PRIMER

E-FORTH SBC+

USER'S GUIDE

REVISION 3
DATE : January 6, 1993

EQUIPMENT MONITOR AND CONTROL
CARBONDALE, IL 62901
618-529-4525

TABLE OF CONTENTS

DISCLAIMER

EMAC has made every attempt to ensure that the information in this document is accurate and complete. However, EMAC assumes no liability for any damages that result from use of this manual or the equipment that it documents. EMAC reserves the right to make changes at any time.

INTRODUCTION

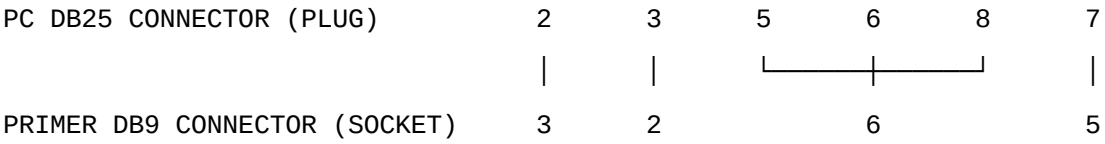
E-FORTH SBC+ creates an optimum environment for software development. The E-FORTH SBC+ Operating System contains an Editor, Assembler, and Target Compiler, all within quick and easy reach of the user. This comprehensive approach to software development provides the user with everything needed to expedite the design and implementation of an application.

Chapter 1

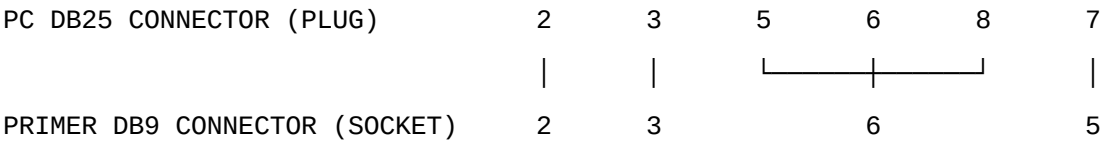
GETTING STARTED

E-FORTH serial communication does not require any form of "handshaking" to operate, although COM1 makes available the handshaking lines DSR and DTR. It is left up to the user to provide drivers in order to make use of these lines. E-FORTH does not support XON/XOFF protocol. Although "Handshaking" lines are not required by E-FORTH, they may be necessary for the IBM PC and compatibles the PRIMER is communicating with. To assure proper handshaking when using E-FORTH, tie RS-232 handshake lines CTS, DSR, and DCD (pins 5, 6, and 8 on the DB25 connector) to DTR (pin 6) of the DB9 PRIMER connector (socket). An alternate method involves wiring a null modem cable. This is easily accomplished by tying CTS, DSR, and DCD to DTR, pin 20 of the DB25 connector that plugs into the PC. One of the above methods should achieve success.

The Receive (RxD pin 3) and Transmit (TxD pin 2) lines also require modification prior to being connected. See diagram below.



Some terminals require pins 2 and 3 of the DB25 go to pins 2 and 3 of the DB9 respectively:



NOTE: Since the RS232 voltages (+9VDc and the -9VDc) are generated on board, a single rail power supply (7VDc - 12Vdc) is all that is needed.

E-FORTH utilizes port COM1 for standard default communications and its serial protocol is: no parity, one stop bit, and 8 data bits. Before the PRIMER can communicate through the serial port, its baud rate must be the same as the PC or terminal the PRIMER is communicating with. The PRIMER's baud rate can be set by placing a jumper in JP1 in the position corresponding to the desired baud rate. The baud rates are labeled 300, 600, 1200, 2400, 4800, 9600 and 19,200 next to JP1.

Now that the PRIMER has been connected to the terminal device and the baud rate is correct, you are ready to apply power. The PRIMER requires a power supply in the range of 7 to 10 volts DC that can supply more than 480 milliamps of current. This power may be taken from a bench power supply, a wall mounted power supply or any other suitable power source. The power supply's output plug tip must be positive and the sleeve must be negative. A wall mounted power supply that meets all of the previous stated requirements may be obtained from EMAC.

Once power has been correctly applied to the PRIMER's power jack, the E-FORTH logon message should appear on the terminal display. If this doesn't occur, remove the power and refer to the troubleshooting section of this manual.

Chapter 2

NEWCOMERS TUTORIAL

SO YOU HAVE POWERED UP YOUR PRIMER AND YOU'RE READY TO START - WELL, WHAT ARE YOU WAITING FOR.....

To begin, press the <return> key a couple of times till the prompt "D0>" appears. The D stands for decimal signifying that any number entered into the system is a decimal number (base 10). Now type the word "HEX" using capital letters (all the words in E-FORTH's dictionary are spelled with capitals) and press <return>. The prompt now has changed from "D0>" to "H0>" signifying we are in Hexadecimal mode or base 16.

By now your probably wondering what the 0 means, this is a very important aspect of FORTH which relates to the FORTH parameter stack. In FORTH a program is generally made up of a number of smaller programs called words. In order for us to pass information from one word to another a stack is used. A stack is analogous to a stack of trays in a cafeteria. You pull a tray off the top of the stack or you can push a tray onto the top of the stack. Simply put, a FORTH stack is a FIRST IN, LAST OUT type of structure which holds numbers instead of trays.

Try this, type "1" <return> and notice that the "H0>" prompt now reads "H1>" . You guessed it, the number to the right of the "H" tells you how many numbers are on the stack. Type "2" <space> "3" <return> (Forth uses spaces to separate words and numbers), the stack now contains three numbers. Type "." <return> to remove the top number off the stack and output it to the display. Now notice the prompt, that's right almost all Forth words that accept a value off the stack, removes that value and thereby decreases the number of values on the stack. Type "+" <space> "." <return> which adds 1 and 2 together and prints their sum.

Moving onto bigger and better things lets define a FORTH word to count from 1 to 20 in decimal and output the count to the display. First we need to be in Decimal Mode. Type "DECIMAL" <return>, next the defining word ":" is needed, followed by the name of the word.

Lets call it SAMPLE and so.....

: SAMPLE 21 1 DO I . LOOP ; <Return>

The name is followed by the bounds of the loop, first 21 is put on the stack then 1 is put on the stack and 'DO' is called. DO pulls the 1 off the stack and then pulls the 21 off the stack. If the numbers seem to be backwards it is due to the nature of the stack, but as you can see when DO pulls the numbers off the stack they are in the correct order. The "I" that follows the DO puts the current index number on the stack which is then pulled off the stack and displayed by ".". The word "LOOP" marks the end of the loop and is followed by ";" which terminates the word.

Now let's execute SAMPLE by typing "SAMPLE" followed by <return>. Of course it works. Now our new word, SAMPLE can be used in any other word we decide to create.

To keep track of all the words at our disposal, another FORTH word is used called VLIST. When VLIST is executed, a list of all the words you have created are listed, followed by the FORTH words available to you within the system. Type "VLIST" <return> and hit any key to stop the display. The word "SAMPLE" is the first word listed. To delete "SAMPLE" from our VLIST type "FORGET" <space> "SAMPLE" <return> which will forget SAMPLE and any other user defined words after "SAMPLE". But you wanted to keep "SAMPLE". Sorry, it's gone forever. If you wanted to keep SAMPLE you should have used the EDITOR.

The EDITOR allows you to store your programs on RAMDISK, where they can subsequently be reviewed and modified at any time. To use the EDITOR we first type the word "EDITOR" <return>. Next we need a clean screen on which to store our program (in FORTH the Disk is divided into 1K sections called screens). Lets try screen number 5. To clean, or as we say in FORTH "CLEAR", screen 5 we type "5" <space> "CLEAR" <return>. CLEARing a screen erases anything that was present on that particular screen. To verify that the screen is now clear type "5" <space> "LIST" <return> which displays the contents of the screen and also makes screen 5 the current screen. Now let's reenter our SAMPLE word by typing:

2 P : SAMPLE 21 1 DO I . LOOP ; <return>

The 2 denotes the line number and the P is an EDITOR word meaning Put. To save this screen to RAMDISK type "L" <return> which will display the current screen and stores it on the disk. Remember all EDITOR words (ie. CLEAR, P, L, etc.) can only be used from within the EDITOR.

To execute SAMPLE after editing, we first have to compile it. This is accomplished by typing "5" <space> "LOAD" <return>, SAMPLE is now safely back on top of our VLIST and can be executed simply by typing its name. The act of LOADing a screen automatically removes you from the EDITOR or any other VOCABULARY and places you in the FORTH VOCABULARY.

I hope that I was of some help in getting you to the ground floor of E-FORTH. Study the other FORTH words available and with a little practice you will be a FORTH wizard in no time. I know I haven't begun to answer all your questions, so I'm going to recommend a book that helped me get started called "STARTING FORTH" by Leo Brodie. So long, and may the FORTH be with you.

Chapter 3

FORTH VOCABULARY

KERNEL NOTATION

In the following description of E-FORTH KERNEL words, you will see some notation which may or may not be familiar. The concept of a STACK is integral to FORTH as well as to the creation of words in FORTH. This being the case, notation was devised to accurately depict the contents of the STACK both before as well as after a word's execution. Let's look at the notation used to describe the FORTH word " * "

* (n1 n2 --- n3)

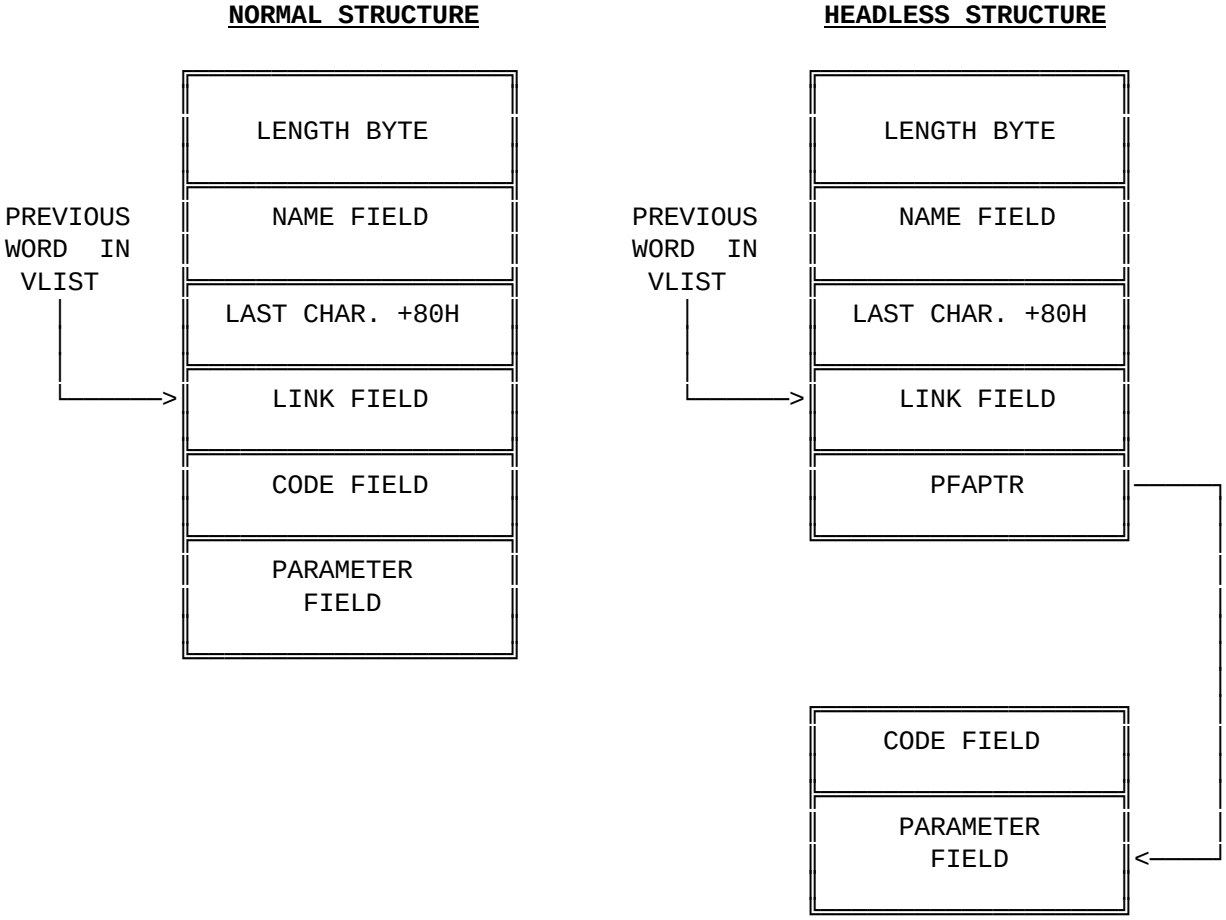
which multiplies together the top two numbers on the stack and upon completion of the multiplication leaves their product on the stack. The notation means that " * " REQUIRES two numbers on the stack prior to execution and WILL LEAVE one number, the product, in their place. The dashes represent the execution of " * ". While in a multiplication operation the order of the operands is not crucial, the notation INDICATES STACK ORDER. On either side of the dashes, reading left to right is actually reading the contents of the STACK from bottom to top.

	BEFORE		**AFTER**
TOP	n2	TOP	n3
	n1		.
	.		.
	.		.
BOTTOM .		BOTTOM .	

ABBREVIATIONS WITHIN A WORD'S DESCRIPTION

addr	-------- address
b	-------- byte
c	-------- character (printable byte)
cfa	-------- code field address
d	-------- 32-bit " double-word " value
f	-------- flag (0 = false , ^0 = true)
ff	-------- false flag
lfa	-------- link field address
n	-------- 16-bit " word " value
nfa	-------- name field address
pfa	-------- parameter field address
pfaptr	-------- pointer to actual pfa (see note on next page)
tf	-------- true flag
ud	-------- unsigned (32-bit) double-word
un	-------- unsigned (16-bit) word

NOTE: Unlike most FORTH systems, E-FORTH possesses a method of generating headless compiled code, i.e. the user is able to, on demand, compile the in-line dictionary header information (called HEADs) in a separate area of memory than the code portion, (called TAILs). To accomplish this end, a different structure to describe a FORTH word was used.



THE USE OF THE POINTER PFAPTR ENABLES US TO SEPARATE THE HEADS FROM THE TAILS AND STORE THEM IN DIFFERENT LOCATIONS BOTH IN THE E-FORTH KERNEL AS WELL AS WHEN NEW DEFINITIONS ARE COMPILED.

KERNEL QUICK REFERENCE

The E-FORTH kernel words listed below are grouped according to their relation to the PRIMER hardware. For details on these words refer to the associated page numbers in the language glossary provided.

MASS STORAGE

VECTORING

INTERRUPTS

COMMUNICATIONS

INPUT/OUTPUT

REAL TIME CLOCK

OVERLAYS

HEADLESS

CASE STRUCTURE

ERROR MESSAGES

" * " DENOTES WORD HAS MORE THAN ONE MODE OF OPERATION.

LANGUAGE GLOSSARY

**** NOTE **** We have listed below all of the E-FORTH KERNEL words including the 79-STANDARD (denoted by 7S), some words from the 79-STANDARD EXTENDED WORD SET (denoted by 7E) and some words from the 79-STANDARD REFERENCE WORD SET (denoted by 7R).

(---)
" null ". Execution code sequence terminating interpretation of terminal input or a line text within one of the disk buffers.

! (n addr ---)
store value n at addr (7S)

(ud1 --- ud2)
generate from an unsigned double-number ud1, the next ASCII character which is placed in an output string. ud2 is the quotient after division b BASE and is maintained for further processing. Used between FORTH words <# and #>. (7S)

#> (ud --- addr n)
end pictured numeric output conversion. DROP ud leaving the text addr, and character count, in preparation for a possible TYPE. (7S)

#S (ud --- 0 0)
convert all digits of an unsigned 32-bit number ud, adding each to the pictured numeric output text, until remainder is zero. A single zero is added to the output string if the number was initially zero. Use only between FORTH words <# and #>. (7S)

' <name> (--- addr)
if executing, leave the parameter field addr of the next word accepted from the input stream. If compiling, compile this addr as a literal; later execution will place this value on the stack. An error condition exists if <name> is not found after a search of the CONTEXT and FORTH vocabularies. Within a colon-definition ' <name> is identical to the FORTH sequence [' <name>] LITERAL. (7S)

'?TERMINAL (--- addr)
leave the addr of the USER variable which contains the compilation address of the current code sequence to be executed when a ?TERMINAL instruction occurs. " addr " is the address of the ?TERMINAL VECTOR.

'EMIT (--- addr)
leave the addr of the USER variable which contains the compilation address of the current code sequence to be executed when an EMIT instruction occurs. " addr " is the address of the EMIT VECTOR.

'KEY (--- addr)
leave the addr of the USER variable which contains the compilation address of the current code sequence to be executed when a KEY instruction occurs. " addr " is the address of the KEY VECTOR.

'R/W (--- addr)
leave the addr of the USER variable which contains the compilation address of the current code sequence to be executed when a R/W instruction occurs. " addr " is the address of the R/W VECTOR.

((---)
accept and ignore comment characters from the input stream, until the next right parenthesis. As a word, the left parenthesis must be followed by one blank. It may be freely used while executing or

compiling. An error condition exists if the input stream is exhausted before the right parenthesis. (7S)

***** (n1 n2 --- n3)
leave the arithmetic product of n1 times n2. (7S)

***/** (n1 n2 n3 --- n4)
multiply n1 by n2, divide the result by n3 and leave the quotient n4. n4 is rounded toward zero. The product of n1 times n2 is maintained as an intermediate 32-bit value for greater precision than the otherwise equivalent sequence: n1 n2 * n3 / (7S)

***/MOD** (n1 n2 n3 --- n4 n5)
multiply n1 by n2, divide the result by n3 and leave the remainder n4 and quotient n5. A 32-bit intermediate product is used as for */. The remainder has the same sign as n1. (7S)

+ (n1 n2 --- n3)
leave the arithmetic sum of n1 plus n2. (7S)

+! (n addr ---)
using the convention for + , add n to the 16-bit value at the addr. (7S)

+- (n1 n2 --- n3)
assign the sign of n2 to the word n1 leaving the word n3.

+LOOP (n ---)
add the signed increment n to the loop index using the convention for +, and compare the total to the limit. Return execution to the corresponding DO until the new index is equal to or greater than the limit (when n > 0), or until the new index is less than the limit (when n < 0). Upon exit from the loop, discard the loop control parameters, continuing execution ahead. Index and limit are signed integers in the range {-32,768...32,767}. (7S)

, (n ---)
allot two bytes in the dictionary, storing n there and increment the dictionary pointer (DP). (7S)

- (n1 n2 --- n3)
subtract n2 from n1 leaving difference n3.(7S)

--> (---)
continue interpretation on the next sequential block. May be used within a colon-definition that crosses a block boundary. (7R)

-TRAILING (addr n1 --- addr n2)
adjust the character count n1 of a text string beginning at addr to exclude trailing blanks, i.e. the characters at addr+n1 - 1 to addr+n2 are blanks. An error exists if n1 is negative. (7S)

. (n ---)
display n converted according to BASE in a free-field format with one trailing blank. Display only a negative sign. (7S)

." (---)
accept the following text from the input stream, terminated by **"** (double quote). If executing, transmit this text to the selected output device. If compiling, compile so that later execution will transmit the text to the selected output device. If the input stream is exhausted before the terminating double-quote, an

error condition exists. (7S)

.LINE (n1 n2 ---)
EMIT, through the current output device, line n1 of screen n2 without trailing blanks.

.S (---)
a non-destructive display, through the current output device, of the contents of the data stack.

.R (n1 n2 ---)
print n1, right aligned, in a field of n2 characters according to BASE. If n2 is less than 1, no leading blanks are supplied. (7R)

/ (n1 n2 --- n3)
divide n1 by n2 and leave the quotient n3. n3 is rounded toward zero. (7S)

/LOOP (n ---)
a DO - LOOP terminating word. The loop index is incremented by the unsigned magnitude of n until the resultant index exceeds the limit. Execution returns to just after the corresponding DO, otherwise, the index and limit are discarded. (7R)

/MOD (n1 n2 --- n3 n4)
divide n1 by n2 and leave the remainder n3 and quotient n4. n3 has the same sign as n1. (7S)

0 (--- 0)
leave the constant 0 on the data stack.

0< (n --- flag)
leave a true flag if n is less than zero (i.e. negative) false flag otherwise. (7S)

0= (n --- flag)
leave a true flag if n is zero. (7S)

0> (n --- flag)
leave a true flag if n is greater than zero (i.e. positive) false flag otherwise. (7S)

1 (--- 1)
leave the constant 1 on the data stack.

1+ (n --- n+1)
increment n by one, according to the operation for +. (7S)

1- (n --- n-1)
decrement n by one, according to the operation for -. (7S)

2 (--- 2)
leave the constant 2 on the data stack.

2! (d addr ---)
store d in 4 consecutive bytes beginning at addr, as for a double number. (7E)

2* (n1 --- n2)
leave n2 which is 2 times n1. (7R)

2+ (n --- n+2)
increment n by two, according to the operation for +. (7S)

2- (n --- n-2)
decrement n by two, according to the operation for -. (7S)

2/ (n1 --- n2)
leave (n1) / 2. (7R)

2@ (addr --- d)
leave on the stack the contents of the four consecutive bytes beginning at addr, as for a double number. (7E)

2CONSTANT (d ---)
a defining word used to create a dictionary entry for <name>, leaving d in its parameter field. When <name> is later executed, d will be left on the stack. (7E)

2DROP (d ---)
DROP the top double number or top two 16-bit numbers from the stack. (7E)

2DUP (d --- d d)
DUP the top double number or top two 16-bit numbers found on the stack. (7E)

2VARIABLE (---)
a defining word used to create a dictionary entry of <name> and allocates 4 bytes of storage for <name>. When <name> is executed, it will leave the address of the first of the 4 bytes allocated. (7E)

3 (--- 3)
leave the constant 3 on the data stack.

4 (--- 4)
leave the constant 4 on the data stack.

79-STANDARD (---)
assures that a FORTH-79 Standard system is available, otherwise an error condition exists. (7S)

: <name> (---)
select the CONTEXT vocabulary to be identical to CURRENT. Create a dictionary entry for <name> in CURRENT, and set compile mode. Words thus defined are called 'colon - definitions'. The compilation addresses of subsequent words from the input stream which are not immediate words are stored into the dictionary to be executed when <name> is later executed. IMMEDIATE words are executed as encountered. If a word is not found after a search of the CONTEXT and FORTH vocabularies, conversion and compilation of a literal number is attempted, with regard to the current BASE; that failing, an error condition exists. (7S)

; (---)
terminate a colon-definition and stop compilation. If compiling from mass storage and the input stream is exhausted before encountering **;** an error condition exists. (7S)

;CODE <name> (---)
stop compilation and terminate a defining word <name>. ASSEMBLER becomes the context vocabulary. (7E)

< (n1 n2 --- flag)
leave a true flag if n1 is less than n2. (7S)

<# (---)
initialize pictured numeric output. The words: <# # #S HOLD SIGN #> can be used to specify the conversion of a double-precision number into an ASCII character string stored in right-to-left order. (7S)

<= (n1 n2 --- f)
leave a flag based on the comparison of n1 <= n2.

<?TERMINAL> (--- 0)
code compiled by an unvectored ?TERMINAL, which at run-time, tests the default communications port for an incoming character returning either a 1 (true flag) or a 0 (false flag)

<BUILDS (---)
used in conjunction with DOES> in defining words, in the form: : <name> <BUILDS...DOES>...; and then <name> <namex>. When <name> executes, <BUILDS creates a dictionary entry for the new <namex>. The sequence of words between <BUILDS and DOES> establishes a parameter field for <namex>. When <namex> is later executed, the sequence of words following DOES> will be executed, with the parameter field address of <namex> on the data stack. (7R)

<EMIT> (c ---)
code compiled by an unvectored EMIT, which at run-time transmits a character c out the COM1 (default) communication port.

<KEY> (--- c)
code compiled by an unvectored KEY which at run-time leaves the next available character c from the COM1 (default) communication port.

<R/W> (addr n f ---)
code compiled by an unvectored R/W which at run-time performs a block read-write. "addr" specifies the source or destination memory address. "n" is the sequential number of the referenced block. "f" is the flag which indicates read (non-zero) or write (zero). "<R/W>" determines the physical mass storage location, performs the read-write and checks for errors.

> (n1 n2 --- flag)
true if n1 is greater than n2. (7S)

>= (n1 n2 --- f)
leave a flag based on the comparison of n1 >= n2.

>IN (--- addr)
leave the addr of the USER variable which contains the present character offset within the input stream {0..1023}. (7S)

>R (n ---)
transfer n to the return stack. Every ">R" must be balanced by a "R>" in the same control structure nesting level of a colon-definition. (7S)

? (addr ---)
display the number at addr, using the format of the FORTH word **"."** (dot). (7S)

?DIP (--- b)
read the current value indicated by the DIP SWITCH settings.

?DUP (n --- n n) else (n --- n)

duplicate n if it is non-zero. (7S)

@ (addr --- n)
leave on the stack the 16-bit number n contained at addr. (7S)

ABORT (---)
clear the data and return stacks, setting execution mode. Return control to the terminal. (7S)

ABS (n1 --- n2)
leave the absolute value n2 of number n1. (7S)

ADCIN (n1 --- n2)
leave the current value n2 of analog channel n1. For faster conversion use ASSEMBLER word A/D.

AGAIN (---)
affect an unconditional jump back to the start of a BEGIN-AGAIN loop. (7R)

ALLOT (n ---)
add n bytes to the Variable Pointer (VP) when in HEADLESS mode. If not in HEADLESS mode ALLOT adds n bytes to (DP). (7S)

ALLOT/ (n ---)
add n bytes to HEADER dictionary pointer (DP/) when in HEADLESS mode. If not in HEADLESS mode ALLOT/ adds n bytes to (DP).
**** NOTE **** FOR ADDITIONAL INFORMATION CONSULT THE DESCRIPTION ON THE CONSTRUCTION OF "HEADLESS" CODE FOUND IN THIS MANUAL.

AND (n1 n2 --- n3)
leave the bitwise logical "AND" of n1 and n2. (7S)

ASSEMBLER (---)
select assembler as the CONTEXT vocabulary. (7E)

B/BUF (--- 256)
a constant leaving 256, the number of bytes per block buffer. (7R)

BASE (--- addr)
leave the addr of the USER variable which contains the current input-output numeric conversion base {2..70}. (7S)

BEGIN (---)
BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until flag is true. A BEGIN-WHILE-REPEAT loop will be repeated until flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. Flag is always DROPPED after being tested. (7S)

BELL (---)
activate a terminal bell appropriate to the device in use. (7R)

BINARY (---)
sets the numeric conversion base to BINARY (base 2) for subsequent input / output.

BL (--- 32)
leaves the constant for an ASCII space.

BLK (--- addr)
leave the addr of the USER variable which contains the number of the mass storage block being interpreted as the input stream. If addr contains a zero, the input stream is taken from the terminal. (7S)

BLOCK (n --- addr)
leave the addr of the first byte in block n. If the block is not already in memory, it is transferred from mass storage into whichever memory buffer has been least recently accessed. If the block occupying that buffer has been UPDATED (i.e. modified), it is rewritten into mass storage before block n is read into the buffer. n is an unsigned number. If correct mass storage read or write is not possible, an error condition exists. Only data within the latest block referenced by BLOCK is valid by byte addr, due to sharing of the block buffers. (7S)

BUFFER (n --- addr)
obtain the next block buffer, assigning it to block n. The block is not read from mass storage. If the previous contents of the buffer has been marked as UPDATED, it is written to mass storage. If correct writing to mass storage is not possible, an error condition exists. The addr left is the first byte within the buffer for data storage. n is an unsigned number. (7S)

C, (n ---)
store the low-order 8 bits of n at the next available byte in the dictionary and advance the dictionary pointer (DP). (7R)

C! (n addr ---)
store the least significant 8-bits of n at addr. (7S)

C/L (--- 64)
leave the constant 64, indicating characters per line, on the data stack

C@ (addr --- byte)
leave on the stack the contents of the byte at addr in a 16-bit field with the 8 high-order bits being zeroes. (7S)

CASE (---)
beginning syntax for a FORTH case statement where the case statement is of the form . . .

```
CASE
n1 OF= . . . ENDOF  ( any of these three    )
n2 OF> . . . ENDOF  ( types of conditionals )
n3 OF< . . . ENDOF  ( may be used within a  )
default             ( CASE statement        )
ENDCASE
```

**** NOTE **** FOR MORE INFORMATION SEE THE SECTION ON THE CASE STATEMENT IN THIS MANUAL.

CFA (pfa --- cfa)
convert the parameter field address (pfa) of a FORTH word to its code field address (cfa).

CLKREG (--- addr)
leave the constant base addr of the REAL TIME CLOCK data registers.
**** NOTE **** FOR MORE INFORMATION SEE APPENDIX C

CMOVE (addr1 addr2 n ---)
move n bytes beginning at addr1 to addr2. The contents of addr1 is moved first proceeding toward

high memory. If n is zero or negative nothing is moved. (7S)

CODE <name> (---)
creates a dictionary entry for <name> to be defined by a following sequence of assembly language words. ASSEMBLER becomes the context vocabulary. (7E)

COLD (---)
performs a cold start including the following :

1. reset of the dictionary pointer (DP) to the first available byte after the KERNEL. This effectively un-links all compiled code including APPLICATION programs.
2. other general system initialization.
3. execution of the FORTH word WARM.

COMPILE (---)
when a word containing COMPILE executes, the 16-bit value following the compilation address of COMPILE is copied (compiled) into the dictionary. i.e., COMPILE DUP will copy the compilation address of DUP. COMPILE [0 ,] will copy zero. (7S)

CONSTANT <name> (n ---)
a defining word used in the form: n CONSTANT <name> to create a dictionary entry for <name>, leaving n in its parameter field. When <name> is later executed, n will be left on the stack. (7S)

CONTEXT (--- addr)
leave the addr of the USER variable which contains vocabulary in which dictionary searches are to be made, during interpretation of the input stream. (7S)

CONVERT (d1 addr1 --- d2 addr2)
convert to the equivalent stack number the 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 non-convertible character. (7S)

COUNT (addr --- addr+1 n)
leave the address addr+1 and the character count of text beginning at addr. The first byte at addr must contain the character count n. The range of n is {0..255}. (7S)

CR (---)
cause a carriage-return and line-feed to occur to the current output device. (7S)

CREATE <name> (---)
builds a dictionary entry for <name>, without allocating any parameter field memory. When <name> is subsequently executed, the address of the first byte of <name>'s parameter field is left on the stack. (7S)

CSP (--- addr)
leaves the addr of the USER variable which temporarily houses the address of the stack pointer during compilation error checking.

CSWAP (b1b2 --- b2b1)
swaps the two top bytes on the data stack.

CURRENT (--- addr)
leave the addr of the USER variable which specifies the vocabulary into which new word definitions are to be entered. (7S)

D+ (d1 d2 --- d3)
leave the arithmetic sum of d1 plus d2. (7S,7E)

D+- (d1 n --- d2)
assign the sign of n to double-word d1 leaving doubleword d2

D. (d ---)
display d converted according to BASE in a free field format, with one trailing blank. Display the sign only if negative. (7E)

D< (d1 d2 --- flag)
true if d1 is less than d2. (7S,7E)

DABS (d1 --- d2)
d1's absolute value, d2, is left on the stack.

DECIMAL (---)
set the input-output numeric conversion base to ten. (7S)

DEFINITIONS (---)
set CURRENT to the CONTEXT vocabulary so that subsequent definitions will be created in the vocabulary previously selected as CONTEXT. (7S)

DEPTH (--- n)
leave the number of the quantity of 16-bit values contained in the data stack, before n was added. (7S)

DIN (--- n)
read in 16-bit value n from the keypad. Returns 0 if keypad buffer is empty and 0100H - 010FH for keys '0' - 'F' respectively and 0114H - 0117H for "STP/RUN", "FUNC.", "DEC" and "ENTER" respectively.

DLITERAL <d> - compilation
(--- d) - execution
inside a colon-definition causes the compilation of the following 32-bit number. During execution this number is pushed on the stack.

DNEGATE (d1 --- d2)
leave the two's complement, d2, of double number d1. (7S)

DO (n1 n2 ---)
begin a loop which will terminate based on control parameters. The loop index begins at n2, and terminates based on the limit n1. At LOOP, +LOOP, or /LOOP the index is modified by a positive or negative value. The range of a DO-LOOP is determined by the terminating word. DO-LOOP may be nested. (7S)

DOES> (---)
define the run-time action of a word created by a high-level defining word. Marks the termination of the defining part of the defining word <name> and begins the definition of the run time action for words that will later be defined by <name>. On execution of the word defined by <name> the sequence of words between "DOES>" and ";" will be executed.

DOUT (n1 n2 ---)
output the hex digit n1 to the display selected by n2. The value of n1 must be in the range of 0 to F hex and n2 must be the in the range of 0 to 5. The displays are numbered 5 to 0 from left to right.

DP (--- addr)
leaves the addr of the USER variable which contains the address of the next available cell in memory above the dictionary. Commonly called the "dictionary pointer "

DP/ (--- addr)
leave the addr of the USER variable which contains the address of the next available cell of memory in the HEADS dictionary area. Commonly called the " HEADS dictionary pointer ".
**** NOTE **** FOR ADDITIONAL INFORMATION CONSULT THE DESCRIPTION ON THE CONSTRUCTION OF "HEADLESS" CODE FOUND IN THIS MANUAL.

DPL (--- addr)
leave the addr of the USER variable which contains the number of places after the fractional point for output conversion. If DPL contains zero, the last character output will be a decimal point. No decimal point is output if DPL contains a negative value. DPL may be set explicitly, or by certain output words, but is unaffected by number input. (7R)

DROP (n ---)
drop the top number from the stack. (7S)

DUMP (addr n ---)
list the contents of n addresses starting at addr each line of values may be preceded by the address of the first value. (7R)

DUP (n --- n n)
duplicate the top stack number. (7S)

EDITOR (---)
the name of the editor vocabulary. When this name is executed, EDITOR is established as the CONTEXT vocabulary. (7R)

ELSE (---)
ELSE executes after the true part following IF. ELSE forces execution to skip till just after THEN it has no effect on the stack. (7S)

EMIT (b ---)
transmit byte b to the current output device. (7S)

EMPTY-BUFFERS (---)
mark all block buffers as empty, without necessarily affecting their actual contents. UPDATED blocks are not written to mass storage. (7S)

ENCLOSE (addr c --- addr n1 n2 n3)
scan the text beginning at addr1 for either delimiter character c or ASCII null. In addition to addr leave :

- n1 offset in bytes from addr to the first character other than a delimiter.
- n2 offset in bytes to the first delimiter after the text.
- n3 offset in bytes to the first character not scanned.

ENDCASE (---)
closing syntactical word in a CASE option structure.
**** NOTE **** FOR MORE INFORMATION SEE THE SECTION ON THE CASE STATEMENT IN THIS MANUAL.

END-CODE (---)
terminate a code definition, resetting the CONTEXT vocabulary to the CURRENT vocabulary. If no errors have occurred, the code definition is made available for use. (7E)

ENDOF (---)
the conditional pair to OF< , OF= or OF> in a CASE structure.
**** NOTE **** FOR MORE INFORMATION SEE THE SECTION ON THE CASE STATEMENT IN THIS MANUAL.

ERASE (addr n ---)
fill an area of memory over n bytes with zeroes, starting at addr. If n is zero or less, take no action. (7R)

ERROR (n ---)
print error MESSAGE n and if USER variable BLK does not contain a 0 (i.e. a LOAD was occurring) print specific information about the location of the error (SCREEN #, LINE #, CHARACTER #).

EXECUTE (addr ---)
execute the dictionary entry whose compilation addr is on the stack. (7S)

EXIT (---)
when compiled within a colon-definition, terminate execution of that definition, at that point. May not be used within a DO...LOOP. (7S)

EXPECT (addr n ---)
transfer characters from the terminal beginning at addr, upward, until a "return" or the count of n has been received. Take no action for n less than or equal to zero. One or two nulls are added at the end of text. (7S)

FENCE (--- addr)
leaves the addr of the USER variable which contains the address below which " FORGETTING " is not allowed.

FILL (addr n byte ---)
fill memory beginning at addr with a sequence of n copies of byte. If the quantity n is less than or equal to zero, take no action. (7S)

FIND <name> (--- addr) or (--- 0)
leave the compilation addr of the next word <name> which is accepted from the input stream. If that word cannot be found in the dictionary after a search of the CONTEXT and FORTH vocabularies leave zero. (7S)

FIRST (--- addr)
leave the addr of the USER variable which contains the lowest memory address associated with block buffers.

FLD (--- addr)
leave the addr of the USER variable which contains the field length reserved for a number during

output conversion. (7R)

FORGET <name> (---)
delete from the dictionary <name> (which is in the CURRENT vocabulary) and all words added to the dictionary after <name>, regardless of their vocabulary. Failure to find <name> in the CURRENT or FORTH vocabularies results in an error condition. (7S)

FORTH (---)
the name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. New definitions become a part of FORTH until a different CURRENT vocabulary is established. User vocabularies conclude by 'chaining' to FORTH, so it should be considered that FORTH is 'contained' within each user's vocabulary. (7S)

H/T (addr ---)
places addr into USER variables DP/ and VP and places a 1 in USER variable HEADLESS. This causes the HEADS of subsequent words to be compiled at addr, TAILS or run-time code to be compiled at the contents of DP, and space for variables/arrays to be allotted at addr. The beginning locations for HEADS and TAILS compilation are displayed using the current BASE.
**** NOTE **** FOR ADDITIONAL INFORMATION CONSULT THE DESCRIPTION ON THE CONSTRUCTION OF "HEADLESS" CODE FOUND IN THIS MANUAL.

HEADLESS (--- addr)
leave the addr of the USER variable which contains a flag indicating compilation mode
flag = 0 NORMAL MODE
flag = 1 HEADLESS MODE
flag = 2 TARGET COMPILE MODE
**** NOTE **** FOR ADDITIONAL INFORMATION CONSULT THE DESCRIPTION ON THE CONSTRUCTION OF "HEADLESS" CODE FOUND IN THIS MANUAL.

HERE (--- addr)
return the addr of the next normal available dictionary location using (DP). If in HEADLESS mode return the next available location using the Variable Pointer (VP). (7S)

HERE/ (--- addr)
test the contents of USER variable HEADLESS. If it does not contain a 0, leave the addr of the next available location in the HEADS dictionary. If it does contain a 0 the address referenced by (DP) is used.
**** NOTE **** FOR ADDITIONAL INFORMATION CONSULT THE DESCRIPTION ON THE CONSTRUCTION OF "HEADLESS" CODE FOUND IN THIS MANUAL.

HEX (---)
set the numeric-output conversion base to sixteen. (7R)

HOLD (c ---)
insert character c into a pictured numeric output string. May only be used between <# and #>. (7S)

I (--- n)
copy the loop index onto the data stack. May only be used in the form: DO...I...LOOP, DO...I...+LOOP, or DO...I.../LOOP. (7S)

ID. (addr ---)
given a FORTH words name field address addr (NFA), print the name of the word.

IF (flag ---)
if flag is true, the words following IF are executed and the words following ELSE are skipped. The

ELSE part is optional. If flag is false, words between IF and ELSE, or between IF and THEN (when no ELSE is used), are skipped. IF-ELSE-THEN conditionals may be nested. (7S)

IMMEDIATE (---)
mark the most recently made dictionary entry as a word which will be executed when encountered during compilation rather than compiled. (7S)

INDEX (n1 n2 ---)
print the first line of each screen over the screen range {n1..n2} provided the line is COMMENTED. This displays the first line of each screen of source of text, which conventionally contains a title in the form of a comment. (7R)

INT5.5 (--- addr)
leave the addr of the USER variable which contains the compilation address of the current code sequence to be executed when a 5.5 interrupt occurs. " addr " is the address of the INT5.5 VECTOR (hardware access to this interrupt is available through the expansion connector)
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON INTERRUPTS IN THIS MANUAL.

INT6.5 (--- addr)
leave the addr of the USER variable which contains the compilation address of the current code sequence to be executed when a 6.5 interrupt occurs. " addr " is the address of the INT6.5 VECTOR (hardware access to this interrupt is available through the expansion connector)
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON INTERRUPTS IN THIS MANUAL.

INT7.5 (--- addr)
leave the addr of the USER variable which contains the compilation address of the current code sequence to be executed when a 7.5 interrupt occurs. " addr " is the address of the INT7.5 VECTOR (hardware access to this interrupt is available through the timer)
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON INTERRUPTS IN THIS MANUAL.

INTERPRET (---)
begin interpretation at the character indexed by the contents of >IN relative to the block number contained in BLK, continuing until the input stream is exhausted. If BLK contains zero, interpret characters from the terminal input buffer. (7R)

INTMASK (b ---)
perform an 8085 SIM instruction using the byte b.
**** NOTE **** THE LOWER FIVE BIT POSITIONS OF b ARE THE BITS AFFECTING THE INTERRUPTS. REFER TO THE SECTION ON INTERRUPTS IN THIS MANUAL.

J (--- n)
return the index of the next outer loop. May be used only within a nested DO-LOOP in the form: DO...DO...J...LOOP...LOOP. (7S)

KEY (--- c)
leave the ASCII value of the next available character c from the current input device. (7S)

LAST (--- addr)
leave the addr of the USER variable which contains the Name Field Address (NFA) of the last word in the FORTH KERNEL VOCABULARY (normally TASK).

LATEST (--- addr)
leave the name field address (nfa) of the last word defined in the current vocabulary.

LEAVE (---)
force termination of a DO-LOOP at the next LOOP, +LOOP, or /LOOP by setting the loop limit equal to the current value of the index. The index itself remains unchanged. EXECUTION PROCEEDS NORMALLY UNTIL THE LOOP TERMINATING WORD IS ENCOUNTERED. (7S)

LFA (pfa --- lfa)
convert the parameter field address (pfa) of a FORTH word to its link field address (lfa).

LIMIT (--- addr)
leave the addr of the USER variable which contains the constant highest memory address available to be used as a disk buffer.

LINE (n --- addr)
leaves the addr of the beginning line n for the screen whose number is contained in SCR. The range of n is {0..15}. (7R)

LIST (n ---)
list the ASCII symbolic contents of the screen n on the current output device, setting SCR to contain n. n is unsigned. (7S)

LIT <n> - compilation
(--- n) - execution
inside a colon-definition causes the compilation of the following 16-bit number. During execution this number is pushed on the stack.

LITERAL (n ---)
if compiling, compile the stack value n as a 16-bit literal, which when executed, will leave n on the stack. (7S)

LOAD (n ---)
begin interpretation of screen n by making it the input stream; preserve the locators of the present input stream (from >IN and BLK). If interpretation is not terminated explicitly it will be terminated when the input stream is exhausted. Control then returns to the input stream containing LOAD, determined by the input stream locators >IN and BLK. (7S)

LOAD/ (n ---)
performs headless compilation of source code beginning with screen n. In addition, the auto-start vector located at 8000H is filled with the code field address of the word to be executed upon power-up. Finally, the contents of " VP " (variable allocation) is set to 8100H.
**** NOTE **** SEE SECTION ON HEADLESS COMPILATION FOR MORE DETAILS.

LOOP (---)
increment the DO-LOOP index by one, terminating the loop if the new index is equal to or greater than the limit. The limit and index are signed numbers in the range {-32,768..32,767}. (7S)

M* (n1 n2 --- d)
multiply n1 by n2 leaving double-word d.

M/ (d n1 --- n2 n3)
divide signed double-word d by n1 leaving signed remainder n2 and signed quotient n3.

M/MOD (ud1 un2 --- un3 ud4)

unsigned division of double-word ud1 by word un2 leaving word remainder un3 and ud4.

MAX (n1 n2 --- n3)
leave n3 the greater of the two numbers n1 and n2. (7S)

MESSAGE (n ---)
print error message n :

0	UNDEFINED WORD
1	EMPTY STACK
2	USE WHEN COMPILING ONLY
3	USE WHEN EXECUTING ONLY
4	FULL STACK
5	PROTECTED DICTIONARY REFERENCE
6	UNDECLARED VOCABULARY
7	NO LINE REFERENCE
8	BAD PARAMETER
10	UNPAIRED CONDITIONALS
11	UNFINISHED DEFINITION
12	USE WHEN LOADING ONLY
13	SCREEN # 0 IS UNUSABLE

MIN (n1 n2 --- n3)
leave n3 the lesser of the two numbers n1 and n2. (7S)

MOD (n1 n2 --- n3)
divide n1 by n2, leaving the remainder n3, with the same sign as n1. (7S)

MOVE (addr1 addr2 n ---)
move n 16-bit memory cells beginning at addr1 into memory at addr2. The contents of addr1 is moved first. If n is negative or zero, nothing is moved. (7S)

NEGATE (n --- -n)
leave the two's complement of a number, i.e. the difference of 0 less n. (7S)

NFA (pfa --- nfa)
convert the parameter field address (pfa) of a FORTH word to its name field address (nfa).

NOOP (---)
execute a FORTH no operation.

NOT (flag1 --- flag2)
reverse the BOOLEAN value of flag1. This is identical to 0=. (7S)

NUMBER (addr --- n)
convert the count and character string at addr to a signed 32-bit integer, using the current base. If numeric conversion is not possible, an error condition exists. The string may contain a preceding negative sign. (7R)

OF< (n1 n2 ---)
part of the syntax found in the body of CASE. If n1 is less than n2 the code between OF< and its paired ENDOF will execute and control will be passed to ENDCASE. If not, n1 and control are passed to a subsequent OF . . . ENDOF, the optional default statement or ENDCASE.
**** NOTE **** FOR MORE INFORMATION SEE THE SECTION ON THE CASE STATEMENT IN THIS MANUAL.

OF= (n1 n2 ---)
part of the syntax found in the body of CASE. If n1 equals n2 the code between OF= and its paired ENDOF will execute and control will be passed to ENDCASE. If not, n1 and control are passed to a subsequent OF . . . ENDOF, the optional default statement or ENDCASE.
**** NOTE **** FOR MORE INFORMATION SEE THE SECTION ON THE CASE STATEMENT IN THIS MANUAL.

OF> (n1 n2 ---)
part of the syntax found in the body of CASE. If n1 is greater than n2 the code between OF> and its paired ENDOF will execute and control will be passed to ENDCASE. If not, n1 AND control are passed to a subsequent OF . . . ENDOF, the optional default statement or ENDCASE.
**** NOTE **** FOR MORE INFORMATION SEE THE SECTION ON THE CASE STATEMENT IN THIS MANUAL

OR (n1 n2 --- n3)
leave the bitwise inclusive-or n3 of the two numbers n1 and n2. (7S)

OUT (--- addr)
leave the addr of the USER variable which contains a value incremented in the course of output display formatting.

OVER (n1 n2 --- n1 n2 n1)
leave a copy of the second number on the stack. (7S)

P! (b n ---)
write byte b out I/O channel n.

P@ (n --- b)
read in byte b from I/O channel n.

PAD (--- addr)
leave the addr of a scratch area used to hold character strings for intermediate processing. The minimum size of PAD is 64 bytes. (7S)

PFA (nfa --- pfa)
convert a compiled FORTH words name field address to its parameter field address (pfa).
**** NOTE **** IN E-FORTH's HEADLESS KERNEL FORMAT THE PFA FOR MOST WORDS IS LOCATED IN A COMPLETELY SEPARATE AREA OF MEMORY AND ARE NORMALLY REFERENCED INDIRECTLY BY A POINTER (PFAPTR). WE HAVE KEPT THE WORD "PFA", ACTUALLY NOTHING MORE THAN A " PFAPTR @ ", IN OUR KERNEL AS A MATTER OF CONVENIENCE.

PFAPTR (nfa --- addr)
given the name field address (nfa) of a word leave its associated pfaptr. Every dictionary entry uses pfaptr in place of (FIG-FORTH) pfa. A subsequent fetch on the pfaptr will yield the pfa.

PICK (n1 --- n2)
return the contents of the n1-th stack value, including n1 itself. An error condition results for n less than one. 2 PICK is equivalent to OVER. (7S)

PITCH (n ---)
outputs a frequency to the speaker that is inversely proportional to the value on the stack. The upper

two bits in 'n' are masked off giving the value a range of 0 to 16383 (decimal) or 0 to 3fff (hex). A value of 0 turns off the speaker.

PREV (--- addr)
leaves the addr of the USER variable which contains the address of the disk buffer that was most recently referenced.

PROMPT (---)
send a prompt to the current output device which indicates :
1. BASE by means of a letter i.e. (B)inary, (O)ctal, (D)ecimal, and (H)exadecimal and
2. the number of items (in decimal) on the data stack.

PTAIN (--- b)
read the contents of the parallel output port A.

PTAOUT (b ---)
send a byte out the parallel port A.

PTBIN (--- b)
input a byte from the parallel port B.

QUERY (---)
accept input of up to 80 characters (or until a <cr>) from the operator's terminal, into the terminal input buffer. WORD may be used to accept text from this buffer as the input stream, by setting >IN and BLK to zero. (7S)

QUIT (---)
clear the return stack, setting execution mode, and return control to the terminal. No message is given. (7S)

R# (--- addr)
leave the addr of the USER variable which contains the offset of the cursor from the start of the screen.

R/W (addr n f ---)
block read-write. "addr" specifies the source or destination memory address. "n" is the sequential number of the referenced block. "f" is the flag which indicates read (non-zero) or write (zero). "R/W" determines the physical mass storage location, performs the read-write and checks for errors.

R0 (--- addr)
leaves the addr of the USER variable which contains the initial (from power up or COLD) address of the return stack.

R> (--- n)
transfer n from the return stack to the data stack. Every "R>" must be balanced by a ">R" in the same control structure nesting level of a colon-definition. (7S)

R@ (--- n)
copy the number on the top of the return stack to the data stack. (7S)

RDSCLK (---)
read the Real Time Clock registers into the eight consecutive RAM locations at CLKREG. FOR MORE

INFORMATION SEE APPENDIX C

REPEAT (---)
at run-time, REPEAT returns to just after the corresponding BEGIN. (7S)

RETURN (b ---)
using byte b, perform a user-defined INTMASK operation, then compile code for a high-level return from interrupts. USE ONLY AT THE END OF A COLON DEFINITION.
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON INTERRUPTS IN THIS MANUAL.

ROLL (n ---)
extract the n-th stack value to the top of the stack, including n itself, moving the remaining values into the vacated position. An error condition results for n less than one. {1..n} 3 ROLL = ROT, 1 ROLL = null operation. (7S)

ROT (n1 n2 n3 --- n2 n3 n1)
rotate the top three values, bringing the deepest to the top. (7S)

RP! (---)
(re)initializes the return stack pointer to the contents of the USER variable R0.

RP@ (--- addr)
fetches the current addr contained in the return stack pointer.

RSCR (n ---)
receive the 1024 bytes being transmitted to the current input device and store them on screen n. After the bytes have been written to the screen, transmit a trailer (ASCII 2AH) from the PRIMER to the host followed by a calculated Longitudinal Redundancy Check (LRC) of the screen.

RST7 (--- addr)
leave the address of the USER variable which contains the compilation address of the current code sequence to be executed upon the occurrence of a low level RST 7 instruction. "addr" is the address of the RST 7 vector.
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON VECTORS AND INTERRUPTS IN THIS MANUAL.

S->D (n --- d)
sign extend word n to form double-word d.

S-FORTH (n ---)
save the created OVERLAY on the screen(s) starting with screen n.
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON OVERLAYS IN THIS MANUAL.

S0 (--- addr)
leaves the addr of the USER variable which contains the address of the bottom of the stack, when the stack is empty. (7R)

SAVE-BUFFERS (---)
write all blocks to mass-storage that have been flagged as UPDATED. An error condition results if mass-storage writing is not completed. (7S)

SCR (--- addr)
leave the addr of the USER variable which contains the unsigned number of the screen most recently listed. (7S)

SIGN (n ---)
insert the ASCII "-" (minus sign) into the pictured numeric output string, if n is negative. (7S)

SMUDGE (---)
used to toggle the smudge bit (b5 of b7 - b0) of the length byte in the name field of a word. " : " initially SMUDGES preventing reference of the word currently being compiled. " ; " ends compilation and checks for success. If successful the word is SMUDGED again thereby enabling reference via a vocabulary / dictionary search.

SP! (---)
re-initializes the data stack pointer to the contents of the USER variable S0.

SP@ (--- addr)
return the addr of the top of the stack, just before SP@ was executed. (7R)

SPACE (---)
transmit an ASCII blank (20H) to the current output device. (7S)

SPACES (n ---)
transmit n spaces to the current output device. Take no action for n <= 0. (7S)

SPLIT (n --- b1 b2)
split word n into two byte values "b1" and "b2". "b1" represents the most significant portion of n and b2 represents the least significant portion of n.

SSCR (n ---)
transmits screen n out the current output device. The transmission is preceded by a header (ASCII 2AH) and followed by a trailer (ASCII 2AH) and the byte Longitudinal Redundancy Check (LRC) for the screen.

STATE (--- addr)
leave the addr of the USER variable containing the current compilation state. A non-zero content indicates compilation is occurring. (7S)

STATUS (--- addr)
leave the addr of the USER variable which contains a 16-bit value indicating the occurrence of various interrupts.

```
b15 - b8  used by system
b7  - b3  unused
b2    indicates ( set high on ) INT7.5
b1    indicates ( set high on ) INT6.6
b0    indicates ( set high on ) INT5.5
```

The user is responsible for resetting these bits.
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON INTERRUPTS IN THIS MANUAL.

SWAP (n1 n2 --- n2 n1)

exchange the top two stack values. (7S)

TASK (---)
a word serving only to mark the end of the E-FORTH " FORTH " vocabulary.

TEXT (c ---)
accept characters from the input stream into the PAD until the non-zero delimiting character c is encountered or the input stream is exhausted. Fill the remainder of the PAD (up to 64 characters) with blanks. (7R)

THEN (---)
THEN is the point where execution resumes after ELSE or IF (when no ELSE is present). (7S)

TIB (--- addr)
leaves the addr of the USER variable which contains the address of the 80 character long Terminal Input Buffer.

TIMER0 (b n ---)
set the length of TIMER0 (IC 8155) count to equal the lower 14 bits (0-13) of word n with bits 14 and 15 indicating the desired TIMER output mode. Byte b is divided up as follows :
bit positions 0-1 are used to specify the TIMER command bit positions 2-7 have no impact on the TIMER and should be set 0 .
**** NOTE **** FOR MORE INFORMATION REFER TO THE TIMER / COUNTER SECTION LOCATED IN THIS MANUAL.

TOGGLE (addr b ---)
complement the byte located at addr by the bit pattern of byte b.

TRAVERSE (addr1 n --- addr2)
beginning at the length byte (addr1) in the name field of a FORTH word determine the location (addr2) of the last letter in the name field by incrementing (n=1) or decrementing (n=-1) addr1 a byte at a time.

TYPE (addr n ---)
transmit n characters beginning at addr to the current output device. No action takes place for less than or equal to zero. (7S)

U* (un1 un2 --- ud3)
perform an unsigned multiplication of un1 by un2, leaving the double number product ud3. All values are unsigned. (7S)

U. (un ---)
display un converted according to BASE as an unsigned number, in a free-field format, with one trailing blank. (7S)

U/MOD (ud1 un2 --- un3 un4)
perform the unsigned division of double number ud1 by un2, leaving the remainder un3, and quotient un4. All values are unsigned. (7S)

U< (un1 un2 --- flag)
leave the flag representing the magnitude comparison of un1 < un2 where un1 and un2 are treated as 16-bit unsigned integers. (7S)

UNTIL (flag ---)
within a colon-definition, mark the end of a BEGIN-UNTIL loop, which will terminate based on a flag. If

the flag is true, the loop is terminated. If the flag is false, continue loop execution. (7S)

UPDATE (---)
mark the most recently referenced block as modified. The block will then be automatically transferred to mass storage should its memory buffer be needed for storage of a different block or upon execution of SAVE-BUFFERS. (7S)

USE (--- addr)
leaves the address of the user variable which contains the address of the block buffer to use next (block buffer least recently written to).

USER <name> (n ---)
a defining word used in the form: n USER <name> which creates a user variable <name>. "n" is the cell offset within the user area where the value for <name> is stored. Execution of <name> leaves its absolute user area storage address. (7R)

VARIABLE <name> (---)
creates a dictionary entry for <name> and allot two bytes for storage in the parameter field. The application must initialize the stored value. When <name> is later executed, it will place the storage address on the stack. If in HEADLESS mode allocate space for the variable using the Variable Pointer (VP). (7S)

VLIST (---)
list the names of the words found in the CONTEXT vocabulary beginning with the most recently defined word. (7R)

VOC-LINK (--- addr)
leave the addr of the USER variable which contains the address of a field in THE DEFINITION THE MOST RECENTLY CREATED VOCABULARY. All vocabulary names are linked by these fields enabling FORGETTING through multiple vocabularies.

VOCABULARY <name> (---)
creates (in the CURRENT vocabulary) a dictionary entry for <name>, which specifies a new ordered list of word definitions. Subsequent execution of <name> will make it the CONTEXT vocabulary. When <name> becomes the CURRENT vocabulary (see DEFINITIONS), new definitions will be created in that list. (7S)

VP (--- addr)
leave the addr of the USER variable which contains the next available location for TARGET COMPILED variables/arrays.

WARM (---)
causes a system warm start which clears the disk buffers and re-initializes the stacks but leaves the dictionary unchanged.

WHILE (flag ---)
select conditional execution based on flag. On a true flag, continue execution through to REPEAT, which then returns back to just after BEGIN. On a false flag, skip execution to just after REPEAT, exiting the structure. (7S)

WIDTH (--- addr)
leaves the addr of the USER variable which contains the maximum number of characters that are saved, for later reference, when a FORTH word is compiled. (in E-FORTH WIDTH contains 31)

WORD (c --- c)
receive characters from the input stream until the non-zero delimiting character c is encountered or the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed string with the character count in the first character position. The actual delimiter encountered (c or null) is stored at the end of the text but not included in the count. If the input stream was exhausted as WORD is called, then a zero length will result. The address of the beginning of this packed string is left on the stack. (7S)

WRSCLK (---)
set the Real Time Clock by writing the 8 consecutive RAM locations at CLKREG into the Real Time Clock.
**** NOTE **** FOR MORE INFORMATION SEE APPENDIX C

X-FORTH (n ---)
LOAD or invoke the OVERLAY stored on screen n.
**** NOTE **** FOR MORE INFORMATION CONSULT THE SECTION ON OVERLAYS IN THIS MANUAL.

XOR (n1 n2 --- n3)
leave the bitwise exclusive-or of two numbers. (7S)

[(---)
end the compilation mode. The text from the input stream is subsequently executed. see] (7S)

[COMPILE]<name> (---)
forces compilation of the following word. This allows compilation of an IMMEDIATE word when it would be otherwise executed. (7S)

] (---)
set the compilation mode. The text from the input stream is subsequently compiled. see [(7S)

**** NOTE **** Constants are include in the E-FORTH KERNEL because they are frequently used, they execute faster than ordinary numbers, and they use less memory. They use less memory because in any number other than a constant, the FORTH word literal is compiled as well, thereby increasing the amount of memory the compiled code will occupy.

EXAMPLES OF SOME FORTH BASICS

THE DO LOOP
Example of a DO LOOP that prints numbers from 0 to 99.

```
: XAMPLE0 100 0 DO I . LOOP ;
```

Example for a DO LOOP with a step size of 2. Prints even numbers from 100 to 1.

```
: XAMPLE1 100 0 DO I . 2 +LOOP ;
```

Example of a DO LOOP that counts down. Prints numbers from 100 to 1.

```
: XAMPLE2 0 100 DO I . -1 +LOOP ;
```

Example of a DO LOOP that prints numbers from 0 to 39999.

```
: XAMPLE3 40000 0 DO I U. 1 /LOOP ;
```

IF THEN ELSE

Example of a DO LOOP containing a IF THEN ELSE statement to determine which loop index numbers are less than 50.

```
: XAMPLE4  100  0  DO  CR  I  DUP  .  50  <  IF  ." IS LESS THAN 50 "
  ELSE  ." IS GREATER OR EQUAL TO 50  "  THEN  LOOP  ;
```

BEGIN UNTIL

Example of a BEGIN UNTIL loop that counts down. Prints numbers from 100 to 1.

```
VARIABLE TEMP

: XAMPLE5  100  TEMP  !  BEGIN  TEMP  @  DUP  .  -1  DUP  TEMP  !  0=  UNTIL  ;
```

Example of a BEGIN UNTIL loop that continues until a key is pressed.

```
: XAMPLE6  BEGIN  CR  ." HIT ANY KEY TO STOP "  ?TERMINAL  UNTIL  ;
```

WHILE REPEAT

Examples of a WHILE REPEAT loop that counts by 5. Prints numbers from 5 to 95.

```
5  CONSTANT  FIVE

: XAMPLE7  0  BEGIN  FIVE  +  DUP  100  <  DUP  .  REPEAT  DROP  ;
```

Chapter 4

THE EDITOR

The 2nd memory slot on your PRIMER is typically populated by a 32K RAM. The Forth system reserves 20K of this RAM to be used as an area in which you can store source code. The 32K RAM may be replaced with a 32K RAMDISK (this may be purchased from EMAC) which has a built-in lithium energy cell which retains the RAM's memory when there is no power to the PRIMER. The RAMDISK may be inserted or removed from the memory slot (with the power to the PRIMER off) without loss of memory, much like a floppy disk.

The RAMDISK is the area in memory used by the programmer to design, modify and LOAD FORTH source code. The RAMDISK is divided up into twenty 1K sections called SCREENS. Each SCREEN is organized as 16 lines each made up of 64 characters.

To invoke the EDITOR and thus have access to EDITOR words simply type "EDITOR" <return>. Loading a screen or compiling a word will remove you from the EDITOR and into the FORTH VOCABULARY.

**** NOTE **** TO USE THE EDITOR, EITHER A RAM OR RAMDISK IS REQUIRED. IF A REGULAR RAM IS USED, POWER TO THE SYSTEM CANNOT BE INTERRUPTED WITHOUT DATA LOSS. ALSO ANY CHANGES MADE TO THE EDITOR ARE NOT GUARANTEED TO BE SAVED UNLESS THE EDITOR WORD " L " OR THE FORTH WORD " FLUSH " ARE PERFORMED AFTER THE CHANGES.

Invoking the EDITOR will automatically change the base to decimal and the user variable HEADLESS to 0. If an error is encountered during LOADing E-FORTH will automatically invoke the editor, LIST the screen that contains the error, and position the cursor at the error.

THE PAD

The PAD is the address of a 64 byte work space designed for use with the EDITOR in the manipulation of character strings of up to 64 bytes (1 line) in size.

**** NOTE **** IF YOU ARE NOT EDITING, THE PAD STILL EXISTS AND MAY BE USED.

GETTING STARTED

Unless you possess a photographic memory, the ability to see what you're editing is extremely important. To display a screen use the command

LIST (n ---)
which displays screen n and selects it for editing.

If you don't like what you see or want to erase a screen for any reason use the command.

CLEAR (n ---)
which clears screen n and selects it for editing.

**** NOTE **** IN BOTH CASES n IS AN INTEGER FROM 1 - 16

The following is a list of other commands within the EDITOR vocabulary.

#CLEAR (n1 n2 ---)
clears n2 screens starting at screen n1. outputs the current screen being cleared in the current base.

#COPY (n1 n2 n3 ---)
copies n3 screens starting at screen n1 to screen n2. outputs the current screen being copied and the screen being copied, to in the current base.

B (---)
backup the cursor by the number of characters in the PAD.

C (---)
insert the following text at the current cursor position.

COPY (n1 n2 ---)
copy screen n1 to screen n2.

D (n ---)
place a copy of line n in the PAD, then delete line n.

DEL (n ---)
delete the n characters preceding the cursor.

E (n ---)
replace every character in line n with a blank.

F <text> (---)
put the following text in the PAD and search from the current cursor position down for a copy of that text.

H (n ---)
place a copy of line n in the PAD.

I (n ---)
perform a spread (S) at line n and insert text from the PAD at line n.

L (n ---)
list and FLUSH screen n.

M (n ---)
move the cursor by the signed amount n and print its line.

N<text> (---)
find the next occurrence of the text in the PAD.

P<text> (n ---)
put the following text on line n of logged screen.

R (n ---)
replace line n with the contents of the PAD.

S (n ---)
spread the screen at line n. lines n through 15 are moved down one line. line 15 is lost and line n is blanked out.

T (n ---)
type line n and place a copy of it in the PAD.
**** NOTE **** CONTRARY TO MANY DESCRIPTIONS OF " T " FOUND IN PUBLICATIONS SUCH AS " STARTING FORTH " BY LEO BRODIE OUR " T " DOES NOT LEAVE THE LINE NUMBER n ON THE STACK.

TILL <text> (---)
delete on current cursor line up to the end of the following text.

TOP (---)
move the cursor to the top of the screen.

X <text> (---)
delete the next occurrence of the following text.

EDITING HINTS AND PITFALLS

(EDITOR and FORTH words are CAPITALIZED)

1. n1 SCR ! n2 H n3 SCR ! n4 R L - moves line n2 on screen n1 to line n4 on screen n3.

2. C followed immediately by a carriage return will store a null character after the cursor.
The EDITOR word sequence . . .

TOP X

followed immediately by a carriage return should remove the null character. A null character can cause strange undefined word errors so beware.

3. Occasionally non-ASCII characters are entered (i.e control characters). They can prove troublesome to find when attempting to LOAD your screens. A good method of finding these characters, without reentering the screen, is to list the screen line by line using the EDITOR command T. (0 T 1 T etc.) All the end-of-line indicators on the right side of the screen should form a column. If one doesn't, that line probably contains an unloadable character. Retype that line and this should correct the problem.

Chapter 5

THE ASSEMBLER

Occasionally, execution speed in a FORTH program is so critical that selective words must be converted to assembly language to achieve the desired results. Included in E-FORTH, is an ASSEMBLER (for use with the 8085 CPU) that will give you the ability to create these " speed " oriented words. In addition to the " normal " 8080 mnemonics we have added the two 8085 mnemonics, SIM and RIM, to the ASSEMBLER. Speed, while important to execution time, is also desirable in programming time. In the course of our involvement with the ASSEMBLER we found the following practices to be invaluable time savers.

1) After you have written your assembly code out on paper, on a separate sheet write the code out backwards removing punctuation as you go.

```
i.e.  MOV   M,A      becomes     A    M MOV    and
      LXI   H,6000   becomes     6000 H LXI
```

Then, using this separate sheet as your guide, enter your modified " code assembly ".

2) The ASSEMBLER will LOAD or COMPILE your FORTH assembly code in whatever base you happen to be in at the time. IF THE BASE IS IMPORTANT, SPECIFY IT. Additionally, if you are using HEX as the BASE place a leading zero in front of any HEX numbers that do not begin with a digit. This will avoid any confusion with ASSEMBLER registers which are defined as constants (A , B , C, etc).

3) If you have used FORTH for even a short period of time, you have probably become accustomed to how FORTH uses the stack in resolving conditionals such as > , < , >= etc. The ASSEMBLER, true to its form, resolves conditional jumps based on the FLAGS REGISTER not the stack. One more time with feeling . . THE ASSEMBLER RESOLVES CONDITIONALS BASED ON THE FLAGS REGISTER (PSW) . . . NOT THE STACK.

CONDITIONALS IN THE ASSEMBLER

The ASSEMBLER handles conditional jumps without using the " normal " mnemonics. The ASSEMBLER words 0= , CS , PE , 0< by themselves or in combination with the ASSEMBLER word NOT enable you to perform all eight conditional jumps. The code and associated 8080 / 8085 mnemonics for all eight are listed below.

ASSEMBLER	8080/8085	ASSEMBLER	8080/8085
0=	JZ	0= NOT	JNZ
CS	JC	CS NOT	JNC
PE	JPE	PE NOT	JPO
0<	JM	0< NOT	JP

If you are thinking in FORTH this can get downright confusing. Our editors think this would be the perfect time for an example of an ASSEMBLER conditional and we'll sneak in an example of ASSEMBLER syntax as well. Take the following line of assembly code . . .

```
   NORMAL ASSEMBLY                  FORTH ASSEMBLY
   ~~~~~~~~~~~~~~~                  ~~~~~~~~~~~~~~

BEGIN:    DCX   D
          MOV   A,D            D DCX  D A MOV  E ORA  0= NOT
          ORA   E                |                          |
          JNZ   BEGIN            |__________________________|
UNTIL:    . . .
          . .
          .
```

In FORTH syntax a BEGIN . . . UNTIL loop terminates when the stack has as it's top-most value a non-zero number. In ASSEMBLER syntax a BEGIN . . . UNTIL loop terminates based on the state of the associated flag. Adding ASSEMBLER syntax the example becomes . . .

. . . BEGIN D DCX D A MOV E ORA 0= NOT UNTIL . . .

0= NOT should be thought of as if the FORTH sequence 0= NOT was performed on the zero flag with the result (0 or 1) interpreted by UNTIL.

Conversely, if the example read as follows:

. . . BEGIN D DCX D A MOV E ORA 0= UNTIL . . .

0= should be thought of as if the FORTH word 0= was performed on the zero flag with the result (0 or 1) interpreted by UNTIL.

A global set of statements can be applied to ASSEMBLER conditional jumps.

ASSEMBLER MNEMONICS 0= , CS , PE , 0<

SHOULD BE THOUGHT OF AS IF THE FORTH SEQUENCE " 0= " WAS PERFORMED ON THE CORRESPONDING FLAG WITH THE RESULT (0 or 1) INTERPRETED BY UNTIL.

ASSEMBLER MNEMONICS 0= NOT , CS NOT , PE NOT , 0< NOT

SHOULD BE THOUGHT OF AS IF THE FORTH WORD " 0= NOT " WAS PERFORMED ON THE CORRESPONDING FLAG WITH THE RESULT (0 or 1) INTERPRETED BY UNTIL.

**** NOTE **** "ASSEMBLER REPEAT" AND "ASSEMBLER AGAIN" UNCONDITIONALLY JUMP TO THEIR ASSOCIATED ASSEMBLER BEGINs.

If you think it is still confusing, you're right, but it becomes clearer with practice. Let's try another example.

ANOTHER EXAMPLE:

```
SAMPLE   ( n --- n )    CODE SAMPLE  H POP  L A MOV  A ORA  0=
                             IF  69 L MVI  ELSE  96 L MVI  THEN
                             HPUSH JMP  END-CODE
```

The sequence "0 SAMPLE ." <return> will yield 96 while "5 SAMPLE . " <return> will produce 69.

Let's examine SAMPLE closer. The H POP pops the initial value off the stack, it is then moved to the Accumulator and ORed with itself in order to set the appropriate flags.

If a 0 is popped off the stack and ORed the Zero Flag is set to 1. 0= can then be thought of as performing a test on the Zero Flag, with the result of that test being False or 0. The IF looks at result of the test and executes the code after the "IF" if the result was True or 1 and branches to the ELSE if the result was 0.

In the above case it branched and executed the code 96 L MVI. This value is pushed on the stack by HPUSH JMP which simply translates H PUSH NEXT JMP.

E-FORTH's USE OF REGISTERS

E-FORTH's uses only the BC Register Pair. Other Registers are free to be used by any Assembly Language Routine. If the BC Register Pair is needed, it must be pushed on the stack before use and popped off the stack before jumping to next.

PASSING INFORMATION TO AND FROM THE ASSEMBLER

Information can be passed to and from an assembled word in several ways. The simplest method is through use of the data stack. The data stack that is readily accessible through the ASSEMBLER, is the same data stack used by FORTH. To pass a number from FORTH to an ASSEMBLER word, place the number on the stack in FORTH and pop it off into a register in ASSEMBLER.

A alternate method is to create a VARIABLE in FORTH which will provide low level access. To access a FORTH VARIABLE simply use the VARIABLE name. . . VAR LDA . . . VAR STA . . etc.

CONSTANTS AND LABELS

Declaring constants within the FORTH ASSEMBLER is no different than in FORTH but the capability to do so is important and this fact will become apparent soon. An example of declaring a constant is as follows . . .

```
7F CONSTANT ASCIIAND
```

In order for an ASSEMBLER word, or sequence of words, to become integrated into the FORTH system, a means has to be provided for the word (or word sequence) to return to the system upon execution completion. This is accomplished via the use of the CONSTANT NEXT which is equated to the address of NEXT residing inside the FORTH interpreter.

LABELS are a useful tool when trying to code complex conditionals in assembly language or when trying to save EPROM space in an application. Our ASSEMBLER provides for their implementation with a few restrictions.

1. Forward referencing is not supported.

2. If a LABEL does not contain an ASSEMBLER RET statement (i.e. it was CALLed) a means for the LABEL to return to the FORTH interpreter after it has finished execution must be provided. This may take the form of NEXT JMP or a CALL or CONDITIONAL BRANCH to a WORD or LABEL containing a NEXT JMP.

THE ASSEMBLER VOCABULARY

In addition to all the 8080-8085 mnemonics and the words used as part of the structure of a CONDITIONAL, the following CONSTANT addresses are found in the ASSEMBLER vocabulary.

A/D (--- addr)
a subroutine providing low level access and thus maximum speed to the 6 bit A/D converter. Receives the channel number in the accumulator and returns the A/D count in the H/L register pair. All registers except the B/C register pair and the E register are used.

DPUSH (--- addr)
DPUSH returns the constant addr of the low-level FORTH linkage routine that, when used in the instruction " DPUSH JMP ", compiles a JMP to the addr DPUSH. At execution time the D-E and H-L register pairs are PUSHED.

HPUSH (--- addr)
HPUSH returns the constant addr of the low-level FORTH linkage routine that, when used in the instruction " HPUSH JMP ", compiles a JMP to the addr HPUSH. At execution time the H-L register pair is PUSHED.

NEXT (--- addr)
NEXT returns the constant addr of the low-level FORTH linkage routine that, when used in the instruction " NEXT JMP ", compiles a JMP to the addr NEXT.

RETINT (--- addr)
returns the constant addr of the " low-level " INTerrupt RETurn

RPP (--- addr)
returns the constant addr that contains the location which is used to hold the FORTH return stack pointer.

WORDS WRITTEN USING THE ASSEMBLER

In order for FORTH to reference a word that you have written with the FORTH ASSEMBLER the ASSEMBLER must provide a way for the programmer to create headers and trailers. The ASSEMBLER words CODE and END-CODE perform these tasks and are used as pairs in the form . . .

. . . CODE <name> mnemonics END-CODE . . .

CODE builds a header for the ASSEMBLER word being defined and enables compilation of the word by invoking the ASSEMBLER vocabulary.

END-CODE terminate compilation and link the just defined word to the rest of the current FORTH vocabulary thereby enabling it's subsequent referencing by any other word in that vocabulary.

EXAMPLES USING THE ASSEMBLER

```
CODE  CSWAP ( swaps high and low bytes of the word on the stack )
      H POP  L A MOV  H L MOV  A H MOV  H PUSH  NEXT JMP
END-CODE



CODE  CMOVE ( addr1 addr2 n --- ) ( moves n bytes from addr1 to addr2 )
      C L MOV  B H MOV  B POP  D POP  XTHL
      BEGIN  B A MOV  C ORA  0=
      WHILE  M A MOV  H INX  D STAX  D INX  B DCX
      REPEAT
      B POP  NEXT JMP
END-CODE



( THIS EXAMPLE IS FOR DEMONSTRATION ONLY AND DOES NOT FUNCTION )

80 CONSTANT CMMD                          ( command byte )
F0 CONSTANT CMDPRT                        ( command port )
F1 CONSTANT STATPRT                       ( status  port )

LABEL DELAY
      BEGIN  D DCX  D A MOV  E ORA  0= NOT  UNTIL
      RET
END-CODE

CODE  CMMDSTAT                          ( performs a continuous check of)
      H POP  CMMD A MVI  CMDPRT B LXI ( the serial input status. If the)
      A OUT  2000 D LXI  DELAY CALL   ( status bit <register L>, when)
      BEGIN  STATPRT IN  L ANA  0=    ( ANDed with the status byte)
      UNTIL                           ( results in the zero flag being)
      NEXT JMP                        ( zeroed, the check is terminated)
END-CODE
```

Chapter 6

VARIABLES AND ARRAYS

You may have noticed within the dictionary of the E-FORTH KERNEL the words ALLOT and VARIABLE. These words are designed to allocate space and create reference addresses for 16-bit variables within a NORMAL or HEADLESS/TARGET compilation. To debug A HEADLESS/TARGET program with these words in place, keep in mind that. . .

> DURING COMPILATION, COMPILED TAILS SHOULD NEVER BE ALLOWED TO OVERWRITE COMPILED HEADS. IF THIS HAPPENS THE PROGRAM COULD CRASH WHEN COMPILING.
>
> ADDITIONALLY, MEMORY ALLOCATED FOR THESE WORDS BEGINS AT THE ADDRESS CONTAINED IN THE USER VARIABLE " VP ". THE WORDS ALLOT, VARIABLE, AND HERE ALL REFERENCE VP WHEN IN HEADLESS MODE RATHER THAN THE NORMALLY REFERENCED DP. ALSO THE WORDS ALLOT/ AND HERE/ REFERENCE DP/ IN HEADLESS MODE RATHER THAN DP.

Since the heads are of no use during run-time, their address is often a convenient place at which to allocate variable and array storage. This being the case, " H/T " also stores the address for heads compilation in the USER VARIABLE " VP " which serves as the base address for variable and array allocation. Remember that just because space is allocated, nothing is physically written into those locations. INITIALIZATION OF ARRAYS AND VARIABLES SHOULD BE PERFORMED AT EXECUTION TIME BY AN INVOKING WORD.

**** NOTE **** IF PROGRAM VARIABLE AND ARRAY STORAGE WRITES OVER COMPILED HEADS, A COLD RESET WILL BE NECESSARY AFTER PROGRAM TERMINATION, IN ORDER TO ACCESS THE OPERATING SYSTEM.

example 1 : you wish to debug a HEADLESS Application Program and have determined (by checking HERE) that the compiled TAILS run from 8002H to 9000H and (by checking HERE/) that the compiled HEADS run from 9800H to 9900H. Further, you have an array that is 200H bytes in length.

5800 H/T will put you in HEADLESS mode 1, compile subsequent HEADS beginning at 9800H and allocate space for your array at 9800H as well. You may, if you desire, alter the contents of " VP " either immediately after the " H/T " command or even interactively on a "LOAD"ing screen. This allows you total control of where variable/array space is allocated.

<scr#> LOAD LOADS your code beginning at <scr#>. You should now be able to run/debug your APPLICATION before burning it.

MORE ON VARIABLES AND ARRAYS

The word VARIABLE in E-FORTH will automatically create VARIABLEs that switch from standard INLINE mode to HEADLESS mode by simply executing the word H/T. No modification to the user's program is generally necessary.

Listed below are 4 ARRAY words, like VARIABLE that require no modification when switching from INLINE mode to HEADLESS mode. The 4 ARRAYs are a one-dimensional byte array " BARRAY ", a (16 bit) word array " WARRAY ", a two-dimensional byte array " DARRAY " and a two-dimensional (16 bit) word array " DWARRAY ". These words are not found in the E-FORTH KERNEL but are provided below for your use. They are designed for use in all compilation modes and as such their function with respect to VP (see above) is the same.

BARRAY <name> (n ---)

creates and allocates memory for an array <name> made up of n 8-bit entries. The base address for the array is found by the FORTH sequence . . .

0 <name> HEX .

```
: BARRAY CREATE 2 DP +! HERE HEADLESS @ 0= IF 2+ THEN , ALLOT
          DOES> @ SWAP + ;
```

WARRAY <name> (n ---)

creates and allocates memory for an array <name> made up of n 16-bit entries. The base address for the array is found by the FORTH sequence . . .

0 <name> HEX .

```
: WARRAY CREATE 2 DP +! HERE HEADLESS @ 0= IF 2+ THEN , 2*
         ALLOT DOES> @ SWAP 2* + ;
```

DARRAY <name> (n1 n2 ---)

creates and allocates memory for a two-dimensional array <name> made up of n1 * n2 8-bit entries. n1 is the number of rows in the array. n2 is the number of columns in the array. The base address for the array is found by the FORTH sequence . . .

0 0 <name> HEX .

to reference an entry use the sequence

n1 n2 <name>

where n1 is the row # and n2 is the column #

```
: DARRAY CREATE 2 DP +! DUP , HERE HEADLESS @ 0= IF 2+ THEN ,
          * ALLOT DOES> ROT OVER @ * ROT + SWAP 2+ @ + ;
```

DWARRAY <name> (n1 n2 ---)

creates and allocates memory for a two-dimensional array <name> made up of n1 * n2 16-bit entries. n1 is the number of rows in the array. n2 is the number of columns in the array. The base address for the array is found by the FORTH sequence . . .

0 0 <name> HEX .

to reference an entry use the sequence

n1 n2 <name>

where n1 is the row # and n2 is the column #

```
: DWARRAY CREATE 2 DP +! DUP 2* , HERE HEADLESS @ 0= IF 2+
           THEN   , * 2* ALLOT DOES> ROT OVER @ * ROT 2* +
SWAP            2+ @ + ;
```

**** NOTE **** UPON POWER-UP "VP" IS INITIALIZED TO 8100H, THIS IS THE RECOMMENDED STARTING ADDRESS FOR TARGET VARIABLES WHEN BURNING A TARGET APPLICATION EPROM. "H/T" INITIALIZES "VP" TO THE SAME ADDRESS YOU SELECT FOR "HEADS" PLACEMENT.

Chapter 7

HEADLESS CODE GENERATION

The E-FORTH operating system possesses an advanced method of FORTH code compilation, TARGET COMPILATION. You, the programmer, have the option of selecting either standard IN-LINE COMPILATION or TARGET COMPILATION (with separated HEADS). The compiled code in the latter case is commonly called HEADLESS CODE.

There are three advantages for using HEADLESS COMPILATION over normal compilation. First, and most obvious, is the space you save in a TARGET EPROM. Secondly, your compiled code is immediately given an additional level of security. Only the most hardcore software pirates will be able to reconstruct your program without access to the HEADS. Lastly, by compiling selective words HEADLESS and others normally, a limited vocabulary or even a new language may be created.

Within a standard FORTH dictionary entry there exists several fields. These fields are referred to as the name, link, code, and parameter fields. By adding a level of indirection to the parameter field the E-FORTH system is able to compile code without dictionary header information (HEADS for short) " IN-LINE " with the entry's code portion (TAILS for short). In order to achieve this desired result the parameter field address, PFA, was altered so as to point to the actual PFA. This pointer field is appropriately named the PFAPTR. In an effort to minimize changes to existing code the FORTH word PFA, nothing more than "PFAPTR @", was left in the E-FORTH kernel.

The 16K E-FORTH EPROM located in the first slot of the PRIMER is divided into two 8K sections. The lower 8K is where the E-FORTH HEADLESS kernel resides. The upper 8K contains various extensions to the kernel such as the ASSEMBLER, EDITOR etc. which, while invaluable tools, are of little use in a TARGET APPLICATION. In addition, the upper 8K houses the HEADS which use their PFAPTR fields to reference associated TAILS in the lower 8K. When creating a TARGET APPLICATION EPROM the lower 8K of the E-FORTH EPROM is copied into the lower 8K of the TARGET EPROM and the upper 8K of the TARGET EPROM is then available for the actual TARGET APPLICATION.

The value contained in the USER variable HEADLESS is used by the E-FORTH system to determine which mode of compilation the programmer has chosen.

0 in HEADLESS --- NORMAL COMPILATION
1 in HEADLESS --- HEADLESS COMPILATION
2 in HEADLESS --- TARGET COMPILATION

A 0 is placed in HEADLESS by COLD or upon power-up and only selected words will alter the contents of HEADLESS. When HEADLESS contains a 0 IN-LINE HEADS and TAILS are generated. However, when HEADLESS contains a 1 or a 2 TAILS are generated in one area of memory referenced through DP and HEADS are generated in another referenced by DP/. Once you have decided to create a TARGET APPLICATION adherence to the following steps should guarantee success.

1. Develop your Application Source Code using In-Line (default mode 0) with Variable Allocation words ALLOT, VARIABLE, etc.

2. Load your program in normal mode. The last word loaded throughout these steps should perform all required initialization. Put another way, only one word (the last word loaded) should be required to invoke your program

3. Run your program to insure that everything is working properly.

4. Debug and repeat steps 2 and 3 if necessary.

5. Check the value of HERE using the FORTH sequence " HERE HEX . " and jot it down for later reference.

6. Now you have to decide where to store the HEADS for a dry run in the HEADLESS mode. This is where you'll utilize the

value you made note of in step 5. That value minus 8002H is the total amount of IN-LINE RAM your compiled program uses. The total IN-LINE RAM used cannot exceed 29D0H bytes on the PRIMER. A good location to store the HEADS is 9800H. This will leave you with 1D00H (approx.) bytes for compiled TAILS and CD0H (approx.) bytes for compiled HEADS.

7. Once these checks have been performed you are ready to dry run your code in HEADLESS mode. Simply enter the following sequence . . . HEX 9800 H/T scr# LOAD . . . substituting the number of your initial screen for scr#.

8. Again check to see that your program is functioning correctly. If it isn't, debug and repeat step 7.

9. If the invoking word (the last word defined) contains a conditional branch back to the operating system (i.e. BEGIN. . .?TERMINAL UNTIL ;), The invoking word should be changed to prevent SYSTEM CRASHES in a TARGET APPLICATION. This can be accomplished by using an absolute branch (i.e. BEGIN. . .AGAIN ;) in your invoking word.

10. If it is working correctly, you are ready to create your TARGET EPROM. After selecting the compilation locations of HEADS and TAILS (H/T), use the E-FORTH word LOAD/ to compile your code in HEADLESS mode as well as stuff the AUTO-START VECTOR located at 8000H with the Code Field Address (CFA) of the last word defined on the last screen loaded. Using either the EMAC EPROM PROGRAMMER BOARD (EMAC PART #E020-8) or EMAC SUPPORT SOFTWARE and your own EPROM programmer you are ready to burn your TARGET EPROM.

****NOTE**** WHEN USING LOAD/, WORDS CREATED BY THE USER MAY NOT BE EXECUTED OFF A SCREEN UNLESS DYNAMICALLY CHANGING FROM TARGET TO INLINE TO TARGET ETC.

TARGET COMPILATION AND THE USE OF VECTORS

A special procedure is required when vectors are to be incorporated into an application program that is designed to be Target compiled and burnt into EPROM. This special procedure computes the correct CFA of any word that is vectored, regardless of the compilation mode selected. The following example demonstrates the typical assignment of a vector not using this technique.

```
.
.
: HEX 3 FFFF TIMER0

: FLASHIT PTAIN 0FF XOR PTAOUT DB RETURN ;

: STUFF_VECT ' FLASHIT CFA INT7.5 ! ;
.
.
```

This program flashes the digital output LED's with each timer interrupt. It will function correctly in compilation modes 0 and 1, but not in Target mode 2. The special procedure previously mentioned will function correctly for all three cases. The following example demonstrates this procedure.

```
.
.
.
HEX 3 FFFF TIMER0

: FLASHIT PTAIN 0FF XOR PTAOUT 0B RETURN ;

' FLASHIT CFA CONSTANT 'FLASHIT_ADD

: STUFF_VECT FLASHIT_ADD INT7.5 ! ;
.
.
```

Chapter 8

PROGRAM VOCABULARY

Ok, you've finally worked all the bugs out of your FORTH program and now you want to burn an application EPROM with it. The PROGRAM VOCABULARY contains all the words required to burn application EPROMS or to backup an existing EPROM.

EMAC's EPROM programmer (E020-7) allows for a variety of EPROM type and voltages. The smallest EPROM supported is a 2764 (8K x 8) and largest EPROM supported is 27512 (64K x 8). Each EPROM type is given a number in order to identify it. The 6 EPROM type numbers supported are as follows:

```
TYPE #1     27512 ( 64K x 8 ) EPROM WHICH PROGRAMS AT 12.5 VOLTS.
TYPE #2     27256 ( 32K x 8 ) EPROM WHICH PROGRAMS AT 12.5 VOLTS.
TYPE #3     27128 ( 16K x 8 ) EPROM WHICH PROGRAMS AT 12.5 VOLTS.
TYPE #4     27128 ( 16K x 8 ) EPROM WHICH PROGRAMS AT 21.0 VOLTS.
TYPE #5     2764  (  8K x 8 ) EPROM WHICH PROGRAMS AT 12.5 VOLTS.
TYPE #6     2764  (  8K x 8 ) EPROM WHICH PROGRAMS AT 21.0 VOLTS.
```

PROGRAM WORDS

BURN (addr1 addr2 n1 n2 --- ff) or (addr1 addr2 n1 n2 --- addr3 addr4 b1 b2 tf)

burns the n1 bytes at addr1 (addr1 must be >= 8000H) into the n1 EPROM locations starting at addr2. n2 specifies one of the above EPROM types with its corresponding programming voltage. a flag is left on the Top of the stack indicating the success (0) or failure (1) of the burn. If the burn was not successful then the EPROM contents b2, the RAM contents b1, the EPROM location addr4, and the RAM location addr3 are left on the stack following the true flag.

DWNHEX (addr1 addr2 addr3)

Send the contents of memory starting at addr1 and ending at addr2 to the COM1 serial port in Intel Hex format. Address addr3 determines the starting address used in creating the Intel Hex file.
**** NOTE **** This word is usually used in conjunction with support communications software running on a PC.

ERASECHK (addr1 n1 --- ff) or (addr1 n1 --- addr2 b1 tf)

checks to see if all locations from 0 to addr in the EPROM are erased. n1 specifies one of the above EPROM types with its corresponding programming voltage. A flag is left (0 if all locations are erased, 1 if not) on the Top of the stack. If the EPROM was not erased then the EPROM contents b1 and the EPROM location addr2, are left on the stack following the true flag.

TBURN (---)

burns a complete application program into an EPROM (27256 only) by first copying the lower 16K of your E-FORTH EPROM into the lower 16K of the target EPROM and then burning your compiled (using LOAD/) code from address 8000H to HERE into the second 16K of the target EPROM. Also performs ERASECHK and VERIFY. If an error is encountered an appropriate message is displayed.

TDUMP (---)

TDUMP transmits a compiled (using LOAD/) application program, addresses 0H to 3FFFH and 8000H to HERE over the current Communication Channel. Addresses 8000H to HERE and their respective checksums are adjusted to form a contiguous transmission. TDUMP is used in conjunction with EMAC support software to create an INTEL HEX FORMAT disk file. This file can then be used to burn the target EPROM with most commercial programmers.

UPHEX (addr1 --- b addr2)

Receives an Intel Hex file from COM1 serial port. The Intel Hex information received is placed in memory starting at addr1. The information is checked for accuracy as it is received and a status byte b is placed on the stack. b breaks down as follows:

bit#	description
0	a record type of 1 was encountered (an end record)
1	A checksum error has taken place
2	A non-ASCII character was encountered
3	An escape character was encountered
4-6	not used
7	The last line of the Intel Hex file was encountered. (Normal termination)

Address addr2 was the last address used when receiving the Intel Hex file. The UPLOAD can be aborted at any time by receiving an 'esc' character.

**** NOTE **** This word is usually used in conjunction with support communications software running on a PC.

VERIFY (addr1 addr2 n1 n2 --- ff) or (addr1 addr2 n1 n2 --- addr3 addr4 b1 b2 tf)

checks to see that the n1 bytes of the E-PAC memory at addr1 match the n1 bytes at EPROM addr2. n2 is used to specify the EPROM type and its corresponding programming voltage. If they match a zero is left on the Top of the stack and if not a one is left on the Top of the stack. If there was an error in verification then the EPROM contents b2, the RAM contents b1, the EPROM location addr4, and the RAM location addr3 are left on the stack following the true flag.

AN EXAMPLE OF CREATING AN APPLICATION EPROM

Let's say you have a FORTH program stored on screens 1 through 5 in the RAMDISK of your PRIMER. Also let's assume you have determined that the headless compiled code and total variable/array storage takes up 800H bytes. Finally, we will assume you have an erased 32K (27256) EPROM. To create the application EPROM you would only have to enter the following line of FORTH code. . .

HEX 9000 H/T 1 LOAD/ PROGRAM TBURN <return>

and in just a few minutes you would have your APPLICATION EPROM.

**** NOTE **** TO INSURE DESIRED RESULTS :

1. WHEN INSERTING OR REMOVING EPROMS, MAKE SURE THE EPROM BURNER POWER LED IS OFF.

2. WHEN PERFORMING AN ERASECHK, VERIFY OR BURN MAKE SURE THE EPROM TYPE WAS CORRECTLY ENTERED.

Chapter 9

OVERLAYS

E-FORTH supports the use of OVERLAYS both during the development of code and during the formation of APPLICATION EPROMs. OVERLAYS provide the user a means of compacting frequently used code in a pre-compiled form for rapid invocation or to simply speed up the LOAD process. As mentioned above, an E-FORTH OVERLAY is a pre-compiled program or section of a program that is stored on a screen or screens of the RAMDISK. Since the code is stored as OBJECT CODE rather than SOURCE CODE a significant savings in storage space and LOAD time can be realized.

In order to maximize the benefits OVERLAYS provide, the SOURCE CODE from which the OVERLAY is to be constructed should be AT LEAST 1 screen in length. Indeed, to fully maximize the benefits, the OBJECT CODE (compiled form) derived from the SOURCE CODE should approach, but not exceed, a screen boundary i.e. 1K, 2K , 3K etc. in length. This is due to the fact that OVERLAYS will always occupy at least 1 screen (1K) irrespective of the amount of OBJECT CODE that is produced.

OVERLAYS are ideal for utilities that are used often, extensions to the FORTH vocabulary and to precompile tested portions of SOURCE CODE in order to accommodate larger programs and speed up LOAD time. While a useful tool in the creation of APPLICATION EPROMS, OVERLAYS are not supported in APPLICATION EPROMS. This is due to the fact that the source code for OVERLAYS, resident in the E-FORTH EPROM, is destroyed when an APPLICATION EPROM is made.

USE OF OVERLAYS

E-FORTH only supports the use of one OVERLAY at a time because OVERLAYS are always LOADED at 8002H and the invocation or LOADING of a second OVERLAY WILL OVERWRITE THE EXISTING OVERLAY AND ANY EXISTING COMPILED DEFINITIONS LOCATED AT 8002H. IN OTHER WORDS THE LOADING OF AN OVERLAY SHOULD PRECEDE THE LOADING OF SOURCE CODE DEFINITIONS IN ORDER TO AVOID LOSING PREVIOUSLY COMPILED SOURCE CODE.

An existing OVERLAY, however, may be expanded at any time because the LOADING of an expanded OVERLAY will merely overwrite the existing overlay.

CREATION OF AN OVERLAY

Type " 5 ALLOT " This ALLOTTED memory is used by the system to keep track of the DICTIONARY POINTER etc.

LOAD program Since OVERLAYS are COMPILED CODE they may not be LISTED like a normal FORTH screen. We suggest the following method for keeping track of your OVERLAYS. Reserve line one of the screen you are creating an OVERLAY from, for a colon definition containing a comment. In other words, on line 1 define a word COM which contains a comment. . .

```
1 : COM ." ( TEST OVERLAY 1 OF 1 ) " ;
```

Placing the comment in a ." --- " insures the comment will remain in text form after being compiled and the FORTH word INDEX will detect the use of parentheses providing an OVERLAY that may be INDEXED.

Save OVERLAY The FORTH word S-FORTH accomplishes this task for you. It expects the screen number where you wish the OVERLAY to be stored on the stack.

<scr#> S-FORTH

S-FORTH responds with . . .

SCREEN n LAST SCREEN USED (Y/N)

This lets the user know which screens will be used and the number of screens it will use in the construction of the OVERLAY. If, from this information, you determine that the OVERLAY is going to OVERWRITE code you wish to keep, enter " N " and the OVERWRITE will not occur. If everything is OK enter " Y " and the OVERLAY will be in place.

HEADLESS OVERLAYS

HEADLESS OVERLAYS are also supported and provide the user with even greater memory savings and allow the user the capability of selectively removing words (by removing their HEADS) from the VLIST thereby creating his own vocabulary for a given utility. The SOURCE CODE for a given utility may contain a great deal of defined words, not all of which are useful to the user when directly referenced. The utility designer may opt to remove the HEADS of these words (remove them from the VLIST and direct access) in order to simplify the utility or " hide " words.

CREATION OF A HEADLESS OVERLAY

Separate HEADLESS words from those you wish to appear in the VLIST and have access to.

By including in your SOURCE CODE for the words HEADLESS OVERLAY, adjustments to the FORTH variable HEADLESS, you are able to create some words with HEADS and others without.

SOURCE CODE	RESULTS IN
0 HEADLESS !	normal compilation of words (WITH HEADS)
1 HEADLESS !	HEADLESS compilation of words (NO HEADS)

example:

```
line#
1   : COM ." ( OVERLAY EXAMPLE 1 OF 1 )" ;
2   0 HEADLESS ! : TTT . . ; 1 HEADLESS !
3   : UUU . . . ; 0 HEADLESS !
4   : SSS . . . ;
```

In the example, the words TTT and SSS are compiled with HEADS and word UUU is compiled without HEADS.

Type " 5 ALLOT " This ALLOTTED memory is used by the system to keep track of the DICTIONARY POINTER etc.

Type " addr H/T " This command will determine where (addr) HEADS to words compiled HEADLESS (UUU) will be compiled. The system will verify your choice by outputting the addresses for the compilation of HEADS and TAILS.

LOAD the program Make sure you're LOADING the program in the BASE you want to be in.

Cut the links Selectively link together the words you wish to appear in the VLIST. This is done by adjusting the Link Field Addresses (LFA) of those words. In our example we

only want the words TTT and SSS to appear in the VLIST. Since we wish to directly link TTT to TASK we will execute the following . . .

```
' TASK NFA ' TTT LFA !
```

. . . and TTT is now directly linked to TASK. Further we don't wish UUU to appear in the VLIST (i.e SSS directly linked to TTT) so . . .

```
' TTT NFA ' SSS LFA !
```

. . . results in the user being allowed access to only the words we want them to have access to. Again these are the only words in our example OVERLAY that will appear in a VLIST.

Save the OVERLAY

The FORTH word S-FORTH accomplishes this task for you. It expects the screen number where you wish the OVERLAY to be stored on the stack.

```
<scr#> S-FORTH
```

S-FORTH responds with . . .

```
SCREEN n LAST SCREEN USED (Y/N)
```

This lets the user know which screens were used and the number of screens used in the construction of the OVERLAY. If, from this information, you determine that the OVERLAY is going to OVERWRITE code you wish to keep, enter " N " and the OVERWRITE will not occur. If everything is OK enter " Y " and the OVERLAY will be in place.

LOADING (INVOKING) OVERLAYS

To load or invoke an OVERLAY simply type the number of the screen on which the OVERLAY begins followed by the FORTH word X-FORTH.

**** CAUTION **** THE LOADING OF AN OVERLAY WILL OVERWRITE EXISTING DEFINITIONS THAT WERE PREVIOUSLY COMPILED. REMEMBER OVERLAYS ARE ALWAYS LOADED AT 8002H.

ADDING TO (EXPANDING) OVERLAYS

Suppose we have an existing OVERLAY located on RAMDISK screens 1 and 2 as well as SOURCE CODE, located on screens 3 through 8, we wish to add to the existing OVERLAY.

```
example:    COMMAND            ACCOMPLISHES
            -------            ------------

            1 X-FORTH          LOAD or invoke the existing OVERLAY

            3 LOAD       LOAD the SOURCE CODE on top of ( memory ) the existing
                               OVERLAY.

            Optionally         See above example
            cut desired
            links

            1 S-FORTH          Save the expanded OVERLAY
```

DELETING OVERLAYS

Suppose we have two utilities, U-ONE and U-TWO, contained within one OVERLAY and we wish to delete U-TWO from the OVERLAY. Let's also assume U-TWO was the last utility LOADED when the OVERLAY was created.

IF U-TWO WAS CREATED WITH HEADS . . .

1. LOAD/INVOKE (USING X-FORTH) THE OVERLAY

2. ENTER THE FORTH SEQUENCE: FORGET <name> WHERE <name> IS THE FIRST WORD DEFINED IN U-TWO

3. SAVE (USING S-FORTH) THE NEW ABBREVIATED OVERLAY

IF U-TWO WAS CREATED WITHOUT HEADS . . .

. . . AND THE FIRST WORD COMPILED IN U-TWO WAS COMPILED WITH IT'S HEAD

1. USE THE ABOVE PROCEDURE

. . . AND THE FIRST WORD COMPILED IN U-TWO DIDN'T HAVE A HEAD COMPILED

2. RECONSTRUCT THE OVERLAY FOR U-ONE FROM SCRATCH. THIS IS DONE TO AVOID HAVING CHUNKS OF UNREFERENCABLE COMPILED CODE TAKING UP VALUABLE MEMORY.

Using the same scenario as described above, the only way to delete OVERLAY U-ONE is to reconstruct U-TWO from scratch.

USING OVERLAYS IN LARGE PROGRAMS

The following is an example of using OVERLAYS to create a program, the SOURCE CODE for which would be 10 screens in length, on an 32K RAMDISK.

1. Segment your SOURCE CODE into several logical blocks.

2. Enter by hand or download (using EMAC's support software) a block of SOURCE CODE on, for example, screens 6 through 8.

3. Load and debug this block.

4. Create an overlay of this block on screen 1.

5. Erase the SOURCE CODE (screens 6 - 8) which has now been duplicated in compiled form.

6. Enter / download the next block. TAKE CARE NOT TO OVERWRITE YOUR OVERLAY.

7. LOAD the new block on top of the existing OVERLAY.

8. Debug this expanded compiled code.

9. Expand the existing OVERLAY to include the new SOURCE CODE.

REPEAT STEPS 6 THROUGH 9 UNTIL YOUR ENTIRE PROGRAM (OR AS MUCH OF IT AS YOU DESIRE) IS IN THE OVERLAY. NOT ONLY WILL THIS PERMIT YOU TO LOAD LARGE PROGRAMS BUT IT WILL ALSO GREATLY DECREASE THE AMOUNT OF TIME IT TAKES TO LOAD YOUR CODE.

Chapter 10

THE E-FORTH CASE STATEMENT

The E-FORTH CASE syntax includes three conditionals as well as an optional DEFAULT (no match) condition. Most syntaxes possess only one conditional (=) and the optional DEFAULT condition. The structure of the E-FORTH CASE statement is depicted below :

value	CASE	header invoking CASE statement
	n1 OF= code1 ENDOF	if value = n1, code1 will be executed and control will pass to the ENDCASE statement. If not the next conditional is parsed, the optional DEFAULT code is executed, or ENDCASE executes.
	n2 OF< code2 ENDOF	if value < n2, code2 will be executed and control will pass to the ENDCASE statement. If not the next conditional is parsed, the optional DEFAULT code is executed, or ENDCASE executes.
	n3 OF> code3 ENDOF	if value > n3, code3 will be executed and control will pass to the ENDCASE statement. If not the next conditional is parsed, the optional DEFAULT code is executed, or ENDCASE executes.
	< optional code4 >	optional, unenclosed (i.e. no OF. . . ENDOF) DEFAULT sequence will be executed if and only if no previous conditional resulted in the execution of code. **** NOTE **** "ENDCASE" WILL EXPECT A VALUE ON THE STACK, SO MAKE SURE THAT THE EXECUTION OF THE OPTIONAL DEFAULT CONDITION LEAVES SUCH A VALUE.
	ENDCASE	DROPS the value being tested in the CASE statement and closes the CASE statement

TWO EXAMPLES

NEG_0_POS (n ---)

will print the value n and its relationship on a number line to zero.

```
: NEG_0_POS DUP
      CASE
      0 OF< . ." IS LESS THAN ZERO    " ENDOF
      0 OF= . ." IS EQUAL TO ZERO     " ENDOF
      0 OF> . ." IS GREATER THAN ZERO " ENDOF
      ENDCASE ;
```

CURRENT_BASE (---)

will print text indicating the current BASE, if the current BASE is 2, 8, 10 or 16. If the current BASE is not one of these a " NOT A NORMAL BASE " message is printed.

```
: CURRENT_BASE BASE @
         CASE
         2  OF=      ." BINARY            " ENDOF
         8  OF=      ." OCTAL             " ENDOF
         10 OF=      ." DECIMAL           " ENDOF
         16 OF=      ." HEXADECIMAL       " ENDOF
         ( default ) ." NOT A NORMAL BASE "
         ENDCASE ;
```

Chapter 11

TIMER AND COUNTER

The PRIMER comes equipped with a TIMER and EVENT COUNTER. This TIMER/COUNTER is resident in the 8155 I/O chip. By loading the TIMER/COUNTER with various TERMINATION COUNTS, the user can select time intervals ranging from 3 microseconds to 53 milliseconds. Upon reaching the TERMINATION COUNT limit, a 7.5 INTERRUPT is transmitted to the CPU and the TIMER/COUNTER may be programmed to either automatically reload itself with the TERMINATION COUNT and begin counting again or cease counting altogether.

ACCESS

E-FORTH provides access to the TIMER/COUNTER via the FORTH word TIMER0. The word TIMER0 expects a 8-bit timer command and a 16-bit timer mode/count length on the stack when it executes. The breakdown of the TIMER0 command byte is as follows. . .

DEC. BYTE	FUNCTION	
0	NOP	does not affect counter operation.
1	STOP	if TIMER has not started, NOP if TIMER is running, STOP COUNTING.
2	STOP AFTER TC	if TIMER has not started, NOP if TIMER is running, STOP counting after current TERMINATION COUNT is reached.
3	START	if TIMER is not running, load mode/count length and START TIMER if TIMER is running, START new mode/count length immediately after current TERMINATION COUNT limit is reached.

The 16-bit TIMER mode/count length value breaks down as follows. . .

BIT POSITION	FUNCTION
0 - 7	Least significant byte of count length
8 - 13	6 most significant bits of count length
14 - 15	TIMER mode

The TIMER mode breaks down as follows:

BIT 15	BIT 14	FUNCTION
0	0	Outputs a low signal during second half of count.
0	1	Outputs a square wave (the period of the square wave is equal to the programmed count length) and reloads automatically when the TERMINATION COUNT limit is reached.
1	0	Outputs a single pulse when TERMINATION COUNT is reached.
1	1	Outputs a single pulse and reloads automatically when TERMINATION COUNT is reached.

EXAMPLES

example 1 : `HEX 3 5400 TIMER0`

. . . will cause the TIMER/COUNTER to output a 60 Hz square wave.

example 2 : `HEX 3 C133 TIMER0`

. . . will cause the TIMER/COUNTER to output a pulse every millisecond.

Chapter 12

INTERRUPTS

The E-FORTH system supports all 8085 HARDWARE INTERRUPTS (5.5 , 6.5 , and 7.5) and one software interrupt (RST 7). Access to the 5.5 and 6.5 INTERRUPTS is through OJ1 (see Appendix A for jumper descriptions). The 7.5 INTERRUPT is tied directly to TIMER (IC 8155) output .

The 5.5, 6.5, 7.5, and RST 7 INTERRUPTS may all be vectored, thus the user may either patch in his own INTERRUPT HANDLER ROUTINE or simply poll HARDWARE INTERRUPTS through the USER variable STATUS.

INTERRUPT TRIGGERS

The 5.5 and 6.5 INTERRUPTS are LEVEL SENSITIVE, meaning they are able to be acknowledged by the processor when their signal is held high. The 7.5 INTERRUPT is RISING-EDGE SENSITIVE, meaning that when a transition from a low-signal to high-signal occurs on the 7.5 input line an internal flip-flop in the 8085 processor acknowledges the occurrence of the 7.5 INTERRUPT. The 7.5 input line, therefore, does not need to be held high. The TRAP INTERRUPT is also RISING-EDGE SENSITIVE but its pulse must be held high until acknowledged by the processor.

MASKING INTERRUPTS

The E-FORTH operating system enables (EI) and disables (DI) INTERRUPTS when necessary. THE USER SHOULD REFRAIN FROM USING THE EI AND DI INSTRUCTIONS (ALTHOUGH THEIR LOW-LEVEL USE IS TOLERATED). The correct method for enabling and disabling interrupts is either via the HIGH-LEVEL (FORTH) words INTMASK and RETURN or via the use of the LOW-LEVEL (ASSEMBLY LANGUAGE) SIM instruction. This method allows the user to selectively disable or enable the various INTERRUPTS. The INTMASK word is basically a HIGH-LEVEL SIM instruction. To use INTMASK simply place the desired value on the stack and enter INTMASK. The breakdown of the value is as follows :

BIT POSITION	FUNCTION
0	5.5 MASK
1	6.5 MASK
2	7.5 MASK
3	MASK SET ENABLE
4	RESET INTERNAL 7.5 FLIP-FLOP
5-15	NOT USED (0)

Placing a 1 in bit positions 0 - 2 will disable the corresponding INTERRUPT provided the MASK SET ENABLE bit (position 3) is also set to a 1. If the MASK SET ENABLE is not set high (to 1) the desired INTERRUPT(S) will not be disabled. A 1 placed in bit position 4 will reset the internal flip-flop of the 8085 which indicates the occurrence of a 7.5 INTERRUPT. This flip-flop, particular to the 7.5 INTERRUPT, will always be set high in the event of a 7.5 INTERRUPT even if the INTERRUPT has been masked. This internal Flip Flop should be reset prior to using the 7.5 INTERRUPT. The TRAP INTERRUPT is not maskable nor is it subject to any enable or disable instructions. (For further details refer to the 8085 USERS MANUAL)

examples :

```
HEX FF INTMASK      WILL DISABLE ALL VECTORABLE INTERRUPTS

HEX 0B INTMASK      WILL ENABLE THE 7.5 INTERRUPT ONLY
```

The word RETURN is basically a combined LOW-LEVEL SIM and RET. This word allows the user to alter the INTERRUPT MASK REGISTER, insuring that no INTERRUPTS will be acknowledged until the return from INTERRUPT part of RETURN has executed. The word RETURN is only to be used within an INTERRUPT handling colon definition which, when appropriately vectored, will process the occurrence of an INTERRUPT.

When using an INTERRUPT handler written in assembly language, the SIM instruction will enable and disable the various INTERRUPTS. To use the SIM instruction simply place the low-order byte of the value described above in the ACCUMULATOR and execute a SIM instruction. Additionally, instead of ending your LOW-LEVEL INTERRUPT handler with the normal "NEXT JMP" use "RETINT JMP". RETINT is an ASSEMBLER constant for the address of the LOW-LEVEL return from INTERRUPT.

VECTORING INTERRUPTS

Vectored execution, put another way, is merely indirect execution and is realized by storing the Code Field Address (CFA) of the word we want executed in an address. This address, which serves as a pointer, is called a VECTOR. The act of storing the CFA in the VECTOR is called VECTORING.

Upon power up, the E-FORTH system vectors all vectorable INTERRUPTS (5.5, 6.5, 7.5, and RST 7) to execute the FORTH word RETURN. For example, without vectoring any INTERRUPTS, the user could write a code definition which would cause the TIMER to output a square wave of 50 Hz., enable the 7.5 INTERRUPT and go about their business unaware that 50 times a second the 7.5 INTERRUPT is occurring. To fully utilize the INTERRUPT capabilities of the E-FORTH system the user should write their own INTERRUPT handling routines and insert the Code Field Addresses (CFA) of the handlers in their respective INTERRUPT vectors. To vector a handler . . .

1. Construct a colon definition to handle the occurrence of a particular INTERRUPT. BE SURE TO END THE COLON DEFINITION WITH THE FORTH SEQUENCE (High-Level) " RETURN ; " OR (Low-Level) " RETINT JMP END-CODE ".

2. Determine the CFA of the handler just defined. For instance, if the handler is named HANDLE, its CFA would be left on the stack as a result of the execution of the following FORTH sequence.

```
' HANDLE CFA
```

3. Store the CFA in its appropriate INTERRUPT vector. The vectors are the aptly named USER variables INT5.5, INT6.5, INT7.5, and RST 7.

4. Enable the appropriate INTERRUPT.

THE BELL EXAMPLE

CODE ENTERED	RESULT
HEX 3 C133 TIMER0	Sets TIMER0 to interrupt every millisecond.
VARIABLE HLDTIM	Create a variable to hold the number of interrupts.
0 HLDTIM !	Initialize HLDTIM to 0.
: TEST 1 HLDTIM +! HLDTIM @ 3E8 = IF BELL 0 HLDTIM ! THEN 1B RETURN ;	The interrupt handler. Rings the BELL every 1000 interrupts (1 sec) and reenables interrupts.
' TEST CFA INT7.5 !	Determines the CFA of TEST and stores it in the TIMER0 INTERRUPT vector.
1B INTMASK	Enable the TIMER0 INTERRUPT.

Immediately after the TIMER0 INTERRUPT has been enabled the bell or noisemaker associated with the host terminal should begin ringing about once a second. The user should notice that, except for the ringing, the system is functioning normally. To stop the bell from ringing enter. . .

```
0F INTMASK
```

**** NOTE **** DUE TO THE ASYNCHRONOUS NATURE OF INTERRUPTS, INTERRUPT HANDLERS ARE FORBIDDEN FROM RETURNING WITH DATA ON THE STACK. THE PASSING OF DATA FROM AN INTERRUPT HANDLER CAN BE ACHIEVED THROUGH THE USE OF VARIABLES, ARRAYS, ETC.

ALL REGISTERS ARE PUSHED UPON THE RECOGNITION OF AN INTERRUPT.

ANOTHER ALTERNATIVE

E-FORTH also supports INTERRUPT POLLING. This provides an easier method of implementing INTERRUPTS for use with tasks that are not time-critical. POLLING is accomplished by " fetching " and then examining the contents of the USER variable STATUS. The breakdown of the 16-bit value store in STATUS is as follows. . .

BIT POSITION	FUNCTION
0	5.5 INTERRUPT
1	6.5 INTERRUPT
2	7.5 INTERRUPT
3-7	UNUSED (0)
8-15	RESERVED FOR SYSTEM USE

A 1 in any of the three low-order bit positions of STATUS indicates the occurrence of an INTERRUPT. It is left up to the user to read the contents of STATUS, handle the occurrence of an INTERRUPT and reset bits that have been set high. The resetting of the bits is critical because subsequent occurrences of the same INTERRUPT will again set the corresponding bit high.

example:

CODE ENTERED	RESULT
STATUS @	Place the 16-bit contents of STATUS on the data stack. **** NOTE **** THE FORTH WORD "C@" COULD BE USED IN PLACE OF THE FORTH WORD "@" BECAUSE THE BITS CRUCIAL TO DETERMINING OCCURRENCE OF "INTERRUPTS" ARE ALL LOCATED IN THE LOW-ORDER BYTE OF STATUS.
4 AND	Leave a flag on the data stack which indicates whether or not a 7.5 INTERRUPT has occurred. flag = 1 7.5 INTERRUPT OCCURRED flag = 0 NO 7.5 INTERRUPT
STATUS DUP @ 3 AND SWAP !	Reset the 7.5 bit in USER variable STATUS.

These three steps may be combined by the user into a colon definition which checks STATUS for an occurrence of a 7.5 INTERRUPT, leaves an appropriate flag on the stack and resets the 7.5 bit position in STATUS.

```
: INT7.5CHK STATUS C@ 4 AND STATUS DUP C@ 3 AND SWAP C! ;
```

Chapter 13

AUTOLOAD

E-FORTH is equipped with an AutoLoad feature. This feature enables Screen #1 to be LOADed automatically upon power-up or the execution of the FORTH word "COLD". To enable the AutoLoad feature, simply flip dip switch #7 to the on position.

By mixing AutoLoad and Overlays Autoexec.bat type functions can be implemented. Also this mixture is ideal for using application programs without the bother of burning a EPROM. For instance, say you have created an application overlay (see chapter on Overlays) called "name" and it resides beginning on RAMDISK Screen #2. The command line 2 X-FORTH "name" present on Screen #1 would cause "name" to execute almost immediately upon power-up with the AutoLoad feature enabled. A simple Reset will terminate the execution of the program and restore the Operating System. Often its desired to run the application program after a reset as well as upon power-up. This can be accomplished by storing 0 in the Warm/Cold Start word located at address AF0AH prior to invoking "name".

Chapter 14

TROUBLESHOOTING

If you don't receive the E-FORTH prompt immediately, it could be caused by one or more of the following problems. These may appear simplistic in nature yet they occasionally happen.

NO POWER

Check to make sure the power switch from your power supply is in the "on" position and make sure power is fed to the unit via power jack J1. The power supply must be a filtered DC power source from seven (7) to ten (10) volts. Make sure that on the power supply that you use the tip is positive and the sleeve is negative. Current consumption will be less than 500 mA. (350mA to 420mA typically), but it is advisable to use a power supply that can provide 500 mA.

Note: Be careful to observe correct type of voltage and polarity, or else the PRIMER may be seriously damaged!

SERIAL COMMUNICATIONS PROBLEMS

See Chapter 1 to verify that the communication cables are connected properly and consult the user's manual of your host terminal or computer to insure proper wiring of the RS232 line (TXD, RXD etc.).

1) JAMMED UART
For some reason the UART may be jammed. Pressing the terminal's RETURN key might free the troublesome UART. If not you may need to power-down then power-up the PRIMER.

2) INCORRECT BAUD RATE
Make sure that the baud rate of the PRIMER matches that of the terminal or PC it is communicating with.

RUNNING PROGRAM

The PRIMER may be executing a program when communication is attempted. A power-down --- power-up sequence should yield the E-FORTH logon message. Also, the PRIMER will perform an Autoload of screen 1 if dip switch 7 is in the ON position. Turn the switch off if you don't want this to occur.

MISSING IC

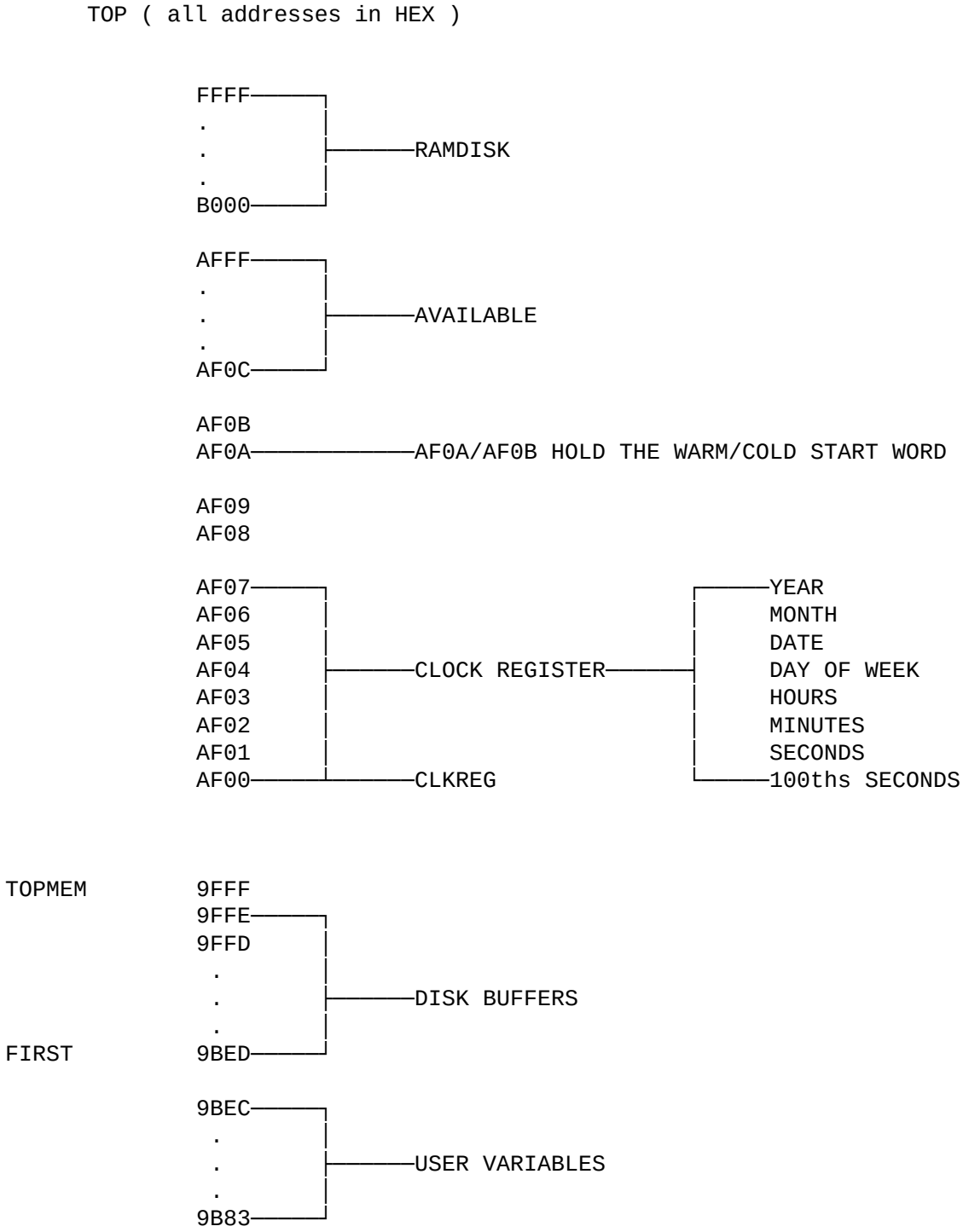
The E-FORTH EPROM or an APPLICATION EPROM must reside in the 1st slot of the PRIMER and an 32K RAM in the 2nd.
NOTE: If the application is not designed to produce a prompt or allow communications, they won't occur.

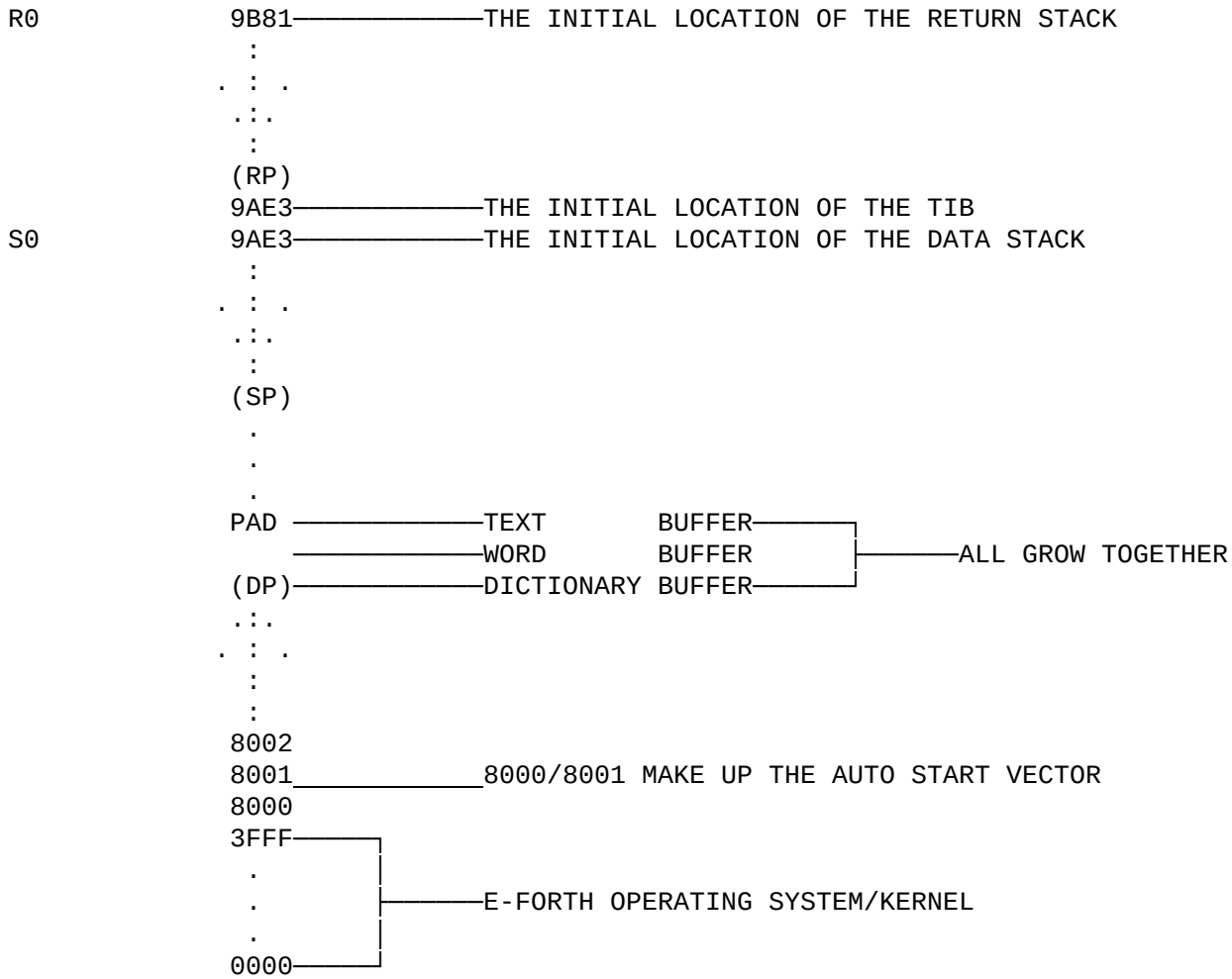
MEMORY JUMPERS

Option jumpers OJ2 and OJ3 must both be in position B.

Chapter 15

MEMORY MAP




R0
9B81
THE INITIAL LOCATION OF THE RETURN STACK
(RP)
9AE3
THE INITIAL LOCATION OF THE TIB
S0
9AE3
THE INITIAL LOCATION OF THE DATA STACK
(SP)
PAD
TEXT BUFFER
WORD BUFFER
(DP)
DICTIONARY BUFFER
ALL GROW TOGETHER
8002
8001
8000/8001 MAKE UP THE AUTO START VECTOR
8000
3FFF
E-FORTH OPERATING SYSTEM/KERNEL
0000

APPENDIX A

JUMPER DESCRIPTIONS

JUMPER DESCRIPTION

JP1 This allows the selection of one of the following baud rates: 300, 600, 1200, 4800, 9600 and 19,200.

OJ1 This is used to select the sources for the 8085's RST 5.5 and RST 6.5 interrupt inputs. The RST 5.5 interrupt pin is connected to the 8279 interrupt request line when there is a connector between pins 4 and 5, or, if a connector is between pins 3 and 4, it is connected to the 8251 receiver ready line. The RST 6.5 interrupt pin is connected to the 8251 receiver ready line when there is a connector between pins 2 and 3. Putting a connector between pins 1 and 2 connects RST 6.5 to +5v. This jumper can also be used to connect RST 5.5 and RST 6.5 to external sources. Pin 2 of the jumper is connected to RST 6.5 and pin 4 is connected to RST 5.5.

0J2 This jumper selects the EPROM size. Position 'A' allows an 8 or 16K EPROM to be placed in slot 0 and position 'B' allows a 32K EPROM to be placed in the slot.

0J3 This selects one of the two memory maps which are as follows:

```
POSITION 'A' MEMORY MAP
   SLOT 0            0000 TO 3FFF
   SLOT 1            4000 TO BFFF
   8155 RAM          C000 TO FFFF

POSITION 'B' MEMORY MAP
   SLOT 0            0000 TO 7FFF
   SLOT 1            8000 TO FFFF
   8155 RAM          (not accessible)
```

APPENDIX B

I/O AND MEMORY ADDRESS DESCRIPTIONS

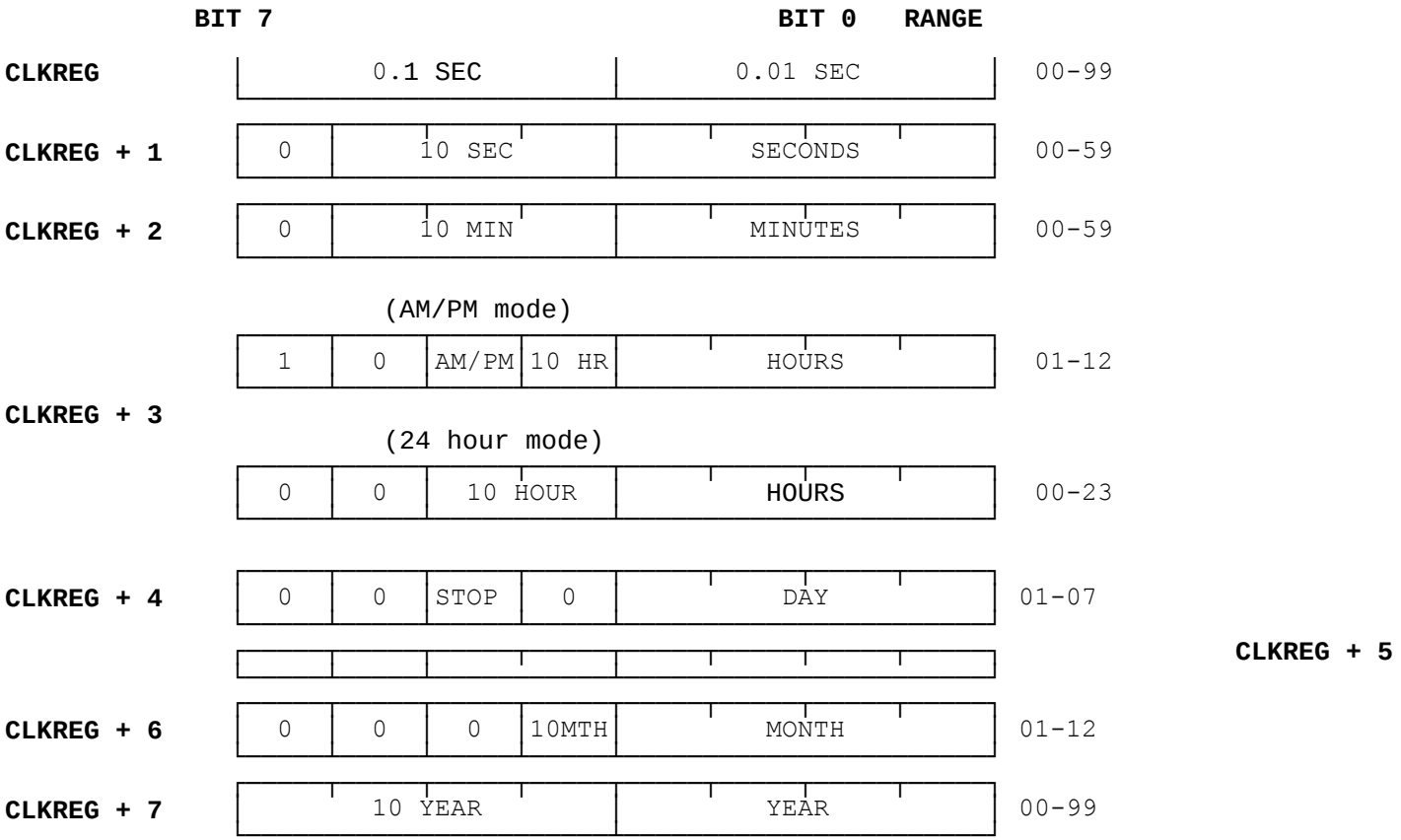
REFERENCE	I/O ADDRESS	DESCRIPTION
8251 DATA REGISTER	80 H	DATA INPUT/OUTPUT
8251 CONTROL REGISTER	81 H	CONFIGURATION
8155 CONTROL REGISTER	10 H	CONFIGURATION
PORT A	11 H	OUTPUT PORT
PORT B	12 H	INPUT PORT
PORT C	13 H	ANALOG OUTPUT PORT
TIMER LOW	14 H	LOW ORDER TIMING BYTE
TIMER HIGH	15 H	HIGH ORDER TIMING BYTE & CONTROL
EXPANSION I/O	C0 - FF H	EXPANSION CONNECTOR

MEMORY ADDRESS			DESCRIPTION
For OJ3 position A			
0000 H	-	3FFF H	EPROM SLOT
4000 H	-	BFFF H	32K RAM SLOT
C000 H	-	FFFF H	256 BYTES IN 8155
For OJ3 position B			
0000 H	-	8000 H	EPROM SLOT
8000 H	-	FFFF H	32K RAM SLOT

APPENDIX C

REAL TIME CLOCK DESCRIPTION

The optional real time clock provides timekeeping information in BCD including hundredths of seconds, seconds, minutes, hours, day, date, month and year information. The date at the end of the month is automatically adjusted for months with less than 31 days, including correction for leap years. The real time clock operates in either 24-hour or 12-hour format with an AM/PM indicator. The data at CLKREG will be stored in the real time clock as follows:



If bit 7 of address CLKREG + 3 is 0 the clock will be in 24 hour mode after WRSCLK is executed. If it is 1 then AM/PM mode is selected and bit 5 of address CLKREG + 3 will select AM or PM (PM is selected if bit 5 is 1). When changing from AM/PM mode to 24 hour mode and vice-versa you must change the hours to match the selected mode. Once the hours are correct, the real time clock will maintain the correct hour for the selected mode.

If bit 5 of address CLKREG + 4 is set to 1 and WRSCL is executed, the real time clock will be stopped. The clock may be restarted by resetting the bit to 0 and executing WRSCL.

Following are a couple of example Forth screens:

```
 Screen # 0
    +---------------------------------------------------------------+
 0  |( INFO FOR USING THE REAL TIME CLOCK WITH THE E-FORTH OS.       |
 1  |  THE CONSTANT CLKREG MARKS THE FIRST CLOCK REG. CLKREG + 7     |
 2  |  MARKS THE LAST CLOCK REG .  TO READ THE CLOCK EXECUTE THE WORD|
 3  |  RDSCLK AND READ THE CLOCK REGS.  TO SET THE CLOCK WRITE THE ) |
 4  |( CLOCK REGS. AND EXECUTE THE WORD WRSCLK. EXAMPLES OF READING  |
 5  |  AND WRITING THE CLOCK ARE LISTED BELOW. )                     |
 6  |                                                               |
 7  |  HEX                                                          |
 8  |: .BS . 08 EMIT ;                                              |
 9  |: TIME_PRINT HEX RDSCLK CR CR CLKREG 7 + C@ CLKREG 5 + C@      |
10  |  CLKREG 6 + C@ .BS 2F EMIT .BS 2F EMIT .BS 5 SPACES CLKREG 1+ |
11  |  C@ CLKREG 2+ C@ CLKREG 3 + C@ .BS 3A EMIT .BS 3A EMIT .BS ;  |
12  |                                                               |
13  |                                                               |
14  |                                                               |
15  |                                                               |
    +---------------------------------------------------------------+

 Screen # 1
    +---------------------------------------------------------------+
 0  |DECIMAL                                                        |
 1  |: NUMBER? 0 0 ROT DUP 1+ C@ 45 = DUP >R + CONVERT C@ 32 =      |
 2  |  IF R> IF NEGATE THEN 1 ELSE R> DROP 2DROP 0 THEN ;           |
 3  |                                                               |
 4  |: #PROMPT BEGIN QUERY BL WORD HERE NUMBER? UNTIL DROP ;        |
 5  |                                                               |
 6  |: SET_TIME HEX                                                 |
 7  |  CR CR ." YEAR > " #PROMPT CLKREG 7 + C!                      |
 8  |  CR CR ." MONTH > " #PROMPT CLKREG 6 + C!                     |
 9  |  CR CR ." DATE > " #PROMPT CLKREG 5 + C!                      |
10  |  CR CR ." DAY OF WEEK > " #PROMPT CLKREG 4 + C!               |
11  |  CR CR ." HOURS > " #PROMPT CLKREG 3 + C!                     |
12  |  CR CR ." MINUTES > " #PROMPT CLKREG 2+ C!                    |
13  |  CR CR ." SECONDS > " #PROMPT CLKREG 1+ C!                    |
14  |  CR CR ." 10ths OF SECONDS > " #PROMPT CLKREG C! WRSCLK CR     |
15  |  TIME_PRINT ;    TIME_PRINT                                   |
    +---------------------------------------------------------------+
```