L10B6
L10B7  DATA >8545,>5252,>4FD2
ERROR  DATA DOCOL,WARNIN,AT,ZLESS,ZBRAN,L10B8-$
       DATA PABORT,BRANCH,L10B9-$
L10B8  DATA ECOUNT,AT,ZEQU,ZBRAN,L10BA-$,ONE,ECOUNT
       DATA STORE,HERE,COUNT,TYPE,PTYPE,>420,>203F
       DATA >2020,MESSAG
L10BA
L10B9  DATA ZERO,ECOUNT,STORE,SPSTOR,IN,AT,BLK
       DATA AT,QUIT,SEMIS
*
*** ID. ***
       DATA L10B7
L10BB  DATA >8349,>44AE
IDDOT  DATA DOCOL,PAD,LIT,>20,LIT,>5F,FILL,DUP
       DATA ONE,TRAVER,OVER,SUB,DUP,TOR,ONEP,PAD
       DATA SWAP,CMOVE,PAD,FROMR,PLUS,LIT,>80,TOGGLE
       DATA PAD,COUNT,LIT,>1F,AND,TYPE,SPACE,SEMIS
*
*** CREATE ***
       DATA L10BB
L10BC  DATA >8643,>5245,>4154,>45A0
CREATE DATA DOCOL,HERE,ECELLS,DP,STORE
       DATA LATEST,COMMA,DFIND,ZBRAN,L10BD-$
       DATA DROP,NFA,IDDOT,LIT,>4,MESSAG,SPACE
L10BD  DATA HERE,DUP,CAT,WIDTH,AT,MIN,ONEP,ECELLS
       DATA ALLOT,DUP,LIT,>A0,TOGGLE,HERE,ONEM
       DATA LIT,>80,TOGGLE,CURREN,AT,STORE,HERE
       DATA TWOP,COMMA,SEMIS
*
*** [COMPILE] ***
       DATA L10BC
L10BE  DATA >C95B,>434F,>4D50,>494C,>45DD
BCOMPI DATA DOCOL,DFIND,ZEQU,ZERO,QERROR,DROP,CFA
       DATA COMMA,SEMIS
*
*** LITERAL ***
```

```
       DATA L10BE
L10BF  DATA >C74C,>4954,>4552,>41CC
LITERA DATA DOCOL,STATE,AT,ZBRAN,L10C0-$,COMPIL
       DATA LIT,COMMA
L10C0  DATA SEMIS
*
*** DLITERAL ***
       DATA L10BF
L10C1  DATA >C844,>4C49,>5445,>5241,>4CA0
DLITER DATA DOCOL,STATE,AT,ZBRAN,L10C2-$,SWAP,LITERA
       DATA LITERA
L10C2  DATA SEMIS
*
*** INTERPRET ***
       DATA L10C1
L10C3  DATA >8949,>4E54,>4552,>5052,>45D4
INTERP DATA DOCOL
L10C4  DATA DFIND,ZBRAN,L10C5-$,STATE,AT,LESS,ZBRAN
       DATA L10C6-$,CFA,COMMA,BRANCH,L10C7-$
L10C6  DATA CFA,EXECUT
L10C7  DATA QSTACK,BRANCH,L10C8-$
L10C5  DATA HERE,NUMBER,DPL,AT,ONEP,ZBRAN,L10C9-$
       DATA DLITER,BRANCH,L10CA-$
L10C9  DATA DROP,LITERA
L10CA  DATA QSTACK
L10C8  DATA BRANCH,L10C4-$,SEMIS
*
*** IMMEDIATE ***
       DATA L10C3
L10CB  DATA >8949,>4D4D,>4544,>4941,>54C5
IMMEDI DATA DOCOL,LATEST,LIT,>40,TOGGLE,SEMIS
*
*** ( ***
       DATA L10CB
L10CC  DATA >C1A8
PAREN  DATA DOCOL,LIT,>29,WORD,SEMIS
*
*** FORTH ***
       DATA L10CC
L10CD  DATA >C546,>4F52,>54C8
FORTH  DATA DOCOL,FORTHL,LIT,>4,SUB,CONTEX,STORE
       DATA SEMIS
*
*** DEFINITIONS ***
**
**       COPY "DSK2.ASMSRC3"
**
       DATA L10CD
L10CE  DATA >8B44,>4546,>494E,>4954,>494F,>4ED3
DEFINI DATA DOCOL,CONTEX,AT,CURREN,STORE,SEMIS
*
*** QUIT ***
       DATA L10CE
L10CF  DATA >8451,>5549,>54A0
QUIT   DATA DOCOL,ZERO,BLK,STORE,LBRCKT
L10D0  DATA RSTOR,CR,QUERY,INTERP,STATE,AT,ZEQU
       DATA ZBRAN,L10D1-$,PTYPE,>320,>6F6B
L10D1  DATA BRANCH,L10D0-$,SEMIS
*
*** ABORT ***
       DATA L10CF
L10D2  DATA >8541,>424F,>52D4
ABORT  DATA DOCOL,SPSTOR,DECIMA,ZERO,ECOUNT,STORE
       DATA CR,PTYPE,>854,>4920
       DATA >464F,>5254,>4820,FORTH,DEFINI,QUIT
       DATA SEMIS
```

```
*
*** +- ***
       DATA L10D2
L10D3  DATA >822B,>2DA0
PM     DATA DOCOL,ZLESS,ZBRAN,L10D4-$,MINUS
L10D4  DATA SEMIS
*
*** D+- ***
       DATA L10D3
L10D5  DATA >8344,>2BAD
DPM    DATA DOCOL,ZLESS,ZBRAN,L10D6-$,DMINUS
L10D6  DATA SEMIS
*
*** DABS ***
       DATA L10D5
L10D7  DATA >8444,>4142,>53A0
DABS   DATA DOCOL,DUP,DPM,SEMIS
*
*** M* ***
       DATA L10D7
L10D8  DATA >824D,>2AA0
MSTAR  DATA DOCOL,OVER,OVER,XOR,TOR,ABS,SWAP,ABS
       DATA MULT,FROMR,DPM,SEMIS
*
*** M/ ***
       DATA L10D8
L10D9  DATA >824D,>2FA0
MSLASH DATA DOCOL,OVER,TOR,TOR,DABS,RR,ABS,DIV
       DATA FROMR,RR,XOR,PM,SWAP,FROMR,PM,SWAP
       DATA SEMIS
*
*** * ***
       DATA L10D9
L10DA  DATA >81AA
TIMES  DATA DOCOL,MULT,DROP,SEMIS
*
*** /MOD ***
       DATA L10DA
L10DB  DATA >842F,>4D4F,>44A0
DMOD   DATA DOCOL,TOR,STOD,FROMR,MSLASH,SEMIS
*
*** / ***
       DATA L10DB
L10DC  DATA >81AF
DDIV   DATA DOCOL,DMOD,SWAP,DROP,SEMIS
*
*** MOD ***
       DATA L10DC
L10DD  DATA >834D,>4FC4
MOD    DATA DOCOL,DMOD,DROP,SEMIS
*
*** */MOD ***
       DATA L10DD
L10DE  DATA >852A,>2F4D,>4FC4
MDMOD  DATA DOCOL,TOR,MSTAR,FROMR,MSLASH,SEMIS
*
*** */ ***
       DATA L10DE
L10DF  DATA >822A,>2FA0
MD     DATA DOCOL,MDMOD,SWAP,DROP,SEMIS
*
*** M/MOD ***
       DATA L10DF
L10E0  DATA >854D,>2F4D,>4FC4
MSLMOD DATA DOCOL,TOR,ZERO,RR,DIV,FROMR,SWAP,TOR
       DATA DIV,FROMR,SEMIS
```

```
*
*** SPACES ***
       DATA L10E0
L10E1  DATA >8653,>5041,>4345,>53A0
SPACES DATA DOCOL,ZERO,MAX,DDUP,ZBRAN,L10E2-$,ZERO
       DATA PDO
L10E3  DATA SPACE,PLOOP,L10E3-$
L10E2  DATA SEMIS
*
*** <# ***
       DATA L10E1
L10E4  DATA >823C,>23A0
STRTCN DATA DOCOL,PAD,HLD,STORE,SEMIS
*
*** #> ***
       DATA L10E4
L10E5  DATA >8223,>3EA0
STOPCN DATA DOCOL,DROP,DROP,HLD,AT,PAD,OVER,SUB
       DATA SEMIS
*
*** SIGN ***
       DATA L10E5
L10E6  DATA >8453,>4947,>4EA0
SIGN   DATA DOCOL,ROT,ZLESS,ZBRAN,L10E7-$,LIT,>2D
       DATA HOLD
L10E7  DATA SEMIS
*
*** # ***
       DATA L10E6
L10E8  DATA >81A3
NUMSGN DATA DOCOL,PAD,HLD,AT,SUB,DPL,AT,EQUAL,ZBRAN
       DATA L10E9-$,LIT,>2E,HOLD
L10E9  DATA BASE,AT,MSLMOD,ROT,LIT,>9,OVER,LESS
       DATA ZBRAN,L10EA-$,LIT,>7,PLUS
L10EA  DATA LIT,>30,PLUS,HOLD,SEMIS
*
*** #S ***
       DATA L10E8
L10EB  DATA >8223,>53A0
NUMS   DATA DOCOL
L10EC  DATA NUMSGN,OVER,OVER,OR,ZEQU,ZBRAN,L10EC-$
       DATA SEMIS
*
*** D.R ***
       DATA L10EB
L10ED  DATA >8344,>2ED2
DDOTR  DATA DOCOL,TOR,SWAP,OVER,DABS,STRTCN,NUMS
       DATA SIGN,STOPCN,FROMR,OVER,SUB,SPACES,TYPE
       DATA SEMIS
*
*** D. ***
       DATA L10ED
L10EE  DATA >8244,>2EA0
DDOT   DATA DOCOL,ZERO,DDOTR,SPACE,SEMIS
*
*** .R ***
       DATA L10EE
L10EF  DATA >822E,>52A0
DOTR   DATA DOCOL,TOR,STOD,FROMR,DDOTR,SEMIS
*
*** . ***
       DATA L10EF
L10F0  DATA >81AE
DOT    DATA DOCOL,STOD,DDOT,SEMIS
*
*** ? ***
```

```
       DATA L10F0
L10F1  DATA >81BF
QMARK  DATA DOCOL,AT,DOT,SEMIS
*
*** UD.R ***
       DATA L10F1
L10F2  DATA >8455,>442E,>52A0
UDDOTR DATA DOCOL,TOR,STRTCN,NUMS,STOPCN,FROMR
       DATA OVER,SUB,SPACES,TYPE,SEMIS
*
*** UD. ***
       DATA L10F2
L10F3  DATA >8355,>44AE
UDDOT  DATA DOCOL,ZERO,UDDOTR,SPACE,SEMIS
*
*** U.R ***
       DATA L10F3
L10F4  DATA >8355,>2ED2
UDOTR  DATA DOCOL,TOR,ZERO,FROMR,UDDOTR,SEMIS
*
*** U. ***
       DATA L10F4
L10F5  DATA >8255,>2EA0
UDOT   DATA DOCOL,ZERO,UDDOT,SEMIS
*
*** +BUF ***
       DATA L10F5
L10F6  DATA >842B,>4255,>46A0
PLSBUF DATA DOCOL,BSLBUF,LIT,>4,PLUS,PLUS,DUP,LIMIT
       DATA EQUAL,ZBRAN,L10F7-$,DROP,FIRST
L10F7  DATA DUP,PREV,AT,SUB,SEMIS
*
*** BUFFER ***
       DATA L10F6
L10F8  DATA >8642,>5546,>4645,>52A0
BUFFER DATA DOCOL,USE,AT,DUP,TOR
L10F9  DATA PLSBUF,ZBRAN,L10F9-$,USE,STORE,RR,AT
       DATA ZLESS,ZBRAN,L10FA-$,RR,TWOP,RR,AT,LIT
       DATA >7FFF,AND,ZERO,RSLW
L10FA  DATA RR,STORE,RR,PREV,STORE,FROMR,TWOP,SEMIS
*
*** UPDATE ***
       DATA L10F8
L10FB  DATA >8655,>5044,>4154,>45A0
UPDATE DATA DOCOL,PREV,AT,AT,LIT,>8000,OR,PREV
       DATA AT,STORE,SEMIS
*
*** FLUSH ***
       DATA L10FB
L10FC  DATA >8546,>4C55,>53C8
FLUSH  DATA DOCOL,LIMIT,FIRST,SUB,BSLBUF,LIT,>4
       DATA PLUS,DDIV,ONEP,ZERO,PDO
L10FD  DATA LIT,>7FFF,BUFFER,DROP,PLOOP,L10FD-$
       DATA SEMIS
*
*** EMPTY-BUFFERS ***
       DATA L10FC
L10FE  DATA >8D45,>4D50,>5459,>2D42,>5546,>4645
       DATA >52D3
EMPTYB DATA DOCOL,FIRST,LIMIT,OVER,SUB,ERASE,FLUSH
       DATA FIRST,USE,STORE,FIRST,PREV,STORE,SEMIS
*
*** CLEAR ***
       DATA L10FE
L10FF  DATA >8543,>4C45,>41D2
CLEAR  DATA DOCOL,DUP,SCR,STORE,BSLSCR,TIMES,OFFSET
```

```
       DATA AT,PLUS,FLUSH,BSLSCR,ZERO,PDO
L1100  DATA I,OVER,PLUS,BUFFER,BSLBUF,BLANKS,UPDATE
       DATA PLOOP,L1100-$,DROP,SEMIS
*
*** BLOCK ***
       DATA L10FF
L1101  DATA >8542,>4C4F,>43CB
BLOCK  DATA DOCOL,OFFSET,AT,PLUS,TOR,PREV,AT,DUP
       DATA AT,RR,SUB,DUP,PLUS,ZBRAN,L1102-$
L1103  DATA PLSBUF,ZEQU,ZBRAN,L1104-$,DROP,RR,BUFFER
       DATA DUP,RR,ONE,RSLW,TWOM
L1104  DATA DUP,AT,RR,SUB,DUP,PLUS,ZEQU,ZBRAN
       DATA L1103-$,DUP,PREV,STORE
L1102  DATA FROMR,DROP,TWOP,SEMIS
*
*** (LINE) ***
       DATA L1101
L1105  DATA >8628,>4C49,>4E45,>29A0
PLINE  DATA DOCOL,TOR,CSL,BSLBUF,MDMOD,FROMR,BSLSCR
       DATA TIMES,PLUS,BLOCK,PLUS,CSL,SEMIS
*
*** .LINE ***
       DATA L1105
L1106  DATA >852E,>4C49,>4EC5
DOTLN  DATA DOCOL,PLINE,DTRAIL,TYPE,SEMIS
*
*** MESSAGE ***
       DATA L1106
L1107  DATA >874D,>4553,>5341,>47C5
MESSAG DATA DOCOL,WARNIN,AT,ZBRAN,L1108-$,DDUP
       DATA ZBRAN,L1109-$,LIT,>4,OFFSET,AT,BSLSCR
       DATA DDIV,SUB,DOTLN
L1109  DATA BRANCH,L110A-$
L1108  DATA PTYPE,>64D,>5347,>2023,>2020,DOT
L110A  DATA SEMIS
*
*** LOAD ***
       DATA L1107
L110B  DATA >844C,>4F41,>44A0
LOAD   DATA DOCOL,DDUP,ZEQU,LIT,>C,QERROR,BLK,AT
       DATA TOR,IN,AT,TOR,ZERO,IN
       DATA STORE,BSLSCR,TIMES,BLK,STORE,INTERP
       DATA FROMR,IN,STORE,FROMR
       DATA BLK,STORE,SEMIS
*
*** --> ***
       DATA L110B
L110C  DATA >C32D,>2DBE
ARROW  DATA DOCOL,QLOADI,ZERO,IN,STORE,BSLSCR
       DATA BLK,AT,OVER,MOD,SUB
       DATA BLK,PSTORE,SEMIS
*
*** R/W ***
       DATA L110C
L110D  DATA >8352,>2FD7
RSLW   DATA DOCOL,BSLBUF,SWAP,ZBRAN,L110E-$,RDISK
       DATA BRANCH,L110F-$
L110E  DATA WDISK
L110F  DATA DUP,QERROR,SEMIS
*
*** ' ***
       DATA L110D
L1110  DATA >C1A7
TICK   DATA DOCOL,DFIND,ZEQU,ZERO,QERROR,DROP,LITERA
       DATA SEMIS
*
```

```
*** UNFORGETABLE ***
       DATA L1110
L1110X DATA >8C55,>4E46,>4F52,>4745,>5441,>424C,>45A0
UNFORG DATA DOCOL,DUP,FENCE,AT,ULESS,OVER,LIT,$TASK1
       DATA ULESS,OR,HERE,ROT,ULESS,OR,SEMIS
*
*** FORGET ***
       DATA L1110X
L1111  DATA >8646,>4F52,>4745,>54A0
FORGET DATA DOCOL,TICK,LFA,DUP,UNFORG,LIT,>15,QERROR
       DATA TOR,VLINK,AT
FORGE1 DATA RR,OVER,ULESS,OVER,UNFORG,ZEQU,AND
       DATA ZBRAN,FORGE2-$,FORTH,DEFINI,AT
       DATA BRANCH,FORGE1-$
FORGE2 DATA DUP,VLINK,STORE
FORGE3 DATA DUP,TWOM
FORGE4 DATA PFA,LFA,AT,DUP,PFA,LFA,RR,ULESS,OVER
       DATA UNFORG,OR,ZBRAN,FORGE4-$
       DATA OVER,LIT,>4,SUB,STORE,AT,DDUP,ZEQU
       DATA ZBRAN,FORGE3-$,FROMR,DP,STORE,SEMIS
*
*** : ***
       DATA L1111
L1112  DATA >C1BA
COLON  DATA DOCOL,QEXEC,STRCSP,CURREN,AT,CONTEX
       DATA STORE,CREATE,RBRCKT,LIT,DOCOL
       DATA HERE,TWOM,STORE,SEMIS
*
*** ; ***
       DATA L1112
L1113  DATA >C1BB
SEMIC  DATA DOCOL,QCSP,COMPIL,SEMIS,SMUDGE,LBRCKT
       DATA SEMIS
*
*** BACK ***
       DATA L1113
L1114  DATA >8442,>4143,>4BA0
BACK   DATA DOCOL,HERE,SUB,COMMA,SEMIS
*
*** BEGIN ***
       DATA L1114
L1115  DATA >C542,>4547,>49CE
BEGIN  DATA DOCOL,QCOMP,HERE,ONE,SEMIS
*
*** ENDIF ***
       DATA L1115
L1116  DATA >C545,>4E44,>49C6
ENDIF  DATA DOCOL,QCOMP,TWO,QPAIRS,HERE,OVER,SUB
       DATA SWAP,STORE,SEMIS
*
*** THEN ***
       DATA L1116
L1117  DATA >C454,>4845,>4EA0
THEN   DATA DOCOL,ENDIF,SEMIS
*
*** DO ***
       DATA L1117
L1118  DATA >C244,>4FA0
DO     DATA DOCOL,QCOMP,COMPIL,PDO,HERE,THREE,SEMIS
*
*** LOOP ***
       DATA L1118
L1119  DATA >C44C,>4F4F,>50A0
LOOP   DATA DOCOL,QCOMP,THREE,QPAIRS,COMPIL,PLOOP
       DATA BACK,SEMIS
*
```

```
*** +LOOP ***
       DATA L1119
L111A  DATA >C52B,>4C4F,>4FD0
PLLOOP DATA DOCOL,QCOMP,THREE,QPAIRS,COMPIL,PPLOOP
       DATA BACK,SEMIS
*
*** UNTIL ***
       DATA L111A
L111B  DATA >C555,>4E54,>49CC
UNTIL  DATA DOCOL,QCOMP,ONE,QPAIRS,COMPIL,ZBRAN
       DATA BACK,SEMIS
*
*** END ***
       DATA L111B
L111C  DATA >C345,>4EC4
END    DATA DOCOL,UNTIL,SEMIS
*
*** AGAIN ***
       DATA L111C
L111D  DATA >C541,>4741,>49CE
AGAIN  DATA DOCOL,QCOMP,ONE,QPAIRS,COMPIL,BRANCH
       DATA BACK,SEMIS
*
*** REPEAT ***
       DATA L111D
L111E  DATA >C652,>4550,>4541,>54A0
REPEAT DATA DOCOL,QCOMP,TOR,TOR,AGAIN,FROMR,FROMR
       DATA TWOM,ENDIF,SEMIS
*
*** IF ***
       DATA L111E
L111F  DATA >C249,>46A0
IF     DATA DOCOL,QCOMP,COMPIL,ZBRAN,HERE,ZERO
       DATA COMMA,TWO,SEMIS
*
*** ELSE ***
       DATA L111F
L1120  DATA >C445,>4C53,>45A0
ELSE   DATA DOCOL,QCOMP,TWO,QPAIRS,COMPIL,BRANCH
       DATA HERE,ZERO,COMMA,SWAP,TWO,ENDIF,TWO
       DATA SEMIS
*
*** WHILE ***
       DATA L1120
L1121  DATA >C557,>4849,>4CC5
WHILE  DATA DOCOL,IF,TWOP,SEMIS
*
*** CASE ***
       DATA L1121
L1122  DATA >C443,>4153,>45A0
CASE   DATA DOCOL,QCOMP,CSP,AT,STRCSP,LIT,>4,SEMIS
*
*** OF ***
       DATA L1122
L1123  DATA >C24F,>46A0
OF     DATA DOCOL,LIT,>4,QPAIRS,COMPIL,POF,HERE
       DATA ZERO,COMMA,LIT,>5,SEMIS
*
*** ENDOF ***
       DATA L1123
L1124  DATA >C545,>4E44,>4FC6
ENDOF  DATA DOCOL,LIT,>5,QPAIRS,COMPIL,BRANCH,HERE
       DATA ZERO,COMMA,SWAP,TWO,ENDIF,LIT,>4,SEMIS
*
*** ENDCASE ***
       DATA L1124
```

```
L1125  DATA >C745,>4E44,>4341,>53C5
ENDCAS DATA DOCOL,LIT,>4,QPAIRS,COMPIL,DROP
L1126  DATA SPAT,CSP,AT,EQUAL,ZEQU,ZBRAN,L1127-$
       DATA TWO,ENDIF,BRANCH,L1126-$
L1127  DATA CSP,STORE,SEMIS
*
*** BASE->R ***
       DATA L1125
L1128  DATA >8742,>4153,>452D,>3ED2
BASTOR DATA DOCOL,FROMR,BASE,AT,TOR,TOR,SEMIS
*
*** R->BASE ***
       DATA L1128
L1129  DATA >8752,>2D3E,>4241,>53C5
RTOBAS DATA DOCOL,FROMR,FROMR,BASE,STORE,TOR,SEMIS
*
*** L/SCR ***
       DATA L1129
L112A  DATA >854C,>2F53,>43D2
LPSCR  DATA DOCOL,BSLSCR,BSLBUF,TIMES,CSL,DDIV
       DATA SEMIS
*
*** PAUSE ***
       DATA L112A
L112AX DATA >8550,>4155,>53C5
PAUSE  DATA DOCOL,QKEY,DUP,TWO,EQUAL
       DATA ZBRAN,PAUSE1-$,DROP,ONE,BRANCH,PAUSE2-$
PAUSE1 DATA ZBRAN,PAUSE3-$
PAUSE4 DATA QKEY,ZEQU,ZBRAN,PAUSE4-$
PAUSE5 DATA QKEY,DDUP,ZBRAN,PAUSE5-$
       DATA TWO,EQUAL,ZBRAN,PAUSE6-$
       DATA ONE,BRANCH,PAUSE7-$
PAUSE6 DATA QKEY,ZEQU,ZBRAN,PAUSE6-$,ZERO
PAUSE7 DATA BRANCH,PAUSE2-$
PAUSE3 DATA ZERO
PAUSE2 DATA SEMIS
*
*** LIST ***
       DATA L112AX
L112B  DATA >844C,>4953,>54A0
LIST   DATA DOCOL,BASTOR,DECIMA,CR,DUP,SCR,STORE
       DATA PTYPE,>553,>4352,>2023,DOT,LPSCR,ZERO
       DATA PDO
L112C  DATA CR,I,THREE,DOTR,SPACE,I,SCR,AT,DOTLN
       DATA PAUSE,ZBRAN,L112CX-$,LEAVE
L112CX DATA PLOOP,L112C-$,CR,RTOBAS,SEMIS
*
*** <BUILDS ***
       DATA L112B
L1139  DATA >873C,>4255,>494C,>44D3
BUILDS DATA DOCOL,CREATE,SMUDGE,SEMIS
*
*** (DOES>) ***
       DATA L1139
L113A  DATA >8728,>444F,>4553,>3EA9
PDOES  DATA DOCOL,FROMR,LATEST,PFA,CFA,STORE,SEMIS
*
*** DOES> ***
       DATA L113A
L113B  DATA >C544,>4F45,>53BE
DOES   DATA DOCOL,LIT,PDOES,COMMA,LIT,>6A0,COMMA
       DATA LIT,DODOES,COMMA,SEMIS
*
*** CONSTANT ***
       DATA L113B
L113C  DATA >8843,>4F4E,>5354,>414E,>54A0
```

```
CONSTA DATA DOCOL,BUILDS,COMMA
DOCON  EQU  $+2
       DATA PDOES,>6A0,DODOES,AT,SEMIS
*
*** USER ***
       DATA L113C
L113D  DATA >8455,>5345,>52A0
USER   DATA DOCOL,BUILDS,COMMA
DOUSER EQU  $+2
       DATA PDOES,>6A0,DODOES,AT,UU,PLUS,SEMIS
*
*** VARIABLE ***
       DATA L113D
L113E  DATA >8856,>4152,>4941,>424C,>45A0
VARIAB DATA DOCOL,BUILDS,COMMA
DOVAR  EQU  $+2
       DATA PDOES,>6A0,DODOES,SEMIS
*
*** VOCABULARY ***
       DATA L113E
L113F  DATA >8A56,>4F43,>4142,>554C,>4152,>59A0
VOCABU DATA DOCOL,BUILDS,CURREN,AT,TWOP,COMMA,LIT
       DATA >81A0,COMMA,HERE,VLINK,AT,COMMA
       DATA VLINK,STORE
DOVOC  EQU  $+2
       DATA PDOES,>6A0,DODOES,CONTEX,STORE,SEMIS
*
*** (;CODE) ***
       DATA L113F
L1140  DATA >8728,>3B43,>4F44,>45A9
PSCODE DATA DOCOL,FROMR,LATEST,PFA,CFA,STORE,SEMIS
*
*** MYSELF ***
       DATA L1140
L1144  DATA >C64D,>5953,>454C,>46A0
MYSELF DATA DOCOL,LATEST,PFA,CFA,COMMA,SEMIS
*
*** ~ ***
       DATA L1144
L1145  DATA >C180
NULL   DATA DOCOL,BLK,AT,ZBRAN,L1146-$,ONE,BLK
       DATA PSTORE,ZERO,IN,STORE,BLK,AT,BSLSCR
       DATA MOD,ZEQU,ZBRAN,L1147-$,QEXEC,FROMR
       DATA DROP
L1147  DATA BRANCH,L1148-$
L1146  DATA FROMR,DROP
L1148  DATA SEMIS
*
*** NOP ***
       DATA L1145
L1166  DATA >834E,>4FD0
NOP    DATA DOCOL,SEMIS
*
*** BLOAD ***
       DATA L1166
L1166X DATA >8542,>4C4F,>41C4
BLOAD  DATA DOCOL
BLOAD1 DATA DUP,ONEP,SWAP,BLOCK
       DATA DUP,LIT,14,PLUS,AT,LIT,29801,EQUAL
       DATA ZBRAN,BLOAD2-$,DUP,AT,TOR
       DATA TWOP,DUP,AT,DUP,TOR,DP,STORE
       DATA TWOP,DUP,AT,CURREN,STORE
       DATA TWOP,DUP,AT,CURREN,AT,STORE
       DATA TWOP,DUP,AT,CONTEX,STORE
       DATA TWOP,DUP,AT,CONTEX,AT,STORE
       DATA TWOP,DUP,AT,VLINK,STORE
```

```
       DATA LIT,12,PLUS,FROMR,FROMR,SWAP
       DATA OVER,SUB,DUP,TOR,LIT,1000,MIN
       DATA CMOVE,FROMR,LIT,1001,LESS,BRANCH,BLOAD3-$
BLOAD2 DATA DROP,DROP,ZERO,ONE
BLOAD3 DATA ZBRAN,BLOAD1-$,ZEQU,SEMIS
*
*** COLD ***
       DATA L1166X
L1167  DATA >8443,>4F4C,>44A0
COLD   DATA DOCOL,UCONS$,AT,U0,AT,LIT,ULNGTH,CMOVE
       DATA LIT,$TASK0,LIT,$TASK1,OVER,SUB,TOR
       DATA HERE,RR,CMOVE,HERE,TWOP,DUP,FORTHL
       DATA LIT,>4,SUB,STORE,FENCE,STORE,LIT,>81A0
       DATA FORTHL,TWOM,STORE,ZERO,FORTHL,STORE
       DATA FORTHL,VLINK,STORE
       DATA FIRST,USE,STORE,FIRST,PREV,STORE,FROMR
       DATA ALLOT,DR0,EMPTYB,LIT,>FFFF,DPL,STORE
       DATA BOOT,ABORT,SEMIS
*
*** BOOT ***
       DATA L1167
BOOTN  DATA >8442,>4F4F,>54A0
BOOT   DATA DOCOL,SPSTOR,DECIMA,ZERO,ECOUNT
       DATA STORE,FORTH,DEFINI,ZERO,BLK,STORE,LBRCKT
       DATA ZERO,THREE,BLOCK,DUP,LIT,>400,PLUS
       DATA SWAP,PDO
BOOT1  DATA I,CAT,DUP,LIT,>20,LESS,SWAP
       DATA LIT,>7F,GREAT,OR,ZBRAN,BOOT2-$
       DATA ONEP,LEAVE
BOOT2  DATA PLOOP,BOOT1-$,ZEQU,ZBRAN,BOOT3-$
       DATA THREE,LOAD
BOOT3  DATA SEMIS
*
*** SYSTEM ***
       DATA BOOTN
L1168  DATA >8653,>5953,>5445,>4DA0
SYST$  DATA $+2
       MOV  *SP+,TEMP1
       MOV  @$SYS(U),LINK
       BL   *LINK
       B    *NEXT
$TASK0 EQU  $
*
*** TASK ***
       DATA L1168
L1169  DATA >8454,>4153,>4BA0
TASK   DATA DOCOL,SEMIS

$TASK1 EQU  $
*
**
       END
```

O.3 BOOT—FORTH

BOOT is assembled to FORTH, the program that starts TI Forth. This program is loaded from Editor/Assembler option 3, **LOAD AND RUN** by giving the **FILE NAME?** prompt "DSK1.FORTH". BOOT, as FORTH, loads FORTHSAVE from DSK1 and copies the first 2230 (**8B6h**) bytes to low memory expansion CPU RAM at **3424h** and the last 7282 (**1C72h**) bytes to high memory expansion CPU RAM at **A000h**. After some additional initializing, TI Forth is cold started.

```
       TITL 'FORTH BOOT PROGRAM'
       IDT  'BOOT'
*************************************************************
*++ The TI Forth workspace registers
TEMP0  EQU  0
TEMP1  EQU  1
TEMP2  EQU  2
TEMP3  EQU  3
TEMP4  EQU  4
TEMP5  EQU  5
TEMP6  EQU  6
TEMP7  EQU  7
U      EQU  8
SP     EQU  9
W      EQU  10
LINK   EQU  11
CRU    EQU  12
IP     EQU  13
R      EQU  14
NEXT   EQU  15
*************************************************************
*++ TI Forth's workspace is 32 bytes at start of PAD (>8300->831F)
MAINWS EQU  >8300        IN CONSOLE CPU RAM
SUBPTR EQU  >8356        POINTS TO SUBROUTINE NAMES IN VDP
DSKERR EQU  >8350        DISK DSR ERROR CODES HERE
*************************************************************
       DEF  BOOT
*++ these REFs needed to access Editor/Assembler routines prior to having the TI Forth versions available
*++ and/or because they don't get trampled by the system load until the COLD start at the end of BOOT
       REF  VWTR,VDPWA,VDPWD
       REF  VMBW,DSRLNK,VMBR
*
FF9900 EQU  >A000        *++ pointer to TI Forth COLD start code
VSPTR  EQU  >836E        *++ pointer to value stack in VDP RAM
KYSTAT EQU  >837C        *++ GPL status byte
FAC    EQU  >834A
GRMWA  EQU  >9C02        *++ GROM write address register
GRMRA  EQU  >9802        *++ GROM read address register
GRMRD  EQU  >9800        *++ GROM read data register
SVGPRT EQU  >3C70        *++ save return to GPL interpreter
GRMSAV EQU  >3CD6        *++ save GROM address during DSRLNK
RTFGPL EQU  >3AB0        *++ return to assembly language from GPL
*++ address of initial values for first 27 user variables—COLD resets user variable table to these
UBASE0 EQU  >3944

ENTLNK EQU  >3A2A        *++ not sure this is actually used
GPLLNK EQU  >3A06        *++ TI Forth's GPLLNK routine
XMLTAB EQU  >0CFA        *++ XML table in console ROM
CIFENT EQU  >3B32        *++ TI Forth's integer to floating point conversion routine
*************************************************************
PAB    EQU  >F80        *++ PAB for loading FORTHSAVE binary from DSK1
*++ PAB data:  LOAD file opcode=5,
*++            VDP address of buffer=1000h,
```

```
*++            maximum number of bytes to transfer=8B6h+1C72h=size of file
*++            name-length byte=Eh (length of file descriptor)
PDATA  DATA >0500,>1000,0,>8B6+>1C72,>000E
*++ PAB data (continued):  file descriptor="DSK1.FORTHSAVE"
       TEXT 'DSK1.FORTHSAVE'
       EVEN
*************************************************************
*++ location of TI Forth's inner interpreter in PAD RAM
       DORG >832E          *++ code at FMOVE will be moved here
DODOES DECT SP             DUMMY COPY TO GET ADRESSES
       MOV  W,*SP
       MOV  LINK,W
DOCOL  DECT R
       MOV  IP,*R
       MOV  W,IP
$NEXT  MOV  *IP+,W
DOEXEC MOV  *W+,TEMP1
       B    *TEMP1
$SEMIS MOV  *R+,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
*
*
       AORG >C000
*++ this is the guts of TI Forth's inner interpreter that gets moved to DODOES in PAD RAM
FMOVE  DECT SP             COPY TO MOVE TO CONSOLE RAM
       MOV  W,*SP
       MOV  LINK,W
       DECT R
       MOV  IP,*R
       MOV  W,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
       MOV  *R+,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
*
*++ beginning of BOOT program code
BOOT   LWPI MAINWS             *++ set up TI Forth workspace in PAD
       LIMI 0
*++ copy PAB data to PAB (VDP F80h)
       LI   TEMP0,PAB
       LI   TEMP1,PDATA
       LI   TEMP2,>20
       BLWP @VMBW
*++ set up and invoke disk DSR to load DSK1.FORTHSAVE to VDP buffer at 1000h
       LI   TEMP6,PAB+9
       MOV  TEMP6,@SUBPTR
       BLWP @DSRLNK
       DATA 8
*++ copy first 8B6h bytes of system from VDP buffer to low memory expansion RAM at 3424h
       LI   TEMP0,>1000
       LI   TEMP1,>3424
       LI   TEMP2,>8B6
       BLWP @VMBR
*++ copy rest of system (1C72h bytes) from VDP buffer to high memory expansion RAM at A000h
       LI   TEMP0,>18B6
```

```
       LI   TEMP1,>A000
       LI   TEMP2,>1C72
       BLWP @VMBR
*++ finding and saving GROM address (682Dh) of XML instruction in E/A cartridge that got us here so
*++ we can use it to "return" from GPL to execute assembly code.  The GPL code in question ("XML >22")
*++ starts the E/A loader that loaded this TI Forth BOOT program, which means the loader address is
*++ stored at CPU RAM address 2004h. until we change it in later code
       MOVB @GRMRA,TEMP1
       SWPB TEMP1
       MOVB @GRMRA,TEMP1
       SWPB TEMP1
       AI   TEMP1,-3
       MOV  TEMP1,@GRMSAV
*++ get the object of the GPL XML instruction, the ">22" of "XML >22", into the high byte of TEMP1
       INC  TEMP1
       MOVB TEMP1,@GRMWA
       SWPB TEMP1
       MOVB TEMP1,@GRMWA
       NOP
       MOVB @GRMRD,TEMP1
*++ calculate the XML vector by using first nybble of ">22" = 2 to look up the table's address in the
*++ console ROM's XML table at 0CFAh + (2 x 2) = 0CFEh (which contains 2000h) and adding the
*++ table's offset (the second nybble (2) x 2 = 4) to get 2004h, which is then stored in TEMP2
       MOV  TEMP1,TEMP2
       SRL  TEMP1,12
       SLA  TEMP1,1
       SLA  TEMP2,4
       SRL  TEMP2,11
       A    @XMLTAB(TEMP1),TEMP2
*++ save E/A loader's return address to the GPL interpreter in console ROM = 061Ch, which is by a
*++ "JMP >05E4" followed by a "B @>0070"
       MOV  @>2030,@SVGPRT       >2030 IS SVGPRT USED BY E/A LOADER
*++ move the address of our return from GPL (RTFGPL=3AB0h) to 2004h, the object of the GROM
*++ "XML >22" instruction noted above, which will be executed every time we return from GPL
       LI   TEMP1,RTFGPL
       MOV  TEMP1,*TEMP2
*++ copy TI Forth's inner interpreter code (26 bytes) from FMOVE to where it will execute in PAD
*++ at 832Eh
       LI   TEMP1,BOOT-FMOVE
       LI   TEMP2,FMOVE
       LI   TEMP3,DODOES
MLOOP  MOV  *TEMP2+,*TEMP3+
       DECT TEMP1
       JNE  MLOOP
*
*** INITIALIZE VDP STUFF
*
       LI   TEMP0,>01B0  BLANK SCREEN
       BLWP @VWTR
       LI   TEMP0,>030E  SET COLOR TABLE AT >0380
       BLWP @VWTR
       LI   TEMP0,>0401  SET PATTERN DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>0506  SET SPRITE ATTRIBUTE TABLE >0300
       BLWP @VWTR
       LI   TEMP0,>0601  SET SPRITE DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>07F4  SET TEXTMODE COLORS
       BLWP @VWTR
       LI   TEMP0,>2000  BLANK
```

```
       LI   TEMP1,>960   TEXT-MODE SCREEN SIZE  *++ this should be either 960 or >3C0
       LI   TEMP2,>0     SCREEN STARTS AT 0
       BL   @FILLER      CLEAR SCREEN
       LI   TEMP0,>FF00  CHAR FF
       LI   TEMP1,>2048  BLOCK SIZE             *++ this should be either 2048 or >800
       LI   TEMP2,>800   STARTING LOCATION IN VDP
       BL   @FILLER      FILL AREA WITH FF'S
*++ force text mode
       LI   TEMP0,>81F0
       SWPB TEMP0
       MOVB TEMP0,@>83D4 USED TO UPDATE VDP REG EACH KEYSTROKE
       MOVB TEMP0,@VDPWA FORCE TEXT MODE
       SWPB TEMP0
       MOVB TEMP0,@VDPWA
*++ load capital letters
       LI   TEMP0,>900   VDP LOCATION
       MOV  TEMP0,@FAC   *++ FAC must contain VDP start address
       CLR  TEMP1        CLEAR GPL STATUS
       MOVB TEMP1,@KYSTAT
       LI   TEMP7,>3E0
       MOV  TEMP7,@VSPTR
       BLWP @GPLLNK      LOAD CAPITAL LETTER SHAPES
       DATA >0018
*++ set location of TI Forth's CIF routine (replaces trampled E/A version)
       LI   TEMP2,CIFENT
       MOV  TEMP2,@>2006
*++ load lowercase letters (actually, small caps)
       LI   TEMP2,>1200
       CB   TEMP2,@3     IF BYTE @3 IN THE CONSOLE IS >12 IT'S A 99/4
       JEQ  FOUR         DON'T LOAD LOWER CASE IN A 99/4
       LI   TEMP0,>0B00
       MOV  TEMP0,@FAC   *++ FAC must contain VDP start address
       MOVB TEMP1,@KYSTAT
       BLWP @GPLLNK      LOAD LOWER CASE IN 99/4A
       DATA >004A
*
*++ finally, we cold start TI Forth
FOUR   LI   TEMP1,UBASE0
       B    @FF9900      BRANCH TO ACMSRC
*
FILLER ORI  TEMP2,>4000  SET BIT FOR VDP WRITE
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
FLLOOP MOVB TEMP0,@VDPWD WRITE A BYTE
       DEC  TEMP1
       JNE  FLLOOP       NOT DONE, FILL ANOTHER
       B    *LINK
*
       END  BOOT
```

O.4 Generating TI Forth from Source Code

To generate TI Forth from the original source code, you must do the following:

1. Assemble the three files, BOOT, DRIVER and ASMSRC. BOOT is self-contained and should be assembled to FORTH. DRIVER includes UTILEQU, UTILROM and UTILRAM when it is assembled. Assemble it to something like DRIVERO to distinguish it from the source file. ASMSRC includes ASMSRC1, ASMSRC2 and ASMSRC3 when it is assembled. Assemble it to something like ASMSRCO.
2. Load the two object files, ASMSRCO and DRIVERO, in that order, with the Editor/Assembler option 3.
3. This step is not so easy. You can write an assembly language program to load next (avoiding clobbering the two files already loaded) that will copy the 9512 bytes of CPU RAM **3424h–3CD9h** (where the relevant part of DRIVERO is loaded) and **A000h–BC71h** (where ASMSRCO is loaded) to a VDP RAM buffer and from there to a file with the disk DSR I/O opcode=6 (SAVE) and name that file FORTHSAVE. The editor has written such a program (FSAVE) and included it in the section following these instructions. [*Note:* This step cannot be accomplished with the Editor/Assembler's SAVE utility due to SAVE's 8-KB limit.]
4. At this point, you can copy FORTH and FORTHSAVE to a copy of the old diskette, if your purpose for regenerating these object files was to replace a defective system. Otherwise, there is more work to do. Check Appendix K and Appendix L for details that will help you set up the system for larger disks and explain in greater detail the purpose for as well as an alternative method for the next step.
5. The TI Forth system screens (blocks) need to be copied to the new system disk. From a working, 90-KB TI Forth system diskette, copy the file named SYS-SCRNS. If the new diskette is also 90 KB, you're done. Otherwise, write a blank record to the last possible record of the file such that writing that record completely fills the disk. The number of sectors occupied by a SYS-SCRNS file that fills the disk can be calculated from the size of the disk my multiplying the disk size in KB by 4 to get total sectors and subtracting the sectors occupied by the FDIR (2), the three FDRs (3) and the first two files (5 + 38 = 43). For a 360-KB diskette, this would be 360 * 4 - 2 - 3 - 43 = 1392 sectors. The number of 128-byte records is twice this (2784 records), so the last record, counting from 0, would be record number 2783. You can expand the file in TI Basic or TI Extended Basic like this:

```
100   OPEN #1:"DSK1.SYS-SCRNS",RELATIVE,UPDATE,FIXED 128
110   LASTREC=2783
120   PRINT #1,REC LASTREC:" "
130   CLOSE #1
```

6. If you performed step 5 for a new system disk larger than 90 KB, you will need to copy TI Forth system screens #2 – #5 to their proper locations from screens #90 – #94. The problem with this is that the source screens are skewed by 12 lines, *i.e.*, we need to start the copy from line 12 of screen #90 and continue through line 11 of screen #94. It will be easier to copy records 572 – 603 of the new SYS-SCRNS file to there proper locations in the SYS-SCRNS file. Doing this requires us to calculate the starting destination record, which will always be the same distance from the end of the SYS-SCRNS file regardless

of its size. There will always be a 20-record space at the end of the file. With 2 lines per record and 16 lines per screen, we will be copying 32 records beginning with the last record - 51, which, for the 360-KB case is 2783 - 51 = 2732. This can be done with another TI Forth system as in Appendix L or with the following TI Extended Basic code (TI Basic won't handle this correctly), which includes extending **SYS-SCRNS** to fill up the disk:

```
100   OPEN #1:"DSK1.SYS-SCRNS",RELATIVE,UPDATE,FIXED 128
110   LASTREC=2783
120   PRINT #1,REC LASTREC:" "
130   FOR I=572 TO 603
140   LINPUT #1,REC I:LINE$
150   PRINT #1,REC LASTREC-623+I:LINE$
160   NEXT I
170   CLOSE #1
```

Remember to change **LASTREC** to whatever is the last record number in **SYS-SCRNS** on the new disk.

O.5 FSAVE Assembly Source Code

The following TMS9900 assembly source code was written by the editor to save the memory images of the previously loaded DRIVERO and ASMSRCO object files to FORTHSAVE. FORTHSAVE is the memory-image file that FORTH (the TI Forth BOOT program) loads to start TI Forth. FSAVE saves FORTHSAVE to DSK2. You can change this in (or near) line 61 to save to a different disk drive, if you like. This code is offered "as is":

```
       TITL 'FORTH SAVE PROGRAM'
       IDT  'FSAVE'
*************************************************************
* Though the names may be different, this program expects
* object files DRIVERO and ASMSRCO to be in memory before it
* is loaded and run.  DRIVERO is assembled from DRIVER and
* ASMSRCO from ASMSRC.  If any modifications are made to
* DRIVER or ASMSRC, starting points and image lengths will
* need to be changed in this program and the Forth BOOT pro-
* gram to match.
*                              ...Lee Stewart, April 18, 2013
*************************************************************
TEMP0  EQU  0
TEMP1  EQU  1
TEMP2  EQU  2
TEMP3  EQU  3
TEMP4  EQU  4
TEMP5  EQU  5
TEMP6  EQU  6
TEMP7  EQU  7
U      EQU  8
SP     EQU  9
W      EQU  10
LINK   EQU  11
CRU    EQU  12
IP     EQU  13
R      EQU  14
NEXT   EQU  15
```

```
*************************************************************
MAINWS EQU  >8300        IN CONSOLE CPU RAM
SUBPTR EQU  >8356        POINTS TO SUBROUTINE NAMES IN VDP
*************************************************************
       DEF  FSAVE
*
FAC    EQU  >834A
KYBD   EQU  >8374
KYCHR  EQU  >8375
KYSTAT EQU  >837C
GRMRA  EQU  >9802
GRMRD  EQU  >9800
GRMWA  EQU  >9C02
KSCAN  EQU  >2108
UTILWS EQU  >2094
VDPWA  EQU  >8C02
VDPWD  EQU  >8C00
VMBR   EQU  >2118
VMBW   EQU  >2110
VSBW   EQU  >210C
VSPTR  EQU  >836E
VWTR   EQU  >211C
XMLTAB EQU  >0CFA
*
*************************************************************
* Set up PAB for disk SAVE operation
*************************************************************
PAB    EQU  >F80
*
       AORG >C000   insure DRIVER and ASMSRC not corrupted
*
PDATA  DATA >0600,>1000,0,>8B6+>1C72,>000E
       TEXT 'DSK2.FORTHSAVE'
       EVEN
*************************************************************
*
PREAM  TEXT 'FSAVE running...Tap a key to start...  '
       EVEN
LOCPY  TEXT '-Copying Forth support to VDP RAM      '
       EVEN
HICPY  TEXT '-Copying resident dictionary to VDP RAM'
       EVEN
SVFIL  TEXT '-Saving memory image to DSK2.FORTHSAVE '
       EVEN
FERROR TEXT '***FILE I/O ERROR:  DSK2.FORTHSAVE***  '
EPILOG TEXT '        ...done.  Press <quit> to exit.'
       EVEN
MCNT   DATA 39  Char count of above messages
ROW    DATA 0   Rowcount initialized to 0
D40    DATA 40  Decimal 40
*
*************************************************************
*** PROGRAM START
*************************************************************
FSAVE  LWPI MAINWS
       LIMI 0
*
*************************************************************
*** INITIALIZE VDP STUFF
*************************************************************
       LI   TEMP0,>01B0  BLANK SCREEN
       BLWP @VWTR
       LI   TEMP0,>030E  SET COLOR TABLE AT >0380
       BLWP @VWTR
       LI   TEMP0,>0401  SET PATTERN DESCRIPTOR TABLE >0800
       BLWP @VWTR
```

```
       LI   TEMP0,>0506  SET SPRITE ATTRIBUTE TABLE >0300
       BLWP @VWTR
       LI   TEMP0,>0601  SET SPRITE DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>07F4  SET TEXTMODE COLORS
       BLWP @VWTR
       LI   TEMP0,>2000  BLANK
       LI   TEMP1,>3C0   TEXT-MODE SCREEN SIZE
       LI   TEMP2,>0     SCREEN STARTS AT 0
       BL   @FILLER      CLEAR SCREEN
       LI   TEMP0,>FF00  CHAR FF
       LI   TEMP1,>800   BLOCK SIZE
       LI   TEMP2,>800   STARTING LOCATION IN VDP
       BL   @FILLER      FILL AREA WITH FF'S
       LI   TEMP0,>81F0
       SWPB TEMP0
       MOVB TEMP0,@>83D4 UPDATES VDP REG EACH KEYSTROKE
       MOVB TEMP0,@VDPWA FORCE TEXT MODE
       SWPB TEMP0
       MOVB TEMP0,@VDPWA
       LI   TEMP7,>3E0   VSPTR location in VDP RAM
       MOV  TEMP7,@VSPTR set VSPTR
       LI   TEMP0,>900   VDP LOCATION
       MOV  TEMP0,@FAC
       CLR  TEMP1        CLEAR GPL STATUS
       MOVB TEMP1,@KYSTAT
       BLWP @GPLLNK      LOAD SMALL CAPITAL LETTER SHAPES
       DATA >0018
*
       LI   TEMP2,>1200
       CB   TEMP2,@3     99/4, IF CONSOLE BYTE @3 = >12
       JEQ  WORK         DON'T LOAD LOWER CASE IN A 99/4
       LI   TEMP0,>0B00
       MOV  TEMP0,@FAC
       MOVB TEMP1,@KYSTAT
       BLWP @GPLLNK      LOAD LOWER CASE IN 99/4A
       DATA >004A
*
       JMP  WORK
*
FILLER ORI  TEMP2,>4000  SET BIT FOR VDP WRITE
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
FLLOOP MOVB TEMP0,@VDPWD WRITE A BYTE
       DEC  TEMP1
       JNE  FLLOOP       NOT DONE, FILL ANOTHER
       B    *LINK
*************************************************************
* Message display routine
*************************************************************
DSP
       MOV    @ROW,TEMP0   get bytecount of row
*--->   TEMP1 already set by caller
       MOV  @MCNT,TEMP2  message bytecount
       BLWP @VMBW
       A    @D40,@ROW    Increment row bytecount by 1 row
       B    *LINK
*************************************************************
* Announce start of program.
*************************************************************
WORK
       LIMI 2
       LIMI 0
```

```
       LI   TEMP1,PREAM
       BL   @DSP
       A    @D40,@ROW    Increment bytecount another row
*************************************************************
* Wait for user to press a key
*************************************************************
GETKY
       CLR  @KYBD          check entire keyboard
       BLWP @KSCAN         check keyboard
       MOVB @KYSTAT,TEMP1  Check flag for key pressed
       SLA  TEMP1,3        Flag value is >20
       JNC  GETKY          No key was pressed, try again
*************************************************************
* Announce LO-mem copy.
*************************************************************
       LI   TEMP1,LOCPY
       BL   @DSP
*************************************************************
* Copy Forth support routines (>3424 - >3CD9) to VDP
*************************************************************
       LI   TEMP0,>1000
       LI   TEMP1,>3424
       LI   TEMP2,>8B6
       BLWP @VMBW
*************************************************************
* Announce HI-mem copy.
*************************************************************
       LI   TEMP1,HICPY
       BL   @DSP
*************************************************************
* Append resident Forth vocabulary (>A000 - >BC71) to above
* copied VDP section
*************************************************************
       LI   TEMP0,>18B6
       LI   TEMP1,>A000
       LI   TEMP2,>1C72
       BLWP @VMBW
*************************************************************
* Announce saving of DSK2.FORTHSAVE.
*************************************************************
       LI   TEMP1,SVFIL
       BL   @DSP
       A    @D40,@ROW     Increment bytecount another row
*************************************************************
* Copy PAB data to VDP RAM in preparation for SAVEing memory
* image to disk in DSK2.FORTHSAVE
*************************************************************
       LI   TEMP0,PAB
       LI   TEMP1,PDATA
       LI   TEMP2,>20
       BLWP @VMBW
*************************************************************
* Perform SAVE of memory image
*************************************************************
       LI   TEMP6,PAB+9
       MOV  TEMP6,@SUBPTR
       BLWP @DSRLNK
       DATA 8
       JNE  EPIDSP        File error?
       LI   TEMP1,FERROR  Yes!  Inform user.
       BL   @DSP
       A    @D40,@ROW     Increment bytecount another row
*************************************************************
* Announce that program is finished.
*************************************************************
EPIDSP LI   TEMP1,EPILOG
```

```
       BL   @DSP
*
DONE   LIMI 2
DONE1  NOP
       JMP  DONE1        Spin our wheels until interrupted
*
*---GPLLNK----(begin)-------------------------------------*
*                                                        *
* A universal GPLLNK--6/21/85--MG Millers Graphics)       *
* This routine will work with any GROM library slot since  *
* it is indexed off of R13 in the GPLWS.  (It does require *
* Mem Expansion) This GPLLNK does NOT require a module to  *
* be plugged into the GROM port so it will work with the   *
* Editor/Assembler, Mini Memory (with Mem Expansion),      *
* Extended Basic, the Myarc CALL LR("DSKx.xxx") or the     *
* CorComp Disk Manager Loaders.  It saves and restores the *
* current GROM Address in case you want to return back to  *
* GROM for Basic or Extended Basic CALL LINKs or to return *
* to the loading module.                                   *
*                                                        *
*    ENTER: The same way as the E/A GPLLNK, i.e.,          *
*             BLWP @GPLLNK                               *
*             DATA >34                                   *
*                                                        *
*    NOTES: Do Not REF GPLLNK when using this routine in   *
*           your code.                                   *
*                                                        *
* 70 Bytes - including the GPLLNK Workspace               *
*--------------------------------------------------------*

GPLWS  EQU  >83E0            GPL workspace
GR4    EQU  GPLWS+8          GPL workspace R4
GR6    EQU  GPLWS+12         GPL workspace R6
STKPNT EQU  >8373            GPL Stack pointer
LDGADD EQU  >60              Load & Execute GROM entry point
XTAB27 EQU  >200E            Low Mem XML table location 27
GETSTK EQU  >166C

GPLLNK DATA GLNKWS    R7      Set up BLWP Vectors
       DATA GLINK1    R8

RTNAD  DATA XMLRTN    R9      GPL XML returns to us here
GXMLAD DATA >176C     R10     GROM Address of GPL "XML 27"
*                                (0F 27 Opcode)
       DATA >50       R11     Initialized to >50 where
*                                PUTSTK address resides
GLNKWS EQU  $->18             GPLLNK's workspace--only
       BSS  >08       R12-R15    registers R7-R15 are used
GLINK1 MOV  *R11,@GR4         Store PUTSTK Address in R4
*                                of GPL workspace
       MOV  *R14+,@GR6        Put GPL Routine Address in
*                                R6 of GPL wprkspace
       MOV  @XTAB27,R12       Save the value at >200E
       MOV  R9,@XTAB27        Save XMLRTN Address at >200E
       LWPI GPLWS             Load GPL workspace
       BL   *R4                 Current GROM Addr to stack
       MOV  @GXMLAD,@>8302(R4)  Push GPL XML Addr on stack
*                                  for GPL ReTurn
       INCT @STKPNT             Adjust the stack pointer
       B    @LDGADD             Execute our GPL Routine

XMLRTN MOV  @GETSTK,R4        Get GETSTK pointer
       BL   *R4               Restore GROM addr from stack
       LWPI GLNKWS            Load our WS
       MOV  R12,@XTAB27       Restore >200E
       RTWP                   All Done - Return to Caller
```

```
*---GPLLNK----(end)----------------------------------------*

*---DSRLNK----(begin)---------------------------------------*
*                                                          *
* A Universal Device Service Routine Link - MG             *
* (Uses console GROM 0's DSRLNK routine)                   *
* (Do not REF DSRLNK or GPLLNK when using these routines)  *
* (This DSRLNK will also handle Subprograms and CS1, CS2)  *
*                                                          *
*    ENTER: The same way as the E/A DSRLNK, i.e.,          *
*                BLWP @DSRLNK                              *
*                DATA 8                                    *
*                                                          *
*    NOTES:    Must be used with a GPLLNK routine          *
*             Returns ERRORs the same as the E/A DSRLNK    *
*             EQ bit set on return if error                *
*             ERROR CODE in caller's MSB of R0 on return   *
*                                                          *
* 186 Bytes total:  GPLLNK, DSRLNK and both Workspaces     *
*----------------------------------------------------------*

PUTSTK EQU  >50          Push GROM Add to stack pointer
TYPE   EQU  >836D        DSRLNK Type byte for GPL DSRLNK
NAMLEN EQU  >8356        Device name length pointer, VDP PAB
VWA    EQU  >8C02        VDP Write Address location
VRD    EQU  >8800        VDP Read Data byte location
GR4LB  EQU  >83E9        GPL Workspace R4 Lower byte
GSTAT  EQU  >837C        GPL Status byte location

DSRLNK DATA DSRWS,DLINK1         Set BLWP Vectors

DSRWS  EQU  $                    Start of DSRLNK workspace
DR3LB  EQU  $+7                  R3 lower byte of DSRLNK WS
DLINK1 MOV  R12,R12         R0   Have we already looked up
*                                the LINK address?
       JNE  DLINK3          R1   YES!  Skip lookup routine
*<<------------------------------------------------------>>*
* This section of code is only executed once to find the   *
* GROM address for the GPL DSRNK, which is placed at DSRADD*
* and R12 is set to >2000 to indicate that the address is  *
* found and to be used as a mask for EQ & CND              *
*----------------------------------------------------------*
       LWPI GPLWS          R2,R3    else load GPL workspace
       MOV  @PUTSTK,R4     R4,R5    Store current GROM
*                                      address on the stack
       BL   *R4            R6
       LI   R4,>11         R7,R8    Load R4 with address of
*                                      LINK routine vector
       MOVB R4,@>402(R13)  R9,R10   Set up GROM with address
*                                      for vector
       JMP  DLINK2         R11      Jump around R12-R15
       DATA 0              R12      has >2000 flag when set
       DATA 0,0,0          R13-R15  has WS, PC & ST for RTWP
DLINK2 MOVB @GR4LB,@>402(R13)       Finish GROM addr setup
       MOV  @GETSTK,R5              Take some time & set up
*                                      GETSTK pointer
       MOVB *R13,@DSRAD1            Get GPL DSR LINK vector
       INCT @DSRADD                 Adjust it to get past
*                                      GPL FETCH instr
       BL   *R5                     GROM address from stack
       LWPI DSRWS                   Reload DSRLNK workspace
       LI   R12,>2000               Set flag to signify
*                                       DSRLNK addr is set
*<<------------------------------------------------------>>*
DLINK3 INC  R14            Adjust R14 to point to
*                             caller's DSR Type byte
```

```
       MOVB *R14+,@TYPE    Move >836D for GPL DSRLNK
       MOV  @NAMLEN,R3     Save VDP addr of Name Length
       AI   R3,-8          Adjust, point to PAB Flag byte
       BLWP @GPLLNK        Execute DSR LINK
DSRADD BYTE >03            High byte of GPL DSRLNK address
DSRAD1 BYTE >00            Lower byte of GPL DSRLNK address
*
*---Error Check & Report to Caller's R0 and EQU bit---------
*
       MOVB @DR3LB,@VWA    LSB of VDP Addr for Error Flag
       MOVB R3,@VWA        MSB of VDP Add for Error Flag
       SZCB R12,R15        Clear EQ bit for Error Report
       MOVB @VRD,R3        Get PAB Error Flag
       SRL  R3,5           Adjust it to 0-7 error code
       MOVB R3,*R13        Put it into Caller's R0 (msb)
       JNE  SETEQ          If it's not zero, set EQ bit
       COC  @GSTAT,R12     Else, CND bit = 0? (Link Error)
       JNE  DSREND         No Error; Just return
SETEQ  SOCB R12,R15        Error, so set Caller's EQ bit
DSREND RTWP                All Done - Return to Caller
*
*---DSRLNK----(end)-----------------------------------------*
*

       END  FSAVE
```