EXTENDED fig-FORTH

by

Patrick L. Mullarky

TRADEMARKS OF ATARI

****

Distributed by

The ATARI Program Exchange
P. O. Box 427
155 Moffett Park Drive, B-1
Sunnyvale, CA 94086

To request an APX Software Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/745-5535.

****

CONTENTS

INTRODUCTION

OVERVIEW

EXTENDED FIG-FORTH fully implements the standard FORTH, as defined in the Forth Interest Group's (fig) Implementation Guide. It roughly follows the 6502 Rev. 1.1 FORTH sources as supplied by the Forth Interest Group (FORTH INTEREST GROUP, P.O. Box 1105, San Carlos, CA 94070). Many changes were incorporated in adapting the sources to the ATARI Home Computer, but the definitions, operation, and user interfaces were implemented exactly as described in the Implementation Guide. Many additional definitions have been added, including extended double-precision words such as 2DUP, 2SWAP, D@, and D!. Further, the standard FORTH Editor, and a complete Assembler for the 6502 are included, as well as a set of ATARI Color/Graphic definitions, ATARI OS definitions, and a set of ATARI Floating-point definitions. One new definition, SAVE , (and CSAVE) allows a self-booting image of FORTH to be made on a diskette or cassette that will include new definitions you add; this feature allows application packages to be produced in volume. Definitions not implemented are DLIST, MON, and TASK. The complete set of ATARI Screen-Editor capabilities is implemented, making editing and changing FORTH programs simple and straightforward.

These instructions assume you are already familiar with FORTH. However, the manual does contain two bibliographies, one for works pertaining to FORTH and a more general one. There is also a two-page FORTH HANDY REFERENCE summary in the back.

If you're a beginning FORTH programmer, an excellent book to help you get started is Starting FORTH, by Leo Brodie, written at FORTH, Inc., and published by Prentice-Hall. There are some differences between FORTH Inc.'s "PolyForth" and fig-FORTH: the word 'S is SP@ in fig-FORTH; the word CREATE cannot be used to create an array name directly, as shown in the book; and the only character that defines a double-precision value in fig-FORTH is the decimal point, whereas PolyForth allows several others, and there are other differences between PolyForth and fig-FORTH.

REQUIRED ACCESSORIES

Cassette version

16K RAM
ATARI 410 Program Recorder

(Note. FORTH as a computer language isn't very workable in a cassette-only environment. But applications software using FORTH can be put onto a self-booting cassette if desired.)

Diskette version

16K RAM
ATARI 810 Disk Drive

OPTIONAL ACCESSORIES

All ATARI peripherals and accessories

(Note. Extended fig-FORTH will work with any ATARI printer using two new definitions, PON , and POFF which turn the printer on and off. The printer does not print the prompts as they occur on the screen, allowing very clean printouts.)

CONTACTING THE AUTHOR

Users wishing to contact the author about Extended fig-FORTH may write to him at:

206 Northside Road
Bellevue, WA 98004

or call him at:

206/453-9698

GETTING STARTED

DISKETTE VERSION

1. Remove any cartridge from the (Left Cartridge) slot of your computer.

2. Place the Extended fig-FORTH diskette in your disk drive and turn on the drive and the computer.

3. The program will load into memory and the prompt "fig-FORTH 1.1" will display when the load is complete. Press the RETURN key to display the standard FORTH prompt " ok ".

4. The Editor, Assembler, Debug, OS, Color/Graphics, and Floating-point packages included with Extended fig-FORTH must be loaded in after booting-up the disk. Instructions for loading and using each package follow.

5. After loading in whichever packages you need, you can make a new copy of FORTH that includes your loaded packages by inserting a formatted diskette into disk drive 1 and typing "SAVE". A self-booting copy will then be written to the new diskette.

6. Now replace the original diskette, type " 14 LIST MARK 15 LIST MARK " , and press the RETURN key. Two screens of error messages will be listed and saved internally.

7 Change diskettes once again and type " FLUSH " and the error messages will be written to your new diskette. You now have a clean diskette for your program development.

8. Store the original FORTH diskette in its folder and put it in a nice safe place. Note that you may make a complete copy of your original diskette using the DISKCOPY routine described later. This will copy the whole diskette, not just the FORTH and error messages.

CASSETTE VERSION

1. Remove any cartridge from the (Left Cartridge) slot of your computer.

2. Turn off the computer and all other peripheral devices. Insert the cassette into the program recorder.

3. Hold down the START key on the computer and turn on the computer. The computer should beep.

4. Press the PLAY button on the program recorder.

5. Press the RETURN key on the computer and the cassette will load itself in. If the program successfully loads, you will see the prompt "fig-FORTH 1.1.

6. SEE THE CASSETTE NOTES AT THE END OF THIS SECTION.

NOTES ON THIS IMPLEMENTATION

Editor and Assembler options

You have several options regarding the EDITOR and ASSEMBLER vocabularies: in addition to the standard EDITOR, a version of the FORTH Inc. Editor has been included. It may be loaded with a 69 LOAD command. Further, the Assembler written by Wm. Ragsdale is supplied (use the command 75 LOAD), which is identical to the assembler used in the Installation Guide.

16K RAM limitation

If you have only 16K of RAM you will not be able to use some of the Color/Graphics higher-level graphics modes without interfering with the screen buffers.

Cold starts with SYSTEM RESET key

The SYSTEM-RESET key calls the "COLD" (cold-start) function directly, so any new word definitions that have not been SAVEd will be erased. This can be a handy feature while debugging: press the SYSTEM RESET key to erase all your old work and leave a clean copy. There is a negative side: if your program wanders off into never-never land, and you have to press SYSTEM RESET, you'll lose all your new definitions unless you've been editing them into new screens. (Using the standard OS screen-editing functions excludes the use of the BREAK key for this purpose. The BREAK key is used to inform the system to ignore the previous input string.)

FORTH and DOS incompatability

There is no compatability between FORTH diskettes and DOS (I or II) diskettes. You may read a DOS diskette with a FORTH program, but unless you know exactly what you're doing, writing to a DOS diskette will, in all probability, make the diskette unworkable from a DOS point of view. The only DOS function applicable to FORTH is that FORTH expects DOS-formatted diskettes.

7-bit and 8-bit output

The word TYPE outputs only 7 bits to the screen or printer. If you want TYPE to output all 8-bits (which includes inverse video characters), you can type in the following sequence:

```
HEX  FF  '  TYPE  14  +  C!  DECIMAL
```

In fact, you can make up a couple of routines if you wish:

```
HEX
:  MODTYPE      ' TYPE 14 + C!  ;

:  8-BITS     FF   MODTYPE   ;
:  7-BITS     7F   MODTYPE   ;
DECIMAL
```

Then, to set your system to type out 7 bits, type 7-BITS, and for 8 bits, type 8-BITS.

Further, you can use these routines in any other programs you wish, just as you would any other word definition. If you type VLIST with TYPE set to 8-BITS then the last character of each word will be in inverse video. The word EMIT always outputs all 8 bits in each byte. TYPE uses EMIT with a mask for 7 or 8 bits.

ERROR Screens

The ERROR screens are 13 and 14 instead of the standard 3 and 4. This is because the self-booting FORTH interpreter, if it is present on the diskette you're using, occupies screens 0 through 7, with 6 screens available for larger versions. If your working diskette doesn't have a bootable FORTH on it, you may use all screens numbered 0 through 89. Disk drive 2 screens are numbered 90 through 179. The second drive may also be accessed by the word DR1 , which sets an offset into the drive addresses for automatically accessing the second drive. The word DR0 accesses the first drive. Alternately, the blocks are numbered 0-719 on the first drive, and 720-1439 on the second drive.

Disk Blocks

This is fig-FORTH, NOT FORTH-79! This means that disk blocks are 128 bytes long and not 1K bytes long. Each screen is 8 blocks long, not 1 block long! A later version will be made available, someday, using the FORTH-79 standard, but Extended fig-FORTH uses the fig-FORTH standard.

DEFINITIONS

SAVE ---

This word, when executed, saves a self-booting copy of the RAM-resident FORTH program to disk drive 1, after setting up new parameters for COLD and FENCE. On booting up, all definitions will be protected by FENCE, and the FORTH vocabulary will be the current dictonary. This word uses (SAVE) described later.

CSAVE ---

This word saves a self-booting copy of the RAM-resident FORTH program to the cassette recorder. The computer will beep twice, indicating that you are to press both the PLAY and the RECORD buttons on the recorder, followed by pressing the RETURN key on the computer.

(SAVE) n ---

This word writes n blocks to disk drive 1, starting at sector 0. This word should not be used by normal FORTH programs.

-DISK addr n2 n3 flag --- n4

This word performs the read/write on a disk, where addr is the starting RAM address, n2 is the diskette sector number (0-719), n3 is the drive number (1-4), and flag is 1 for a read, and 0 for a write. On return, n4 will contain a zero if everything went all right, or it will contain the DOS error number returned by DOS if an error occurred. It is not expected that the normal FORTH program will use this word. The usual disk I/O word used is R/W, which is documented in the Implementation Guide.

ASCII --- c --> n

This word places the binary value of character c on the top of the stack.

BEEP ---

This word sounds the "beep" tone on the computer's speaker.

BOOT ---

When executed, this word causes a cold-boot of the computer exactly as if the power were turned off.

(FMT) n1 --- n2

This word formats disk drive n1 and returns the DOS status byte upon completion in n2. This word is used by the word FORMAT in the OS definitions. No error checks are made and no warnings aregiven by this word. Those functions are performed by the FORMAT word. For more information, see the OS section in this manual.

ok ---

This word allows the Screen Editor (E:) to handle the standard FORTH prompt properly. The interpreter can "eat" the previous "ok" prompt with no other effect. It allows you to repeat the same input stream by placing the cursor anywhere in a previous line and pressing the RETURN key.

PON ---

This word enables the printer. PFLAG is set to 1, and thereafter every character put to the screen will be echoed on the printer except the prompts.

POFF ---

This word disables the printer. It sets PFLAG to zero.

PFLAG --- addr

This word is the printer-flag. See PON.

GFLAG --- addr

This word is the graphics-mode, cursor-control flag. When GFLAG is set to non-zero, FORTH will use the alternate cursor-address variables required by the Operating System to handle the text-window at the bottom of the screen. This variable is handled automatically by the various graphic commands in the Color/Graphics package.

PROMPT ---

This word was added to handle the extended complexities of excluding the prompt from the printer when PFLAG is non-zero. Basically it types "ok".

Words for using the Assembler

A series of words are defined for the ASSEMBLER:

```
NEXT
PUSH
PUT
PUSH0A
POP
POPTWO
BINARY
IP
W
N
XSAVE
UP
```

Please refer to the ASSEMBLER documentation for their descriptions.

NOTES

THE CASSETTE VERSION

The cassette version of fig-FORTH contains the ASSEMBLER and DEBUG vocabularies already loaded. Because no diskette is used, the EDITOR vacabulary is essentially useless. However, printouts of the EDITOR, OS, and COLOR/GRAPHICS screens are included so that you may type them in if you wish. The cassette version is primarily for use as an introduction to the FORTH language, and not as a software development system. Nevertheless, the CSAVE feature allows you to develop permanent versions of your FORTH programs. See the following section for how to erase old definitions. Note that error messages in the cassette version type only a number. Refer to the printout of the error message screens for their meaning. The error numbers start sequentially at screen 14, line 1 (error 1).

MODIFYING THE DICTIONARY

To erase a definition in your FORTH dictionary that is locked in (you get an "in protected dictionary" message when you try to FORGET a definition) do the following: using VLIST, find the name of the first word that you want to keep, call it XXX , and type ' XXX FENCE ! <RETURN>. This will set the dictionary protection to your XXX word. Then you may type FORGET name <RETURN>, where "name" is the name of the word you wish deleted. Note that all words above "name" are deleted. You can actually instruct FORTH to forget everything, so be careful. If you make an error in a new definition that FORTH rejects for one reason or another, you may find that you cannot FORGET the new definition, and, in fact, only VLIST seems able to find it at all! In such cases, type the word SMUDGE and you'll be able to FORGET the word. By the way, you can interrupt VLIST anywhere you want by pressing any key except BREAK while it is typing out the dictionary.

"Go FORTH and conquer"

"May the FORTH be with you"

INTRODUCTION

The ASSEMBLER vocabulary included in Extended fig-FORTH is a full-featured 6502 assembler, capable of assembling the range of assembler op-codes. It is similar to W. Ragsdale's assembler used in the fig Installation Manual. To load it, type:

```
39 LOAD
```

As is usual in any FORTH product, the notation used in this assembler is in Reverse Polish Notation (RPN). This brief outline assumes you know assembly language programming very well, particularly in regard to the 6502. The RPN notation will seem very awkward at first, but it allows the full power of FORTH to be brought to bear in an assembler-level routine. The op-codes are very similar to standard 6502 op-codes, except that every one ends with a comma, a FORTH convention for assembler-level codes. Some examples will help describe the assembler:

```
LDA  123   is written as  123  LDA,

similarly,

STA  3BC0    is        3BC0  LDA,
LDA  33,X    is        33   ,X   LDA,
AND  (45,X)  is        45   X)   AND,
STA  (74),Y  is        74   )Y   STA,
LDA  3374,Y  is        3374   ,Y   LDA,
LDX  #7F     is        7F   #   LDX,     or    #  7F   LDA,
```

The current BASE value (radix) of FORTH determines whether the assembler creates hex, decimal, or octal values (or any radix, for that matter).

Non-standard op-codes are the A-register shifts only, which are expressed as:

```
ROL.A,
```

instead of the standard:

```
ROL  A
```

and the op-code for an indirect JMP instruction, which is:

```
nnnn   JMP(),
```

instead of:

```
JMP  (nnnn).
```

Loop constructs use the words BEGIN, and END, (note the commas) and an alias for the latter UNTIL, . The END, is preceeded by a 0= or 0= NOT construct to determine loop termination. The termination test actually assembles as a BNE or BEQ instruction, as in the following example:

```
0   ,X   LDY,     BEGIN,     INY,     0=   END,     NEXT   JMP,
```

The above routine increments the Y-Register until it is zero and exits to a routine named NEXT. It will be assembled as:

```
LDY   0,X
INY
BNE   *-1
JMP   NEXT
```

The Branch instructions have been integrated into a generalized IF construct so that they may be readily incorporated into an unlabeled branch capability. The syntax is:

```
IFxx,   ...   ...   ...   THEN,
```

or

```
IFxx,   ...   ...   ...   ENDIF,
```

where xx is the last two letters of the standard 6502 branch instructions (IFEQ, IFNE, IFMI, etc.). The test will be made on the Status Register as appropriate to the sense of the conditional branch, and if the test is TRUE, the code enclosed between the IFxx, and the THEN, or ENDIF, will be executed; otherwise, the enclosed code will be skipped. The operation of the construct is almost identical to the IF ... THEN at the higher-level FORTH definitions, except that nothing is popped off the stack by the IFxx, words. Instead, a Branch instruction is assembled.

LEGAL EXITS

There are only a few legal exits from assembly language FORTH routines to the main FORTH inner interpreter. These addresses are predefined in the main FORTH dictionary and need no further definition by the assembly language itself. These returns use a cccc JMP, sequence, as shown in later examples. The legal exits are :

NEXT

This is the normal return. It takes no stack action.

PUT

This places the A-Register and the first item on the hardware stack on the top of the stack. That is, it does a 1 ,X STA, PLA, 0 ,X STA, NEXT JMP, sequence. This action overwrites whatever was previously on the top of the stack.

PUSH

This pushes down the stack and does a PUT . This action adds one item to the stack.

POP

This performs the DROP function.

POPTWO

This performs DROP DROP .

PUSHOA

This first pushes the A-Register, followed by a zero. Essentially, it pushes one byte, the A-Register, onto the stack, adding a 16-bit word to the stack with the one byte in the lower half.

BINARY

This word takes two words off the stack and replacea=s them with one word. The best example is the add word + . This routine does a DROP followed by a PUT , which overwrites the old top of the stack.

CALLING THE ASSEMBLER

The word CODE is used to call the assembler automatically when defining a new assembly level routine. The character string following CODE will become a new FORTH word having directly executable assembly level code. Two examples follow that do the same thing--they multiply the top of the stack by two, using a single left shift across the two bytes that are the top of the stack:

```
CODE  2*    0   ,X   ASL,    1   ,X   ROL,    NEXT   JMP,

CODE  *2    0   ,X   LDA,    ASL.A,    PHA,    1   ,X   LDA,    ROL.A,
      PUT   JMP,
```

The first routine shifts the actual memory locations of the top of the stack. This procedure is quite short and very fast. The second routine is the more universal method, in that the arguments are first loaded to the A-Register and later stored. Notice that the low order byte is pushed to the hardware stack and the high-order byte is left in the A-Register on the return to PUT . The second example shows how words are retrieved from the stack and how a return is made. To reach the second word down on the stack, you would use 2 ,X LDA, to access the low byte and 3 ,X LDA, to access the high byte, and so on. You can increment the stack pointer (push the stack) with a DEX, DEX, sequence, and pop the stack with an INX, INX, pair. In fact, the DROP word does a simple INX, INX, NEXT JMP, sequence.

If your routines need the X-Register for any reason, you must save it off someplace. A very convenient place called XSAVE is provided. Do a XSAVE STX, later followed by a XSAVE LDX, instruction.

Several other addresses are made available as "hooks" into the FORTH system. These are predefined words you use at your own risk (you'd better study up a bit before doing so), but some routines, such as in the assembler itself, need these addresses.

IP

This is the Intepreter Instruction Pointer, which points to the next word to be executed.

W

This is the actual execution address of the current word being executed.

N

This is a convenient eight-byte (4-word) save area where you may save your words and bytes by storing them in N+0 , N+1 , N+2 ... N+7 . You can use the following sequence to call an internal routine called SETUP, # 2 LDA, SETUP JSR, if you want to copy the top two stack words into N+0 ... N+3, low bytes first. Use # 3 for the top three stack words, and so on. This does not change the stack itself; it only extracts copies of however many words you want.

On entry to your routine, the Y-Register will contain a zero. This fact can be handy for clearing out bytes or registers. For example, you can clear the A-Register with a simple TAY, instruction.

Using the assembler, like in almost any assembly level programming, is playing with fire, and you'll probably get burned from time to time. But, one of the delights of FORTH is that you can simply re-boot and try again. Careful examination of your code will probably clear up your problems.

Note. A good descripton of Wm. Ragsdale's assembler is in Dr. Dobb's Journal, Vol. 6, No. 9 (Sept. '81). This assembler is quite similar on the surface. Internally, they are totally different approaches to solving the same problem using FORTH. Reading Ragsdale's code and reading the code for this assembler could be very instructive in the area of assembly level FORTH programming.

COLOR/GRAPHICS (& SOUND)

INTRODUCTION

You must have already loaded the ASSEMBLER Vocabulary into your FORTH dictionary before the COLOR/GRAPHICS definitions will LOAD properly. Once you have the ASSEMBLER loaded, type:

```
50 LOAD
```

A small demo program will draw a box and FIL it in Graphics Mode 5 when you enter the word FBOX . Type:

```
57 LOAD
FBOX
```

Type 57 LIST to examine the program itself.

NOTE. As in BASIC, a color value of zero is used to erase a point. Also, note that in Graphics Mode 8, there are only two color values: zero or one.

DEFINITIONS

The following words have been defined for use with Extended fig-FORTH in programming color graphics. Most resemble the commands used in ATARI BASIC.

SETCOLOR n1 n2 n3 ---

Color register n1 (0..4) is set to color n2 (0..15) at luminance n3 (0..7). This word is very similar to ATARI BASIC's SETCOLOR command.

SE. n1 n2 n3 ---

This is a synonym for SETCOLOR using an the abbreviation used in ATARI BASIC.

GR. n ---

This word selects Graphics Mode n where n is defined as in ATARI BASIC's "GRAPHICS n" command. (plus modes 9, 10, and 11).

XGR ---

This word allows easy exit from Graphics Modes 1-8. It essentially does a " 0 GR ".

POS n1 n2 ---

This word sets the X (n1) and Y (n2) coordinates for the next point to be plotted. It does not plot anything by itself. It is primarily used in the FIL word definition.

PLOT n1 n2 n3 ---

This word uses the color value given by n1 to plot the point at position X (n2), Y (n3).

DRAW n1 n2 n3 ---

This word draws a line from the last plotted point, using color value n1 to the point X (n2), Y (n3) .

FIL n ---

This word fills the enclosed area just drawn with color value n. The ATARI BASIC FILL command is somewhat awkward to use. Careful reading of the ATARI BASIC Reference Manual is recommended.

G" --- cccc"

In Graphics Modes 1 or 2 this word performs the way the word ." does in text mode. The character string cccc will be compiled if in compiler mode or typed out if in interpreter mode. The POS word may be used to position the output.

SOUND

The sound command definition is practically identical to ATARI BASIC's SOUND definition. But another word not present in ATARI BASIC lets you alter the "filter" values described in the HARDWARE MANUAL as AUDCTL. The word FILTER! sets this control register.

SOUND n1 n2 n3 n4 ---

This word is used as: chan freq dist vol SOUND . n1 is the channel number (0-3); n2 is the frequency, as described in the ATARI BASIC Reference Manual; n3 is the distortion control (an even number between 0 and 14); and n4 is the volume (0-15).

FILTER! n1 ---

This word stores a value between 0 and 255 into audio control register AUDCTL. The default condition is 0 FILTER!. Using this control is not at all straightforward. Please refer to the HARDWARE MANUAL if you wish to alter the contents of this control register. Or, you can try a few different values and see what happens!

INTRODUCTION

Load the DEBUG package by typing:

```
21 LOAD
```

The package includes several very useful features for testing and debugging your FORTH programs.
Each function is described below, in standard FORTH terminology.

DEFINITIONS

B? ---

This word types out the current BASE value (radix) without changing it. It overcomes an intrinsic difficulty in typing only BASE ? , which always returns the value 10 no matter what the current radix is. (10 is the right answer, always.) This word types out the value Base 10, so that if your current base is hex, B? will type out 16 .

CDUMP addr n ---

This word types out n bytes in character format, starting at addr. For example, to display the characters in any disk block, say, sector 34, type 34 BLOCK 128 CDUMP .

DUMP addr n ---

This word types out n bytes in numerical format using the current value of BASE. You can go from a decimal dump to a hex dump by typing HEX first (and vice-versa).

DECOMP cccc ---

This word decompiles the previously entered, colon definition cccc for debugging purposes. Use this word cautiously. It is defined for the purpose of decompiling colon definitions only, and it can go off to never-never land if you try to decompile things like dictionary headers (e.g., FORTH), words terminated by ;CODE or words whose definitions do not end in ; , such as ABORT . Most non-colon definitions will cause the message " Primitive " to display if you try to decompile them. Try DECOMP VLIST and DECOMP @ to see the different results.

FREE ---

This word types out the number of free bytes of dictionary space left. NOTE that this number will vary depending on the current graphics mode.

H. n ---

This word outputs the top of the stack in hexadecimal, no matter what the current value of BASE is. It is similar to U. (unsigned type-out).

S. ---

This word prints out the contents of the stack in unsigned form using the current BASE (radix). It doesn't change the contents of the stack in any way. This is easily the most useful debugging tool. During program development you will probably use it very frequently.

DISKCOPY

The diskette copying routine supplied with this package is minimal. Load it into memory by typing

```
36  LOAD
```

To invoke the copy routine, type DISKCOPY and you will be prompted for what to do.

This routine requires 32K of RAM to operate, and uses one drive to copy 90 sectors at a time. You may interrupt the copy routine by pushing the SYSTEM RESET key when you think it has copied enough sectors for your application. Or, you may copy single FORTH screens, two at a time, by using the LIST and MARK words as described in the introduction.

INTRODUCTION

The Editor in Extended fig-FORTH is the Screen Editor described in the Forth Interest Group's Installation Manual, complete and unchanged. It isn't the most sophisticated editor around, and it has some quirks that take getting used to. For example, it's difficult to insert spaces into a line of text. But the Editor is specifically designed to work with FORTH screens, and it's handy for that purpose.

To load the Editor into your system, put the Extended fig-FORTH diskette into drive 1 and type:

```
27 LOAD
```

Ignore any errors regarding duplicate names. To use the Editor, you must first type EDITOR to set the context to the Editor vocabulary. To edit a given screen, first type n LIST to load the screen into memory.

One new word has been added to the Editor vocabulary : MARK . This word will mark every line in the current screen (the one you last used the LIST command with) as having been modified, so that when a subsequent FLUSH command is given, the whole screen will be written out. It is used primarily to update backup diskettes and to duplicate single screens onto other diskettes.

Whenever you've finished an editing session, type the word FLUSH to save your work. It is quite important to get into the habit of doing this. If you fail to do so, and subsequently your program bombs out, you can lose the last screen you edited.

COMMANDS

WORD FORM DOES

L L

This word Lists the current screen. The current screen is changed by n LIST which will list out screen n and make it the current screen.

T n T

This word Types out line n and puts the cursor at the beginning of that line.

E n E

This word Erases line n .

D n D

This word Deletes line n and moves up all following lines. Save the contents of the line in a buffer so that you can use an I command later, if desired.

P n P cccc

This word Puts the character string cccc into line n and erases the previous contents, if any. Use this command to create new lines. The string cccc may be any combination of characters and spaces up to 64 characters.

I n I

This word Inserts the buffer from the previous D command into a line created immediately above line n and then moves all following lines (including n) down one line. The last line is lost.

F F cccc

This word Finds character string cccc in the current screen starting from the current cursor position.

B B

This word Backs up the cursor over the word you just found using the F command.

C C cccc

This word inserts Character string cccc into the current line at the current cursor position. This is the primary character-entry command (see also P).

M n M

This word Moves the cursor n characters forward or backward (backward if n is negative).

S n S

This word Spreads the current screen at line n , creating a new line immediately preceeding line n and moving all following lines down one. The last line will be lost.

X X cccc

This word eXtracts the character string cccc and shortens up the line. This is the primary find-and-delete command. The X command uses the F command, which means that the string search will commence from the current cursor position.

CLEAR n CLEAR

This word CLEARs screen n by completely filling it with blanks. It destroys any previous information on that screen. Note that an unused, unCLEARed screen will be filled with hearts, which is the ATARI null character. CLEAR will replace the hearts with spaces.

COPY n m COPY

This word COPYs screen n onto screen m. It destroys any old information on screen m.

MARK MARK

This word MARKs the current screen as having been modified. A subsequent FLUSH command will cause the entire screen to be written out. Use it to copy a single screen to another diskette.

The best way to learn the Editor is to pick an arbitrary unused screen and use the LIST and CLEAR commands to erase it and make it the current screen. Then use the P command to put several lines of text into the new screen. Then, try out the various commands, one at a time, until they become somewhat familiar. Use the command FLUSH if you want to keep the results of your work handy; otherwise, use the command EMPTY-BUFFERS to erase all traces of your screen editing.

INTRODUCTION

The floating-point package uses the ATARI floating-point routines in OS ROM, exactly as ATARI BASIC does. The routines aren't very fast, but they are easily accessible and fairly complete (there are no transcendental functions except LOG and EXP). Most of the floating-point word definitions follow the conventions for double-precision words as far as spelling goes, making them very easy to remember.

Before loading the floating-point package, first make sure that you have already loaded the ASSEMBLER. Then put in the master diskette and type:

```
60 LOAD
```

The floating-point routines will be loaded into the current dictionary.

All floating-point operations assume three-word variables (fn) with few exceptions. The only real variant from standard FORTH nomenclature occurs in the definition of floating-point constants and variables (FCONSTANT and FVARIABLE) in that these operations expect a floating-point number to be on the stack already. Therefore, the syntax is a bit different from single-precision or double-precision constants and variables.

A single-precision variable would, for example, be written:

```
1234 VARIABLE MYNUM
```

whereas a floating-point variable would be written:

```
FLOATING 1234 FVARIABLE MYNUM
```

To reduce typing, the word FLOATING has been given the synonym FP :

```
FP 1234 FVARIABLE MYNUM
```

In fact, the word FLOATING or FP should precede any floating number if you wish that number to be placed on the stack in floating-point format.

You may enter floating-point numbers in any standard Fortran "E" format:

```
1.234
.00000001
-7.8945E-31
9999999
5
```

All the above numbers are legal floating-point numbers as long as they are preceeded by FP or FLOATING . The decimal point is optional for integer values. The package is easy to use. Here's an example of a square-root function definition:

```
: FSQRT      FLOG   FP   2.0   F/   FEXP   ;
```

The routine expects a floating-point value on the top of the stack (top three words), takes the natural log of the value, enters the floating-point value 2.0, divides the numbers, and raises the result to the power "e". This is the standard "slow" square-root routine used in mathematics.

DEFINITIONS

The following definitions conform to the standard FORTH nomenclature, with the addition of the symbol fn (e.g., f1, f2), which represents a three-word floating-point number.

FCONSTANT f1 --- cccc

The character string cccc will be a new word, which will place the floating-point constant f1 on the stack. f1 is normally preceeded by the word FLOATING or FP.

FVARIABLE f1 --- cccc

The character string cccc will be a new word, which will return the address of the floating-point variable whose initial value will be f1. f1 is normally preceeded by the word FLOATING or FP.

FDUP f1 --- f1 f1

This word duplicates the floating-point number on the top of the stack.

FDROP f1 f2 --- f1

this word drops the floating-point number on the top of the stack.

FSWAP f1 f2 --- f2 f1

this word reverses the order (swap) of the top two floating-point numbers on the stack.

FOVER f1 f2 --- f1 f2 f1

This word copies the second floating-point number and places it on the top of the stack.

FLOATING --- cccc --> f1

This word converts the character string cccc to a floating-point number and places it on the top of the stack. cccc must be in valid Fortran-style, floating-point number representation, such as, 1.23 or .67E9 or -9.876E-21 or 5 . There is no error check. If the string cccc is invalid, the value of f1 will be undetermined.

FP --- cccc --> f1

This is a synonym for FLOATING.

F@ addr --- f1

This word loads the floating-point number whose address is on the top of the stack.

F! f1 addr ---

This word stores the floating-point number at the address on the top of the stack. A total of 4 words will be dropped from the stack at the completion of F! .

F. f1 ---

This word types out the floating-point number on top of the stack. The output format will be identical to ATARI BASIC's output format. The floating-point number will then be dropped from the stack.

F? addr ---

This word types out the floating-point number whose address is on top of the stack.

F+ f1 f2 --- f3

This word adds the top two floating-point numbers and places the result on the top of the stack.

F- f1 f2 --- f3

This word subtracts the floating-point number f2 from the floating-point number f1 and places the result on the top of the stack.

F* f1 f2 --- f3

This word multiplies the top two floating-point numbers and places the result on the top of the stack.

F/ f1 f2 --- f3

This word divides the floating-point number f1 by the floating-point number f2 and places the result on the top of the stack.

FLOAT n --- f1

This word converts the integer on top of the stack is to a floating-point number and places the result on the top of the stack.

FIX f1 --- n

This word fixes the floating-point number on the top of the stack (after rounding) and places it on the top of the stack. The range of the integer result must be between -32768 and 32767.

FLOG f1 --- f2

This word replaces the floating-point number on the top of the stack with the number's natural logarithm.

FLOG10 f1 --- f2

This word replaces the floating-point number on the top of the stack with the number's log base 10.

FEXP f1 --- f2

This word raises the floating-point number on the top of the stack to the power "e" and replaces the top of the stack.

FEXP10 f1 --- f2

This word raises the floating-point number on the top of the stack to the power 10 and replaces the top of the stack.

F0= f1 --- flag

This word drops the floating-point num ber from the stack and tests it. If the number is equal to zero, a true flag (1) is placed on the stack; otherwise, a false flag (0) is placed on the stack.

F= f1 f2 --- flag

This word drops the top two floating-point numbers from the stack and compares them. If they're equal, a true flag (1) is placed on the stack; otherwise, a false flag (0) is placed on the stack.

F=< f1 f2 --- flag

This word drops the top two floating-point numbers from the stack and compares them. If f1 is strictly less than f2, then a true (1) flag is placed on the stack; otherwise, a false (0) flag is placed on the stack.

COMMENTS

This package isn't meant to be exhaustive, nor is any claim made for its level of usefulness. However, if you need floating-point capabilities, the package works quite well to extend the range of numbers, particularly in scientific calculations. Trignometric functions could be added by a clever programmer. A sufficient set is SIN, COS, and ATN. A random-number generator could also be added. In fact, any number of features could be added.

In summary, if you can't implement your program specifications using the double-precision capability of FORTH, then try this floating-point package.

INTRODUCTION

This vocabulary package implements the full set of ATARI computer's OS I/O routines. It also adds a FORMAT command, as well as a BOOT850 command, which downloads the RS-232 I/O package into the system so that you may use the asynchronous I/O supplied in ROM in the ATARI 850 Interface Module (devices "R1", "R2", etc.).

Load the OS definitions package by typing:

```
81 LOAD
```

Load the BOOT850 package by typing:

```
83 LOAD
```

Be aware that the ATARI 850 I/O routines take up nearly 2K of RAM, and they are loaded directly into the dictionary.

DEFINITIONS

OPEN addr n1 n2 n3 --- n4

This word opens the device whose name is at addr on channel n1 with AUX1 value n2 and AUX2 value n3. Upon return, it places the OS STATUS byte on top of the stack. The address of the name may be obtained by storing the character name in PAD and then referencing PAD in the OPEN command. EXAMPLE: ASCII S PAD C! will set the character "S" into the PAD buffer. Then, PAD 3 12 0 OPEN will open "S:" on channel 3, with AUX1 = 12 (read-and-write), and AUX2 = 0 .

CLOSE n1 --- n2

This word closes channel n1 and returns the status byte at the top of the stack (n2). The status byte will always be a 1 (operation complete, no errors).

PUTC char n1 --- n2

This word outputs the character char on channel n1 and returns status byte n2.

GETC n1 --- char n2

This word gets one character from channel n1 and returns it and the status byte n2.

GETREC addr n1 n2 --- n3

This word inputs record to address addr but no more than n1 characters from channel n2. It returns status byte n3.

PUTREC addr n1 n2 --- n3

This word outputs n1 characters from a buffer whose address is addr to channel n2. It returns status byte n3.

STATUS n1 --- n2

This word gets the status byte from channel n1.

DEVSTAT n1 --- n2 n3 n4

This word gets the device status bytes n2 and n3 and the normal status byte n4 from channel n1.

SPECIAL n1 n2 n3 n4 n5 n6 n7 n8 --- n9

This command is the OS "Special" command that does anything any of the others can't. n1 thru n6 are the values of AUX1 thru AUX6 , n7 is the command byte (whatever your device wants), and n8 is the channel number. The command returns the status byte n9.

FORMAT ---
This word formats a diskette. The command is self-prompting.

BOOT850 ---
This word boots the Atari 850 Interface Module software drivers into the dictionary. Screen 83 must be loaded to execute this command. DO NOT TRY TO EXECUTE THIS COMMAND TWICE IN A ROW. THE SYSTEM WILL LOCK UP IF YOU DO.

FORTH BIBLIOGRAPHY

In order of technical level

1. Starting FORTH, Leo Brodie, Prentice-Hall

The best all-around book for anyone beginning programming...and not just in FORTH. This quite new book is everything one could want in a FORTH primer. It begins by assuming that you know absolutely nothing about computers at all and leads you to some quite sophisticated programs at the end. Even experienced programmers will learn a great deal from this fine work. HOWEVER, the text is not too compatible with fig-FORTH. There are many examples that will cause trouble when using fig-FORTH. Nevertheless...buy this book !! .... and read it !!!

2. Invitation to FORTH, Harry Katzan, Jr., Petrocelli Books

This book is for the total novice, and deals primarily with introducing the first-time computer user to the fundamental concepts of computer programming, and explores FORTH somewhat casually as it moves along. Non-novice users will become impatient with the long elementary discussions and the awkward type-face (no descenders).

3. BYTE Magazine, Vol.5 No.6 (Aug. '80)

The FORTH-dedicated issue which helped bring the concepts of FORTH to thousands of people who might not otherwise have ever heard of the language. While the presentations are somewhat erratic in their technical content, the whole issue deserves reading to acquire a taste for FORTH.

4. Dr. Dobb's Journal, Vol.6 No.9 (Sept. '81)

A second "dedicated issue" on the FORTH Language. This issue approaches FORTH from quite a philosophical point of view, and is excellent reading for the somewhat advanced programmer who, say, already knows several languages. The issue is a wealth of ideas and solid FORTH programs ... the Ragsdale Assembler, for one !

5. A FORTH PRIMER, W. Richard Stevens, Kitt Peak Nat'l Observatory

This is a "self-study" guide to FORTH from the place where it all started. The FORTH described differs somewhat from fig-FORTH, but the book is quite good. It includes some floating-point words which are not too different from the package included with this product.

6. Systems Guide to fig-FORTH, C. H. Ting, Offete Enterprises.

A complete, in-depth analysis of every fig-FORTH word used in the entire fig-FORTH vocabulary. If you ever wondered just exactly how a word such as 'INTERPRET' works ... it's all here !! For the advanced FORTH programmer.

7. Threaded Interpretive Languages, R. G. Loeliger, McGraw-Hill

This is a definitive work for those who want to write their own FORTH Language processor. It uses 8080 code for its examples, but the routines are so well explained that it would be quite easy to translate the code to any other processor. The FORTH isn't exactly fig-FORTH, but the differences are quite minor, and are easily accomodated.

8. FORTH Dimensions, the journal of the Forth Interest Group (fig) All Vols.

These bound journals are available from the Forth Interest Group, P.O. Box 1105, San Carlos, CA 94070. The FORTH Language at its best and its worst. A highly-technical journal for the FORTH addict.

ALL OF THE ABOVE ARE AVAILABLE FROM:
Mountain View Press
P.O. Box 4656, Mountain View, CA 94040
(415)-961-4103

BIBLIOGRAPHY

GOOD BOOKS FOR LEARNING TO PROGRAM IN FORTH:

Using FORTH
FORTH Inc.
Hermosa Beach, CA 90254

Starting FORTH
by Leo Brodie
FORTH, Inc.
Hermosa Beach, CA 90254
Prentice-Hall, Inc. 1981

REFERENCES FOR DEVELOPING GOOD STRUCTURED PROGRAMMING TECHNIQUES:

1. D.L. Mills, "Executive systems and software development for mini computers," Proc. IEEE, vol. 61, pp. 1556-1562, November 1973.

2. J. Koudela, Jr., "The past, present and future of minicomputers," Proc. IEEE, vol. 61, pp. 1526-1534, November 1973.

3. R. Burns and D. Savitt, "Microprogramming and stack architecture ease the minicomputer programmer's burden," Electronics, vol. 46, 15 February 1973.

4. D.E. Knuth, The Art of Computer Programming, vol. I. Reading, Mass.: Addison-Wesley, 1968.

5. G.A. Korn, Minicomputers for Scientists and Engineers. New York: McGraw-Hill, 1973.

THE FOLLOWING ARE AVAILABLE FROM THE FORTH INTEREST GROUP P.O. Box 1105 SAN CARLOS, CA 94070.:

Membership in FORTH Interest Group and Volume 2 (6 issues: #7 through #12) of FORTH DIMENSIONS.

fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map, and installation instruction.

Assembly language source listing of fig-FORTH for specific CPU's. The above manual is required for installation. Specify the desired CPU.

FORTH HANDY REFERENCE

Stack inputs and outputs are shown; top of stack on right.
This card follows usage of the Forth Interest Group (S.F. Bay Area); usage aligned with the *Forth 78* International Standard.
For more info: Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070.

Operand key:

n, n1, . . .		16-bit signed numbers
d, d1, . . .		32-bit signed numbers
u		16-bit unsigned number
addr		address
b		8-bit byte
c		7-bit ascii character value
f		boolean flag

STACK MANIPULATION

DUP	(n — n n)	Duplicate top of stack.
DROP	(n —)	Throw away top of stack.
SWAP	(n1 n2 — n2 n1)	Reverse 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 ?)	Duplicate only if non-zero.
>R	(n —)	Move top item to "return stack" for temporary storage (use caution).
R>	(— n)	Retrieve item from return stack.
R	(— n)	Copy top of return stack onto stack.

NUMBER BASES

DECIMAL	(—)	Set decimal base.
HEX	(—)	Set hexadecimal base.
BASE	(— addr)	System variable containing number base.

ARITHMETIC AND LOGICAL

+	(n1 n2 — sum)	Add.
D+	(d1 d2 — sum)	Add double-precision numbers.
-	(n1 n2 — diff)	Subtract (n1–n2).
*	(n1 n2 — prod)	Multiply.
/	(n1 n2 — quot)	Divide (n1/n2).
MOD	(n1 n2 — rem)	Modulo (*i.e.* remainder from division).
/MOD	(n1 n2 — rem quot)	Divide, giving remainder and quotient.
*/MOD	(n1 n2 n3 — rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate.
*/	(n1 n2 n3 — quot)	Like */MOD, but give quotient only.
MAX	(n1 n2 — max)	Maximum.
MIN	(n1 n2 — min)	Minimum.
ABS	(n — absolute)	Absolute value.
DABS	(d — absolute)	Absolute value of double-precision number.
MINUS	(n — -n)	Change sign.
DMINUS	(d — -d)	Change sign of double-precision number.
AND	(n1 n2 — and)	Logical AND (bitwise).
OR	(n1 n2 — or)	Logical OR (bitwise).
XOR	(n1 n2 — xor)	Logical exclusive OR (bitwise).

COMPARISON

<	(n1 n2 — f)	True if n1 less than n2.
>	(n1 n2 — f)	True if n1 greater than n2.
=	(n1 n2 — f)	True if top two numbers are equal.
0<	(n — f)	True if top number negative.
0=	(n — f)	True if top number zero (*i.e.*, reverses truth value).

MEMORY

@	(addr — n)	Replace word address by contents.
!	(n addr —)	Store second word at address on top.
C@	(addr — b)	Fetch one byte only.
C!	(b addr —)	Store one byte only.
?	(addr —)	Print contents of address.
+!	(n addr —)	Add second number on stack to contents of address on top.
CMOVE	(from to u —)	Move u bytes in memory.
FILL	(addr u b —)	Fill u bytes in memory with b, beginning at address.
ERASE	(addr u —)	Fill u bytes in memory with zeroes, beginning at address.
BLANKS	(addr u —)	Fill u bytes in memory with blanks, beginning at address.

CONTROL STRUCTURES

DO . . . LOOP	do: (end+1 start —)	Set up loop, given index range.
I	(— index)	Place current index value on stack.
LEAVE	(—)	Terminate loop at next LOOP or +LOOP.
DO . . . +LOOP	do: (end+1 start —) +loop: (n —)	Like DO . . . LOOP, but adds stack value (instead of always '1') to index.
IF . . . (true) . . . ENDIF	if: (f —)	If top of stack true (non-zero), execute. [Note: *Forth 78* uses IF . . . THEN.]
IF . . . (true) . . . ELSE . . . (false) . . . ENDIF	if: (f —)	Same, but if false, execute ELSE clause. [Note: *Forth 78* uses IF . . . ELSE . . . THEN.]
BEGIN . . . UNTIL	until: (f —)	Loop back to BEGIN until true at UNTIL. [Note: *Forth 78* uses BEGIN . . . END.]
BEGIN . . . WHILE . . . REPEAT	while: (f —)	Loop while true at WHILE; REPEAT loops unconditionally to BEGIN. [Note: *Forth 78* uses BEGIN . . . IF . . . AGAIN.]

TERMINAL INPUT-OUTPUT

.	(n —)	Print number.
.R	(n fieldwidth —)	Print number, right-justified in field.
D.	(d —)	Print double-precision number.
D.R	(d fieldwidth —)	Print double-precision number, right-justified in field.
CR	(—)	Do a carriage return.
SPACE	(—)	Type one space.
SPACES	(n —)	Type n spaces.
."	(—)	Print message (terminated by ").
DUMP	(addr u —)	Dump u words starting at address.
TYPE	(addr u —)	Type string of u characters starting at address.
COUNT	(addr — addr+1 u)	Change length-byte string to TYPE form.
?TERMINAL	(— f)	True if terminal break request present.
KEY	(— c)	Read key, put ascii value on stack.
EMIT	(c —)	Type ascii value from stack.
EXPECT	(addr n —)	Read n characters (or until carriage return) from input to address.
WORD	(c —)	Read one word from input stream, using given character (usually blank) as delimiter.

INPUT-OUTPUT FORMATTING

NUMBER	(addr — d)	Convert string at address to double-precision number.
<#	(—)	Start output string.
#	(d — d)	Convert next digit of double-precision number and add character to output string.
#S	(d — 0 0)	Convert all significant digits of double-precision number to output string.
SIGN	(n d — d)	Insert sign of n into output string.
#>	(d — addr u)	Terminate output string (ready for TYPE).
HOLD	(c —)	Insert ascii character into output string.

DISK HANDLING

LIST	(screen —)	List a disk screen.
LOAD	(screen —)	Load disk screen (compile or execute).
BLOCK	(block — addr)	Read disk block to memory address.
B/BUF	(— n)	System constant giving disk block size in bytes.
BLK	(— addr)	System variable containing current block number.
SCR	(— addr)	System 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

: xxx	(—)	Begin colon definition of xxx.
;	(—)	End colon definition.
VARIABLE xxx	(n —) xxx: (— addr)	Create a variable named xxx with initial value n; returns address when executed.
CONSTANT xxx	(n —) xxx: (— n)	Create a constant named xxx with value n; returns value when executed.
CODE xxx	(—)	Begin definition of assembly-language primitive operation named xxx.
;CODE	(—)	Used to create a new defining word, with execution-time "code routine" for this data type in assembly.
<BUILDS . . . DOES>	does: (— addr)	Used to create a new defining word, with execution-time routine for this data type in higher-level Forth.

VOCABULARIES

CONTEXT	(— addr)	Returns address of pointer to context vocabulary (searched first).
CURRENT	(— addr)	Returns address of pointer to current vocabulary (where new definitions are put).
FORTH	(—)	Main Forth vocabulary (execution of FORTH sets CONTEXT vocabulary).
EDITOR	(—)	Editor vocabulary; sets CONTEXT.
ASSEMBLER	(—)	Assembler vocabulary; sets CONTEXT.
DEFINITIONS	(—)	Sets CURRENT vocabulary to CONTEXT.
VOCABULARY xxx	(—)	Create new vocabulary named xxx.
VLIST	(—)	Print names of all words in CONTEXT vocabulary.

MISCELLANEOUS AND SYSTEM

(	(—)	Begin comment, terminated by right paren on same line; space after (.
FORGET xxx	(—)	Forget all definitions back to and including xxx.
ABORT	(—)	Error termination of operation.
' xxx	(— addr)	Find the address of xxx in the dictionary; if used in definition, compile address.
HERE	(— addr)	Returns address of next unused byte in the dictionary.
PAD	(— addr)	Returns address of scratch area (usually 68 bytes beyond HERE).
IN	(— addr)	System variable containing offset into input buffer; used, *e.g.*, by WORD.
SP@	(— addr)	Returns address of top stack item.
ALLOT	(n —)	Leave a gap of n bytes in the dictionary.
,	(n —)	Compile a number into the dictionary.

Screens

SCR # 21

```
( DEBUGGER AIDS -- DUMP , CDUMP )
BASE @ HEX
 02FE CONSTANT DSPFLG
: DSP.ON   0 DSPFLG ! ;
: DSP.OFF  1 DSPFLG ! ;
( USED BY "DUMP" )
: H. BASE @ HEX U. BASE ! ;
: B?   BASE @ DECIMAL . BASE ! ;
-->
```

SCR # 22

```
( DEBUGGER AIDS -- DUMP , CDUMP )
   DECIMAL
: ?EXIT  ?TERMINAL
       IF LEAVE ENDIF ;
: U.R    0 SWAP D.R ;
: LDMP  DUP 8 + SWAP DO I C@ 4 .R
       LOOP ;
: DUMP   OVER + SWAP DO CR I 5 U.R I
       LDMP ?EXIT 8 +LOOP CR ;
: CDMP   DUP 16 + SWAP DO
        I C@ EMIT LOOP ;
: CDUMP   OVER + SWAP DO CR I 5 U.R I
      SPACE DSP.OFF CDMP DSP.ON
      ?EXIT 16 +LOOP CR ;
-->
```

SCR # 23

```
( STACK PRINTER )
HEX
: DEPTH SP@ 12 +ORIGIN @ SWAP - 2 / ;
: S.  ( PRINTS THE STACK )
  DEPTH -DUP IF
     0 DO  CR  ." TOP+" I .
     SP@ I 2 * + @ U. LOOP
  ELSE ." Stack Empty" THEN CR ;
BASE !
-->
```

SCR # 24

```
( DEFINITION TRACER )
   BASE @ HEX
0 VARIABLE .WORD
' CLIT CFA CONSTANT .CLIT
' 0BRANCH CFA CONSTANT ZBRAN
' BRANCH CFA CONSTANT BRAN
' ;S CFA CONSTANT SEMIS
' (LOOP) CFA CONSTANT PLOOP
' (+LOOP) CFA CONSTANT PPLOOP
' (.") CFA CONSTANT PDOTQ
```

```
: PWORD 2+ NFA ID. ;
: 1BYTE  PWORD .WORD @ C@ . 1 .WORD +! ;
: 1WORD PWORD .WORD @ @  . 2 .WORD +! ;
: NP DUP SEMIS = IF PWORD CR CR
   PROMPT QUIT THEN ?TERMINAL IF
    PROMPT QUIT THEN ;    -->
```

SCR # 25

```
( DEFINITION TRACER )
: BRNCH PWORD ." to " .WORD @ .WORD @ @ + . 2 .WORD +! ;
: STG  PWORD 22 EMIT .WORD @   DUP COUNT TYPE 22 EMIT
  C@ .WORD @ + 1+ .WORD ! ;
' LIT CFA CONSTANT .LIT
: CKIT DUP ZBRAN = OVER BRAN =
 OR OVER PLOOP = OR OVER PPLOOP =
OR IF BRNCH ELSE DUP .LIT =
IF 1WORD ELSE DUP .CLIT =
 IF 1BYTE ELSE DUP PDOTQ = IF STG
 ELSE PWORD THEN THEN THEN THEN ;
-->
```

SCR # 26

```
( DEFINITION TRACER )
   ' : 12 + CONSTANT DOCOL
: T?PR CR CR ." Primitive" CR CR ;
: ?DOCOL DUP 2 - @ DOCOL -  IF
 T?PR PROMPT QUIT THEN ;
: SETUP [COMPILE] ' ?DOCOL .WORD ! ;
: NXT1  .WORD @ U. 2 SPACES .WORD
   @ @ 2 .WORD +! ;
: DECOMP  SETUP CR CR BEGIN NXT1 NP
     CKIT CR AGAIN ;
BASE !    ;S
```

SCR # 27

```
( ** EDITOR ** )
 BASE @  HEX
( THIS EDITOR IS PATTERNED AFTER
( THE EXAMPLE EDITOR IN THE fig
( "INSTALLATION MANUAL" 8/80 WFR
: TEXT HERE C/L 1+ BLANKS WORD
     HERE PAD C/L 1+ CMOVE ;
: LINE DUP FFF0 AND 17 ?ERROR SCR
     @ (LINE) DROP ;
: MARK  10 0 DO I LINE UPDATE
   DROP LOOP ;       -->
```

SCR # 28

```
( LINE EDITOR DEFS )
VOCABULARY EDITOR IMMEDIATE
: WHERE DUP B/SCR / DUP SCR ! ." SCR # " DECIMAL .
SWAP C/L /MOD C/L * ROT BLOCK + CR C/L -TRAILING TYPE CR HERE
```

```
- SPACES 5E EMIT [COMPILE] EDITOR QUIT ;

EDITOR DEFINITIONS
: #LOCATE R# @ C/L /MOD ;
: #LEAD #LOCATE LINE SWAP ;
: #LAG  #LEAD DUP >R + C/L R> - ;

: -MOVE LINE C/L CMOVE UPDATE ;

-->
```

SCR # 29

```
( LINE EDITING COMMANDS )
: H LINE PAD 1+ C/L DUP PAD C!
    CMOVE ;
: E LINE C/L BLANKS UPDATE ;
: S DUP 1 - 0E DO I LINE I 1+
   -MOVE -1 +LOOP E ;
: D DUP H 0F DUP ROT
   DO I 1+ LINE I -MOVE LOOP E ;

  -->
```

SCR # 30

```
( LINE EDITING COMMANDS )

: M   R# +! CR SPACE #LEAD TYPE
      17 EMIT #LAG TYPE #LOCATE
      . DROP ;
: T   DUP C/L * R# ! DUP H 0 M ;
: L   SCR @ LIST 0 M ;
: R   PAD 1+ SWAP -MOVE ;
: P   1 TEXT R ;
: I   DUP S R ;
: TOP   0 R# ! ;

 -->
```

SCR # 31

```
( SCREEN EDITOR COMMANDS )

: CLEAR  SCR ! 10 0 DO FORTH I
        EDITOR E LOOP ;

: COPY   B/SCR * OFFSET @ + SWAP
        B/SCR * B/SCR OVER +
        SWAP DO DUP FORTH I
        BLOCK 2 - ! 1+ UPDATE
        LOOP DROP FLUSH ;
 -->
```

SCR # 32

```
( STRING EDITING COMMANDS )
: 1LINE   #LAG PAD COUNT MATCH R#
        +! ;
: FIND   BEGIN 3FF R# @ < IF TOP
        PAD HERE C/L 1+ CMOVE 0
        ERROR ENDIF 1LINE UNTIL
        ;
: DELETE   >R #LAG + FORTH R -
       #LAG R MINUS R# +! #LEAD
       + SWAP CMOVE R> BLANKS
       UPDATE ;
-->
```

SCR # 33

```
( SCREEN EDITING COMMANDS )
:N   FIND 0 M ;
:F   1 TEXT N ;
:B   PAD C@ MINUS M ;
:X   1 TEXT FIND PAD C@ DELETE
     0 M ;
: TILL   #LEAD + 1 TEXT 1LINE 0=
        0 ?ERROR #LEAD + SWAP -
        DELETE 0 M ;
-->
```

SCR # 34

```
( SCREEN EDITING COMMANDS )
:C   1 TEXT PAD COUNT #LAG ROT
     OVER MIN >R FORTH R R# +!
     R - >R DUP HERE R CMOVE
     HERE #LEAD + R> CMOVE R>
     CMOVE UPDATE 0 M ;
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN !
HERE 28 +ORIGIN !
HERE 30 +ORIGIN !
' EDITOR 6 + 32 +ORIGIN !
HERE FENCE !     ;S
```

SCR # 35

SCR # 36

```
( DISK COPY ROUTINE  40K RAM )
( 40 K RAM AND DRIVES #1 AND #2 )
24576 CONSTANT BUFHEAD
0 VARIABLE BLK#  0 VARIABLE ADRS
: GET   ADRS @ BLK# @ ;
: RD    GET DUP 718 = IF LEAVE THEN 1 R/W ;
: WRT   GET 720 + DUP 1438 = IF LEAVE THEN  0 R/W ;
: +BLK  1 BLK# +!  128 ADRS +! ;
: SETUP   BLK# ! BUFHEAD ADRS ! ;
: RDIN   SETUP 90 0 DO RD +BLK
        LOOP ;
: WRTO   SETUP 90 0 DO WRT +BLK
        LOOP ;
-->
```

SCR # 37

```
( DISK COPY ROUTINE )
( INSERT SOURCE DISK IN DRIVE #1
( AND NEW DISK IN DRIVE #2. THEN,
( SIMPLY TYPE  "DISKCOPY" !
: MS1 CR CR
." INSERT SOURCE IN DRIVE #1 AND"
CR ." NEW DISK IN DRIVE #2" CR
."   HIT ANY KEY WHEN READY..."
KEY DROP ;
: %COPY     0 DO I 90 *
            DUP DUP RDIN WRTO
            90 +  . LOOP ;
-->
```

SCR # 38

```
( DISK COPY ROUTINE )
: DISKCOPY   CR MS1 CR 8 %COPY ;
: FORTHCOPY  CR MS1 CR 5 %COPY ;
;S
```

SCR # 39

```
( ** ASSEMBLER **   IN FORTH )
(  ASSEMBLER COMFORMS TO THE
( fig "INSTALLATION GUIDE" WITH
```

```
( THE FOLLOWING EXCEPTIONS:
(  SHIFTS ARE: "XXX.A" FOR A-REG.
( SHIFTS.
(  CONDITIONAL BRANCHES ARE
(  PATTERNED AFTER THE BRANCH OP-
(  CODES:   "IFEQ," IS USED IN-
( STEAD OF "0= IF," FOR BETTER
( CLARITY.  SEE SCREEN 43.
-->
```

SCR # 40

```
( ASSEMBLER VOCABULARY )
VOCABULARY ASSEMBLER IMMEDIATE
BASE @  HEX
: CODE  [COMPILE] ASSEMBLER
        CREATE SMUDGE ;
ASSEMBLER DEFINITIONS
: SB  <BUILDS C, DOES> @ C, ;
       ( SINGLE BYTE OPERATORS)
-->
```

SCR # 41

```
( SINGLE-BYTE OPERANDS )
00 SB BRK, 18 SB CLC, D8 SB CLD,
58 SB CLI, B8 SB CLV, CA SB DEX,
88 SB DEY, E8 SB INX, C8 SB INY,
EA SB NOP, 48 SB PHA, 08 SB PHP,
68 SB PLA, 28 SB PLP, 40 SB RTI,
60 SB RTS, 38 SB SEC, F8 SB SED,
78 SB SEI, A8 SB TAX, BA SB TSX,
8A SB TXA, 9A SB TXS, 98 SB TYA,
0A SB ASL.A,  2A SB ROL.A,
4A SB LSR.A,  6A SB ROR.A,
: NOT  0= ;  ( REVERSE LOGICAL )
: 0=  1 ; ( PUSH A TRUE )  -->
```

SCR # 42

```
( JMP, JSR, BRANCH CODES )
: 3BY  <BUILDS C, DOES> @ C, , ;
4C 3BY JMP,   6C 3BY JMP(),
20 3BY JSR,
: ?ER5    5 ?ERROR ;
: IF.  <BUILDS C, DOES> C@ C, 0
        C, HERE ;
: THEN,   DUP HERE SWAP - DUP
        7F > ?ER5 DUP -80 < ?ER5
        SWAP -1 + C! ;
: ENDIF,   THEN, ;
-->
```

SCR # 43

```
( CONDITIONAL BRANCH CODES )
10 IF, IFPL, ( BPL )
30 IF, IFMI, ( BMI )
50 IF, IFVC, ( BVC )
70 IF, IFVS, ( BVS )
90 IF, IFCC, ( BCC )
B0 IF, IFCS, ( BCS )
D0 IF, IFNE, ( BNE )
F0 IF, IFEQ, ( BEQ )
: BEGIN,   HERE ;
: END,   IF D0 ELSE F0 THEN  C,
         HERE 1+ - DUP
        -80 < ?ER5 C, ;
: UNTIL,  END,  ;      -->
```

SCR # 44

```
( MEMORY-REFERENCE INST. )
0D VARIABLE MODE ( ABS. MODE )
: MODE=   MODE @ = ; ( CK MODE )
: 256<   DUP 100 ( HEX) < ;
: MODEFIX  256< IF -08 MODE +!
          THEN ;
    ( MODE=MODE-8 IF ADR<256 )
: CKMODE   MODE= IF MODEFIX
              THEN ;
: M0 <BUILDS C, DOES> SWAP
    0D CKMODE  1D CKMODE SWAP
    C@ MODE @ OR C,    256< IF
    C, ELSE , THEN 0D MODE ! ;
-->
```

SCR # 45

```
( MEMORY REF. INST. )
: X)  01 MODE ! ;   ( [ADDR,X]  )
: #   09 MODE ! ;   ( IMMEDIATE )
: )Y  11 MODE ! ;   ( [ADDR],Y  )
: ,X  1D MODE ! ;   ( ADDR,X    )
: ,Y  19 MODE ! ;   ( ADDR,Y    )
00 M0 ORA, 20 M0 AND, 40 M0 EOR,
60 M0 ADC, 80 M0 STA, A0 M0 LDA,
C0 M0 CMP, E0 M0 SBC,
: BIT,   256< IF 24 C, C, ELSE
        2C C, , THEN ;
-->
```

SCR # 46

```
( MEMORY REF.  INC, CPX, ETC. )
: STOREADD  C, 256<  IF C, ELSE ,
          THEN 0D MODE ! ;
: ZPAGE     OVER 100 < IF F7 AND
          THEN  ;
: XYMODE  MODE @ 19 = MODE @ 1D
        = OR ;
: M1  <BUILDS C, DOES> C@ MODE @
```

```
      1D = IF 10 ELSE 0 THEN OR
      ZPAGE STOREADD ;

0E M1 ASL, 2E M1 ROL, 4E M1 LSR,
6E M1 ROR, CE M1 DEC, EE M1 INC,
-->
```

SCR # 47

```
( MEMORY REF. INST. )

: OPCODE  C@ ZPAGE XYMODE IF 10
        OR THEN ;
: M2  <BUILDS C, DOES> OPCODE
     MODE @ 9 = IF 4 - THEN
     STOREADD ;

AC M2 LDY,   AE M2 LDX,
CC M2 CPY,   EC M2 CPX,

: M3    <BUILDS C, DOES> OPCODE
        STOREADD ;

8C M3 STY,   8E M3 STX,
-->
```

SCR # 48

```
( END OF ASSEMBLER )

FORTH DEFINITIONS


LATEST 0C +ORIGIN !  ( NTOP )

HERE   1C +ORIGIN !  ( FENCE )

HERE   1E +ORIGIN !  ( DP )





BASE !     ;S
```

SCR # 49

SCR # 50

```
( COLOR COMMANDS )
BASE @ HEX
: SETCOLOR  2 * SWAP 10 * OR SWAP
        02C4 ( COLPF0 ) + C! ;
```

```
: SE. SETCOLOR ; ( ALIAS )
( REGISTER#-3, COLOR-2, LUM-1
(   0-3        0-F    0-7
-->
```

SCR # 51

```
( GRAPHICS COMMANDS )
E456 CONSTANT CIO
  1C VARIABLE MASK
 340 CONSTANT IOCB
  53 VARIABLE SNAME

CODE GR. 1 # LDA, GFLAG STA,
         XSAVE STX,   0 ,X LDA,
 # 30 LDX,     IOCB 0B + ,X STA,
# 3 LDA,  IOCB 2 + ,X STA,
SNAME FF AND # LDA,  IOCB 4 + ,X
 STA,   SNAME 100 / # LDA,
IOCB 5 + ,X STA,   MASK LDA,
IOCB 0A + ,X STA,    CIO JSR,
XSAVE LDX,   0 # LDY,  POP JMP,
     -->
```

SCR # 52

```
( GRAPHICS COMMANDS )

CODE &GR     XSAVE STX, # 30 LDX,
             # C LDA,  IOCB 2 +
             ,X STA,   CIO JSR,
             XSAVE LDX, 0 # LDA,
             GFLAG STA, NEXT JMP,

: XGR  &GR  0 GR  &GR ;
  ( EXIT GRAPHICS MODE )

-->
```

SCR # 53

```
( GRAPHICS I/O )

CODE CPUT   0 ,X LDA,  PHA,
  XSAVE STX,  # 30 LDX,
  # B LDA,  IOCB 2 + ,X STA, TYA,
  IOCB 8 + ,X STA, IOCB 9 + ,X
  STA, PLA,  CIO JSR,  XSAVE LDX,
  POP JMP,

54 CONSTANT ROWCRS
55 CONSTANT COLCRS

: POS  ROWCRS C! COLCRS ! ;
: PLOT   POS CPUT ;

-->
```

SCR # 54

```
( GRAPHICS I/O )
: GTYPE  -DUP IF OVER + SWAP
      DO I C@ CPUT LOOP ELSE
      DROP ENDIF ;
: (G")  R COUNT DUP 1+ R> + >R
      GTYPE ;
: G"  22 STATE @ IF COMPILE (G")
     WORD HERE C@ 1+ ALLOT
    ELSE WORD HERE COUNT GTYPE
    ENDIF  ; IMMEDIATE
-->
```

SCR # 55

```
( DRAW, FIL )
2FB CONSTANT ATACHR
2FD CONSTANT FILDAT
CODE GCOM    XSAVE STX, 0 ,X LDA,
  # 30 LDX,  IOCB 2 + ,X STA,
  CIO JSR,  XSAVE LDX, POP JMP,
: DRAW   POS ATACHR C! 11 GCOM ;
: FIL  FILDAT C!  12 GCOM ;
BASE !   ;S
```

SCR # 56

```
( GRAPHICS TESTS )
: BOX  0 10 10 PLOT  1 50 10 DRAW
     1 50 25 DRAW  1 10 25 DRAW
     1 10 10 DRAW ;
: FBOX  XGR   5 GR.   BOX
    10 25 POS  2 FIL ;
```

SCR # 57

```
SCR # 58
```

```
SCR # 59
```

```
SCR # 60
( FLOATING POINT WORDS )
 BASE @  DECIMAL
: FDROP   DROP DROP DROP ;
: FDUP    >R >R DUP R> DUP ROT
          SWAP R ROT ROT R> ;
CODE FSWAP
  XSAVE STX, # 6 LDY,
BEGIN,  0 ,X LDA, PHA, INX, DEY,
0= END, XSAVE LDX,  # 6 LDY,
BEGIN, 6 ,X LDA,  0 ,X STA, INX,
DEY, 0= END, XSAVE LDX,  # 6 LDY,
 BEGIN, PLA, 11 ,X STA, DEX, DEY,
0= END,   XSAVE LDX,  NEXT JMP,
HEX
XSAVE 100 * 86 + CONSTANT XSAV
: XS,   XSAV , ;      -->
```

```
SCR # 61
( FLOATING POINT WORDS )
CODE FOVER     DEX, DEX, DEX,
 DEX, DEX, DEX, XSAVE STX,
 # 6 LDY,  BEGIN,  12 ,X LDA,
```

```
0 ,X STA, INX, DEY, 0= END,
XSAVE LDX,  NEXT JMP,
XSAVE 100 * A6 + CONSTANT XLD
: XL,   XLD  ,  ;
CODE AFP   XS, D800 JSR, XL, NEXT JMP,
CODE FASC  XS, D8E6 JSR, XL, NEXT JMP,
CODE IFP   XS, D9AA JSR, XL, NEXT JMP,    -->
```

SCR # 62

```
( FLOATING POINT WORDS )
CODE FPI   XS, D9D2 JSR, XL, NEXT JMP,
CODE FADD  XS, DA66 JSR, XL, NEXT JMP,
CODE FSUB  XS, DA60 JSR, XL, NEXT JMP,
CODE FMUL  XS, DADB JSR, XL, NEXT JMP,
CODE FDIV  XS, DB28 JSR, XL, NEXT JMP,
CODE FLG   XS, DECD JSR, XL, NEXT JMP,
CODE FLG10 XS, DED1 JSR, XL, NEXT JMP,
CODE FEX   XS, DDC0 JSR, XL, NEXT JMP,
CODE FEX10 XS, DDCC JSR, XL, NEXT JMP,
CODE FPOLY XS, DD40 JSR, XL, NEXT JMP,
 -->
```

SCR # 63

```
( FLOATING POINT WORDS )
D4 CONSTANT FR0
E0 CONSTANT FR1
FC CONSTANT FLPTR
F3 CONSTANT INBUF
F2 CONSTANT CIX
 DECIMAL
-->
```

SCR # 64

```
( FLOATING POINT )
: F@ >R R @ R 2+ @ R> 4 + @ ;
: F! >R R 4 + ! R 2+ ! R> ! ;
  HEX
: F.TY  BEGIN INBUF @ C@ DUP
      7F AND EMIT 1 INBUF +!
      80 > UNTIL ;
  DECIMAL
: F. FR0 F@ FSWAP FR0 F! FASC
   F.TY SPACE FR0 F! ;
: F?   F@ F. ;
-->
```

SCR # 65

```
( FLOATING POINT )
: <F   FR1 F! FR0 F! ;
: F>  FR0 F@ ;
: FS  FR0 F! ;
: F+   <F FADD F> ;
: F-   <F FSUB F> ;
: F*   <F FMUL F> ;
: F/   <F FDIV F> ;
: FLOAT   FR0 ! IFP F> ;
: FIX   FS FPI FR0 @ ;
: FLOG   FS FLG F> ;
: FLOG10 FS FLG10 F> ;
: FEXP  FS FEX F> ;
: FEXP10 FS FEX10 F> ;  -->
```

SCR # 66

```
( FLOATING POINT )
  HEX
: ASCF 0 CIX ! INBUF ! AFP F> ;
: FLIT R> DUP 6 + >R F@ ;
: FLITERAL STATE @ IF
 COMPILE FLIT HERE F! 6 ALLOT
 ENDIF ;
: FLOATING  ( FLOAT FOLLOWING CONSTANT )
  BL WORD HERE 1+ ASCF
  FLITERAL ;  IMMEDIATE
( EX:  FLOATING 1.2345 )
( OR   FLOATING  -1.67E-13 )
: FP [COMPILE] FLOATING ;
IMMEDIATE     -->
```

SCR # 67

```
( FLOATING POINT )
     HEX
: FVARIABLE
 <BUILDS HERE F! 6 ALLOT DOES> ;
: FCONSTANT
  <BUILDS HERE F! 6 ALLOT DOES>
 F@ ;
: F0=   OR OR 0= ;
: F=    F- F0= ;
: F<    F- DROP DROP 80 AND 0 > ;
BASE !  ;S
```