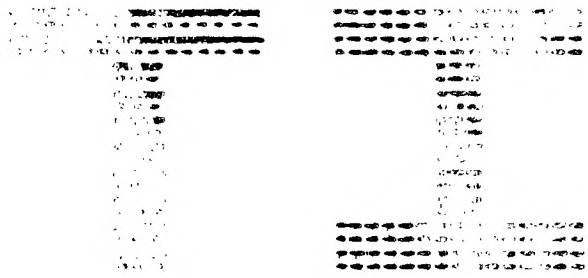
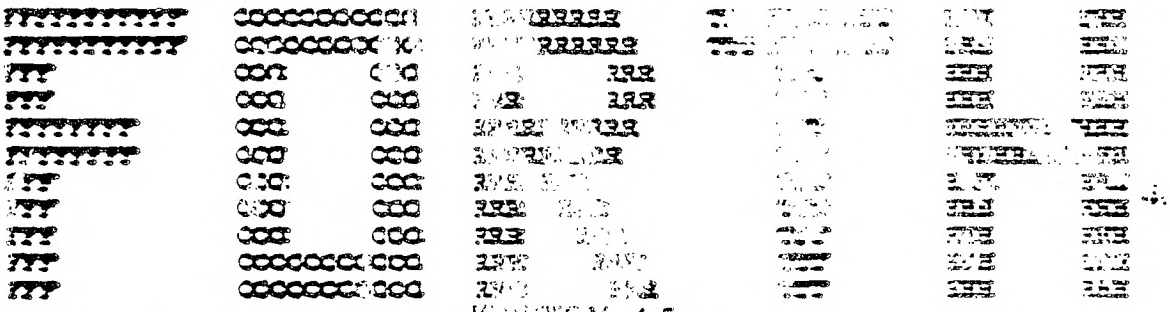
TI-WARE™





VERSION 4.5

REQUIRES 32K EXTRA MEMORY, DISK DRIVE AND EDITOR ASSEMBLER MODULE

IMPORTANT NOTICE TO USERS

The user assumes complete responsibility for any decisions made or actions taken based on information contained using these programs and book materials which are made available solely on an "AS IS" basis.

TEX-SOFT does not warrant or represent that the programs and book materials will be free from error or will meet the specific requirements of the user.

PUBLISHED BY
TEX-SOFT
P.O. BOX 33084
GRANADA HILLS, CA 91344

TI FORTH INSTRUCTION MANUAL

TABLE OF CONTENTS

CHAPTER 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 compatability 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
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 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 "words" or procedures 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 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. When multi-precision values are stored on the stack they are always stored with the most significant part most accessable. 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.

STARTING FORTH

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

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 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 DSK1, 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 Routines 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 TI FORTH editor.	3
-COPY	Copy FORTH screens 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 MULTI-COLOR 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 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 HEX.	9
-ASSEMBLER	Write routines in TI FORTH assembler	9
-GRAPH	Utilize the graphics capabilities of the 99/4A.	6
-BSAVE	Save dictionary overlays to diskette.	11
-CRU	Access the FORTH equivalents of 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.

CHAPTER 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" by 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 RETURN. The following convention will be used when referring to the stack in FORTH:

(a b --- c)

This diagram shows the stack contents before and after the execution of a word. In this case the stack contains two values, a and b, before execution of a word. The execution is denoted by --- and the stack contents after execution is c. The 'most accessible' stack element is always on the right, e.g. b is 'more accessible' than a. There may be values on the stack that are less accessible than a but these are unaffected by the execution of the word in question. In addition the following symbols are used as operands for clarity:

SYMBOLS USED IN THIS DOCUMENT

n, n1, ...	16-bit signed numbers
d, d1, ...	32-bit signed numbers
u	16-bit unsigned number
ud	32-bit unsigned number
addr, addr1 ...	addresses
b	8-bit byte (in right half of word)
c	7-bit character (in right end of word)
f	boolean flag (0=false, non-0=true)

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	(n1 n2 -- n2 n1)	Exchange top two stack items
OVER	(n1 n2 -- n1 n2 n1)	Make copy of second item on top
ROT	(n1 n2 n3 -- n2 n3 n1)	Rotate third item to top
-DUP	(n -- n (n))	Duplicate only if non-zero
>R	(n --)	Move top to Return stack for storage
R>	(-- n)	Retrieve item from Return stack
R	(-- n)	Copy top of Return stack to stack

Note: >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.

ARITHMETIC AND LOGICAL OPERATIONS

The following are the most common arithmetic and logical operations:

+	(n1 n2 -- sum)	Add
D+	(d1 d2 -- dsum)	Add double precision numbers
-	(n1 n2 -- diff)	Subtract (n1-n2)
1+	(n -- n+1)	Increment by 1
2+	(n -- n+2)	Increment by 2
1-	(n -- n-1)	Decrement by 1
2-	(n -- n-2)	Decrement by 2
*	(n1 n2 -- prod)	Multiply
/	(n1 n2 -- quot)	Divide (n1/n2)
MOD	(n1 n2 -- rem)	Modulo (remainder from n1/n2)
/MOD	(n1 n2 -- rem quot)	Divide giving remainder and quotient
*/MOD	(n1 n2 n3 -- rem quot)	n1*n2/n3 with 32-bit intermediate
*/	(n1 n2 n3 -- quot)	Like */MOD but giving quot only
U*	(u1 u2 -- ud)	Unsigned * with double product
U/	(ud u1 -- urem uquot)	Unsigned / with remainder
MAX	(n1 n2 -- max)	Maximum
MIN	(n1 n2 -- min)	Minimum
ABS	(n -- absolute)	Absolute value
DABS	(d -- dabsolute)	Absolute value of 32-bit number
MINUS	(n -- -n)	Leave two's complement
DMINUS	(d -- -d)	Leave two's complement of 32-bits
AND	(n1 n2 -- and)	Bitwise logical AND
OR	(n1 n2 -- or)	Bitwise logical OR
XOR	(n1 n2 -- xor)	Bitwise logical exclusive-OR
SWPB	(n1 -- n2)	Swap the bytes of n1 producing n2
SRC	(n1 n2 -- n3)	Shift n1 right circular n2 bits giving n3
SRL	(n1 n2 -- n3)	Shift n1 right logical n2 bits giving n3
SRA	(n1 n2 -- n3)	Shift n1 right arithmetic n2 bits giving n3
SLA	(n1 n2 -- n3)	Shift n1 left arithmetic n2 bits giving n3

COMPARISON OPERATIONS

The following are the most common comparisons.

<	(n1 n2 -- f)	True if n1 less than n2 (signed)
=	(n1 n2 -- f)	True if top two numbers are equal
>	(n1 n2 -- f)	True if n1 greater then n2
0<	(n -- f)	True if top number is negative
0=	(n -- f)	True if top number is 0 (e.g. NOT)
U<	(u1 u2 -- f)	Unsigned integer compare

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 to u --)	Block move u bytes. FROM & TO = addr
FILL	(addr u b --)	Fill u bytes with b beginning at addr
ERASE	(addr u --)	Fill u bytes beginning at addr with 0's
BLANKS	(addr u --)	Fill u bytes with blanks beginning at addr.

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:	(end+1 start --)	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.
I	(-- n)	Used between DO and LOOP. Places value of loop counter on stack.
J	(-- n)	Used when DO LOOPs 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: (end+1 start ——) loop: (n ——)	DO as above. +LOOP adds top stack value to loop counter (index)
IF ... ENDIF if: (f ——)	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 ... ELSE ... ENDIF if: (f ——)	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.
THEN	May be used as a synonym for ENDIF.
BEGIN ... UNTIL until: (f ——)	Loop which executes the words between BEGIN and 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. (Note: this loop may be exited by executing R> DROP one level below)
BEGIN ... WHILE ... REPEAT while: (f ——)	Executes words between BEGIN and WHILE leaving a flag which is tested by WHILE. If the flag is non-zero (true) execute words between WHILE and REPEAT, then jump back to BEGIN. If flag is zero (false), continue execution after the REPEAT.

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	(n1 n2 —)	Print n1 right-justified in field of width n2
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
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	(— f)	Test if BREAK (CLEAR on 99/4)
?KEY	(— n)	Read keyboard. If no key pressed n=0 else n is 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

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
#	(d1 — d2)	Convert next digit of d1 leaving d2
#S	(d — 0 0)	Convert all significant digits
SIGN	(n d — d)	Insert sign of n into number
#>	(d — addr u)	Terminate conversion, ready for TYPE
HOLD	(c —)	Insert ASCII character into string

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)	Read disk block to memory at addr
B/BUF	(— n)	Constant giving disk block size in bytes
BLK	(— addr)	User variable containing current block number
SCR	(— addr)	User variable containing current screen number
UPDATE	(—)	Mark last buffer accessed as updated
FLUSH	(—)	Write all updated buffers to disk
EMPTY-BUFFERS	(—)	Erase all buffers

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 ;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
;CODE	(—)	Create new defining word with execution-time code-routine
<BUILDS ... DOES>		Create new defining word using high
does:	(— addr)	level FORTH.

MISCELLANEOUS WORDS

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

CONTEXT	(— addr)	Return address of pointer to context vocabulary (searched first)
CURRENT	(— addr)	Return address of pointer to current vocabulary (new def'ns 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)	Return address of xxx. If compiling compile address (apostrophe)
HERE	(— addr)	Returns addr of next unused byte in the dictionary
PAD	(— addr)	Returns address of scratch area
IN	(— addr)	User variable containing offset into input buffer
SP@	(— addr)	Returns address of top stack item
ALLOT	(n —)	Leave n byte gap in dictionary
	(n —)	Compile n into the dictionary (comma)

Several SCREENS on the FORTH System Disk serve special purposes. SCREEN 0 may not be modified because it is used by the disk Device Service Routine to locate the object code of the FORTH kernel. SCREEN 3 is the BOOT SCREEN (see BOOT in APPENDIX D), and SCREENS 4 and 5 contain error messages used by several FORTH words. Any disk placed in DSK1 must contain a copy of 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 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) before they become available.

CHAPTER 3

HOW TO USE THE FORTH EDITOR

WORDS INTRODUCED IN THIS CHAPTER

CLEAR
ED@
EDIT
FLUSH
TEXT
WHERE

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

** NOTE: The word "SCREEN" in upper case will refer to a FORTH SCREEN while "screen" will refer to the monitor screen.

You must read the chapter titled "SYSTEM SYNONYMS" 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 DSK2 and your FORTH disk in DSK1. It is necessary to copy SCREENS 4 and 5 from the 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 3 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  \
        .                  |
        .                  \   up to 5 SCREENS because
        .                  /   the system has 5 disk
                           |   buffers
scr#  BLOCK  DROP  UPDATE  /
```

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 99/4A files and FORTH SCREENS. Editing the disk may destroy its integrity!

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. Each 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.

EDITING INSTRUCTIONS

Initialization fills each 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 SCREEN must be filled with blanks before it can be used. Typing a SCREEN number and CLEAR will fill a SCREEN with blanks.

```
1 CLEAR
```

will prepare SCREEN 1 for use by the EDITOR.

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

```
EDIT
```

The above instruction will bring the contents of 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 function key on your keyboard. To clear the SCREEN, type the SCREEN number and the word CLEAR.

To reenter the EDITOR, you do NOT have to type 1 EDIT again. A special FORTH word,

ED@

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 "(" and end with a ")". Note: The left parenthesis MUST be followed by at least 1 space. Press ENTER to move to the next line.

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
LEFT ARROW	- moves the cursor one position to the LEFT.
RIGHT ARROW	- moves the cursor one position to the RIGHT.
UP ARROW	- moves the cursor UP one position.
DOWN ARROW	- moves the cursor DOWN one position.
DELETE	- deletes the character on which the cursor is placed.
INSERT	- 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	- 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	- 40-column editor: moves the cursor 28 positions to the RIGHT if the cursor is on the LEFT half of a SCREEN. Otherwise, it moves the cursor 28 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 REDO	- 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.
** c-8	- will insert a blank line just above the line the cursor is on.
** c-W	- will TAB forward by 4 cols.
** f-W	- will TAB backwards by 4 cols.

** f = function
c = control

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 will read in the succeeding SCREEN. Pressing PROCEED will read in the preceeding 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 occured.

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.

** NOTE: 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.

CHAPTER 4

MEMORY MAPS

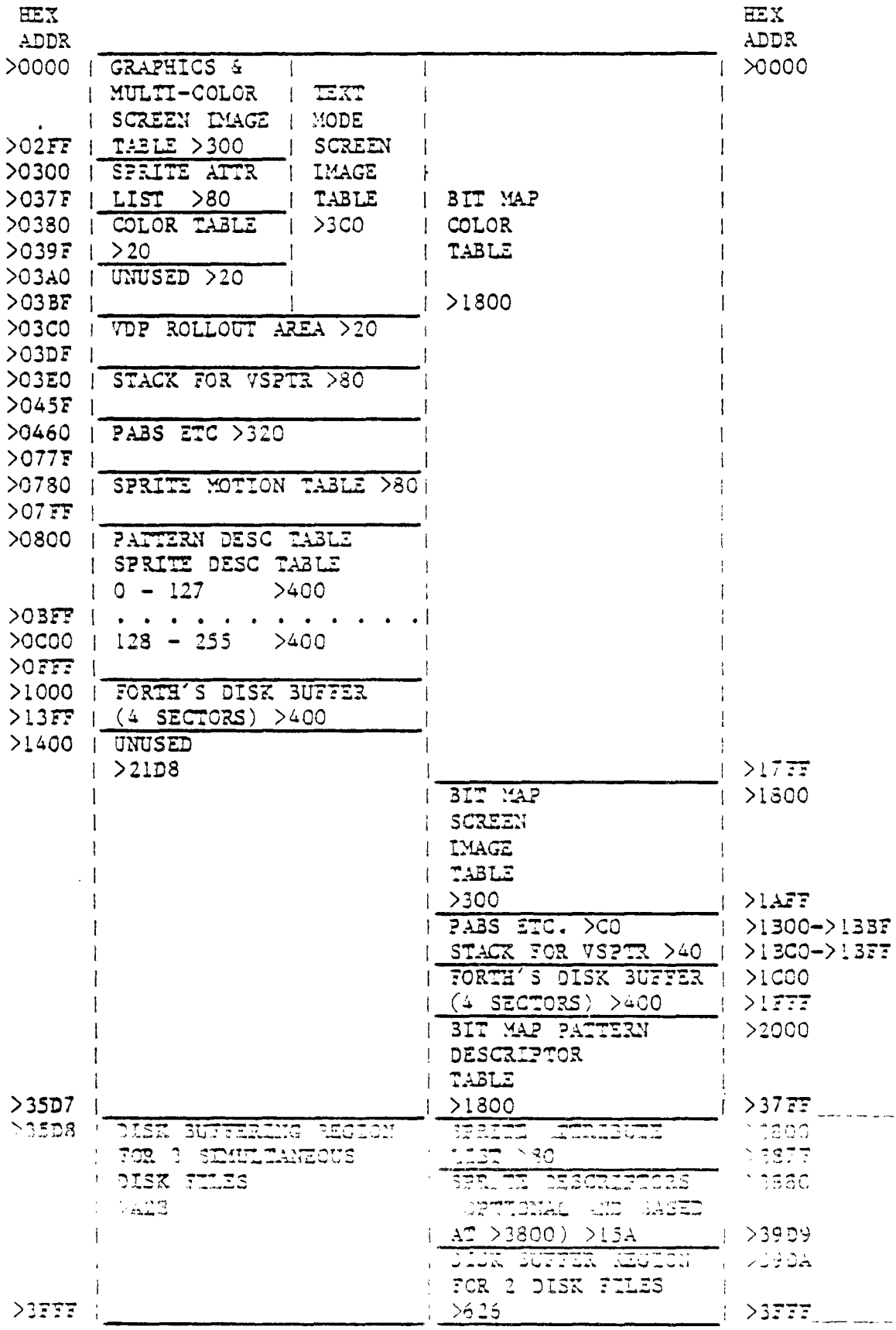
The following diagrams illustrate the memory allocation in the 99/4A system. For more detailed information, see the EDITOR/ASSEMBLER manual.

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, MULTI-COLOR 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 this figure, the GRAPHICS2 mode on the right half. The area at >03C0 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.

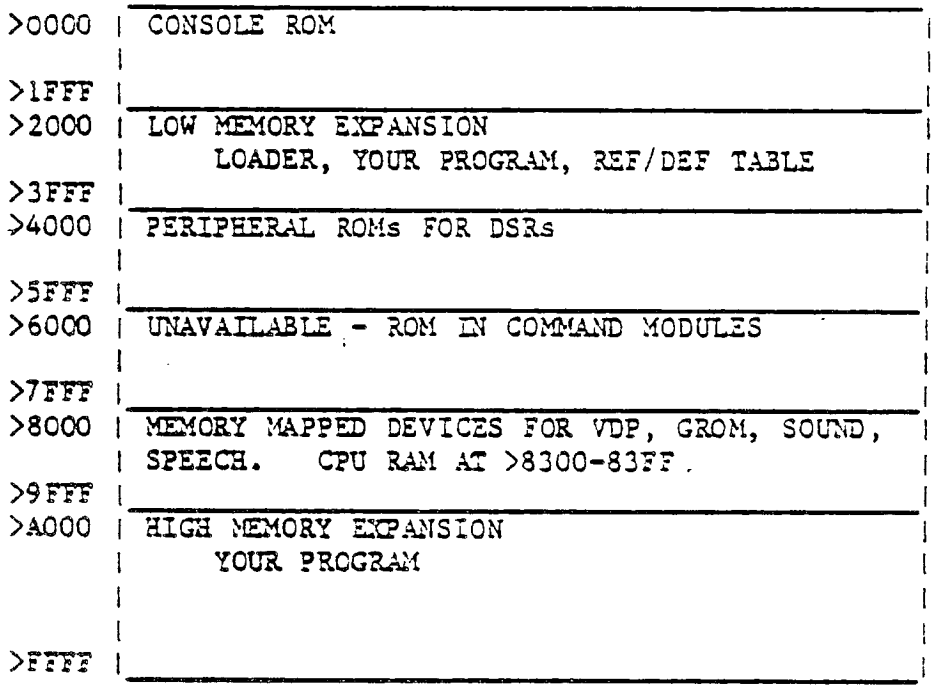
The only CPU RAM on a true 16-bit data bus is in the console at >8300. Because this is the fastest RAM in the system, the FORTH Workspace and the most frequently executed code of the interpreter are placed in this area to maximize the speed of the TI FORTH system. The use of the remainder of the RAM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

The 32K byte memory expansion is divided into an 8K piece at >2000 and a 24K piece at >A000. 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.

VDP MEMORY MAP



CPU MEMORY


>0000
CONSOLE ROM
>1FFF
>2000
LOW MEMORY EXPANSION
LOADER, YOUR PROGRAM, REF/DEF TABLE
>3FFF
>4000
PERIPHERAL ROMs FOR DSRs
>5FFF
>6000
UNAVAILABLE - ROM IN COMMAND MODULES
>7FFF
>8000
MEMORY MAPPED DEVICES FOR VDP, GROM, SOUND, SPEECH. CPU RAM AT >8300-83FF.
>9FFF
>A000
HIGH MEMORY EXPANSION
YOUR PROGRAM
>FFFF

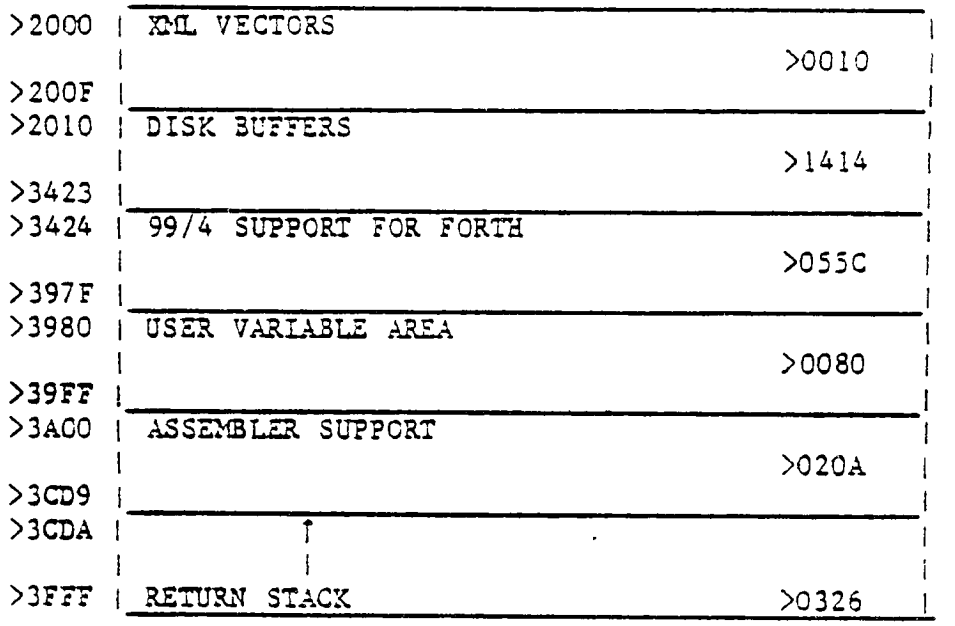
CPU RAM PAD

	HEX ADDR	
	>8300	FORTH'S WORKSPACE
	>831F	
*	>832E	FORTH'S INNER INTERPRETER ETC.
	>8347	
*	>834A	FAC (floating point accumulator)
	>8351	
*	>8356	SUBROUTINE POINTER FOR DSR's
	>8357	
*	>835C	ARG (floating point argument register)
	>8363	
*	>8370	HIGHEST AVAILABLE ADDRESS OF VDP RAM
	>8371	
	>8372	LEAST SIGNIFICANT BYTE OF DATA STACK PTR
	>8373	LEAST SIGNIFICANT BYTE OF SUBR STACK PTR
	>8374	KEYBOARD NUMBER TO BE SCANNED
	>8375	ASCII KEYCODE DETECTED BY SCAN ROUTINE
	>8376	JOYSTICK Y-STATUS
	>8377	JOYSTICK X-STATUS
*	>8379	VDP INTERRUPT TIMER
	>837A	NUMBER OF SPRITES THAT CAN BE IN AUTOMOTION
	>837B	VDP STATUS BYTE ** BIT 0 ON DURING VDP INTERRUPT BIT 1 ON WHEN 5 SPRITES ON A LINE BIT 2 ON WHEN SPRITE COINCIDENCE BIT 3-7 NO. OF 5TH SPRITE ON A LINE
	>837C	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
*	>8380	THE DEFAULT SUBROUTINE STACK ADDRESS
*	>83A0	THE DEFAULT DATA STACK ADDRESS
*	>83C0	RANDOM NO. SEED (BEGIN INTERRUPT WORKSPACE)
	>83C2	BIT 0 DISABLE ALL OF THE FOLLOWING BIT 1 DISABLE SPRITE MOTION BIT 2 DISABLE AUTO SOUND BIT 3 DISABLE SYSTEM RESET KEY (QUIT)
	>83C4	LINK TO ISR HOOK
	>83D4	CONTENTS OF VDP REGISTER 1
*	>83E0	BEGIN GPL WORKSPACE
	>83FF	

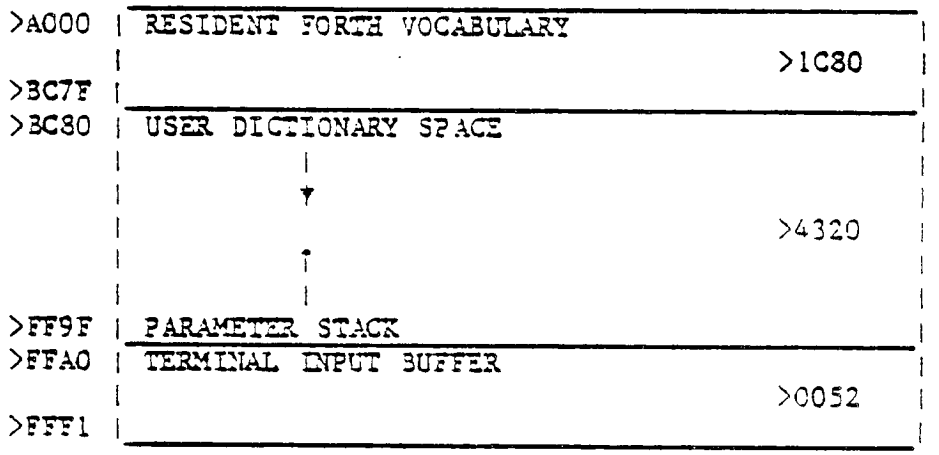
LOCATIONS OMITTED ARE NOT USED BY FORTH BUT MAY BE USED BY SYSTEM ROUTINES

** BIT 0 = HIGH ORDER BIT

LOW MEMORY EXPANSION



HIGH MEMORY EXPANSION



CHAPTER 5

SYSTEM SYNONYMS AND MISCELLANEOUS UTILITIES

WORDS INTRODUCED IN THIS CHAPTER

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

SYSTEM SYNONYMS

Several utilities are available to give you simple access to many resources of the TI 99/4A HOME COMPUTER. These utilities allow you to change the display, access the Device Service Routines for peripheral devices such as RS232s and disk drives, link your program to GPL and ASSEMBLER routines, and perform operations on VDP memory locations.

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

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 - Writes a single byte to VDP RAM. It requires 2 parameters on the stack; a byte to be written and a and a VDP address.

base	byte	vaddr	instr
HEX	A3	380	VSBW

places the value HEX A3 into VDP address HEX 380.

VMBW - 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	cnt	instr
HEX	PAD	808	4	VMBW

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

VSBR - 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 HEX 781 on the stack.

VMBR - 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	cnt	instr
HEX	300	PAD	20	VMBR

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

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 - Allows you to link your program to Graphics Programming Language 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 HEX 16 which loads the standard character set into VDP RAM. It then returns to your program.

XMLLNK - 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 HEX 800, and returns to your program.

DSRLNK - Links a FORTH program to any Device Service Routine in ROM. Before this instruction is used, a Peripheral Access Block must be set up in VDP RAM. A PAB contains information about the file to be accessed. See the EDITOR/ASSEMBLER manual and Chapter 9 of this manual for additional setup information. DSRLNK needs no parameters.

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 requires 2 parameters; a byte to be written and a VDP register number.

base	byte	reg	instr
HEX	F5	7	VWTR

The above instruction writes a HEX F5 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 lt. blue.

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 require 2 parameters; a byte to be used as the second operand and the VDP address at which the operation is to be performed. The result of the operation is placed back into this address.

base	byte	vaddr	instr
HEX	F0	304	VAND
HEX	F0	304	VOR
HEX	F0	304	VXOR

Each of the above instructions reads the byte stored at HEX 804 in VDP RAM, performs an AND, OR, or XOR on that byte and HEX F0, and places the result back into VDP RAM at HEX 804.

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	cnt	byte	instr
HEX	300	20	0	VFILL

fills 32 locations, starting at HEX 300, with zeros.

DISK UTILITIES

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 DSK1, put a 0 on the stack, etc. Next, type FORMAT-DISK.

```
0  FORMAT-DISK
```

will initialize the disk in DSK1, thus preparing it for use by the FORTH system. Disks initialized by the DISK MANAGER are properly formatted and may be used.

The TI FORTH System Disk, or any disk which contains a copy of SCREENS 0 thru 19 of the System Disk, may be copied with the Disk Manager. Any other disk may be copied with the Disk Manager only after a special header has been written on it by the TI FORTH instruction DISK-HEAD.

Any disk which can be copied by the Disk Manager can also be accessed from TI BASIC. If you access a FORTH disk which contains the FORTH kernel, 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 basic, but the location of the records will be different. The file 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.

To copy an entire FORTH disk without using the Disk Manager, you must place the new disk in DSK1 and the source disk in DSK2. Typing FORTH-COPY will copy the entire contents of the disk in DSK2 onto the disk in DSK1. NOTE: 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 !
```

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

base	source	destin	instr
DECIMAL	5	17	SCOPY

The above instructions 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.

The SMOVE instruction acts as a multiple SCOPY. It allows you to copy a group of 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, overlaping 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	source	destin	cnt	instr
DECIMAL	11	36	7	SMOVE

These instructions 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 DSK1, SCREENS 90 - 179 are located on the disk in DSK2, 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 (2x90). This is accomplished by executing the following instruction:

```
180  DISK_HI  !
```

Therefore, to copy SCREEN 6 on DSK1 to SCREEN 20 on DSK2, you would type:

base	source	destin	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.

If you have reason to suspect that a 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 DSK1. 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.

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. The first, called TRIAD, requires a 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 which 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.

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.

The INDEX instruction allows you to list to your terminal the comment line 0's of a specified range of SCREENS. INDEX expects a beginning and and 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 to exit execution prematurely.

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 as in INDEX above.

DEBUGGING

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	cnt	instr
HEX	2F26	100	DUMP

will list 256 (>100) bytes of memory beginning at address >2F26 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 prematurely, press BREAK.

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.

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 * * ;  OK   ( ROUTINE TO BE TRACED )
UNTRACE  OK   ( DON'T COMPILE NEXT DEF. 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 EXEC. 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
```

A more complex TRACE example involves a recursive routine. Normally, a FORTH word can not 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. 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 example uses a recursive factorial routine for illustration:

```
DECIMAL   OK
TRACE   OK    ( COMPILE FOLLOWING DEF. UNDER TRACE OPTION )
: FACT DUP 1 > IF DUP 1 - MYSELF * ENDIF ;   OK
UNTRACE   OK
TRON   OK    ( ACTIVATE TRACE )
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.

RANDOM NUMBERS

Two different random number functions are available in FORTH. The first, RND generates a positive random integer between , and a specified range. The second, RND1, generates a random word (2 bytes). No range is specified

for RNDW.

base	range	instr
DECIMAL	13	RND

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

```
RNDW
```

will place on the stack a number from 0 to HEX FFFF.

To gaurantee 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.

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.

MISCELLANEOUS INSTRUCTIONS

To store a string at a specified address, the !" instruction is used. !" 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.

base	addr	instr	string
HEX	PAD	!" HOW ARE YOU?"	

To clear the display screen, the word CLS is used. This may be used inside a colon definition or directly from the keyboard. CLS will not clear bit-map displays or SPRITES.

CHAPTER 6

AN INTRODUCTION TO GRAPHICS

WORDS INTRODUCED IN THIS CHAPTER:

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

GRAPHICS MODES

The TI HOME COMPUTER posesses 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 can not be defined.

4) BIT-MAP MODE (GRAPHICS2) - This mode is available only on the 99/4A. BIT-MAP MODE allows you to set any pixel on the screen and to assign its color rather than 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 mode 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, or SPLIT2.

FORTH GRAPHICS WORDS

Many FORTH words have been defined to make graphics handling much easier for the user. As these 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) 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 HEX than in DECIMAL.

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, set one contains 8 - 15, etc. Sixteen colors are available on the TI HOME COMPUTER.

color	hex value	color	hex value
TRANSPARENT	0	MED. RED	8
BLACK	1	LT. RED	9
MED. GREEN	2	DK. YELLOW	A
LT. GREEN	3	LT. YELLOW	B
DK. BLUE	4	DK. GREEN	C
LT. BLUE	5	MAGENTA	D
DK. RED	6	GRAY	E
CYAN	7	WHITE	F

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 -

```
HEX       B  SCREEN   or
DECIMAL  11  SCREEN
```

(G)

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 instructions will change character set 26 (characters 208 - 215) to have a foreground color of dk. blue and a background color of magenta.

PLACING CHARACTERS ON THE SCREEN

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 repititions 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	cnt	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 and row 17, that will wrap from right to left on the screen.

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.

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)	

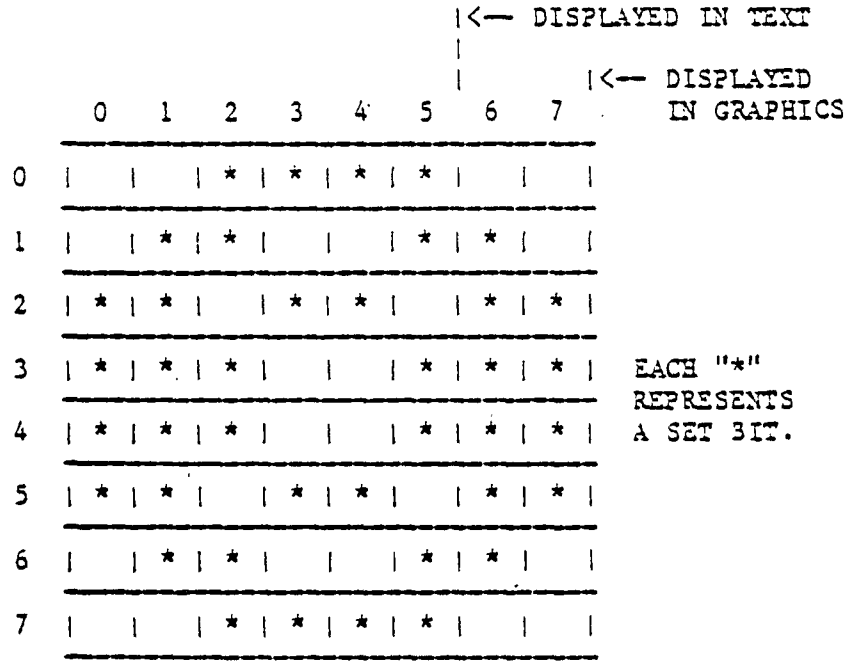
a 2A HEX or 42 DECIMAL would be left on the stack.

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 -

	3C66	DBE7	E7DB	663C
rows	0-1	2-3	4-5	6-7

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

base	w1	w2	w3	w4	char#	instr	
HEX	3C66	DBE7	E7DB	663C	7B	CHAR	or
DECIMAL	15462	56295	59355	26172	123	CHAR	
						(T G)	

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

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 O )
```

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

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.

MAGNIFICATION

SPRITES may be defined in 4 different sizes or magnifications.

magnification factor

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.

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 The SPRITE fills 16 positions.

The default magnification is 0. To alter SPRITE magnification, use the FORTH word MAGNIFY.

```
mag  instr
---  ------
 2   MAGNIFY
       ( G M B )
```

will change all SPRITES to double size and unmagnified.

SPRITE INITIALIZATION

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 Ch. 4) or moving the SPRITE Descriptor Table elsewhere in memory. This move is highly recommended to avoid confusion. HEX 2000 is usually a good location, but any available 2K (>800) boundary will do.

base	addr	instr	
HEX	2000	SSDT	or
DECIMAL	8192	SSDT	
		(G M 3)	

will move the SPRITE Descriptor Table to 2000 HEX. Use the value HEX 800 with the SSDT instruction if you do not want to move the Descriptor Table.

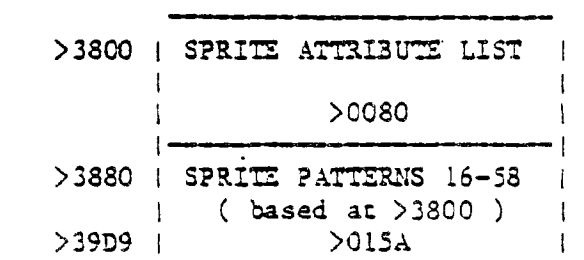
** NOTE: Whether or not you choose to move the table, you MUST execute this instruction before you can use SPRITES in your program!!!

USING SPRITES IN BIT-MAP MODE

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 HEX 1B00 as follows.

HEX 3800 ' SATR

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 (>3800), 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.)



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.

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.

base	w1	w2	w3	w4	char#	instr
HEX	0F0F	2424	F0F0	4242	0	SPCHAR or

```
DECIMAL  3855   9252   61680  8770   0   SPCHAR
                                          ( G M B )
```

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

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	dc	dr	col	char	spr#	instr	
HEX	6B	4C	5	10	1	SPRITE	or
DECIMAL	107	76	5	16	1	SPRITE	

(G M B)

defines SPRITE #1 to be located at column 107 and row 76, to be lt. 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.

base	char#	spr#	instr	
HEX	14	1	SPRPAT	or
DECIMAL	20	1	SPRPAT	

(G M B)

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

base	col	spr#	instr	
HEX	C	2	SPRCOL	or
DECIMAL	12	2	SPRCOL	

will change the color of SPRITE #2 to lt. green.

base	dc	dr	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.

SPRITE AUTOMOTION

In GRAPHICS or MULTI-COLOR 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 HEX 0 to 7F are positive velocities (down for vertical and right for horizontal), and from FF to 80 are taken as two's complement negative velocities.

```
          base      xv   yv   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.

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.

no	instr
6	#MOTION

(G M)

will set SPRITES #0 - #5 in motion.

no	instr
0	#MOTION

will stop all SPRITE AUTOMOTION, but motion will resume when another #MOTION instruction is executed.

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.

DISTANCE AND COINCIDENCES BETWEEN SPRITES

It is possible to determine the distance between two SPRITES or between a SPRITE and a point on the screen. This capability comes in handy when writing game programs.

[illegible]	[illegible]	[illegible]
2	4	SPRDIST

(G M B)

returns on the stack the SQUARE of the distance between

SPRITE #2 and SPRITE #4.

base	dc	dr	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.

To find a coincidence between two SPRITES, the FORTH word COINC is used.

spr#	spr#	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.

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

base	dc	dr	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.

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.

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.

spr#	instr
2	DELSPR

will remove SPRITE #2 from the screen by altering its description in the SPRITE Attribute List (see VDP Memory

Map in Ch. 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
(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.

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.

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 1/4 the size of a standard character; therefore more of them fit across and down the screen.

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
DECIMAL	11	26	44	MCHAR

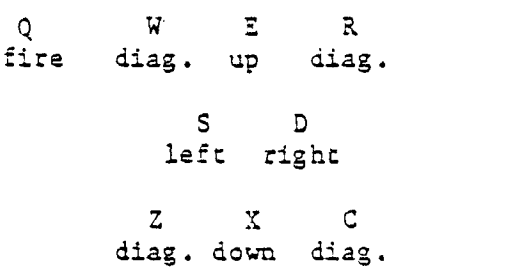
The above instruction will place a lt. yellow square at (26,44).

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

USING JOYSTICKS

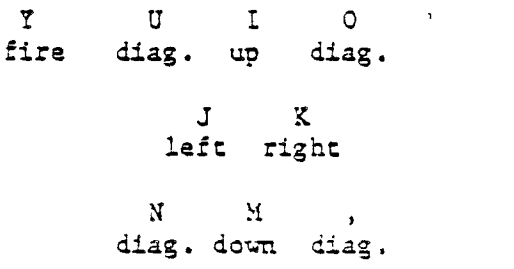
The JOYST instruction allows you to use Joysticks in your FORTH program. JOYST requires only one parameter; a Keyboard number. The Keyboard number tells the computer which Joystick or which side of the Keyboard to scan for input. When Keyboard #1 is specified, both Joystick #1 and the left side of the Keyboard are scanned. When Keyboard #2 is specified, 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. The following diagram shows which keys have which function.

When Joystick #1 is specified:



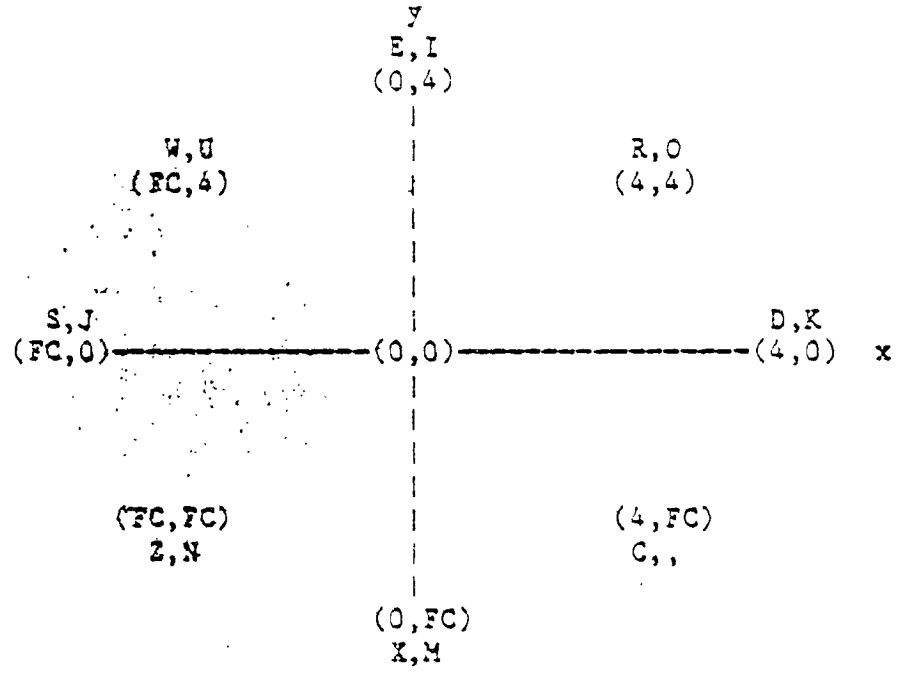
Legal input keys on the LEFT side of the Keyboard. Q is used as the FIRE button.

When Joystick #2 is specified:



Legal input keys on the RIGHT side of the Keyboard. Y is used as the FIRE button.

The JOYST instruction returns 3 numbers on the stack: an ASCII code (on the bottom of the stack), an X Joystick status, and a Y Joystick status (on the top of the stack). The Joystick positions are illustrated in the following diagram.



Hex FC equals decimal 252.

The capital letters indicate which keys on the left and right side of the keyboard return these values.

****NOTE**** The ascii value of all FIRE buttons is 18.

If no Keyboard key is pressed, the returned values will be an ascii code 255, and the current X and Y Joystick positions. If a Keyboard key was pressed, the ascii value of that key will be returned along with its translated directional meaning (see above diagram).

If an illegal Keyboard key is pressed, three 0's will be returned. If the FIRE button is pressed, an ascii 18 along with two 0's will be returned.

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

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:

1) DRAW - plots dots in the 'on' state
2) UNDRAW - plots dots in the 'off' state
3) DTOG - toggles dots between the 'on' and 'off' state. If the dot is 'on', DTOG will turn it 'off' and vice-versa.

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.

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	dc	dr	instr
DECIMAL	34	12	DOT

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

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 HEX 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.

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	dc1	dr1	dc2	dr2	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 MODE and DCOLOR before using LINE.

SPECIAL SOUNDS

Two special sounds can be used to enhance your graphics application. The first is called BEEP and produces a pleasant high pitched sound. The other, called HONK, produces a less pleasant low tone. To use these noises in your program, simply type the name of the sound you want to hear. No parameters are needed.

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 Meory Map in Chapter 4.

name	type	description	default
COLTAB	C	VDP address of Color Table	HEX 380
DMODE	V	Dot graphics drawing mode	0
PDT	C	VDP address of Pattern Desc. Table	HEX 800
SATR	C	VDP address of Sprite Attrib. Table	HEX 300
SMTN	C	VDP address of Sprite Motion Table	HEX 780
SPDTAB	C	VDP address of Sprite Desc. Table	HEX 800

CHAPTER 7

THE FLOATING POINT SUPPORT PACKAGE

WORDS INTRODUCED IN THIS CHAPTER

>ARG	F0<	FMUL
>F	F0=	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 effeciency of these operations by optimizing the code for this implementation the details are given in the latter portion of this chapter.

The floating point numbers in the 99/4A occupy 4 words (8 bytes) each. In order to simplify stack manipulations with these numbers the following stack manipulation words are presented: FDUP FDROP FSWAP AND FOVER. Floating point numbers can be stored and fetched by using the F! and F@ words. 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.)

The following words put floating point numbers on the stack so that the above operations can be used. A 16-bit number can be converted to floating point by using the S->F word. It functions by replacing the 16-bit number on the stack by a floating point number of equal value. Its inverse is F->S which starts with a floating point number on the stack and leaves a 16-bit integer. In addition the word >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:

FLOATING POINT NUMBER ENTRY		
>F 123	or	123 S->F
>F 123.456		
>F -123.456789		
>F 1.234E-6		
>F 9876E88		
>F 0	or	0 S->F

Floating point arithmetic can now be performed on the stack just as it is with integers. The four arithmetic operators are: F+ , F- , F* and F/ . The word PI is available to place 3.141592653590 on the stack.

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.

FLOATING POINT COMPARISON WORDS

F0<	True if f1 on stack is negative
F0=	True if f1 on stack is zero
F>	(f11 f12 -- f) f is true if f11 > f12
F=	(f11 f12 -- f) f is true if f11 = f12
F<	(f11 f12 -- f) f is true if f11 < f12

The word F. is used to print the floating point number on the top to 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 (scientific) format is used for very large or very small numbers.

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 R where R is a 16-bit word added to the top of the stack for this purpose.

Two additional words are used for more specific formating. They are FF. and FF.R . FF. requires two integers on the stack above the floating point number. They control the maximum number of digits to convert and the number of digits following the decimal point. FF.R adds the printing field width to this to make a total of three

integers

(f1 max-digits dig-after-. field width --)

The following transcendental functions are also available:

TRANSCENDENTAL FUNCTIONS

INT	fl1 —— fl2	Returns largest integer not larger then the input
*	fl1 fl2 —— fl3	Fl3 is fl1 raised to the fl2 power
SQR	fl1 —— fl2	Fl2 is the square root of fl1
EXP	fl1 —— fl2	Fl2 is e (2.718281828...) raised to the fl1 power
LOG	fl1 —— fl2	Fl2 is the natural log of fl1
COS	fl1 —— fl2	Fl2 is the cosine of fl1 (in radians)
SIN	fl1 —— fl2	Fl2 is the sin of fl1 (in radians)
TAN	fl1 —— fl2	Fl2 is the tangent of fl1 (in radians)
ATN	fl1 —— fl2	Fl2 is the arctangent (in radians) of fl1.

CAUTION! A conflict exists when using transcendentals and floating point prints while in bit-map mode. The contents of the VDP Rollout Area (>3C0 - >3DF) must be saved before a transcendental or floating point print is executed, and restored upon completion.

** NOTE: The transcendentals also use the area known as the stack for VSPTR (See VDP Memory Map in Ch. 4). This area is pointed to by >836E.

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

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. The words >FAC and >ARG move floating point data from the

stack to these two locations. FAC> is used to move data from FAC to the stack. Each of the binary floating point operations require that two numbers be moved from the stack to FAC and ARG. SETFL does this by calling >FAC and >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.

When conversion from 16-bit integer to floating point is performed, it is done in the FAC . If the user desires the result to remain there rather than to be brought back to the stack the word S->FAC can be used.

Several miscellaneous words include FAC->S to convert the contents of FAC to a 16-bit integer on the stack. FAC>ARG moves the contents of FAC to ARG . VAL is used to convert a string at PAD to a floating point number. FLERR is used to fetch the contents of the floating point error register (see Editor/Assembler manual) to the stack. If there is a possibility of a floating point error condition ?FLERR can be used to test for and flag such a condition.

CHAPTER 8

ACCESS TO FILE I/O USING 99/4A DEVICE SERVICE ROUTINES

WORDS INTRODUCED IN THIS CHAPTER

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

This chapter will explain the means by which different types of data files native to the 99/4A are accessed with TI-FORTH. To further illustrate the material, two commented examples have been included on the last pages of this chapter. The first demonstrates the use of a Relative disk file, and the other 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 and EDITOR/ASSEMBLER manuals. 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 that [illegible] they are initialized [illegible] the [illegible] [illegible] and [illegible] NOT contain FORTH SCREENS.

Before any file access can be achieved, a Peripheral Access Block (PAB) must be set up which 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 begining of this region. DO NOT use the first 2 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. The following diagram illustrates the structure of a PAB.

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

All Device Service Routines (DSRs) on the 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.

VARIABLES USED BY FILE I/O

PAB-ADDR	Points into VDP RAM to first byte of the PAB
PAB-BUF	Points into CPU RAM to first byte in FORTH'S memory where allocation has been made for this buffer
PAB-VBUF	Points into VDP RAM to first byte of a region of adequate length to store data temporally while it is transfered between the file and FORTH. The area of VDP which is used for this purpose is labled "UNUSED" on the VDP Memory Map in Chapter 4. If working in bit-map mode, be cautious as to where PAB-VBUF is placed.

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 above buffers before the word FILE may be used. The values to be used for the above variables 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
                              ( will be set as defined here.)
JOE                           ( Use the word before using any )
                              ( other FILE I/O words )
```

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

Files on the 99/4A have various characteristics which 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 one referenced is the one most recently named.

FILE ATTRIBUTE WORDS

Attribute Type	Options From BASIC	From 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
Data Type	DISPLAY	DSPLY	* File contains printable or displayable characters
	INTERNAL	INTRNL	File contains data in machine 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

To specify the record length for a file, the desired length should be on the stack when the word REC-LEN is executed. The length will be placed in the PAB. Every file must have a name to specify the device and file to be accessed. This is performed with the 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"
```

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 a conflict with one or more existing FORTH words.

WORDS THAT PERFORM FILE I/O

From BASIC	From FORTH	DSR Opcode
OPEN	OPN	0
CLOSE	CLSE	1
READ	RD	2
WRITE	WRT	3
RESTORE	RSTR	4
LOAD	LD	5
SAVE	SV	6
DELETE	DLT	7
SCRATCH	SCRTCH	8
STATUS	STAT	9

OPN opens the file specified by the currently selected PAB. CLSE works similarly for closing a file.

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

The RD instruction will transfer the contents of the record into your PAB-BUF and leave a character count on the stack. WRT takes a character count from the stack and moves

that number of characters from the PAB-BUF to the desired file. RSTR takes a record number from the stack and restores a relative file to that record. LD and SV are used to read and write program files respectively. They each require a byte count on the stack. For SV this is the number of bytes to save; for LD it is the maximum number of bytes to read. Both of these commands expect or place the data in the VDP RAM at the address specified in PAB-VBUF. OPN and CLSE need not be used with LD and SV . DLT is used to delete a file. SCRTCH is used to remove a relative record. It requires a record number on the stack. STAT returns the status of the specified device/file.

The words GET-FLAG , PUT-FLAG , CLR-STAT , CHK-STAT , I/OMD , CHAR-CNT! , CHAR-CNT@ , N-LEN! and DOI/O are available for the advanced user and their utility will be obvious to that user when the definitions on disk are examined.

Examples of File I/O in use are available on the SCREENS that define the Alternate I/O capabilities for printing to the RS232.

ALTERNATE INPUT AND OUTPUT

The words SWCH and UNSWCH make it possible to send output that would normally go to the monitor to the RS232 printer. For example, the LIST instruction normally outputs to the monitor. By typing

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

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 a conflict with one or more existing FORTH words.

WORDS THAT PERFORM FILE I/O

From BASIC	From FORTH	DSR Opcode
OPEN	OPN	0
CLOSE	CLSE	1
READ	RD	2
WRITE	WRT	3
RESTORE	RSTR	4
LOAD	LD	5
SAVE	SV	6
DELETE	DLT	7
SCRATCH	SCRTCH	8
STATUS	STAT	9

OPN opens the file specified by the currently selected PAB. CLSE works similarly for closing a file.

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

The RD instruction will transfer the contents of the record into your PAB-BUF and leave a character count on the stack. WRT takes a character count from the stack and moves

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
FXD	Records will be of Fixed length
DSPLY	Records will contain printable information
40 REC-LEN	Record length is 64 (>40) bytes
P-D" DSK2.TEST"	Will create a disk file called TEST
OPN	Open the file

. . . . . .

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 ( >10) records
       DUP                  Duplicate address
       BUFR 40 CMOVE        Move 64 bytes of the 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
```

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 over 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 accesed 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 address in memory of the information to be sent to the printer
20 0 DO	Will write 32 records
DUP	Duplicate address
MY-BUF 50 CMOVE	Move 80 characters from FILE-STUFF to MY-BUF
50 WRT	Write one record to printer
50 +	Increment address on stack
LOOP DROP	Clear stack
	[illegible]
PRNT	Execute write procedure
CLSE	Close the file called PRNTR

CHAPTER 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 9900 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 following table.

9900 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 all 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 9900 assembly code.

FORTH'S WORKSPACE REGISTERS

Reg Name	Usage
0	\
1	|
2	|
3	\ These registers are available.
4	/ They are used only within FORTH
5	| words written in CODE.
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 NEXT instruction fetch routine

When the assembler is loaded, it is loaded into the ASSEMBLER vocabulary. To use the assembler, type ASSEMBLER to make it the context vocabulary. Assembly definitions begin with either the word CODE or ;CODE These are used in the following way:

```
ASSEMBLER
CODE    EXAMPLE
```

This begins the definition of a code routine named EXAMPLE. The above words would be followed by assembly mnemonics as desired. ;CODE is used as very much like the word DOES> :

```
ASSEMBLER
:  DEF-WRD
    .
    . an existing defining word must be included
    . here to create the dictionary header.
    .
   ;CODE
   assembly mnemonics
```

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 .

We will now introduce those words that permit this assembler to perform the various addressing modes of which the 9900 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.

WORKSPACE REGISTER ADDRESSING

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

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

SYMBOLIC MEMORY ADDRESSING

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

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

WORKSPACE REGISTER INDIRECT ADDRESSING

Forth	Conventional Assembler		
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

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

WORKSPACE REGISTER INDIRECT AUTOINCREMENT ADDRESSING

Forth	Conventional Assembler		
2000 CONSTANT XRAM	XRAM	EQU	>2000
CODE EX4		DEF	EX4
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]		[illegible]	[illegible]
[illegible]		[illegible]	[illegible]

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

INDEXED MEMORY ADDRESSING

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

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.

ADDRESSING MODE WORDS FOR SPECIAL REGISTERS

Register Addr	Indirect	Indexed	Indirect Autoincrement
W	*W	@(W)	*W+
RP	*RP	@(RP)	*RP+
IP	*IP	@(IP)	*IP+
SP	*SP	@(SP)	*SP+
UP	*UP	@(UP)	*UP+
NEXT	*NEXT	@(NEXT)	*NEXT+

This assembler also permits the user to write structured (label-less) code. 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:

STRUCTURED ASSEMBLER CONSTRUCTS

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

The three conditional words in the previous list (IF, UNTIL, WHILE,) must each be preceeded by one of the following jump tokens:

ASSEMBLER JUMP TOKENS

Token	Comment	
EQ	True if =	(uses JNE)
GT	True if signed >	(uses JGT $+1 JMP)
GTE	True if signed > or =	(uses JLT)
H	True if unsigned >	(uses JLE)
HE	True if unsigned > or =	(uses JL)
L	True if unsigned <	(uses JHE)
LE	True if unsigned < or =	(uses JH)
LT	True if signed <	(uses JLT $+1 JMP)
LTE	True if signed < or =	(uses JGT)
NC	True if No Carry	(uses JOC)
NE	True if equal bit not set	(uses JEQ)
NO	True if No Overflow	(uses JNO $+1 JMP)
NP	True if Not odd Parity	(uses JOP)
OC	True if Carry bit is set	(uses JNC)
OO	True if Overflow	(uses JNO)
OP	True if Odd Parity	(uses JOP $+1 JMP)

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

ASSEMBLY EXAMPLE FOR STRUCTURED CONSTRUCTS

```
     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     *NEXT
```

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.

CHAPTER 10

INTERRUPT SERVICE ROUTINES (ISR's)

The TI-99/4A has a 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 ISR's. 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:

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 (hex) 83C4 (decimal) 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 a 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 the ISR after any keyboard character or not a key has been pressed.

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.

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 1/60th second to do meaningful computing.

AN EXAMPLE OF AN INTERRUPT SERVICE ROUTINE

```
0 VARIABLE TIMER      ( TIMER WILL HOLD THE CURRENT COUNT )
: UP 100 ALLOT ;      ( MOVE HERE AND THUS PAD UP 100 BYTES )
: DOWN -100 ALLOT ;   ( RESTORE PAD TO ITS ORIGINAL LOCATION )
: DEMO UP             ( MOVE PAD TO AVOID CONFLICT )
       1 TIMER +! TIMER @  ( INC 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 )
```

To execute this for the following code may be

INSTALLING THE ISR

```
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 )
83C4 !          ( PUT ISR HOOK INTO CONSOLE INTERRUPT ROUTINE )
                ( NOTE: THE CFA MUST BE IN USER VARIABLE ISR )
                ( BEFORE WRITING TO 83C4 )
```

To reverse the installation of the ISR one can either write a 0 to 83C4 or place the CFA of NOP (a do nothing instruction) in User Variable ISR.

Some additional thoughts concerning the use of ISR's:

1) ISRs are uninterruptable. 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.
2) Caution must be exercised when using PAD, 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.
3) An ISR must never expect nor leave anything on the stacks. It may however use them in the normal manner during execution.
4) Disk activity stops interrupts as do most of the other DSRs in the 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.

CHAPTER 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.

BLOAD and BSAVE

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

1) The lowest memory address in the dictionary image to be saved to disk.

2) The SCREEN number to which the saved image will be written.

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

Each SCREEN of the saved image has the following format:

Byte #	Contents
0 - 1	Address at which the first image byte of this 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 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 SCREEN number 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 0 on the stack signifying a successful load. If the letters "ti" are not found, then the BLOAD is not performed and a 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. 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.

To save an entire application to the disk starting with screen 30, the user would enter the following into the TI FORTH system:

```
TASK 30 BSAVE
```

The number of the next available screen will be printed. To reload this application you would place on Screen 3 (the auto boot screen):

```
30 BLOAD
```

CONDITIONAL LOADS

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

WLITERAL is used to compile SLIT and the desired character string into the current dictionary definition.

To use CLOAD there must always be a 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 SCREEN number on the stack is loaded.

```
27 CLOAD FOO
```

This instruction, for example, will load 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 screen. This is done by using the command:

```
0 CLOAD TESTWORD
```

If this line of code was located on screen 30, 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.

MEMORY RESIDENT MESSAGES

If the user desires, he may elect to use a version of MESSAGE which is provided on the system disk (SCR 34). 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 SCREENS with the routine. Installing "message" will remove the 5th disk buffer from the system and use that memory for storing the error mesages. 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 either 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 CFA's of WARNING and @ .

CRU WORDS

Five words have been included to assist in performing CRU related functions. They allow the FORTH programmer to perform the LDCR, STCR, TB, SBO and SBZ operations of the 9900 without using the assembler. The functions of these words will be apparent when someone familiar with these instructions on the 9900 examines their definitions in the GLOSSARY.

APPENDIX A

ASCII KEYCODES

(Mapping of keystrokes to ASCII sequential order)

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

NOTE f- press function key
c- press control key

continued on next table

ASCII KEYCODES (continued from previous page)

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

NOTE f- press function key
c- press control key

APPENDIX B

ASCII KEYCODES
(Mapping of keystrokes to ASCII in keyboard order)

control key	hex	decimal	function key	hex	decimal
c-1	31	49	f-1	03	3
c-2	32	50	f-2	04	4
c-3	33	51	f-3	07	7
c-4	34	52	f-4	02	2
c-5	35	53	f-5	0E	14
c-6	36	54	f-6	0C	12
c-7	37	55	f-7	01	1
c-8	1E	30	f-8	06	6
c-9	1F	31	f-9	0F	15
c-0	30	48	f-0	3C	60
c-=	1D	29	f-=	05	5
c-Q	11	17	f-Q	39	57
c-W	17	23	f-W	7E	126
c-E	05	5	f-E	0B	11
c-R	12	18	f-R	5B	91
c-T	14	20	f-T	5D	93
c-Y	19	25	f-Y	46	70
c-U	15	21	f-U	5F	95
c-I	09	9	f-I	3F	63
c-O	0F	15	f-O	27	39
c-P	10	16	f-P	22	34
c-/	3B	59	f-/	3A	58
c-A	01	1	f-A	7C	124
c-S	13	19	f-S	08	8
c-D	04	4	f-D	09	9
c-F	06	6	f-F	7B	123
c-G	07	7	f-G	7D	125
c-H	08	8	f-H	3F	63
c-J	0A	10	f-J	40	64
c-K	0B	11	f-K	41	65
c-L	0C	12	f-L	42	66
c-;	1C	28	f-;	3D	61
c-Z	1A	26	f-Z	5C	92
c-X	18	24	f-X	0A	10
c-C	03	3	f-C	60	96
c-V	16	22	f-V	7F	127
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
c-M	0E	14	f-M	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
c-.	1B	27	f-.	39	57

NOTE f- press function key
c- press control key

APPENDIX C

DIFFERENCES BETWEEN THE FORTH IN "STARTING FORTH" AND TI FORTH

PAGE	WORD	CHANGES REQUIRED
10	BACKSPACE	Function-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- .
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 this 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 your TI FORTH manual describing the Editor.
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 ;
84-85		Ignore Editor words.
89ff	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 ;
101ff	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 synonym for I) : I' R> R> R SWAP >R SWAP >R ;

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 preceed SWAP instead of LOOP.
137	XX	When an error is detected in TI FORTH, the stack is clear 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 ff on pg. 163 for definition.
164-165		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.
[illegible]	[illegible]	[illegible]
177	D-	This word is not in TI FORTH, but can be created with the following definition: : D- DMINUS D+ ;

173 DNEGATE — This word is not in TI FORTH, but can be created with the following definition:

```
: DNEGATE   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 ;
```

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 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.
183ff		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 e.g. 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.
204	DUMP	TI FORTH already has a dump instruction which must be loaded from the disk. DUMPS are always printed in HEX. See APPENDIX D for location of DUMP.
207	CREATE	The CREATE word of TI FORTH behaves somewhat differently. Hackers should consult file-FORTH documentation.
216	EXECUTE	Because this word operates a little differently in TI FORTH, it must be preceded by the CFA (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 the TI FORTH 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 the TI FORTH 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-multiprogrammed systems. Not present in TI FORTH.

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": : 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 283.
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

EXPLANATION OF ABBREVIATIONS

ABBR	MEANING
addr, addr1, ...	memory address
b	byte
c	column position
cccc	string representation
cfa	code field address
ch	ascii character code
cnt	count (length)
d, d1, d2, ...	double precision number
dc, dc1, dc2, ...	dot column position
dr, dr1, dr2, ...	dot row position
dsk	refers to DSK1, DSK2, or DSK3
f	boolean flag
ff	boolean false flag
fl, fl1, fl2, ...	floating point number
lfa	link field address
mod	modulo
n, n1, n2, ...	single precision signed number
nfa	name field address
nnnn	string representation
pfa	parameter field address
r	row positopn
rem	remainder
scr#	screen number
spr#	sprite number
tf	boolean true flag
tol	tolerance limit
u	unsigned single precision number
ud	unsigned double precision number
vaddr	VDP address

! n addr — RESIDENT

Store 16 bits of n at address. Pronounced "STORE".

!" addr — SCR 39 -COPY

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 CSP. Used as part of the compiler security.

d1 — d2 RESIDENT

Generate from a double number d1, the next ASCII character which is placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between <# and #>. See #S.

#> d — addr cnt RESIDENT

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

#MOTION n — SCR 59 -GRAPH

Sets SPRITES number 0 to n-1 in AUTOMOTION.

#S d1 — d2 RESIDENT

Generates ASCII text in the text output buffer, by the use of #, until a zero double number d2 results. Used between <# and #>.

' (TICK) —— pfa RESIDENT

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".

((PAREN) —— RESIDENT

Used in the form:

(cccc)

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

(.") (PDOTQ) —— RESIDENT

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

(;CODE) (PSEMICODE) —— RESIDENT

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

(+LOOP) (PPLOOP) n —— RESIDENT

The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.

'ABORT' (TABORT) —— RESIDENT

Executes after an error from QUIT. This word normally executes ABORT, but may be altered (with care) to a user's alternative procedure.

(DO) — RESIDENT

The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO.

(DOES>) — RESIDENT

The run time procedure compiled by DOES>.

(FIND) addr1 addr2 — pfa b tf (ok) RESIDENT
addr1 addr2 — ff (bad)

Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field, and boolean true for a good match. If no match is found, only a boolean false is left.

(LINE) n scr# — addr cnt RESIDENT

Convert the line number n and the screen scr# to the disk buffer address containing the data. A count of 64 indicates the full line text length.

(LOOP) — RESIDENT

The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.

(NUMBER) d1 addr1 — d2 addr2 RESIDENT

Convert the ASCII text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.

(OF) — RESIDENT

The run time procedure compiled by OF.

n1 n2 — n3 RESIDENT

Leave the signed product of two signed numbers.

*/ n1 n2 n3 -- n4 RESIDENT

Leave the ratio n4=n1*n2/n3 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 n1 n2 n3 -- n4 n5 RESIDENT

Leave the quotient n5 and remainder n4 of the operation n1*n2/n3 . A 31 bit intermediate product is used as for */.

+ n1 n2 -- n3 RESIDENT

Leave the sum of n1 + n2.

+! n addr -- RESIDENT

Add n to the value at the address. Pronounced "PLUS STORE".

- n1 n2 -- n3 RESIDENT

Apply the sign of n2 to n1, which is left as n3.

-BUF addr1 -- addr2 f RESIDENT

Advance the disk buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by variable PREV.

+LOOP n1 —— (run) RESIDENT
addr n2 —— (compile)

Used in a colon-definition in the form:

DO ... n1 +LOOP

At run time, +LOOP selectively controls branching back to the corresponding DO based on n1, the loop index and the loop limit. The signed increment n1 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 (n1>0), or until the new index is equal to or less than the limit (n1<0). Upon exiting the loop, the parameters are discarded and execution continues ahead.

At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. n2 is used for compile time error checking.

, n —— RESIDENT

Store n into the next available dictionary memory cell, advancing the dictionary pointer. (comma)

- n1 n2 —— n3 RESIDENT

Leave the difference of n1 - n2.

--> —— RESIDENT

Continue interpretation with the next disk screen. Pronounced "NEXT SCREEN".

-DUP n1 —— n1 (if zero) RESIDENT
n1 —— n1 n1 (non-zero)

Reproduce n1 only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

-FIND —— pfa cnt f (found) RESIDENT
       —— ff (not found)

Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true are left. Otherwise, only a boolean false is left.

-TRAILING addr n1 —— addr n2 RESIDENT

Adjusts the character count n1 of a text string beginning at addr to suppress the output of trailing blanks. i.e. the characters at addr+n1 to addr+n2 are blanks.

. n —— RESIDENT

Print a number from a signed 16 bit two's complement value, converted according to the numeric 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 n scr# —— RESIDENT

Print on the terminal device, a line of text from the disk by its line (n) and screen number. Trailing blanks are supressed.

.R n1 n2 —— RESIDENT

Print the number n1 right aligned in a field whose width is n2. No following blank is printed.

.S —— SCR 43 -DUMP

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

/ n1 n2 —— n3 RESIDENT

Leave the signed quotient of n1/n2.

/MOD n1 n2 —— rem n3 RESIDENT

Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 —— n RESIDENT

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

0< n —— f RESIDENT

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

0= n —— f RESIDENT

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

0BRANCH f —— RESIDENT

The run-time procedure to conditionally branch. If f 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+ n1 —— n2 RESIDENT

Increment n1 by 1.

1- n1 —— n2 RESIDENT

Decrement n1 by 1.

2+ n1 —— n2 RESIDENT

Leave n1 incremented by 2.

2- n1 —— n2 RESIDENT

Leave n1 decremented by 2.

: —— RESIDENT
SCR 44 -TRACE

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 (P) are executed rather than being compiled.

When colon definitions are compiled under the TRACE option, : takes on an alternate definition which allows the colon definition to be traced.

; —— RESIDENT

Terminates a colon-definition and stops further compilation. Compiles the run-time ;S.

;CODE — SCR 74 -CODE

Used in the form:

```
: cccc    ...    ;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 following mnemonics.

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 cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S — RESIDENT

Stop interpretation of a screen. ;S is also the run-time word compiled at the end of a colon-definition which returns execution to the calling procedure.

< n1 n2 — f RESIDENT

Leave a true flag if n1 is less than n2; 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.

<BUILDS —— RESIDENT

Used within a colon-definition:

```
: cccc <BUILDS   ...
       DOES>     ...   ;
```

Each time cccc is executed, <BUILDS defines a new word with a high level execution procedure. Executing cccc in the form:

```
cccc   nnnn
```

uses <BUILD to create a dictionary entry for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time procedures to be written in high-level rather than in assembler code (as required by ;CODE).

<CLOAD> —— SCR 21 BOOT SCR

The run-time procedure compiled by CLOAD.

= n1 n2 —— f RESIDENT

Leave a true flag if n1=n2; otherwise leave a false flag.

=CELLS addr —— n2 RESIDENT

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.

> n1 n2 —— f RESIDENT

Leave a true flag if n1 is greater than n2; otherwise leave a false flag.

>ARG —— SCR 43 >FLOAT

Moves a floating point number from the stack into the ARG register

>F —— fl SCR 48 -FLOAT

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 fl —— SCR 45 -FLOAT

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

>R n —— RESIDENT

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

? addr —— RESIDENT

Print the value contained at the address in free format according to the current BASE. This word must preceed the address.

?COMP —— RESIDENT

Issue error message if not compiling.

?CSP —— RESIDENT

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

?ERROR f n —— RESIDENT

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.

?KEY — ch RESIDENT

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 — n RESIDENT

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 n1 n2 — RESIDENT

Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.

?STACK — RESIDENT

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

?TERMINAL — f RESIDENT

Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. On the TI 99/4A, the CLEAR key is used as the BREAK key.

@ addr — n RESIDENT

Leave the 16 bit contents of addr.

ASSM — SCR 62 -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 operators terminal, printing an appropriate message.

ABS n1 — n2 RESIDENT

Leave the absolute value of n1 as n2.

AGAIN addr n — (compiling) RESIDENT

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 this loop (unless R> DROP is executed one level below).

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

ALLOT n — RESIDENT

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

ALTIN — addr RESIDENT

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 for the alternate input device is located.

ALTOUT — addr RESIDENT

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 for the alternate output device is located.

AND n1 n2 — n3 RESIDENT

Leave the bitwise logical AND of n1 and n2 as n3.

APPND — SCR 69 -FILE

Assigns the APPEND attribute to the file whose PAB is pointed to by PAB-ADDR.

ARG — addr SCR 45 -FLOAT

A constant which contains the address of the ARG register.

ATN f11 — f12 SCR 50 -FLOAT

Performs the arctangent function leaving a floating point result on the stack.

B/BUF — n RESIDENT

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

B/BUF$ — addr RESIDENT

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

B/SCR — n RESIDENT

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/SCRS — addr RESIDENT

A user variable which contains the number of blocks per SCREEN.

BACK addr — RESIDENT

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

BASE — addr RESIDENT

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

BASE->R — RESIDENT

Place the current base on the return stack. See R->BASE.

BEEP — SCR 60 -GRAPH

Produces the sound associated with correct input or prompting.

BEGIN — addr n (compiling) RESIDENT

Occurs in a colon-definition in the form:

```
BEGIN  ...  UNTIL
BEGIN  ...  AGAIN
BEGIN  ...  WHILE  ...  REPEAT
```

At run-time, 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 and n for compiler error checking.

BL — ch RESIDENT

A constant that leaves the ascii value for "blank".

BLANKS addr cnt — RESIDENT

Fill an area of memory beginning at addr with cnt blanks.

BLK — addr RESIDENT

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

BLOAD scr# — f RESIDENT

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 n --- addr RESIDENT

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 rewritten to disk before block n is read into the buffer. See also BUFFER, R/W, UPDATE, and FLUSH.

BOOT --- RESIDENT

Examines the SCREEN designated as the booting SCREEN (SCR 3). If it contains only displayable characters (32-127) it performs a LOAD on that SCREEN.

BRANCH --- RESIDENT

The run-time 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, and REPEAT.

BSAVE addr scr# --- scr# SCR 83 -BSAVE

Places in a binary image (starting at scr# and going as far as necessary) all dictionary contents between addr and HERE. The next available SCREEN number is returned on the stack. See BLOAD.

BUFFER n --- addr RESIDENT

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! b addr --- RESIDENT

Store 8 bits at addr. Bytes always occupy the low order bits when on the stack.

C, b --- RESIDENT

Store 8 bits of b 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 --- n RESIDENT

Returns on the stack the number of characters per line.

C/L$ ___ addr RESIDENT

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

C@ addr --- b RESIDENT

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

CASE n --- RESIDENT

Initiates the construct:

CASE...OF...ENDOF...ENDCASE

CFA pfa --- cfa RESIDENT

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

CHAR n1 n2 n3 n4 ch --- SCR 57 -GRAPH

Defines character # ch to have the pattern specified by the 4 words on the stack. The definition for character #0 by default resides at HEX 800. Each character definition is 8 bytes long.

CHAR-CNT n --- SCR 69 -FILE

Used in file I/O to store the character count of a record to be transmitted.

CHAR-CNT@ —— n SCR 69 -FILE

Used in file I/O to retrieve the character count of a record that has been read.

CHARPAT ch —— n1 n2 n3 n4 SCR 57 -GRAPH

Places the 4 word pattern of a specified character (ch) on the stack. By default, the definition for character #0 resides at HEX 800.

CHK-STAT —— SCR 68 -FILE

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

CLEAR scr# —— RESIDENT

Fills the designated screen with blanks.

CLINE addr cnt n —— SCR 66 -64SUPPORT

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

CLIST scr# —— SCR 66 -64SUPPORT

Lists the specified SCREEN in Tiny CHARactrs to the monitor. CLIST executes a multiple call to CLINE. See CLINE and TCHAR.

CLOAD scr# —— SCR 21 BOOT SCR

Used in the form:

scr CLOAD nnnn

CLOAD will load 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 screen if the specified word has already been compiled.

CLR-STAT --- SCR 68 -FILE

Zeroes the status field of the PAB 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 is pointed to by PAB-ADDR.

CMOVE addr1 addr2 cnt --- RESIDENT

Move the specified quantity of bytes beginning at addr1 to addr2. The contents of addr1 is moved first proceeding toward high memory.

CODE --- SCR 74 -CODE

A defining word initializing the definition of a code (assembly) word.

COINC spr# spr# tol --- f SCR 61 -GRAPH

Detects a coincidence between two given SPRITES within a specified tolerance limit. A true flag indicates a coincidence.

COINCALL --- f SCR 61 -GRAPH

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

COINCXY dc dr spr# tol --- f SCR 61 -GRAPH

Detects a coincidence between a specified SPRITE and a given point on the screen within tolerance limits. A true flag indicates a coincidence.

COLD --- RESIDENT

The COLD start procedure to adjust the dictionary pointer to the mininum 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 n1 n2 n3 --- SCR 58 -GRAPH

Causes a specified character set (n3) to have the given foreground (n1) and background (n2) colors.

COLTAB --- vaddr SCR 57 -GRAPH

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

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 n --- RESIDENT

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 --- addr RESIDENT

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

COS f1 --- f2 SCR 50 -FLOAT

Performs the cosine operation on the floating point f1 with result on the stack.

COUNT addr1 -- addr2 n RESIDENT

Leave the byte address (addr2) and byte count (n) of a message text beginning at addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with 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 -- addr RESIDENT

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

CURPOS -- addr RESIDENT

A user variable that stores the current VDP cursor position.

CURRENT -- addr RESIDENT

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

D+ d1 d2 -- d3 RESIDENT

Leave the double number sum of two double numbers.

D— d1 n --- d2 RESIDENT

Apply the sign of n to the double number d1, leaving it as d2.

D. d --- RESIDENT

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

D.R d n — RESIDENT

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

DABS d1 — d2 RESIDENT

Leave the absolute value of a double number.

DCOLOR — addr SCR 63 -GRAPH

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.

DDOT dc dr — vaddr SCR 63 -GRAPH

The assembly code routine called by DOT. It expects a dot column and a dot row on the stack and returns a VDP address.

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 made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.

DELALL —— SCR 61 -GRAPH

Delete all SPRITES.

DELSPR spr# —— SCR 61 -GRAPH

Delete the specified SPRITE.

DIGIT ch n1 —— n2 tf (ok) RESIDENT
ch n1 —— ff (bad)

Convert the ascii character ch (using BASE n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leave only a false flag.

DISK_BUF —— addr RESIDENT

A user variable that points to the first byte in VDP RAM of the 1K disk buffer.

DISK-HEAD —— SCR 40 -COPY

Writes a disk-head on SCREEN 0 that makes the disk compatable with the TI 99/4A DISK MANAGER and with TI BASIC.

DISK_HI —— addr RESIDENT

A user variable which contains the SCREEN number immediately above the SCREEN range wherein SCREEN writes are permitted.

DISK_LO —— addr RESIDENT

A user variable which contains the first SCREEN number of the range wherein disk writes are permitted.

DISK_SIZE —— addr RESIDENT

A user variable whose value is the number of SCREENS logically assigned to a diskette.

DLITERAL d —— d (executing) RESIDENT
d —— (compiling)

If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it to the stack. If executing, the number will remain on the stack.

DLT —— SCR 71 -FILE

The FILE I/O routine which deletes the file whose PAB is pointed to by PAB-ADDR.

DMINUS d1 —— d2 RESIDENT

Convert d1 to its double number two's complement.

DMODE —— addr SCR 63 -GRAPH

A variable which 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 instructions.

DO n1 n2 --- (execute) RESIDENT
addr n --- (compile)

Occurs in a colon-definition in the form:

```
DO  ...  LOOP
DO  ...  +LOOP
```

At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. 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 n1 and n2 are determined at run-time 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.

When compiling within the colon-definition, DO compiles (DO), leaving the following address (addr) and n for later error checking.

DOES> --- RESIDENT

A word which defines the run-time 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, multi-dimensional arrays, and compiler generation.

DOT dc dr --- SCR 63 -GRAPH

Plots a dot at (dc,dr) in whatever mode is selected by DMODE and in whatever color is selected by DCOLOR.

DP --- addr RESIDENT

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 — addr RESIDENT

A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used to hold output column location of a decimal point, in user generated formating. The default value on single number input is -1.

DR0 — RESIDENT
DR1
DR2

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 supressed 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 n — RESIDENT

Adjusts OFFSET so that the drive number on the stack becomes the first drive in the system.

DROP n — RESIDENT

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 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 DSK1 by attempting to read each 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 addr n — SCR 43 -DUMP

Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in hexadecimal. See PAUSE.

DUP n — n n RESIDENT

Duplicate the value on the stack.

DXY dc1 dr1 dc2 dr2 — n1 n2 SCR 59 -GRAPH

Places the square of the x distance (n1) and the square of the y distance (n2) between the points (dc1,dr1) and (dc2,dr2) on the stack.

ECOUNT — addr RESIDENT

A user variable which contains an error count. This is used to prevent error recursion.

ED@ — SCR 38 -EDITOR
SCR 29 -64SUPPORT

Brings you back into the EDITOR on the last SCREEN you edited. This SCREEN is pointed to by SCR.

EDIT scr# — SCR 38 -EDITOR
SCR 29 -64SUPPORT

Brings you into the EDITOR on the specified SCREEN.

ELSE addr1 n1 --- addr2 n2 RESIDENT
(compiling)

Occurs within a colon-definition in the form:

IF ... ELSE ... ENDIF

At run-time, 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.

At compile-time, ELSE emplaces BRANCH reserving a branch offset and leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing it at addr1.

EMIT ch --- RESIDENT

Transmit ascii character ch to the selected output device. OUT is incremented for each character output.

EMIT8 ch --- RESIDENT

Transmit an 8-bit character 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 addr1 ch --- addr1 n1 n2 n3 RESIDENT

The text scanning primitive used by WORD. From the text address addr1 and an ascii delimiting character ch, is determined the byte offset to the first non-delimiter character n1, the offset to the delimiter after the text n2, and the offset to the first character not included. This procedure will not process past an ascii null , treating it as an unconditional delimiter.

END f --- RESIDENT

This is an 'alias' or duplicate definition for UNTIL.

ENDCASE --- RESIDENT

Terminates the CASE construct.

ENDIF addr n --- (compile) RESIDENT

Occurs in a colon-definition in the form:

```
IF  ...  ENDIF
IF  ...  ELSE  ...  ENDIF
```

At run-time, 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.

ERASE addr n --- RESIDENT

Clear a region of memory to zero from addr over n bytes.

ERROR n1 --- n2 n3 RESIDENT

Execute error notification and restart of system. WARNING is first examined. If 1, the text of line n1, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING=0, n1 is just printed as a message number (non-disk installation). If WARNING is -1, the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-FORTH saves the contents of IN (n2) and BLK (n3) to assist in determining the location of the error. Final action is execution of QUIT.

EXECUTE cfa —— RESIDENT

Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXP f1 —— f2 SCR 50 -FLOAT

Raises e to the power specified by the floating point number on the stack and leaves the result on the stack.

EXPECT addr cnt —— RESIDENT

Transfer characters from the terminal to addr, until a 'ENTER' or the count of characters has been received. One or more nulls are added at the end of the text.

F! f1 addr -- SCR 45 -FLOAT

Stores a floating point number, f1, into the 4 words beginning with the specified address.

F* f11 f12 —— f13 SCR 46 -FLOAT

Multiplies the top two floating point numbers on the stack and leaves the result on the stack.

F+ f11 f12 —— f13 SCR 46 -FLOAT

Adds the top two floating point numbers on the stack and places the result on the stack.

F- f11 f12 —— f13 SCR 46 -FLOAT

Subtracts f12 from f11 and places the result on the stack.

F->S f1 —— n SCR 46 -FLOAT

Converts a floating point number on the parameter stack into a single precision number.

F. fl --- SCR 48 -FLOAT

Prints a floating point number in BASIC format to the output device.

F.R fl n --- SCR 48 -FLOAT

Prints the floating point number in BASIC format right justified in a field of width n.

F/ fl1 fl2 --- fl3 SCR 46 -FLOAT

Divides fl1 by fl2 and leaves the floating point quotient on the stack.

F0< fl --- f SCR 49 -FLOAT

Compares the floating point number 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= fl --- f SCR 49 -FLOAT

Compares the floating point number 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< fl1 fl2 --- f SCR 49 -FLOAT

Leaves a true flag if fl1 < fl2. Else leaves a false flag.

F= fl1 fl2 --- f SCR 49 -FLOAT

Leaves a true flag if fl1 = fl2. Else leaves a false flag.

F> fl1 fl2 --- f SCR 49 -FLOAT

Leaves a true flag if fl1 > fl2. Else leaves a false flag.

F@ addr --- fl SCR 45 -FLOAT

Retrieves the floating point contents of the given address (4 words) and places it on the stack.

FAC —— addr SCR 45 -FLOAT

A constant which contains the address of the FAC register.

FAC->S —— n SCR 46 -FLOAT

Converts a floating point number in FAC to a single precision number and places it on the parameter stack.

FAC> —— fl SCR 45 -FLOAT

Brings a floating point number 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 fl —— SCR 45 -FLOAT

Drops the top floating point number from the stack.

FDUP fl —— fl fl SCR 45 -FLOAT

Duplicates the top floating point number on the stack.

FENCE —— addr RESIDENT

A user variable containing an address below which FORGETting is trapped. To FORGET below this point the user must alter the contents of FENCE.

FF. f1 n1 n2 -- SCR 48 -FLOAT

Prints the floating point number with n2 digits following the decimal point and a maximum of n1 digits.

FF.R f1 n1 n2 n3 -- SCR 48 -FLOAT

Prints the floating point number, with n2 digits following the decimal point, right justified in a field of width n3 with a maximum of n1 digits.

F-D" -- SCR 70 -FILE

Expects a file descriptor ending with a " to follow. This instruction places the file descriptor in the PAB pointed to by PAB-ADDR.

FILE addr1 addr2 vaddr -- SCR 68 -FILE

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:

addr1 addr2 vaddr FILE cccc

When cccc executes, PAB-ADDR, PAB-BUF, and PAB-VBUF are set to addr1, addr2, and vaddr, respectively.

FILL addr cnt b -- RESIDENT

Fill memory beginning at addr with the specified number (cnt) of bytes b.

FIRST -- addr RESIDENT

A constant that leaves the address of the first (lowest) block buffer.

FIRSTS -- addr RESIDENT

A user variable which contains the first byte of the disk buffer area.

FLD — addr RESIDENT

A user variable for control of number output field width. Presently unused in fig-FORTH and TI FORTH.

FLERR — n SCR 49 -FLOAT

Returns on the stack the contents of the floating point status register.

FLUSH — RESIDENT

Rewrites to the disk all disk buffers that have been 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 definition named cccc from the dictionary with all entries physically following it.

FORMAT-DISK dsk — SCR 33 -SYNONYMS

Initializes the disk in DR0, DR1, or DR2 for use with the Forth system. CAUTION: all data on the disk (if any) will be destroyed. Also, disks initialized by the DISK MANAGER may be used without any changes. DSK number 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 part of FORTH. 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 DSK2 onto the disk in DSK1.

FORTH-LINK —— addr RESIDENT

A user variable used for vocabulary linkage.

FOVER f11 f12 —— f11 f12 f11 SCR 45 -FLOAT

Copies the second floating point number on the stack to the top of the stack.

FRND —— f1 SCR 46 -FLOAT

Generates a 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 f11 f12 —— f12 f11 SCR 45 -FLOAT

Swaps the top two floating point numbers on the stack.

FXD —— SCR 68 -FILE

Assigns the attribute FIXED to the file whose PAB is pointed to by PAB-ADDR.

GCHAR c r —— ch SCR 58 -GRAPH

Returns on the stack the ascii code of the character currently at (c, r). NOTE: Rows and columns are numbered from 0.

GET-FLAG —— b SCR 68 -FILE

Retrieves the flag byte from the current PAB and places it on the stack.

GOTOXY c r -- RESIDENT

Places the cursor at the designated column and row position. NOTE: Rows and columns are numbered from 0.

GPLLNK addr -- SCR 33 -SYNONYMS

Links a FORTH program to the Graphics Programming Language routine located at the given address.

GRAPHICS -- SCR 52 -GRAPH1

Converts from present screen mode into standard GRAPHICS mode configuration.

GRAPHICS2 -- SCR 54 -GRAPH2

Converts from present screen mode into standard GRAPHICS2 mode configuration.

HCHAR c r cnt ch -- SCR 57 -GRAPH

Prints a horizontal stream of a specified character beginning at (c,r) and having a length cnt. NOTE: Rows and columns are numbered from 0.

HERE -- addr RESIDENT

Leave the address of the next available dictionary location.

HEX -- RESIDENT

Set the numeric conversion base to sixteen (hexadecimal).

HLD -- addr RESIDENT

A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD n -- RESIDENT

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 — n RESIDENT

Used within a DO-LOOP to copy the loop index to the stack. Other use is implementation dependent. See R .

ID. nfa — RESIDENT

Print a definition's name from its name field address.

IF f — (run-time) RESIDENT
— addr n (compile)

Occurs in a colon-definition in the form:

IF (tp) ... ENDIF
IF (tp) ... ELSE (fp) ... ENDIF

At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f 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.

At compile time, IF compiles OBRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.

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 preceeding it with [COMPILE].

IN —— addr RESIDENT

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 n1 n2 —— SCR 73 -PRINT

Prints to the terminal a list of the line #0 comments from SCREEN n1 thru SCREEN n2. See PAUSE.

INPT —— SCR 69 -FILE

Assigns the atribuce INPUT to the file whose PAB is pointed to by PAB-ADDR.

INT f1 —— f2 SCR 50 -FLOAT

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 —— addr RESIDENT

A user variable which is a pointer to the Interrupt Service linkage.

INTRPL —— SCR 69 -FILE

pointed to by PAB-ADDR.

ISR —— addr RESIDENT

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 HEX 83C4 must be modified to point to this address. Note, the interrupt service linkage code address is also available in INTLNK.

J —— n RESIDENT

Copies the loop index of the second innermost loop to the stack.

JOYST n1 —— ch n2 n3 SCR 60 -GRAPH

Allows you to accept input from JOYSTICK #1 or #2 (n1) or from the left and right sides of the keyboard, respectively. Values returned are the ascii value of the key pressed (ch), the x status (n2) and the y status (n3).

KEY —— ch RESIDENT

Leave the ascii value of the next terminal key struck.

KEY8 —— ch RESIDENT

Leave the 8-bit value of the next terminal key struck.

L/SCR —— n RESIDENT

Returns on the stack the number of lines per SCREEN.

LATEST —— nfa RESIDENT

Leave the name field address of the topmost word in the CURRENT vocabulary.

LD n —— SCR 71 -FILE

The file I/O process to load a program file from a disk into VDP RAM. The parameter n specifies the maximum number of bytes to be loaded.

LDCR n1 n2 addr --- SCR 88

Performs a 9900 LDCR instruction. The CRU base (addr) will be shifted left one bit by the LDCR instruction. n1 is the value to be transferred to the CRU and n2 is the field width of n1 (in 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 execution proceeds normally until LOOP or +LOOP is encountered.

LFA pfa -- lfa RESIDENT

Convert the parameter field address of a dictionary definition to its link field address.

LIMIT -- addr RESIDENT

A constant which leaves the address just above the highest memory available for a disk buffer.

LIMITS -- addr RESIDENT

A user variable which contains the address just above the highest memory available for a disk buffer.

LINE dc1 dr1 dc2 dr2 -- SCR 64 -GRAPH

The high resolution graphics routine which plots a line from (dc1,dr1) to (dc2,dr2). DCOLOR and DMODE must be set before this instruction is used.

LIST scr# -- RESIDENT

Lists the specified SCREEN to the output device. See PAUSE.

LIT -- RESIDENT

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 n --- (compiling) RESIDENT

If compiling, then compile the stack value n as a 16 bit literal. This will execute during a colon-definition. The intended use is:

: xxx [calculate] LITERAL ;

Compilation is suspended for the compile-time calculation of a value. Compilation is resumed and LITERAL compiles this value.

LOAD n --- RESIDENT

Begin interpretation of SCREEN n. Loading will terminate at the end of the SCREEN or at ;S. See ;S and -->.

LOG f11 --- f12 SCR 50 -FLOAT

The floating point operation which returns the LOG of the floating point number on the stack.

LOOP addr n --- (compiling) RESIDENT

Occurs in a colon-definition in the form:

DO ... LOOP

At run-time, 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* n1 n2 --- d RESIDENT

A mixed magnitude math operation which leaves the double number signed product of two signed numbers.

M/ d n1 -- n2 n3 RESIDENT

A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor, n1. The remainder takes its sign from the dividend.

M/MOD ud1 u2 -- u3 ud4 RESIDENT

An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and a single divisor u2.

MAGNIFY n1 -- SCR 60 -GRAPH

Alters the SPRITE magnification factor to be n1. The value of n1 must be 0 , 1 , 2 , or 3.

MAX n1 n2 -- n3 RESIDENT

Leave the greater of the two numbers.

MCHAR n c r -- SCR 62 -GRAPH

Places a square of color n at (c,r). Used in MULTICOLOR mode.

MENU -- SCR 20 BOOT SCR

Displays the available Load Options.

MESSAGE n -- RESIDENT

Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (disk un-available).

MIN n1 n2 -- n3 RESIDENT

Leave the smaller of the two numbers.

MINIT — SCR 62 -GRAPH

Initializes the SCREEN for use with MCHAR.

MINUS n1 — n2 RESIDENT

Leave the two's complement of a number.

MOD n1 n2 — mod RESIDENT

Leave the remainder of n1/n2, with the same sign as n1.

MON — SCR 33 -SYNONYMS

Exit to the TI 99/4A color bar screen.

MOTION n1 n2 spr# — SCR 59 -GRAPH

Assigns a horizontal (n1) and vertical (n2) velocity to a specified SPRITE.

MOVE addr1 addr2 n — RESIDENT

Move the contents of n memory cells (16 bit contents) beginning at addr1 into n cells beginning at addr2. The contents of addr1 is moved first.

MULTI — SCR 53 -GRAPH

Converts from present screen mode into standard MULTICOLOR mode configuration.

MYSELF — RESIDENT

Used in a colon definition. Places the CFA of a routine into itself. This permits recursion.

NFA pfa — nfa RESIDENT

Convert the parameter field address of a definition to its name field address.

NOP — RESIDENT

A do nothing instruction. NOP is useful for patching as in assembly code.

NUMBER addr --- d RESIDENT

Convert a character string left at addr with a preceeding count, to a signed double number, 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 n --- RESIDENT

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.

OFFSET --- addr RESIDENT

A user variable which may contain a bIock 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, DRO, MESSAGE.

OPN — SCR 71 -FILE

Opens the file whose PAB is pointed to by PAB-ADDR.

OR n1 n2 --- n3 RESIDENT

Leave the bit-wise logical OR of two 16 bit values.

OUT --- addr RESIDENT

A user defined variable that contains a value incremented by EMIT and EMIT8. The user may alter and examine OUT to control display formating

OUTPT — SCR 69 -FILE

Assigns the attribute OUTPUT to the file whose PAB is pointed to by PAB-ADDR.

OVER n1 n2 — n1 n2 n1 RESIDENT

Copy the second stack value, placing it as the new top.

PAB-ADDR — addr SCR 68 -FILE

A variable containing the VDP address of the first byte of the current PAB.

PAB-BUF — addr SCR 68 -FILE

A variable which holds the address of the area in CPU RAM used as the source or destination of the data to be transferred. This is a file I/O word.

PAB-VBUF — addr SCR 68 -FILE

A variable pointing to a VDP RAM buffer which serves as a temporary buffer when transferring data.

PABS — addr RESIDENT

A user variable which points to a region in VDP RAM which has been set aside for creating PABs.

PAD — addr RESIDENT

Leave the address of the text output buffer, which is a fixed offset above HERE.

PAUSE — RESIDENT

The words LIST, INDEX, DUMP and VLIST all call the word PAUSE. 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 —— vaddr SCR 57 -GRAPH

A constant which contains the VDP address of the Pattern Descriptor Table. Default value is >800.

PFA nfa —— pfa RESIDENT

Convert the name field address of a compiled definition to its parameter field address.

PI —— fl SCR 50 -FLOAT

A floating point approximation of PI to 14 decimal places. (3.141592653590)

PREV —— addr RESIDENT

A 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 b —— SCR 68 -FILE

Writes the flag byte into the appropriate PAB referenced by PAB-ADDR.

QUERY —— RESIDENT

Input 80 characters of text (or until a "enter") from the operator's terminal. Text is positioned at the address contained in TIB with IN set to zero.

QUIT —— RESIDENT

Clear the return stack, stop compilation, and return control to the operator's terminal. No message is given.

R —— n RESIDENT

Copy the top of the return stack to the parameter stack.

R# —— addr RESIDENT

A user variable which may contain the location of an editing cursor, or other file related function.

R->BASE RESIDENT

Restore the current base from the return stack. See BASE->R.

R/W addr n1 f —— RESIDENT

The fig-FORTH standard disk read-write linkage. addr specifies the source or destination block buffer, n1 is the sequential number of the referenced block; and f is a flag for f=0 write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs error checking.

R> —— n RESIDENT

Remove the top value from the return stack and leave it on the parameter stack. See >R and R.

R0 —— addr RESIDENT

A user variable containing the initial location of the return stack. Pronounced "R zero". See RP!

RANDOMIZE —— SCR 33 -SYNONYMS

Creates an unpredictable seed for the random number generator.

RD —— cnt SCR 71 -FILE

The file I/O instruction that reads from the current PAB. This instruction uses PAB-BUF and PAB-VBUF.

RDISK addr n1 n2 —— n3 RESIDENT

The primative routine that performs disk reads. addr is the address where the block is to be written in CPU RAM. n1 is the block number, n2 is the number of bytes per block, and n3 is the returned error code.

REC-LEN b --- SCR 69 -FILE

Stores the length of the record for the upcoming write into the appropriate byte in the current PAB.

REC-NO n --- SCR 69 -FILE

Writes a record number n into the appropriate location in the current PAB.

REPEAT addr n --- (compiling) RESIDENT

Used within a colon-definition in the form:

BEGIN ... WHILE ... REPEAT

At run-time, 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 n1 --- n2 SCR 33 -SYNONYMS

Generates a positive random integer (n2) greater than or equal to 0 and less than n1.

RNDW --- n SCR 33 -SYNONYMS

Generates a random word. The value of the word may be positive or negative depending on whether the sign bit is set.

ROT n1 n2 n3 --- n2 n3 n1 RESIDENT

Rotate the top three values on the stack, bringing the third to the top.

RP! — RESIDENT

A procedure to initialize the return stack pointer from user variable R0.

RSTR n --- SCR 71 -FILE

Restores the file whose PAB is pointed to by the current PAB to the specified record number, n.

S->D n — d RESIDENT

Sign extend a single number to form a double number.

S->F n — fl SCR 46 -FLOAT

Converts a single precision number on the stack to a floating point number.

S->FAC n — SCR 46 -FLOAT

Takes a single precision number from the stack, converts it to floating point, and leaves it in FAC.

S0 — addr RESIDENT

Pronounced "S zero". See SP!.

SATR — vaddr SCR 57 -GRAPH

A constant whose value is the VDP address of the SPRITE Attribute List. Default value is >300.

SBO addr — SCR 88 -CRU

This word expects to find on the stack the CRU address (addr) of the bit to be set to 1. Note that the SBO instruction will itself shift the address before using R12.

SBZ addr — SCR 88 -CRU

This word expects to find on the stack the CRU address (addr) of the bit to be set to 0. Note that the SBZ instruction will itself shift the address before using R12.

SCOPY scr#1 scr#2 --- SCR 39 -COPY

Copies the source SCREEN (scr#1) to the destination SCREEN (scr#2). Does not destroy the source SCREEN.

SCR --- addr RESIDENT

A user variable containing the screen number most recently referenced by LIST or EDIT.

SCREEN n --- SCR 58 -GRAPH

Changes the screen color to the color specified (n).

SCRN_END --- addr RESIDENT

A user variable containing the address of the byte immediately following the last byte of the screen image table to be used as the logical screen.

SCRN_START --- addr RESIDENT

A user variable containing the address of the first byte of the screen image table to be used as the logical screen.

SCRN_WIDTH --- addr RESIDENT

A user variable which contains the number of characters which will fit accross the screen. (32 or 40) Used by the screen scroller.

SCRTCH n --- SCR 71 -FILE

Removes the specified record from the RELATIVE file whose PAB is pointed to by PAB-ADDR.

SEED n --- SCR 33 -SYNONYMS

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 pointed to by PAB-ADDR and places the contents of PAB-VBUF into the appropriate word of the PAB. This initializes the PAB.

SETFL f11 f12 --- SCR 45 -FLOAT

Performs a >FAC on f12 and a >ARG on f11.

SIGN n d --- d RESIDENT

Stores an ascii "-" sign at the current location in a converted numeric output string in the text output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>.

SIN f11 --- f12 SCR 50 -FLOAT

Finds the SIN of the floating point number on the stack and leaves the result on the stack.

SLA n1 cnt --- n2 RESIDENT

Arithmetically shifts the number on the stack cnt bits to the left, leaving the result on the stack.

SLIT --- addr SCR 20 BOOT SCR

SLIT is similar to LIT but acts on strings instead of numbers. SLIT places the address of the string following it on the stack. It modifies the top of the return stack to point to just after the string.

SMASH addr cnt n --- addr vaddr cnt SCR 65 -64SUPPORT

The assembly code routine which formats a line of tiny characters. It expects the address of the line in memory, the number of characters per line, and the line number to which it is to be written. It returns on the stack the line buffer address, a VDP adddress, and a character count. See CLIST and CLINE.

SMOVE scr#1 scr#2 cnt --- SCR 39 -COPY

Copies cnt SCREENS beginning with the source SCREEN (scr#1) to the destination SCREEN (scr#2). Overlapping SCREEN ranges may be specified without detrimental effects.

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 compiling is completed without error.

SMTN --- vaddr SCR 57 -GRAPH

A constant whose value is the VDP address of the SPRITE MOTION TABLE. Default value is >780.

SP! --- RESIDENT

A procedure to initialize the stack pointer from S0.

SP@ --- addr RESIDENT

A procedure to return the address of the stack position to 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 an ascii blank to the output device.

SPACES n --- RESIDENT

Transmit n ascii blanks to the output device.

SPCHAR n1 n2 n3 n4 ch --- SCR 58 -GRAPH

Defines a character (ch) in the SPRITE Descriptor Table to have the pattern composed of the 4 words on the stack.

SPDTAB —— vaddr SCR 57 -GRAPH

A constant whose value is the VDP address of the SPRITE Descriptor Table. Default value is >800. Notice that this coincides with the Pattern Descriptor Table.

SPLIT —— SCR 55 -SPLIT

Converts from present screen mode into standard SPLIT mode configuration.

SPLIT2 —— SCR 55 -SPLIT

Converts from present screen mode into standard SPLIT2 mode configuration.

SPRCOL n spr# —— SCR 58 -GRAPH

Changes the given SPRITE to the color (n) specified.

SPRDIST spr#1 spr#2 —— n SCR 60 -GRAPH

Returns on the stack the square of the distance (n) between two specified SPRITES. Distance is measured in pixels and the maximum distance that can be detected accurately is 181 pixels.

SPRDISTXY dc dr spr# —— n SCR 60 -GRAPH

Places on the stack the square of the distance between the point (dc,dr) and a given SPRITE. Distance is measured in pixels and the maximum distance that can be detected accurately is 181 pixels.

SPRGET spr# —— dc dr SCR 59 -GRAPH

Returns the dot column and dot row position of a SPRITE.

SPRITE dc dr n ch spr# —— SCR 59 -GRAPH

Defines SPRITE number spr# to have the specified location (dc,dr), color (n), and character pattern (ch). The size of the SPRITE will depend on the magnification factor.

SPRPAT ch spr# --- SCR 59 -GRAPH

Changes the character pattern of a given SPRITE to ch.

SPRPUT dc dr spr# --- SCR 59 -GRAPH

Places a given SPRITE at location (dc,dr).

SQNTL --- SCR 69 -FILE

Assigns the attribute SEQUENTIAL to the file whose PAB is pointed to by PAB-ADDR.

SQR fl1 --- fl2 SCR 50 -FLOAT

Finds the square root of a floating point number and leaves the result on the stack.

SRA n1 cnt --- n2 RESIDENT

Arithmetically shifts n1 cnt bits to the right and leaves the result on the stack. cnt will be modulo 16, except when cnt=0, when 16 bits will be shifted. To create a word which permits shifts when cnt could be zero, use the following definition: : SRA0 -DUP IF SLA ENDIF ;

SRC n1 cnt --- n2 RESIDENT

Performs a circular right shift of cnt bits on n1 leaving the result on the stack.

SRL n1 cnt --- n2 RESIDENT

Performs a logical right shift of cnt bits and leaves the result on the stack. cnt will be modulo 16, except when cnt=0, when 16 bits will be shifted. To create a word which permits shifts when cnt could be zero, use the following definition: : SRL0 -DUP IF SLA ENDIF ;

SSDT vaddr --- SCR 58 -GRAPH

Places the SPRITE descriptor table at the specified VDP address and initializes all SPRITE tables. The address given must be an even 2K boundary. This instruction MUST be executed before SPRITES can be used.

STAT —— b SCR 71 -FILE

Reads the status of the current PAB and returns the status byte to the stack. See the EDITOR/ASSEMBLER manual for the meaning of each bit of the status byte.

STATE —— addr RESIDENT

A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

STCR n1 addr —— n2 SCR 88

Performs the 9900 STCR instruction. n1 is the field width, addr is the CRU base address, and n2 is the returned value. CRU base will be shifted left by the STCR instruction.

STR —— SCR 47 -FLOAT

Converts the number in FAC to a string which is placed in PAD. The string is in BASIC format.

STR. n1 n2 n3 —— SCR 47 -FLOAT

See the STR function in the EDITOR/ASSEMBLER manual. n1 corresponds to the byte at FAC+13, n2 corresponds to the byte at FAC+12, and n3 corresponds to the byte at FAC+11.

SV cnt —— SCR 71 -FILE

Performs the file I/O save operation. cnt equals the number of bytes to be saved.

SWAP n1 n2 —— n2 n1 RESIDENT

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 n1 --- n2 RESIDENT

Reverses the order of the two bytes in n1 and leaves the new number as n2.

SYS$ --- addr RESIDENT

A user variable that contains the address of the system support entry point.

SYSTEM n --- RESIDENT

Calls the system synonyms. You must specify an offset n into a jump table for the routine you wish to call. n must be one of the predefined even numbers.

TAN f11 --- f12 SCR 50 -FLOAT

Finds the TANgent of the floating point number on the stack and leaves the result.

TASK --- RESIDENT

A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.

TB addr --- f SCR 88 -CRU

The address addr is tested by this instruction. The value (1 or 0) is returned to the stack. Note that the TB instruction will itself shift the address before using R12.

TCHAR --- addr SCR 67 -64SUPPORT

The array that holds the Tiny CHARacter definitions. See CLIST.

TEXT --- SCR 51 -TEXT

Converts from present screen mode into standard TEXT mode configuration.

THEN — RESIDENT

An alias for ENDIF.

TIB — addr RESIDENT

A user variable containing the address of the terminal input buffer.

TOGGLE addr b — RESIDENT

Complement the contents of the byte at addr by the bit pattern b.

TRACE — SCR 44 -TRACE

Forces the following colon definitions to be compiled in such a way that they can be traced. See TRON, TROFF, and UNTRACE.

TRAVERSE addr1 n — addr2 RESIDENT

Move accross the name field of a fig-FORTH variable length name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward high memory; if n=-1, the motion is toward low memory. The addr2 resulting is the address of the other end of the name.

TRIAD scr# — SCR 72 -PRINT

Display on the RS232 the three SCREENS which include that number scr#, beginning with a 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.

TRIADS scr# scr# — SCR 73 -PRINT

May be thought of as a multiple TRIAD. You must specify a SCREEN range. TRIADS will perform as many TRIAD's as necessary to cover that range.

TROFF — SCR 44 -TRACE

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.

TRON — SCR 44 -TRACE

See TROFF.

TYPE addr cnt — RESIDENT

Transmit count characters from addr to the selected output device.

U — n RESIDENT

Places the contents of register U on the stack. Register U contains the base address of the user variable area.

U* u1 u2 — ud RESIDENT

Leave the unsigned double number product of two unsigned numbers.

U. u — RESIDENT

Prints an unsigned number to the output device.

U.R u n — RESIDENT

Prints an unsigned number right justified in a field of width n.

U/ ud u1 — u2 u3 RESIDENT

Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

UO —— addr RESIDENT

A user variable that points to the junction between the user variable area and the return stack.

U< u1 u2 —— f RESIDENT

Leaves a true flag if u1 is less than u2, else leaves a false flag.

UCONS$ —— addr RESIDENT

A user variable which contains the base address of the user variable default area which is used to initialize the user variables at COLD.

UD. ud —— RESIDENT

Prints an unsigned double number to the output device.

UD.R ud n —— RESIDENT

Prints an unsigned double number right justified in a field of length n.

UNDRAW —— SCR 63 -GRAPH

Sets DMODE to 1. This means that dots are plotted in the 'off' mode.

UNFORGETABLE addr —— f RESIDENT

Decides whether or not a word can be forgotten. A true flag is returned if the address is not located between FENCE and HERE.

UNSWCH —— SCR 72 -PRINT

Causes the computer to send output to the screen instead of an RS232 device. See SWCH.

UNTIL f --- (run-time) RESIDENT
addr n --- (compile)

Occurs within a colon-definition in the form:

BEGIN ... UNTIL

At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead.

At compile-time, UNTIL compiles (OBRANCH) and an offset from HERE to addr. n is used for error tests.

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.

UPDT --- SCR 69 -FILE

Assigns the attribute UPDATE to the file whose FAB is pointed to by FAB-ADDR.

USE --- addr RESIDENT

A variable containing the address of the block buffer to use next, as the least recently written.

USER n --- RESIDENT

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 pointer register UP 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.

VAL --- SCR 47 -FLOAT

Causes the string at PAD to be converted into a floating point number and put into FAC.

VAND b vaddr --- SCR 33 -SYNONYMS

Performs a logical AND on the contents of the specified VDP location and the given byte. The result is stored back into the VDP address.

VARIABLE n --- RESIDENT

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 c r cnt ch --- SCR 57 -GRAPH

Prints a vertical stream of length cnt of the specified character. The first character of the stream is located at (c,r). NOTE: Rows and columns are numbered from 0.

VFILL vaddr cnt b --- SCR 33 -SYNONYMS

Fills cnt locations beginning at the given VDP address with the specified byte.

VLIST --- SCR 43 -DUMP

Prints the names of all words defined in the CONTEXT vocabulary. See PAUSE.

VMBR vaddr addr cnt --- SCR 33 -SYNONYMS

Reads cnt bytes beginning at the given VDP address and places them at addr.

VMBW addr vaddr cnt --- SCR 33 -SYNONYMS

Writes cnt bytes from addr into VDP beginning at the given VDP address.

VOC-LINK --- addr RESIDENT

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 thru 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 b vaddr --- SCR 33 -SYNONYMS

Performs a logical OR on the contents of the specified VDP address and the given byte. The result 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 vaddr --- b SCR 33 -SYNONYMS

Reads a single byte from the given VDP address and places it on the stack.

VSBW b vaddr --- SCR 33 -SYNONYMS

Writes a single byte into the given VDP address.

VWTR b n --- SCR 33 -SYNONYMS

Writes the given byte into the specified VDP write-only register (n).

VXOR b vaddr --- SCR 33 -SYNONYMS

Performs a logical XOR on the contents of the specified VDP address and the given byte. The result is stored back into the VDP address.

WARNING --- addr RESIDENT

A user variable containing a value controlling messages. If =1 disk is present, and screen 4 of drive 0 is the base location for messages. If =0, no disk is present and messages will be presented by number. If =-1, execute (ABORT) for a user specified procedure. See MESSAGE, ERROR.

WDISK addr n1 n2 --- n3 RESIDENT

The primative routine which performs a disk write. addr is the CPU RAM location of the block to be written. n1 is the block number, n2 is the number of bytes per block, and n3 is the returned error code.

WHERE n1 n2 --- SCR 38 -EDITOR
-c-SUPPORT

When an error occurs on a LOAD instruction, typing WHERE will bring you into the EDITOR and place the cursor at the exact location of the error.

WHILE f --- (run-time) RESIDENT
addr1 n1 --- addr1 n1 addr2 n2 (compile)

Occurs in a colon-definition in the form:

BEGIN ... WHILE(tp) ... REPEAT

At run-time, WHILE selects conditional execution based on boolean flag f. If f is true (non-zero), WHILE continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution skips to just after REPEAT, exiting the structure.

At compile time, WHILE emplaces (OBRANCH) and leaves addr2 of the reserved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr RESIDENT

A user variable containing the maximum number of letters saved in the compilation of a definition's name. It must be 1 thru 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

Used in the form: WLITERAL cccc

A compiling word which compiles SLIT and the string which follows WLITERAL into the dictionary.

WORD ch --- RESIDENT

Read the text characters from the input stream being interpreted, until a delimiter ch is found, storing the packed character string beginning at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurances of ch 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 cnt --- SCR 71 -FILE

Performs the file I/O write operation. You must specify the number of bytes to be written.

XMLLNK addr -- SCR 33 -SYNONYMS

Links a FORTH program to a routine in ROM or to a routine located in the memory expansion. A ROM address or XML vector must be specified as in the Editor/Assembler.

XOR n1 n2 -- n3 RESIDENT

Leave the bitwise logical EXCLUSIVE OR of two values.

[-- 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,].

[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 [.

f1 f2 -- f3 SCR 30 -FLOAT

Returns on the stack f1 raised to the f2 power. The operands must be floating point numbers

message —— SCR 84

A replacement for MESSAGE which contains the error messages in memory instead of on the disk. When screen #84 is loaded, the error messages are compiled into the space formerly occupied by the fifth disk buffer. 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 complete description of each of these variables is provided in Appendix D. The table is located on the following page.

The user may use even numbers >68 through >7E to create his own user variables. See the definition of USER in Appendix D.

TI FORTH USER VARIABLES

Name	Offset	Initial Value	Description
UCONSS	>6		Base of User Var initial value table
S0	>8		Base of Stack
R0	>A		Base of Return Stack
U0	>C		Base of User Variables
TIB	>E		Terminal Input Buffer addr
WIDTH	>10	31	Name length in dictionary
DP	>12		Dictionary Pointer
SYSS	>14		Addr of System Support
CURPOS	>16		Cursor location in VDP RAM
INTLNK	>18		Pointer to Interrupt Service linkage
WARNING	>1A	1	Message Control
C/L$	>1C	64	Characters per Line
FIRST$	>1E		Beginning of Disk Buffers
LIMIT$	>20		End of Disk Buffers
B/BUFS	>22	1024	Bytes per Buffer
B/SCRS	>24	1	Blocks per Screen
DISK_LO	>26	1	Low end Disk Fence
DISK_HI	>28	90	High end Disk Fence
DISK_SIZE	>2A	90	Logical Disk Size in Screens
DISK_BUF	>2C	>1000	VDP location of 1K Buffer
PABS	>2E	>460	VDP location for PABs
SCRN_WIDTH	>30	40	Screen Width in Characters
SCRN_START	>32	0	Screen Image Start in VDP
SCRN_END	>34	960	Screen Image End in VDP
ISR	>36		Interrupt Service Pointer
ALTIN	>38	0	Alternate Input Pointer
ALTOUT	>3A	0	Alternate Output Pointer
FENCE	>3C		Dictionary Fence
BLK	>3E		Block being interpreted
IN	>40		Byte offset in text buffer
OUT	>42		Incremented by EMIT
SCR	>44		Last Screen referenced
OFFSET	>46		Block offset to disks
CONTEXT	>48		Pointer to Context Vocabulary
CURRENT	>4A		Pointer to Current Vocabulary
STATE	>4C		Compilation State
BASE	>4E		Number Base for Conversions
DPL	>50		Decimal Point Location
FLD	>52		Field Width (unused)
CSP	>54		Stack Pointer for error checking
R#	>56		Editing Cursor location
HLD	>58		Holds addr during numeric conversion
USE	>5A		Next Block Buffer to Use
PREV	>5C		Most recently accessed disk buffer
	>5E		Don't use
	>60		Don't use
FORTH_LINK	>62		FORTH Vocabulary base
ECOUNT	>64		Error control
VOC-LINK	>66		Vocabulary linkage

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 -SYNONYMS. 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.

OPTION: -SYNONYMS

Starting Screen: 33
Loads:

VSBW	VMBW	VSBR
VMBR	VWTR	GPLLNK
XMLLNK	DSRLNK	CLS
FORMAT-DISK	VFILL	VAND
VOR	VXOR	MON
RNDW	RND	SEED
RANDOMIZE		

OPTION: -EDITOR

Starting Screen: 34
Loads: -SYNONYMS and

EDIT	EDG	WHERE

OPTION: -COPY

Starting Screen: 39
Loads:

!"	DTEST	SCOPY
SMOVE	FORTH-COPY	DISK-HEAD

OPTION: -DUMP

Starting Screen: 42
Loads:

DUMP	.S	VLIST

OPTION: -TRACE

Starting Screen: 44
Loads: -DUMP and

TRACE	UNTRACE	TRON
TROFF	: (alternate)	

OPTION: -FLOAT

Starting Screen: 45
Loads: -SYNONYMS and

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.	FO<
FO=	F>	F=
F<	FLERR	?FLERR
INT	^	SQR
EXP	LOG	COS
SIN	TAN	ATN
PI		

OPTION: -TEXT

Starting Screen: 51
Loads: -SYNONYMS and TEXT

OPTION: -GRAPH1

Starting Screen: 52
Loads: -SYNONYMS and GRAPHICS

OPTION: -MULTI

Starting Screen: 53
Loads: -SYNONYMS and MULTI

OPTION: -GRAPH2

Starting Screen: 54
Loads: -SYNONYMS and GRAPHICS2

OPTION: -SPLIT

Starting Screen: 55
Loads: -SYNONYMS, -GRAPH2 and

SPLIT SPLIT2

OPTION: -VDPMODES

Starting Screen: 51
Loads: -SYNONYMS, -TEXT, -GRAPH1, -MULTI, -GRAPH2 and -SPLIT

OPTION: -GRAPH

Starting Screen: 57
Loads: -SYNONYMS, -CODE and

GCHAR	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

OPTION: -FILE

Starting Screen: 68
Loads: -SYNONYMS and

FILE	GET-FLAG	PUT-FLAG
SET-FAB	CLR-STAT	CHK-STAT
FXD	VRBL	DSPLY
INTRNL	I/CND	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
STAT		

OPTION: -PRINT

Starting Screen: 72
Loads: -SYNONYMS, -FILE and

SWCH	UNSWCH	?ASCII
TRIAD	TRIADS	INDEX

OPTION: -CODE

Starting Screen: 74
Loads:

CODE	;CODE

OPTION: -ASSEMBLER

Starting Screen: 75
Loads: -CODE and Entire Assembler Vocabulary. See Chapter 9.

OPTION: -64SUPPORT (64 Column Editor)

Starting Screen: 22
Loads: -SYNONYMS, -GRAPH, -TEXT, -GRAPH2, -SPLIT and

EDIT	ED@	WHERE
CLIST	CLINE	

OPTION: -BSAVE

Starting Screen: 83
Loads: BSAVE

OPTION: -CRU

Starting Screen: 88
Loads: -CODE and

SBO	SBZ	TB
LDCR	STCR	

APPENDIX G

ASSEMBLY SOURCE FOR CODED WORDS

Several words on the FORTH System Disk have been written in 9900 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.

```
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  0F 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  0 VARIABLE DTAB
  2
  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
 11
 12
 13
 14
 15 R->BASE

SCR #44
  0 ( SOURCE FOR SMASH )  BASE->R  HEX
  1 TCHAR 7C - CONSTANT TC
  2 CODE SMASH ( ADDR #CHAR LINE# ---- LB VADDR CNT )
  3   *SP+ 1 MOV,  *SP+ 2 MOV,  *SP 3 MOV,  4 LB    LI,
  4   4 *SP  MOV,  SP    DECT,  1    SWPB,  1 2000 AI,
  5   1 *SP  MOV,  2 1   MOV, 1 INC, 1 FFFE ANDI,  SP   DECT,
  6   1 2 SLA,  1 *SP  MOV,
  7    3 2    A,  BEGIN,  2 3 C,  GT  WHILE,  5 CLR,  6 CLR,
  8              3 *?+ 5 MOVB,  3 *?+ 6 MOVB,   5 6 SRL,  6 6 SRL,
  9   BEGIN,  TC 5 @(?) 0 MOV,  TC 6 @(?) 1 MOV,  1 4 SRC,
 10   C 4 LI, BEGIN, 0 B MOV, B FOCO ANDI,  1 7 MOV,  7 FOC ANDI,
 11   B 7 SOC,  7 4 *?+ MOVB,  0 C SRC,  1 C SRC, C DEC, EQ UNTIL,
 12   5 INCT,  6 INCT,  5 C MOV,  C 2 ANDI,  EQ UNTIL,  REPEAT,
 13   NEXT,
 14
 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 , 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 , 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 , ( L)
  3 0AEE , AAAA , ( M)  0AAE , EEAA , ( N)  0EAA , AAAE , ( O)
  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)  0C04 , 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) 000A , 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

TI FORTH ERROR MESSAGE EXPLANATIONS

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.
-	full stack	The proceedure 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 >837C 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. Non-existing file opened, etc.

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 trignometric 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 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	conditionalls not paired	A DO has been left without 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 preceeding [illegible] example, -->
23	off current editing screen	Not used in TI FORTH.
24	declare vocabulary	Not used in TI FORTH due to the 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

SCR #2

T I F O R T H

THIS VERSION OF THE FORTH LANGUAGE
IS BASED ON THE fig-FORTH MODEL

THE ADDRESS OF THE FORTH INTEREST GROUP IS:

FORTH INTEREST GROUP
P.O. BOX 1105
SAN CARLOS, CA 94070

TEXAS INSTRUMENTS PERSONNEL WITH SIGNIFICANT
INPUT TO THIS VERSION INCLUDE:
LEON TIETZ
LESLIE O'HAGAN
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


TI FORTH --- a fig-FORTH extension
```

```
CR #20
   ( CONDITIONAL LOAD  )
   : MENU CR 272 265 DO I MESSAGE CR LOOP CR CR CR ;
   : SLIT ( --- ADDR OF STRING LITERAL )
        R> DUP C@ 1+ =CELLS OVER + >R ;

   : WLITERAL ( WLITERAL word )
        BL STATE @
        IF COMPILE SLIT WORD HERE C@ 1+ =CELLS ALLOT
        ELSE WORD HERE ENDIF ; IMMEDIATE  -->
   -SYNONYMS    -EDITOR      -COPY
   -DUMP        -TRACE       -FLOAT
   -TEXT        -GRAPH1      -MULTI
   -GRAPH2      -SPLIT       -VDPMODES
   -GRAPH       -FILE        -PRINT
   -CODE        -ASSEMBLER   -64SUPPORT
   -BSAVE       -CRU


 FORTH --- a dig-FORTH extension
```

```
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 ;
 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
  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   0000 0000 0000 00F0 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 : GOTOXY ( COL ROW --- ) 64 * + !CUR PLACE ;
 14
 15 R->BASE -->

TI FORTH --- a fig-FORTH extension
```

```
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 1+ R/C DROP C/L SWAP - + SWAP 32
 13   1 C@ LOOP DROP PTR DUP R/C DROP C/L SWAP - + 1- SWAP 1- SWAP
 14   DO 1 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 CINIT ENDIF .!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 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
( SYSTEM CALLS 09JUL82 LCT)  0 CLOAD RANDOMIZE
BASE->R DECIMAL 74 R->BASE CLOAD ;CODE
BASE->R  DECIMAL
: VSBW 0 SYSTEM ; : VMBW 2 SYSTEM ;
: VSBR 4 SYSTEM ; : VMBR 6 SYSTEM ;
: VWTR 8 SYSTEM ; : GPLLNK 0 33660 C! 10 SYSTEM ;
: XMLLNK 12 SYSTEM ; : DSRLNK 8 14 SYSTEM ;
: CLS 16 SYSTEM ; : FORMAT-DISK 1+ 18 SYSTEM ;
: VFILL 20 SYSTEM ; : VAND 22 SYSTEM ; : VOR 24 SYSTEM ;
: VXOR 26 SYSTEM ;        HEX
CODE MON 0200 , 4E4F , 0201 , 2000 , CC40 , 0281 , 4000 , 16FC ,
            0420 , 0000 ,
: RNDW 83C0 DUP @ 6FE5 * 7AB9 + 5 SRC DUP ROT ! ;
: RND RNDW ABS SWAP MOD ;  : SEED 83C0 ! ;
: RANDOMIZE 8802 C@ DROP 0 BEGIN 1+ 8802 C@ 80 AND UNTIL SEED ;
R->BASE
```

```
SCR #34
( SCREEN EDITOR 09JUL82 LCT) 0 CLOAD ED@
BASE->R DECIMAL 33 R->BASE CLOAD RANDOMIZE
BASE->R   HEX VOCABULARY EDITOR1 IMMEDIATE EDITOR1 DEFINITIONS
  : BOX 8F7 8F1 DO 84 I VSBW LOOP ;
  : CUR R# ;
  : !CUR 0 MAX B/SCR B/BUF * 1- MIN CUR ! ;
  : +CUR CUR @ + !CUR ;
  : +LIN CUR @ C/L / + C/L * !CUR ;
  0 VARIABLE S_H     DECIMAL
  : FTYPE 40 * 124 + SWAP VMBW ;
  : LISTA DECIMAL 0 0 GOTOXY DUP SCR !
     ." SCR # " . CR CR CR 16 0 DO I 3 .R CR LOOP ;
  : ROWCAL S_H @ IF 29 + ENDIF ;
  : LINE. DO I SCR @ (LINE) DROP ROWCAL 35 I FTYPE LOOP ;
  : LISTB L/SCR 0 LINE. ;
R->BASE -->
```

```
SCR #35
( SCREEN EDITOR 09JUL82 LCT)
: LISTL BASE->R LISTA 4 1  GOTOXY
."           1          2          3      " 4 2 GOTOXY
." ....+....0....+....0....+....0....+"
0 S_H ! LISTB R->BASE ;
: LISTR BASE->R DROP 4 1 GOTOXY
." 3          4          5          6     " 4 2 GOTOXY
." 0....+....0....+....0....+....0...."
1 S_H ! LISTB R->BASE ;
: BCK 0 L/SCR 2+ GOTOXY QUIT ;
: PTR SCR @ B/SCR * CUR @ B/BUF /MOD ROT + BLOCK + ;
: R/C CUR @ C/L /MOD ;   ( --- COL ROW )
: DELHALF PAD 64 BLANKS PTR PAD C/L R/C DROP - CMOVE ;
-->
```

```
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 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 * SCR + 1+ VED ;
 14 : EDIT EDITOR1 0 VED ;  ' EDIT EDITOR1 8 !CSP ;
 15 R->BASE


TI FORTH --- a fig-FORTH extension
```

```
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 + 16B 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 IF 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- .* I * 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.SB 260CT82 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 ]  ( PFA ) LITERAL
14  HERE 2- !   TRACF @ IF ' (TRACE) CFA DUP @ HERE 2- ! , ENDIF ]
15   ; IMMEDIATE

'I FORTH --- a fig-FORTH extension
```

```
CR #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  833C 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  -->

CR #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 3 ;
 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 -->

CR #47
 0 ( FLOATING POINT CONVERSION ROUTINES CONTINUED 12JUL82 LCT)
 1 BASE->R HEX
 2 : DOSTR FAC 3 + C! 14 GPLLNK
 3   FAC 3 + 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 3 + C! FAC C + C! DOSTR ;
11
12   STRING AT PAD CONVERTED TO NUMBER IN FAC.
13 : VAL PAD 1+ 1000 DUP FAC C + ! PAD C@ OVER OVER + >R SWAP VSBW
14   VMBW 1000 XMLLNK ;
15 R->BASE -->

( FORTH --- a fig-FORTH extension
```

```
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 3 83C4 ! ;
  4 : INT >FAC 22 GLNK FAC> ;
  5 : ^   SETFL ARG 836E @ 9 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


TI FORTH --- a fig-FORTH extension
```

```
SCR #51
 0 ( CONVERT TO TEXT MODE CONFIGURATION 14SEP81 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 14SEP81 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 14SEP81 LAO)
 1 0 CLOAD MULTI  BASE->R DECIMAL 56 R->BASE CLOAD VDPSETC
 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    MON SET VDP REGISTERS
12      3 VWTR   11 7 VWTR
13 0E2 SETVDP2 ;
14
15 R->BASE


1 FORTH --- a dig-FORTH extension
```

```
SCR #54
  0 ( CONVERT TO GRAPHICS2 MODE CONFIG 14SEP82 LAO)
  1 0 CLOAD GRAPHICS2 BASE->R DECIMAL 56 R->BASE CLOAD VDPSET2
  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 ! 1D00 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 VDPSET2
  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 GOTOX? 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


TI FORTH --- a fig-FORTH extension
```

```
SCR #57
 0 ( GRAPHICS PRIMITIVES 12JUL83 LCT)  0 CLOAD CHAR BASE->R DECIMAL
 1  33 B->BASE CLOAD RANDOMIZE  BASE->R DECIMAL 74 B->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 ---  )
 0   >R >R SCRN_WIDTH @ * + SCRN_END @ SCRN_START @ - SWAP
 1   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 B->BASE -->
15


 0 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   SATR 20 0 DO DUP >R D000 SP@ R> 2 VMBW DROP 4 + LOOP DROP
 0   VDPMDE @ 4 < IF SMTN 80 0 VFILL 300 ! SATR ! ENDIF
 1   ( INIT ALL SPRITES ) ;
 2 : SPCHAR  ( W1 W2 W3 W4 CH#  ---  )
 3   8 * SPDTAB + >R -2 6 DO PAD I + ! -2 +LOOP PAD R> 8 VMBW ;
 4 : SPRCOL ( COL #  ---  )  4 * SATR 3 - + DUP >R VSBR 0F0 AND OR
 5   R> VSBW ;                R->BASE -->


 0 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 00FF AND OR SP@ R> 2 VMBW DROP ;
10 : #MOTION ( NO ---  )  837A C! ;
11 : SPRGET ( # --- DX DY )
12   4 * SATR + DUP VSBR 1+ SWAP 1+ SWAP VSBR SWAP ;
13 : DX? ( X2 Y2 X1 Y1 --- X^2 Y^2 )
14   ROT - ABS ROT ROT - ABS DUP * SWAP DUP * ;
15 B->BASE -->


  FORTH --- a dig-FORTH extension
```

SCR #60

```
( GRAPHICS PRIMITIVES 12JUL82 LCT)
BASE->R HEX : BEEP 34 GPLLNK ;     : HONK 36 GPLLNK ;
: SPRDIST ( #1 #2 --- DIST^2 ) ( DISTANCE BETWEEN 2 SPRITES )
  SPRGET ROT SPRGET DXY OVER OVER
  + DUP >R OR OR 8000 AND IF R> DROP 7FFF ELSE R> ENDIF ;
: SPRDISTXY ( X Y # --- DIST^2 ) SPRGET DXY OVER OVER
  + DUP >R OR OR 8000 AND IF R> DROP 7FFF ELSE R> ENDIF ;
: MAGNIFY ( MAG-FACTOR ---  )
  83D4 C@ 0FC AND + DUP 83D4 C! 1 VWTR ;
: JOYST ( KEYBDNO --- ASCII XSTAT YSTAT )  8374 C!
  ?KEY DROP 8375 C@ DUP DUP 12 = IF DROP 0 0 ELSE 0FF =
  IF 8377 C@ 8376 C@ ELSE 8375 C@
  CASE 4 OF 0FC 4  ENDOF  5 OF 0 4 ENDOF  6 OF 4   4 ENDOF
        2 OF 0FC 0  ENDOF  3 OF 4 0 ENDOF  0 OF 0 0FC ENDOF
       0F OF 0FC 0FC ENDOF 0E OF 4 0FC ENDOF DROP DROP 0 0 0 0
  ENDCASE ENDIF ENDIF  4 8374 C! ; R->BASE -->
```

SCR #61

```
( GRAPHICS PRIMITIVES 12JUL82 LCT)  BASE->R HEX
: COINC ( #1 #2 TOL  --- F ) ( 0= NO COINC  1= COINC )
  DUP * DUP + >R SPRDIST R> > 0= ;
: COINCXY ( DX DY # TOL --- F )
  DUP * DUP + >R SPRDISTXY R>  > 0= ;
: COINCALL  ( --- F ) ( BIT SET IF ANY TWO SPRITES OVERLAP )
  8802 C@ 20 AND 20 = ;
: DELSPR ( # --- )
  4 * DUP SATR + >R 0 C001 SP@ R> 4 VMBW DROP DROP
  SMTN + >R 0 0 SP@ R> 4 VMBW DROP DROP ;
: DELALL ( --- )
  0 #MOTION SATR 20 0 DO DUP DO SWAP VSBW 4 + LOOP DROP
  SMTN 80 0 VFILL ;



  R->BASE -->
```

SCR #62

```
( GRAPHICS PRIMITIVES 24NOV82 LAO) BASE->R  HEX  0 VARIABLE ADR
: MINIT 18 0 DO 0 I 4 / 20 * DUP 20 + SWAP
                DO DUP J 1 I HCHAR 1+ LOOP  DROP LOOP ;
: MCHAR ( COLOR C R --- ) DUP >R 2 / SWAP DUP >R 2 / SWAP
  DUP >R GCHAR  DUP 20 /  100 U* DROP 800 + >R 20 MOD
  8 * R> + R>  4 MOD 2 * + ADR ! R> 2 MOD R> 2 MOD SWAP
  IF  IF 3 ELSE 1 ENDIF  ELSE  IF 2 ELSE 0 ENDIF   ENDIF
  DUP 2 MOD 0= IF SWAP 10 * SWAP ENDIF
  CASE 0 OF ADR @ VSBR OF ENDOF  1 OF ADR @ VSBR F0 ENDOF
  2 OF 1 ADR +! ADR @ VSBR OF ENDOF
  3 OF 1 ADR +! ADR @ VSBR F0 ENDOF
  ENDCASE AND +   ADR @  VSBW ;
0 VARIABLE DMODE   -1 VARIABLE DCOLOR
: DRAW 0 DMODE ! ;   : UNDRAW 1 DMODE ! ;    : DTOG 2 DMODE ! ;
8040 VARIABLE DTAB 2010 , 804 , 201 , 7FBF , DFEF , F7FD ,
FDFE , 8040 , 2010 , 804 , 201 ,   R->BASE -->
```

SCR #63

```
( GRAPHICS PRIMITIVES ) BASE->R  HEX
CODE DDOT   C079 ,
     COD9 , C081 , C103 , 0241 ,
     0007 , 0243 , 0007 , 0242 ,
     00F8 , 0244 , 00F8 , 0A52 ,
     A042 , A044 , 0221 , 2000 ,
     04C4 , D123 , D7A8 , 06C4 ,
     C644 , 0649 , C641 , 045F ,
: DOT ( X Y --- )
  DDOT DUP 2000 - >R DMODE @
  CASE  0 OF VOR  ENDOF  ( DRAW )
        1 OF SWAP FF XOR SWAP VAND ENDOF  ( UNDRAW )
        2 OF VXOR ENDOF  ( TOGGLE )
  DROP DROP ENDCASE R>
  DCOLOR @ 0 < IF DROP ELSE DCOLOR @ SWAP VSBW ENDIF ;
 R->BASE  -->
```

SCR #64

```
( GRAPHICS PRIMITIVES 12JUL82 LCT) BASE->R HEX
: SGN DUP IF DUP 0< IF -1 ELSE 1 ENDIF ELSE 0 ENDIF + ;
: LINE >R R ROT >R R - SGN SWAP >R R ROT >R R - SGN OVER ABS
  OVER ABS < >R R 0= IF SWAP ENDIF 100 ROT ROT */ R>
  IF ( X AXIS ) R> R> OVER OVER >
     IF ( MAKE L TO R ) SWAP R> DROP R>
     ELSE R> R> DROP
     ENDIF 100 + ROT ROT 1+ SWAP
     DO I OVER 0 100 M/ SWAP DROP DOT OVER + LOOP
  ELSE ( Y AXIS ) R> R> R> R> ROT >R ROT >R OVER OVER >
     IF ( MAKE T TO B ) SWAP R> DROP R>
     ELSE R> R> DROP
     ENDIF 100 + ROT ROT 1+ SWAP
     DO DUP 0 100 M/ SWAP DROP I DOT OVER + LOOP
  ENDIF DROP DROP ;
R->BASE
```

SCR #65

```
( COMPACT LIST )
0 CLOAD SMASH  BASE->R DECIMAL 74 R->BASE CLOAD :CODE
BASE->R DECIMAL 33 R->BASE CLOAD RANDOMIZE  BASE->R DECIMAL
0 VARIABLE TCHAR 382 ALLOT 67 BLOCK TCHAR 384 CMOVE HEX
TCHAR 7C - CONSTANT TC  0 VARIABLE BADDR  0 VARIABLE INDX
( SMASH EXPECTS ADDR #CHAR LINE# --- L3 VADDR CNT )
0 VARIABLE L3 FE ALLOT
CODE SMASH
  C079 , C039 , COD9 , 0204 , L3   , C644 , 0649 , 06C1 ,
  0221 , 2000 , C641 , C042 , 05B1 , 0241 , FFFE , 0649 ,
  0A21 , C641 , A083 , 80C2 , 1501 , 1020 , 04C5 , 04C6 ,
  [illegible]
  [illegible]
  0F00 , E1C3 , DD07 , 03C0 , 03C1 , 060C , 16F4 , 05C5 ,
  05C6 , C305 , 024C , 0002 , 16E7 , 10DD , 045F ,
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 + 83D6 !
  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 : [illegible] ;
 13 : [illegible] ;
 14
 15 R->BASE

FI FORTH --- a fig-FORTH extension
```

```
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     ;
 13
 14
 15


TI FORTH --- a fig-FORTH extension
```

```
SCR #75
( ASSEMBLER 12JUL82 LCT)  0 CLOAD ASSM
BASE->R DECIMAL 74 R->BASE CLOAD ;CODE
BASE->R HEX
ASSEMBLER DEFINITIONS
: GOP' OVER DUP 1F > SWAP 30 < AND
        IF + , , ELSE + , ENDIF ;
: GOP <BUILDS , DOES> @ GOP' ;
0440 GOP B,       0680 GOP BL,       0400 GOP BLWP,
04C0 GOP CLR,     0700 GOP SETO,     0540 GOP INV,
0500 GOP NEG,     0740 GOP ABS,      06C0 GOP SWPB,
0580 GOP INC,     05C0 GOP INCT,     0600 GOP DEC,
0640 GOP DECT,    0480 GOP X,
: GROP <BUILDS , DOES> @ SWAP 40 * + GOP' ;
2000 GROP COC,   2400 GROP CZC,   2800 GROP XOR,
3800 GROP MPY,   3C00 GROP DIV,   2C00 GROP XOP,
-->

SCR #76
( ASSEMBLER 12JUL82 LCT)
: GGOP <BUILDS ,
     DOES> @ SWAP DUP DUP 1F > SWAP 30 < AND
             IF 40 * + SWAP >R GOP' R> ,
             ELSE 40 * + GOP' ENDIF ;
A000 GGOP A,     B000 GGOP AB,
8000 GGOP C,     9000 GGOP CB,
6000 GGOP S,     7000 GGOP SB,
E000 GGOP SOC,   F000 GGOP SOCB,
4000 GGOP SZC,   5000 GGOP SZCB,
C000 GGOP MOV,   D000 GGOP MOVB,

: OOP <BUILDS , DOES> @ , ;
0340 OOP IDLE,   0360 OOP RSET,  03C0 OOP CKOF,
03A0 OOP CKON,   03E0 OOP LREX,  0380 OOP RTWP,
-->

SCR #77
( ASSEMBLER 12JUL82 LCT)

: ROP <BUILDS , DOES> @ + , ;

02C0 ROP STST,  02A0 ROP STWP,

: IOP <BUILDS , DOES> @ , , ;

02E0 IOP LWPI,  0300 IOP LIMI,

: RIOP <BUILDS , DOES> @ ROT .

0220 RIOP AI,   0240 RIOP ANDI,
[illegible]
0260 RIOP ORI,
-->
```

```
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 1D01 , 0 ENDOF
 13            DUP 0< OVER 10 > OR IF ." ERROR END-IF DUP
 14       ENDCASE 100 * 1000 + , ;
 15 -->


TI FORTH --- a fig-FORTH extension
```

```
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


10 : NEXT, *NEXT B, ;
11
12 FORTH DEFINITIONS
13
14 : A$$M ;              B->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   ; B->BASE


FORTH --- a Big-FORTH extension
```

```
SCR #84
  0 ( NEW MESSAGE ROUTINE  1636P82   LOT )     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 - + - 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


TI FORTH --- a fig-FORTH extension
```

```
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 , 1e01 , 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
```