FORTH for the Complete Idiot

by C. H. Ting , PhD

OFFETE ENTERPRISES, INC.

1984

First Edition, September 1983 (First Printing)
FEBRUARY 1984 (Second Printing)

****

****

UNITED STATES COPYRIGHT OFFICE

REGISTRATION NUMBER

TX 1-254-698

TX TXU

EFFECTIVE DATE OF REGISTRATION

21 DEC 1983

Month Day Year

Printed in the United States of America

by

Offete Enterprises, Inc.

1306 SOUTH "B" STREET
SAN MATEO, CALIFORNIA 94402
TEL: (415) 574-6250

CONTENTS

APPDENDIX

THE FORTH MYTH

FORTH is a language of mystic quality. The learning proceess for many FORTH programmers can be describe only in religious terms. However, what I consider to be the greatest myth about FORTH is that FORTH IS AN ELITIST'S LANGUAGE. This myth is perpetuated because most people think that FORTH is difficult to learn, FORTH programs are difficult to read and to comprehend, and that a FORTH computer is expensive, at least comparing to most other personal computers running BASIC.

My opinion is that FORTH IS THE IDIOT'S LANGUAGE which is best suited for common people who are being shuffled into this age of computer literacy, kicking and screaming. Why? Because FORTH is actually a very simple language to learn. It has the simplest grammar and the fewest syntax rules than any other computer languages. Above all, FORTH can be extended and modified so that you can use your own language or terms to converse with it. I was asked what is the qualification of a person to learn FORTH. My answer was that he must have at least one finger, to hit keys on a keyboard. Well, what else does he need to learn FORTH?

What he needs are a good book on FORTH and a cheap and reliable computer which has a FORTH in it. FORTH is small and eminently ROMmable. A FORTH computer can be built simply and cheaply just as those BASIC personal computers. Lately, we have seen a number of implementations like Jupiter ACE, and VIC 20 FORTH ROM Cartridge. I am offering here my implementation for Apple II in the form of a ROM card. It works inside an Apple of minimal resources, namely 16 Kbytes of RAM memory. Now we have also a number of good books on FORTH, notably 'Starting FORTH' by Leo Brodie. With a good book and a good FORTH computer, I think all we common people, or idiots, should be able to learn FORTH and use it to enjoy the company of a computer without much help from high priests of the FORTH religion.

This booklet is my attempts to describe this FORTH ROM CARD to people who owned a very minimal Apple II computer without a disk drive. If you can afford a disk drive, you will be much better off by purchasing a regular FORTH system on a disk, which allows you much more freedom to explore the capability of this language. The best place where this FORTH ROM CARD can be of substantial contribution is in primary and secondary schools where the use of disks increases administrative burdens and reliability is of primary concerns. Like BASIC systems residing in ROM, this FORTH ROM CARD system is reliable and doesn't mind to be abused.

My hope is that this FORTH system will provide you a simple tool to get acquainted with this exciting language and give you many hours of fun. In time you will outgrow this system and move on to other more expesive and more sophisticated FORTH implementations to be productive in your own profession. By then, you will be able to choose the best system for your specific requirements and specifications.

INSTALL THE FORTH ROM CARD

For a person not having the detailed knowledge of computers, this is probably the most difficult task that I will ask you to do. Install the FORTH ROM CARD all by yourself! It is not as difficult as you might think, but you do have to read very carefully the instructions and followed them precisely so that you will not hurt yourself or damage the computer or the ROM CARD. The instructions, however, are very simple and easy to follow.

1. Turn off the power to the computer and remove the plug from the wall socket. You will get inside the computer enclosure. The chances that you will be electrocuted is nil. The chances that you will get an electric shock by touching parts inside the computer is also nil. It is good practice to turn off power to an equipment before your open the enclosure. You might drop a screwdriver or a nail into the computer and short out some circuitry.
2. Open the top cover of your Apple II.
3. Observe that there are 8 long printed circuit card slots at the back of the Apple II main circuit board. This FORTH ROM CARD does not need any of the circuit cards and I assume that you have none in your Apple. If you are using one or more of the circuit cards, like disk interface, printer interface, 16 K RAM card, etc., I advice you to remove them all. Gently pull these cards straight up out of the slots and store them away.
4. Take the FORTH ROM CARD and align the connector with one of the circuit card slot. Push the card firmly into the slot and make sure that the card is seated securely into slot. The FORTH ROM CARD will work in any of the eight slots. However, I prefer that you will install it in the leftmost slot or Slot 0.
5. Replace the top cover of Apple II. This completes the installation of FORTH ROM CARD.
6. Insert the power plug into a wall socket so that Apple II will received electricity when we need it.
7. Turn on your monitor or the TV set. Make sure that the output cable from the Apple II is connected to the input of the monitor or TV, and that the connection is correct.
8. Turn on your Apple II now. You will hear a beep from the speaker in Apple, and see a sign on message appearing on the monitor or TV screen:

```
FORTH-79   2.2
OK
```

The installation procedure is simple and I am sure you will have no problem in doing it if you follow the instructions above. You should at least read the relevant sections in the Apple II manual to familiar youself with the parts inside the Apple enclosure and identify the circuit card slots. That's about all you really have to know about the Apple in order to install and use the FORTH ROM CARD.

TROUBLE SHOOTING THE APPLE II

What could you do if the FORTH 79 sign on message didn't appear on the monitor or TV screen? Well, you are in trouble. However, don't panic yet. There are quite a few thing you can do to identify where the trouble is. Here is a list of thing that you must check and make sure that all parts are working before blaming FORTH or Apple:

1. Make sure that power is supplied to the monitor or TV, and that the monitor or TV is turned on.
2. If you are using a TV set, the video output from Apple must be modulated by a RF modulator. The RF modulator convert the video signal so that it can be received by a TV set through one of its channels. Try channal 3, 4, or 13 because these are channels used most often by modulators you bought from local hobby shop.
3. Turn the TV channel select switch around, searching the correct channel used by the modulator.
4. Check that the Apple receives power and that the on-off switch at the back of Apple is in the ON position.
5. If the sign on message does not appear when Apple is turn on, turn off power to Apple, re-seat the FORTH ROM CARD, and make sure that the card is inserted all the way into the Slot. The conducting fingers on the ROM CARD must be aligned with the mating connectors in the slot. Turn Apple on to see if the sign on message appear on the screen.
6. If the FORTH 79 sign on message still does not appear, turn off power to Apple and remove the FORTH ROM CARD. Be absolute certain that the power is off when you pull out the ROM CARD. If the ROM CARD is removed while power is still on, components on the ROM CARD or on the Apple main board might be damaged. After the ROM CARD is removed, turn on power to the Apple and see if the Apple BASIC sign on message will appear on the screen.
7. If the Apple works in the BASIC mode without the ROM CARD, Apple is healthy. Insert the FORTH ROM CARD into another slot and try again. If you don't get the FORTH 79 sign on message, the ROM CARD is defective. Call the dealer or the manufacturer for service or replacement.
8. If the Apple BASIC sign on message didn't appear either, the problem is in the Apple. Apple sometimes does not work because many IC's develop bad contacts within the sockets, or are getting loose. Turn off power to the Apple. Push and re-seat all the socketed IC's on the Apple main board, particularly the CPU, the ROM chips storing the Applesoft BASIC, and the 16 K RAM chips.
9. Turn on power to Apple. If the Apple BASIC sign on message is not displayed on the monitor or TV screen, the Apple itself is having problems. Call your Apple dealer for service.
10. Now is the time to panic.

CHECK OUT FORTH LIKE AN EXPERT

You don't have to know much about FORTH in order to act like an FORTH expert in checking out or testing a FORTH computer. A few commands can exercise the FORTH computer very rigorously to demonstrate that the FORTH computer is in proper working conditions. If you just try these command on other people's FORTH computers, they will have a hard time to figure out whether you are an expert or an idiot.

1. The Carriage Return

Push the Carriage Return key, the Return key, or the down arrow key at the righthand side of the keyboard. The computer will echo 'OK' on the screen and move the cursor down by one line. Keep on tapping the Return key and a column of 'OK's will roll up on the lefthand side of the screen. The computer went through thousands of instruction cycles in receiving one tap on the Retrun key and sending back the 'OK' message. When the FORTH computer says 'OK', it is truely OK.

2. The VLIST command

Type VLIST on the keyboard and follow it by a Return. The computer will print a very long list of words on the screen. Many of these words appear to be regular English word. Many others have strange punctuation characters embedded in them. VLIST asks the FORTH computer to spell out all the commands or instructions that it understands. This command is very useful in finding out what words were defined in this system and the sequence by which the words are stored in the computer memory.

3. The DUMP Command

You can cause the computer to pour its guts out using this DUMP command. However, you must tell the computer which part of its memory you want to examine. You have to type the starting address and the number of bytes you wanted to examine. Thus the following commands:

```
0 100 DUMP
```

will display on your screen the contents of memory from address 0 to address 99. The numbers display on the screen may not have much meaning to you at this moment. Nevertheless, this is the method to examine the contents of computer memory.

When an expert examines the memory, most likely he wants the memory contents to be displayed in the hexadecimal number system in which each digit represents one of 16 values. The values are represented by the numerals 0 to 9, and the upper case letters A to F. You can switch the FORTH computer from the regular decimal mode to hexadecimal mode by the command:

```
HEX
```

Once the computer is in the hexadecimal mode, all the input numbers and the output numbers are converted according to the base of 16. Since the dictionary or the main body of FORTH program is located between memory D000 and EFFF, in hexadecimal, you can dump this range of memory to the CRT screen by these commands:

```
D000 2000 DUMP
```

You will see a huge amount of numbers scrolling across the CRT screen. Don't worry about what them mean. On the lefthand side of the screen, the memory contents are presented in Ascii characters. You might be able to find many command names in this column, showing how the dictionary is constructed. After the dumping exercises, it is a good practice to switch the computer back to the more familiar decimal number system by:

DECIMAL

4. Examine Disk Blocks

Most regular FORTH computers require at least one disk drive so that programs and data can be stored on the floppy disks. In this FORTH-79 system, instead of a physical disk, we use a large chunk of the RAM memory to simulate a disk as it is used in a regular FORTH system. I call it a pseudo disk. However, the methods to access data or program in this pseudo disk are identical to those used in regular FORTH systems. To display the contents of any disk block, you can type the command:

4 LIST

The number before LIST specifies the serial number of a block of disk memory to be displayed. You will see a screen full of letters or numbers displayed on the CRT screen arranged in 16 lines, each line consists of 32 characters. Since we haven't put in any meaninful data into this block, the data display are those happened to be in the memory when you turn on the computer. You can look at the contents of other disk blocks by giving different numbers before the command LIST.

Regular FORTH systems display disk data in 16 lines, each of 64 characters. Because the Apple screen can show only 40 columns of characters, this FORTH system was designed to show 32 characters per line. Shorter lines tend to force the user to write shorter codes, which is probably a desirable constraint.

Another interesting command is to list the index of a range of disk blocks. As an example, suppose we want to see the index of blocks 0 to 12. We can give the following commands:

0 12 INDEX

13 lines of index will appear on the CRT screen. In FORTH terminology, the index line is the first line of text data in a disk block. A comment is usually put there to serve as an index indicating the contents of this disk block. INDEX thus serves the function of DIRECTORY or CATALOG in the conventional operating systems.

The commands we mentioned above, VLIST, DUMP, LIST, and INDEX are only a few commnads among the entire set of commands available in this FORTH-79 system. They are used here as tools to check out the FORTH system because they produce visible results on the CRT screen. They are very useful commands during programming and testing and are used very often. You should get yourself thoroughly familiar with them and their functions, as the first step into this versatile and powerful computer language.

ENTER COMMANDS AND DATA

You control and use the computer by entering information into it through a typewiter-like keyboard. The FORTH computer will accept two and only two types of information, commands and data. Commands are English-like words and data are usually in the form of numbers. As shown in the last section, VLIST displays all the commands the computer can recognize and follow. To learn to use a computer is to learn its command set and use it to solve problems. We will discuss the command set in this FORTH computer at a slow and ordered pace to lead you to the level where you will be able to use it to solve simple programming problems.

Information is put into the computer by typing on the keyboard. Type in the word HELLO. The CRT screen will display whatever you typed in. If HELLO is misspelled, the spelling may be erased by pressing the backspace (<--) key. The backspace key will move one space to the left everytime it is pressed. This erases, letter by letter, anything that has been typed in. The spelling may then be corrected. This is the basic procedure to talk to the computer.

The computer will not respond to the information you typed in until the RETURN key is pressed. With HELLO typed in, now press the Return key. The computer prints "?" and drops down one line. The prompting cursor appears at the beginning of the next line. Computer responds only to exact commands, not to idle chit-chat. This FORTH computer has not been told how to respond to HELLO, so it prints what is called an error message. The error message "?" occurs when the information entered is not understood by the computer.

To distinguish the information you type into the computer and the responses the computer prints on the CRT screen, we will underline all the letters the computer prints. But remember, the computer responds to your commands only after you press the Return key. This Return is implied between your input and the computer's output and will not be shown in prints here.

RULE 1. COMPUTER RESPONDS TO INFORMATION TYPED IN ONLY WHEN THE RETURN KEY IS PRESSED.

RULE 2. COMPUTER RESPONDS TO TWO TYPES OF INFORMATION: COMMANDS AND DATA.

Let us do some exercises. Type in HELLO with a Return:

HELLO <u>HELLO ?</u>

The computer does not understand HELLO as a command. Therefore, it responds with an error message "HELLO ?". Now type in:

." HELLO"

After you enter the command and press Return key, the computer will respond:

<u>HELLO OK</u>

The computer was told to print the word HELLO, so it did. The command ." tells the computer to print the following letters upto but not including the sign " . This quotation mark indicates to the computer the end of the letter sequence to be printed. The

"OK" after HELLO is printed by the computer to indicate that it had successfully carried out the command. The prompting cursor drops down to the beginning of the next line, ready for the next line of commands.

The computer can do wonderful things with numbers. Afterall, manipulating numbers was the primary reason for computers to be invented. Here we will use this FORTH computer like a pocket calculator or an adding machine. Type in the commands 4 5 + and then press Return:

```
4 5 + OK
```

The computer accepts the two numbers 4 and 5. + is a command telling the computer to add these two numbers. The computer did all these. However, it displayed only the OK message because it was not told what to do with the result of addition. Type in a period and follow by the Return:

```
. 9 OK
```

. is a command telling the computer to print the last result it held in its memory, which is the sum of 4 and 5, on the top of a stack. When the computer accepts a number you typed in, it put the number on the stack in a last-in-first-out fashion, similar to a card stack in which the topmost card is the first one to be used. In the last example, the + command takes the topmost two numbers, removes them from the stack, and put the sum back on the top of the stack. The . command removes the sum and prints it on the screen.

RULE 3. NUMBERS ARE ENTERED ON A STACK IN THE LAST-IN-FIRST-OUT FASHION.

RULE 4. COMMANDS REMOVE THEIR NUMERIC DATA FROM THE TOP OF THE STACK AND LEAVE THEIR RESULTS ON THE STACK.

When information is entered into the computer in one line, commands and numbers must be separated by at least one space. Spaces are used by the FORTH computer to isolate commands or numbers so that appropriate action can be taken. Letter sequences separated by spaces are generally called "words". Words must be either commands or numbers.

RULE 5. WORDS MUST BE SEPARATED BY ONE OR MORE SPACES.

Words not separated by spaces will be considered as a single word by the FORTH computer. For example, if you type:

```
4+5 .
```

The computer will reply:

```
4+5 ?
```

because the computer took in the sequence 4+5 as one word. It couldn't execute it because it was not a command nor a number. All the computer could do was to give you an error message.

Numbers and commands can be entered in a long line. However, you have to remember that they have to be separated by spaces and terminated with a Return key. Type in the following line:

```
1 2 3 4 5678 . . . . .
```

The computer will reply:

5678 4 3 2 1 OK
Notice that the last number entered was printed out first because numbers are pushed on to the stack and removed from the top.

Try the following line:

10 11 12 . . . .

We pushed three numbers on the stack and ask the computer to print out four numbers. What will the computer do? Most likely it will reply:

12 11 10 0 . ? STACK EMPTY
The computer printed out four numbers as you requested. After it printed out the last number, it checked the stack and found that you used more numbers than you put on the stack. It then printed the error message 'STACK EMPTY' to warn you that you did something wrong. Using more numbers than you put on the stack or leaving unwanted numbers on the stack are the most common mistakes made even by the most experienced FORTH programmers. Keeping track of the numbers on the stack requires attention and practise. This is a very important part in learning FORTH language.

MATH FUNCTIONS

The computer can perform many math functions when you give it the appropriate commands. The most commonly encountered math commands are: + for addition, - for subtraction, * for multiplication, and / for division. Each of these commands consumes the topmost two numbers on the stack, performs the math function, and leaves the result back on the stack so that the next command will make use of it. I will give you a number of examples to show you how the computer works out math problems. Type in:

```
3 5 * 6 + 4 2 * + .
```

The computer will respond with:

```
29 OK
```

What the computer did was multiplying 3 and 5 to get 15, add 6 to 15 to get 21, multiplying 4 and 2 to get 8, add 8 to 21 to get 29, and finally print out 29 on the CRT screen. A few variations in the sequence will yield different results:

```
3 5 * 6 4 + 2 * + .   35 OK
3 5 * 6 4 2 * + + .   29 OK
3 5 6 4 + 2 * + * .   75 OK
```

It is quite simple to follow the computer actions as the computer carries out the commands from left to right in the sequence as we gave to it. Using this scheme to do math calculations, no parenthesis is necessary and the calculation sequence is specified unambiguously. This is one of the many benefits in using a stack to hold numbers to be processed. There must be enough numbers on the stack for all the math commands. If the commands use more numbers than what had been put on the stack, the computer will issue an error message after it completes the calculations. For example:

```
3 5 * 6 4 + * +
```

The computer will print

```
STACK EMPTY
```

The user must know how many numbers are on the stack so that he will be able to do the correct calculations. The computer only checks whether the stack is empty or not at the end of a line. The user is responsible in making sure the right numbers are available at every stage of calculation.

FORTH computer operates on integers, normally ranging from -32768 to 32767. Numbers outside of this range are truncated and only the least significant 16 bits are preserved. For a very large number of applications with microcomputers, this range is sufficient, but one should always be careful in handling large numbers. Special commands and methods are available to handle large numbers and fractions. However, we must reserve these topics to more advanced texts.

Let me summarize the most useful math commands in a list:

+	(n1 n2 --- sum)	Add n1 and n2. Return sum.
-	(n1 n2 --- dif)	Subtract n2 from n1.
*	(n1 n2 --- prod)	Multiply n1 and n2.
/	(n1 n2 --- quot)	Divide n1 by n2. Return quotient.
MOD	(n1 n2 --- rem)	Divide n1 by n2. Return remainder
/MOD	(n1 n2 --- rem quot)	Divide n1 by n2. Return both remainder and quotient.
*/	(n1 n2 n3 --- quot)	Multiply n1 and n2, divide the product by n3.
*/MOD	(n1 n2 n3 --- rem quot)	Multiply n1 and n2, divide the product by n2. Return both the remainder and quotient.
MAX	(n1 n2 --- max)	Return greater of two items.
MIN	(n1 n2 --- min)	Return lesser of two items.
ABS	(n --- abs)	Return absolute value.
NEGATE	(n --- -n)	Return negative value.

*/ and */MOD are two special commands useful in doing scaling calculations. The intermediate product is a 32 bit double precision integer so that the range of product is not truncated as would be in using * and / separately. you will see a few examples using these compound commands later.

CONSTANTS AND VARIABLES

Constants and variables are storage locations in the computer memory which allow data to be stored and to be acted upon later. A constant will put its numeric value on the stack, and a variable will leave its address on the stack, so that subsequent commands can use the address to retrieve stored data or change the data value.

Constants are defined by a special command CONSTANT , which assigns a name and a value to a constant. The name thus defined becomes a new command. When this name is invoked, the value assigned to this name is pushed on the stack. The value assigned to a constant cannot be changed. A variable is defined by a command VARIABLE and is also assigned a name and an address. Invoking the name of a variable will put its assigned address on the stack. Other commands can then store data into this address, or fetch the data currently stored in this address. After an new value is stored in a variable, the old value of this variable is destroyed forever.

Examples:

```
5  CONSTANT  A    OK
VARIABLE  B    OK
```

To use a constant:

```
A  .
```

The computer will print:

```
5  OK
```

To use a variable:

```
10  B  !
```

The command ! , pronounced as 'store', stores the value 10 into the address assigned to variable B .

```
B  @  .    10  OK
20  B  !    B  @  .    20  OK
```

The command @ , pronounced as 'fetch', fetches the value stored in B. This value can be changed as ofter as desired. @ and ! are two extremely useful commands in FORTH. They allow us to examine the contents of any memory location and change their contents at will.

Mathematical calculations can be performed with the constants or variables supplying numeric data. Results of calculations can also be stored away in variables for later uses. Some examples are given below:

```
6  4  + CONSTANT  C    OK
3  5  *  C  2  *  +  .    35  OK
6  4  2  *  +  CONSTANT  D    OK
C  D  +  .    24  OK
C  D  +  B  !    OK
B  @  .    24  OK
```

DEFINING NEW COMMANDS

Up to this point, we use the computer on a one shot basis. One of the main advantages of computer is that it can execute a series of commands repeatedly. The process of writing such sets of commands is called programming. Programs or sets of commands can be very simple or incredibly complicated.

The FORTH computer is programmed by adding new commands to the existing command set provided in the ROM CARD. New commands are created or defined by a powerful command : , commonly called 'colon command'. : tells the computer to construct a new command and store it in its memory. : command is followed by a name given to the new command and a sequence of commands and numbers. The sequence is terminated by another command ; . The definition of a new command can span over several lines. During the defining process, pressing the Return key merely causes the prompting cursor to drop down a line. The regular 'OK' message will not appear until after ; command is encountered.

Type in the following lines and define several new commands:

```
: HELLO  ." HELLO"  ;  OK
: ADD  ." 4+5="  4  5  +  ;  OK
: CALCULATE  ." 2*(+5)="  2  4  5  +  *  .  ;  OK
```

The new commands are now stored in the memory of the computer. You can use VLIST command to see that their names are included in the list of command names in this FORTH computer. To execute a newly defined command, simply type the name and press Return key.

```
HELLO  HELLO  OK
ADD  4+5=  9  OK
CALCULATE  2*(4+5)=  18  OK
```

Suppose we want a program which will convert temperatures from Celsius scale to Fahrenheit scale and vice versa. Here are the new commands which will do it:

```
: C->F  9  5  */  32  +  ;  OK
: F->C  32  -  5  9  */  ;  OK
```

Now we can use them to do temperature conversions:

```
100  C->F  212  OK
0  C->F  32  OK
0  F->C  -17  OK
98  F->C  36  OK
```

The special command */ takes three numbers off the stack, multiplies the lower two numbers, and divides the product by the topmost number. This is an interesting command to allow us doing precise ratios with only integer numbers. Another good example is to calculate the circumference of a circle, given its diameter:

```
: CIRCLE  31416  10000  */  ;  OK
```

It takes the diameter value off the stack, multiplies it with the ratio of 31416 and 10000, which is 3.1416. Some tests are:

```
10  CIRCLE  .  31  OK
2000  CIRCLE  .  6283  OK
```

EDITOR

Everybody makes mistakes. When you type a line of information into the computer and found something in the line you want to change, there are two ways to do it:
1. Before you press the Return key, the line may be changed by backspacing to the place you wanted to change and retyping the line from there.
2. After the Return key is pressed, the line is stored in the computer's memory and you cannot change it conveniently. However, you can retype the correct line and enter it by pressing the Return key. A new version of a command will be entered on top of the old one. Only the last entered command will be effective.

The following sequence of commands and responses will demonstrate this second method:

```
: SO  ." SO WHATT"  ;   OK
SO   SO WHATT OK
: SO  ." SO WHAT"  ;  OK
SO   SO WHAT OK
: DON'T  ." DON'T SAY THAT"  ;   OK
DON'T   DON'T SAY THAT  OK
SO  DON'T   SO WHATDON'T SAY THAT OK
SO  CR  DON'T   SO WHAT OK
DON'T SAY THAT OK
SO  SPACE DON'T   SO WHAT DON'T SAY THAT OK
```

Commands in the same line are executed from left to right after the Retrun key is pressed. Executing SO and DON'T together prints two messages without a break. The command CR separates the two messages with a carriage return and the command SPACE puts a space between the messages. CR and SPACE are useful in formatting the output display.

To remove commands from the computer's memory, use the command FORGET followed by the name of the command to be deleted. The computer will delete this command together with all the commands defined after it. FORGET is useful in reclaiming the computer memory after you have completed the desired task.

```
SO   SO WHAT OK
DON'T   DON'T SAY THAT OK
FORGET  SO   OK
DON'T   ?
SO   DO WHATT OK
HELLO   HELLO OK
```

There were two versions of SO in the computer's memory. FORGET SO deleted the second version of SO with DON'T. Executing SO now thus recalls the first version of SO which is still in memory. Commands defined before SO are still in the memory and are accessible, like HELLO.

To write short commands and test them out, one can type in the definitions directly via the keyboard. For long definitions and projects which requires many many definitions, it is better to have the computer remember all the texts which can be changed and updated conveniently. Texts in computer's memory can be processed or compiled into executable commands similar to the

command texts you type in on the keyboard. These functions were programmed into the FORTH-79 ROM CARD in the form of an Editor. The Editor is called into service by the command EDITOR. After you type:

EDITOR OK

all the editing commands are available to you to create and modifiy texts in a special area of memory I call a 'pseudo disk'.

This 'pseudo disk' contains 24 Kbytes of memory, and is divided into 48 blocks, each having 512 bytes. To select any block in this pseudo disk to be used to store texts, you can give the following command:

1 LIST

Any number between 0 and 47 can be used before LIST, specifying the particular block you want to used. After you press the Return key, the computer will display on the CRT screen the current contents of this block. The display format is 16 lines of 32 columns of texts. Since the disk was not used before, a block of random characters will appear in the display. Let us first clear it so that we can do some useful work:

1 CLEAR OK
1 LIST

After clearing the whole block, 1 LIST will show us a clean screen for us to work on.

The following is a list of commands which are useful in inputing lines of texts into the disk block and modifying them. The lower case letters in front of some commands show the numbers required by the commands. The string xxxxx shows that a string of characters up to 32 characters will be used in the string processing command.

n T	Type the nth line of texts on screen. Make this line the current line.
P xxxxx	Put the string in the current line.
U xxxxx	Put the string under the current line. Subsequent lines will be pushed down. The 15th line will be lost.
X	Extract the current line from block. Subsequent lines move up by one line. The 15th line will be blank filled.
F xxxxx	Find the string starting at the current cursor. The cursor will be put after the found string.
D xxxxx	Find the string and delete it.
I xxxxx	Insert the string at the current cursor.
TILL xxxxx	Delete the text from the cursor till the end of the string in current line.
L	List the current block.
n LIST	List the nth block and make it current.
TOP	Move the cursor to top of block.
n CLEAR	Clear the nth block.
n1 n2 COPY	Copy block n1 to block n2.

These commands are sufficient to perform most of the editing tasks. The commands are nearly identical to those described in

Brodie's 'Starting FORTH'. However, there are several peculiar behavior in this implementation you have to know in order to use this editor successfully:

1. Each line is only 32 characters long. Longer strings will be truncated to 32 characters.
2. Do not leave blank lines between texts. The computer will not process texts below a blank line.
3. String commands must have strings following them. Blank string, a string command followed immediately by Return will not be processed according to the description in 'Starting FORTH'.

The following exercise shows you how to use these commands to write a program in a disk block.

```
1  LIST
1  CLEAR   OK
0  T
P  ( LARGE LETTER F )
U  : STAR   42  EMIT  ;
U  : STARS   0 DO START LOOP  ;   OK
U  : MARGIN   CR 30 SPACES   ;   OK
U  : BLIP   MARGIN STAR  ;   OK
U  : BAR   MARGIN 5 STARS  ;   OK
U  : F   BAR BLIP BAR BLIP BLIP CR  ;   OK
U  EXIT   OK
1  LIST
```

The program will be show on the CRT screen. If you find any error in it, use the T command to position the cursor at the beginning of that line, use the D command to delete the erroneous string, and use the I command to insert the correct string. In the old days, each FORTH system had its own editing command set and it was rather frustrating in getting the editor to work according to your wish. This editor command set here in the FORTH-79 ROM CARD should be familiar to most FORTH users due to the popularity of 'Starting FORTH'.

After the texts are typed in and checked, you can use the following commands to compile the commands into the memory:

```
1  LOAD   OK
```

You can use the VLIST command to verify that all the commands you defined in block 1 had been compiled into memory. To see what you have accomplished, you should type in:

```
F
```

and a big F character will be show on your screen.

RULE 6. A STRING COMMAND MUST BE FOLLOWED BY THE STRING IT USES.

LOGIC COMMANDS

Logic is the basis of digital computer. In FORTH logic is represented by numbers. Although numbers may have many different values, in logic we distinguish only two types: zero and non-zero. When numbers are used in logic operations, they are called flags. If it is non-zero, it is called a true flag. If it is zero, it is called a false flag. Logic commands often produce flags according to some math operations which produce results having only two different states, like equal, greater than, and less than. The result of an logic operation is represented by zero if false and by one if it is true.

The logic commands defined in this FORTH-79 ROM CARD are summerized in the following list:

<	(n1 n2 --- f)	True if n1 is less than n2.
>	(n1 n2 --- f)	True if n1 is greater than n2.
=	(n1 n2 --- f)	True if n1 is equal to n2.
0<	(n --- f)	True if n is lesss than 0.
0>	(n --- f)	True if n is greater than 0.
0=	(n --- f)	True if n is equal to 0.
AND	(n1 n2 --- n3)	Bitwise logical AND.
OR	(n1 n2 --- n3)	Bitwise logical OR.
XOR	(n1 n2 --- n3)	Bitwise logical exclusive OR.

The logic commands are simple but very important for computers because they allow a computer to make logic decisions. The best example is in the IF-ELSE-THEN structures and the BEGIN-UNTIL structures, where the computer must decides which execution path it must follow between two choices. IF uses the top number on the stack as a flag. If the number is non-zero, a true flag, the commputer will execute the true clause between IF and ELSE. Otherwise, the computer will execute the false clause between ELSE and THEN. UNTIL does very much the same thing. If the top number on the stack is non-zero or true, UNTIL will terminate the loop. Otherwise, it will loop back and repeat the repeat clause.

Examples in using these logic commands are:

```
1  3  >  .    0 OK
-1  10  <  .    1 OK
-101  101  =    0 OK
9  9  =    1 OK
0  0=    1 OK
-1  0=    0 OK
-2  0<    1 OK
2  0>    1 OK
7  4  AND    4 OK
1  2  OR    3 OK
1  3  XOR    2 OK
```

Note that the commands AND, OR, and XOR operate on whole 16 bit numbers. The logic operations are performed to every bit pairs on the top two numbers on the stack. FORTH was originally designed for instrumental control applications. In these applications, performing logic operations on single bits and

representing true/false conditions with whole 16 bit numbers are quite wasteful. These commands operate on 16 bit numbers are very convenient and more efficient.

STACK COMMANDS

Most modern computers use stacks as temprorary storage of numbers which will be used and forgotten after their functions are served. Using stacks save memory space and allow many neat tricks to use the computer efficiently. High level languages, however, hide this facility from the user because mismanagement of the data on stack is the easiest way to make the computer produce garbage, or worse, to crash the computer. FORTH, on the other hand, let the user have full control over the stack in an effort to simplify the language structure and speed up the execution. Open up the stack facility to the user place a great burden on the user to manage it correctly. The reward is that the user is given a very powerful tool to interact with and to control his computer.

The purpose of the stack is to store numbers temporarily for the appropriate commands to use. Since most math commands use the topmost numbers on the stack in some specific orders, you will have to make sure that the right numbers are available for the right command. When you work on real problems, you will find that half of the time, even if you can get the required numbers on the top of the stack, they are in a wrong order as required by the command. The stack commands allow you to rearrange the ordering of the topmost items on the stack so that math operations can be performed correctly. Here is the list of the most useful stack command:

DUP	(n --- n n)	Duplicate top of stack.
DROP	(n ---)	Discard top of stack.
SWAP	(n1 n2 --- n2 n1)	Exchange top two stack items.
OVER	(n1 n2 --- n1 n2 n1)	Copy second item to top.
ROT	(n1 n2 n3 --- n2 n3 n1)	Rotate third item to top.

Using these commands, you can rearange the top three items on the stack to any order you want. If we have 1 2 3 on the stack, ROT will rearrange them to 2 3 1 . Can you order them in to 3 2 1 ?

The command sequence to do that is:

```
: <ROT  ROT TOT SWAP  ;   OK
1  2  3  <ROT   OK
.  .  .   1 2 3 OK
```

That's fine. How about the fourth item on the stack, or the fifth item? There are commands that allows you to dig very deep into the stack:

PICK	(n --- n1)	Copy the nth item to top.
ROLL	(n ---)	Rotate nth item to top.
>R	(n ---)	Move top item to 'return stack'.
R>	(--- n)	Retrieve item from 'return stack'.
R	(--- n)	Copy top of return stack onto stack.

DUP is thus equivalent to 1 PICK , and OVER is equivalent to 2 PICK. SWAP is equivalent to 2 ROLL , and ROT is equivalent to 3 ROLL.

Now, let me tell you a secret: FORTH actually has two stacks. What we have been talking about is only one of them, the data stack where numeric data are stored. The other stack is called 'return stack' because it stores return addresses which allows a FORTH command to call other FORTH commands, and keeps track of the execution sequence. I like to tell you more about this return stack but it will take too much space and time. It is sufficient at this moment to tell you its existence and what we can do with it.

The only way you can access the return stack is by the three commands, >R , R> , and R . >R copies the top item on the data stack and pushes it onto the return stack, and R> does the reverse. R copies the top of return stack and pushes it onto the data stack, without removing this item from the return stack. Thus we can move data from the data stack and store them temporarily on the return stack, getting them out of the way as we uses the data items below the third item on the data stack. However, you have to be reminded that the FORTH system uses the return stack to keep track of the execution addresses, and you should never mess up the information on the return stack. If you mess up the data stack, the computer doesn't mind it at all. It just happily gives you the wrong results, or at the worst, an error message. If you mess up the return stack by leaving garbage on it, you will most surely crash the FORTH system! Try it a couple of times and you will appreciate this warning.

RULE 7. RESTORE THE RETURN STACK WITHIN A COLON DEFINITION AND WITHIN EVERY LEVEL OF DO-LOOP.

Don't warry too much about crashing the system. In this FORTH 79 ROM CARD system, all the system resides in ROM memory. When you crash the system, all you have to do is push the Reset key or cycle the power off and on. You can get back to FORTH immediately. It is idiot-proof.

DO-LOOP also uses the return stack to store the index and limit values. Therefore, it is important that you restore the return stack before the LOOP command.

INPUT AND OUTPUT COMMANDS

FORTH is an interactive computer system, meaning that you can operate the computer in a conversational manner. You type in some commands and data, the computer will process them immediately and ask you for more commands. FORTH supplies you the entire set of commands which it uses to converse with you so that you can use them to handle any input and output tasks.

Some of the often used commands handling keyboard input are:

KEY	(--- c)	Read key and leave Ascii on stack.
EXPECT	(addr n ---)	Accept n characters from keyboard an store them at addr.
?TERMINAL	(--- f)	Return true if a key is presses.

KEY command will stop the computer and wait for you to type a key on your keyboard. After you type a key, the corresponding Ascii character code is pushed on the data stack. This is the most elementary command to ask user to respond in the conversation with the FORTH computer. If you want to input a line of information, the best command to use is EXPECT . You specify how many characters you want and also the address in memory which this line of information will be stored.

An example here demonstrates the use of KEY which loops around to monitor the key board. Whenever you press a key, the computer will print the Ascii code on CRT screen. This is useful if you are not familiar with the Ascii codes. When you press the Return key, the loop will be terminated.

```
: ASCII   BEGIN KEY DUP . 13 = UNTIL ;   OK
ASCII
```

Now you can press any key and the computer will tell you its Ascii code. When you get tired of it, just press Return and you will be back in FORTH again.

?TERMINAL examines the keyboard interface and returns a flag without waiting for your response. If you had pressed any key on the keyboard, the flag returned would be true. If the computer couldn't detect any activity on the keyboard, a false flag would be returned. This command is often used in an indefinite loop so that the user can terminate the loop by pressing a key on the keyboard.

The output commands are summerized in the following list:

EMIT	(c ---)	Output Ascii from stack.
CR	(---)	Output a carriage return.
SPACE	(---)	Type one space.
SPACES	(n ---)	Type n spaces.
TYPE	(addr n ---)	Type n characters from addr.
.	(n ---)	Print n with one trailing space.
U.	(n ---)	Print n as an unsigned number.
.R	(n1 n2 ---)	Print n1 right justified in a field of n2 columns.

EMIT is the fundamental output command. Other commands are for your convenience. If you have a long line of characters stored in memory, you can used TYPE to print them out. TYPE is the reverse of EXPECT . I am showing here an example you can use to get a message from keyboard and later print it out on demand.

```
VARIABLE  NAME  20  ALLOT   OK
:  NAME?   NAME  20  EXPECT  ;   OK
:  .NAME   NAME  20  TYPE  ;   OK
NAME?   LUKE SKYWALKER   OK
.NAME   LUKE SKYWALKER      OK
```

From now on, everytime you type .NAME , the computer will answer LUKE SKYWALKER . You can change the name by executing NAME? . The computer will halt and wait for you to type a string up to 20 characters, and then store this string into the memory starting at NAME .

. is the command we have been using all the time to display the topmost number on the stack. U. prints this number as an unsigned number with a range from 0 to 65535. It is useful to display address in the computer. The addresses in the top half of the computer memory will be displayed as negative numbers if you display them using the . command. U. will print them as addresses.

.R allows you to format the numeric output. You specify the width of the field you want the number to be displayed. The number will appear in this field, right justified. An example of its use was shown in the printing of the multiplication table in the next section.

STRUCTURED PROGRAMMING

The : command is the most powerful command in FORTH, because it allows us to construct many many new commands using the commands supplied in the FORTH system and the commands we defined previously by ourselves. Very large and complicated problems can be solved by designing and building a command set which best describes the problem and arrives at its solution. So far, you have been told only to write very simple : definitions in which commands are executed in a linear sequence. FORTH does have many special commands which are used to build structures inside an : definition, so that execution sequence can be controlled at your will. Let me first summerize these special commands in a list and then explain how to use them with a few examples.

```
( f ---)  IF  true-clause  ELSE  false-clause  THEN
( lim ind ---)  DO  repeat-clause  LOOP
( lim ind ---)  DO  repeat-clause  ( n ---) +LOOP
BEGIN  repeat-clause  ( f ---)  UNTIL
BEGIN  repeat-clause-1  ( f ---)  WHILE  repeat-clause-2  REPEAT
```

When IF is executed inside a : definition, the top item on the stack is testes as a flag. If this flag is true or non-zero, the true-clause, or the commands between IF and ELSE are then executed. If the flag is false or zero, the false-clause, or the commands between ELSE and THEN will be executed. In either case, the execution continues on after THEN. An example to use this structure is:

```
:  AGE  21  >  IF  ." ADULT "  ELSE  ." CHILD "  THEN  ;   OK
30  AGE   ADULT OK
10  AGE   CHILD OK
20  AGE   CHILD OK
```

21 > performs a comparison between the top item on the stack with 21 and leaves a true flag on the stack if that number is greater than 21. It leaves a false flag on the stack if that number is equal to or less than 21. IF picks up the flag and decides which of the two alternate paths will be executed.

This structure can be very complicated. Let's show another example:

```
:  POUNDS
   DUP  100  >  IF
      200 >  IF  ." TOO FAT"  ELSE  ." NORMAL"  THEN
   ELSE  ." TOO THIN"  DROP
   THEN   ;   OK
```

It can now be tested:

```
80 POUNDS   TOO THIN OK
120 POUNDS   NORMAL OK
300 POUNDS   TOO FAT OK
```

In this definition, if the weight given on the stack is greater than 100, a second test will be performed to select a correct response. The structures can thus be nested and many levels of tests can be carried out before some action is taken.

Computers are at their bests when you give them a set of commands to be executed repeatedly. They just love to loop

around and around. Generally, loops are classified into two classes: the definite loops and the indefinite loops. In definite loops, the number of times a sequence of commands will be executed is known beforehand and we can give the computer an upper bound and a lower bound, and commad it to loop a fixed number of times. The indefinite loop means that the number of repetition is not known beforehand, and it is decided depending upon some conditions tested when the program is actually running.

The DO-LOOP structure is designed to perform the definite loops. When the execution sequence gets to DO , DO will take the top two items off the stack and uses them as the upper and lower bounds of repetition. The topmost number on the stack is the lower bound, and the second number is the upper bound. The repeat clause will then be executed to LOOP . LOOP will then increment the lower bound value by one and compare it with the upper bound. If the lower bound value or the index is equal or greater than the upper bound, the loop will be terminated and execution proceed to the commands after LOOP . If the index is smaller than the upper bound, the repeat clause will be executed and the looping will continue.

+LOOP is very similar to LOOP. The difference is that +LOOP will use the top number on the stack as the increment, so that the loop can be stepped through at a faster pace. Let's use an example which prints the multiplication table to explain how the loops are constructed.

```
: PRODUCTS   10  1  DO  DUP  I  *  4  .R  LOOP  DROP  ;   OK
1 PRODUCTS     1   2   3   4   5   6   7   8   9 OK
2 PRODUCTS     2   4   6   8  10  12  14  16  18 OK
: TABLE   10  1  DO  CR  I  PRODUCTS  LOOP  ;   OK
TABLE
   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
   3   6   9  12  15  18  21  24  27
   4   8  12  16  20  24  28  32  36
   5  10  15  20  25  30  35  40  45
   6  12  18  24  30  36  42  48  54
   7  14  21  28  35  42  49  56  63
   8  16  24  32  40  48  56  64  72
   9  18  27  36  45  54  63  72  81 OK
```

The command I recalls the current index value and put it on the stack to be used in a loop. I can only be used in this way inside a loop structure. The above example can be written in a single definition:

```
: TABLE  10 1  DO  CR 10 1 DO  I  J  *  4  .R  LOOP  LOOP  ;  OK
```

where I recalls the index of the inner loop while J recalls the index of the next outer loop. This definition will print exactly the same table as the previous one.

To print an algebraic series of numbers, we will have to use the +LOOP command:

```
:  ALGEBRAIC   DO  I  .  DUP  +LOOP  DROP  ;  OK
2  20  0  ALGEBRAIC   0 2 4 6 8 10 12 14 16 18 OK
3  20  0  ALGEBRAIC   0 3 6 9 12 15 18 OK
5  20  0  ALGEBRAIC   0 5 10 15 OK
```

The first number used is taken as the incremental value by +LOOP in the loops. 20 and 0 are taken by DO as limit and index.

In the BEGIN-UNTIL loop structure, the repeat clause will be repeatedly executed until a test condition is true. This test is done by UNTIL at the end of the loop. UNTIL tests the topmost item on the stack. If the number is false or zero, the loop body will be repeated. If the number is true or non-zero the loop will be terminated and the commands following UNTIL will the executed. In the BEGIN-WHILE-REPEAT structure, the test is done by WHILE . It takes the top item off the stack. If the number is true, the second clause will be executed and the loop will be repeated. If the number is false, execution will continue after REPEAT, thus terminates the loop.

To print a geometric series of numbers:

```
: GEOMETRIC   BEGIN  DUP  .  OVER  *  DUP  100  >  UNTIL
              DROP  DROP  ;  OK
3  1  GEOMETRIC   1 3 9 27 81 OK
2  1  GEOMETRIC   1 2 4 8 16 32 64 OK
```

In the DO-LOOP and BEGIN-UNTIL structures the repeat clause will be executed at least once because the test for termination is at the end of the loop. BEGIN-WHILE-REPEAT allows us to exit the loop in the middle.

ASSEMBLER

FORTH is fast, compact, and totally structured. Therefore, it is quite unlikely that you will use assembly or machine codes to solve your programming problems. Only in the most time critical applications one will resort to assembly coding to squeeze the uttermost out of a computer. Nonetheless, because an assembler is very easy to write in FORTH, and I did have extra space in the ROM memory of this FORTH 79 ROM CARD, I implemented a 6502 assembler in this system, just in case you might want to experiment with it.

You should first try to solve you problem entirely in high level FORTH, using mainly the : definitions. After you solve the problem and you want to speed up the execution, you should analyze your program and try to identify where the computer spends most of its time. These are called critical routines, which are ideal targets for assembly coding. If you just code these routines in assembly, you will realize a quantum jump in the performance of the program. If this is still not fast enough for your application, I would claim that you had chosen a wrong computer. Instead of optimizing you codes on a slow computer, you are much better off starting look for a faster computer for your application.

This assembler was implemented according to the one published by William F. Ragsdale, in FORTH Dimensions, Vol. III, p. 143-150. Instead of repeating what he had done and said, I am reproducing his paper in the appendix for your reference. Happy coding!

ERROR MESSAGES

If you make a mistake in typing or in writing a definition, the computer usually will abort the operations and return the system to accept another command with an error message like MSG# 1 or something like it. The number tells you what type of error the computer encountered. The meaning of the error numbers are in the following list:

MSG#	0	? (Command not recognized)
MSG#	1	Stack empty
MSG#	2	Dictionary full
MSG#	4	Command redefined
MSG#	17	Compilation only
MSG#	18	Execution only
MSG#	19	Structures not paired
MSG#	20	Definition not finished
MSG#	21	Protected dictionary
MSG#	22	Use only when loading
MSG#	24	Declare vocabulary

Refer to this table if you receive an error message. Error numbers are usually rather frightening if you do not know what they mean. It is a good practice to let the computer print out the actually error message instead of an error number. This can be done by storing these error messages in disk blocks 4 and 5. If you clear these two blocks and used the editor commands to put the messages in proper place, i. e., MSG# 0 should be stored in the 0th line in block 4, MSG# 1 in the line 1, and so forth. MSG# 17 should go to line 1 in block 5, MSG# 18 to line 2 , etc.

After all the error messages are put in right places, you can type in the following commands:

```
1 WARNING ! OK
```

These commands store a one into a system variable named WARNING. From now on, when an error is detected by the computer, the computer will go to block 4 or 5 to find the message and print it on the CRT screen instead of the message number. It makes the computer look much more friendly and human-like. You have the freedom to turn off this message output mechanism by the following commands:

```
0 WARNING ! OK
```

which reverts the computer to the condition at the cold start. Since our disk is a pseudo disk residing in the computer's memory, you will lose these messages you put in blocks 4 and 5 when you turn the computer off.

SAVE PROGRAMS ON TAPE

One of the reasons in producing this FORTH system in ROM memory is to avoid the costly disk drive, which is taking for granted by most FORTH implementation. The problem in not having a disk drive is that you cannot save your programs very conveniently on a tangible medium. However, we can still make use of the cassette interface built in the Apple computer to save the programs you spent so much time in developing. It is not very convenient. If you have ever used the cassette tapes on Apple, you would also know it is not the most reliable medium for recording large programs. Well, we just have to make use of it and stay with it if that all we've got.

What we will have to do is to get into the Monitor program in the Apple computer and use the commands there to save programs and reload them back into the Apple. The command to enter into the Apple Monitor is:

MON

and the computer will respond with an asterisk character, indicating that it is in the Monitor mode:

*

You can now type in monitor commands to do all sorts of wonderful things if you know how. It is helpful if you have an Apple Reference Manual which has all the information on the monitor commands. Here we only need a few to accomplish what we wanted, that is to save our FORTH programs on tape. At this point, I have to assume that you know a little bit about the hexadecimal number notations, which you will have to use to talk to the Apple in its monitor mode.

The FORTH pseudo disk starts at location $6000 in the memory of Apple and extends to the limit of the RAM memory, which is at the location $BFFF. Assuming that we want to save the entire contents of this pseudo disk, the commands to be given is as follows:

*6000.BFFFW

Before you press the Return key, prepare the tape recorder and the connection cables, rewind the tape to the starting position, and press down both the PLAY and RECORD buttons. After the header of the tape clears the recording head, press the Return key on Apple and the recording will proceed until the entire pseudo disk is dumped on to the tape.

To read program stored on tape back into the pseudo disk in the Apple memory, the commands are:

*6000.BFFFR

You have to prepare the tape recorder carefully. Adjust the volume and tone, make the correct connections, and rewind the tape to the beginning. Press the PLAY button on the recorder and then press the Return key on Apple to start the reading process. If everything works, after about 3 minutes you will hear the Apple beeping and the * appearing on the CRT screen. Many times the Apple will show a read error message. You will have to start all over again.

After the program is loaded back into the Apple, you can return to FORTH by pressing the Reset key on the Apple keyboard or type the following command in the monitor mode:

```
*D000G
FORTH 79  2.2
OK
```

Bingo! You are back in FORTH again. Now you can list out the blocks and see if the programs you recorded on the tape are in place.

If the tape recorder works reliably with the Apple, this is a great way to do personal computing. The pseudo disk in FORTH has a capacity of 24 Kbytes, which is really very large for FORTH programs. You can store very large program with lots of data in this pseudo disk and back it up with cassette tape. For normal usage, you can load a large application into the pseudo disk and operate only on the pseudo disk. At the end of the task, the contents of the pseudo disk can be put back on tape for storage.

If you are not quite satisfied with the 24 Kbytes capacity in the pseudo disk, you can make use of the 16 Kbytes memory normally reserved for the high resolution graphic display in Apple. This of cause assumes that you will not use the high resolution graphic memory for graphic operation. The command to move the starting point of the pseudo disk from $6000 to $2000 is

```
-32  OFFSET  !   OK
```

After this command is executed, the first 32 blocks of pseudo disk are mapped into the high resolution graphics memory for you to use. To get back to the normal pseudo disk area, you must execute the following command:

```
0  OFFSET  !   OK
```