HesWare™

64 FORTH

By Tom Zimmer

Instruction Manual

64FORTH User's Manual

By Tom Zimmer

DISCLAIMER

64FORTH Dedication

This software is the result of the combined efforts of many people. Here is a list of some of these people and their specific contribution.

Bob Reigling	Entered and debugged the original 6502 source.
Lyle Thompson	Wrote all of the relative file disk interface primitives.
Ken Madell	Wrote the sprite control words, and the sprite editor.
Bill Ragsdale	Provided the Assembler and assembler documentation.
Forth Inc. & Leo Brodie	Created STARTING FORTH and the line editor which is the primary basis of the editor included in this package.
Forth Interest Group	Provides support and assistance to the Forth community.
FORTH DIMENSIONS III/2 & SYN-1 Users Group	Provided the basic form of the decompiler provided in this package.

TABLE OF CONTENTS

1 INTRODUCTION

MAY THE FORTH BE WITH YOU

FORTH is a high level language designed to develop compact, efficient, well-organized programs. The FORTH language offers program development tools such as an operating system, a compiler, an assembler, an interpreter, and a text editor.

The flexibility of the FORTH language forms the basis for its software design philosophy. FORTH consists of a powerful set of standard commands, and provides a method for you to create, or "define" your own commands. These commands can be built upon your previous "definitions."

You build functions which are similar to subroutine calls in other languages. You define new commands built upon your previous definitions until ultimately one word defines and can perform the whole procedure. You can even re-define some of the standard FORTH words to adapt the language to fit your needs.

This extensibility feature makes it easy for you to add new data types, features, operators, or other programming constructs to the language. You customize FORTH to suit your own particular circumstances.

As a result, 64FORTH gives you a more powerful programming environment than most other languages by putting you in direct control of your computer. However, this also places a higher level of responsibility on you, the programmer. FORTH may be more difficult to learn than the BASIC language, but you have greater control over your computer.

UNIQUE FEATURES

In FORTH, the term WORD refers to an identifiable function or command, which in some computer languages is known as subroutine or procedure.

In FORTH, words may be any length from one to thirty one characters (letters, or symbols). You can use descriptive names in writing your programs. As a result, FORTH programs can be well-documented and easy to follow. FORTH is a structured language—it does not use GOTO statements or statement labels. Instead, FORTH uses a stack which encourages the storage of working variables on the stack rather than in variables with names. Using the 6502 assembler provided, you can link a FORTH program to a machine language subroutine for high speed functions.

WHAT IS 64FORTH?

64FORTH is an implementation of the FORTH Interest Group (fig) version of the FORTH language with the addition of many words to control the unique sprite, color, and sound capabilities of the Commodore 64. Over 500 WORDS (Commands) are included, and because FORTH is extensible, you can add many of your own commands to 64FORTH. 64FORTH also contains words that allow you to control the character, border, and background colors of the screen.

64FORTH supports both multi-color and high resolution sprites with the ability to set their position and color and to edit, save, and retrieve sprites from cassette or disk storage.

You can control the special sound capabilities of the Commodore 64's SID chip using over thirty 64FORTH words provided.

If you have the 1541 disk drive, 64FORTH provides a true virtual disk interface, which uses the Commodore relative file structure. This means you can have FORTH files co-resident with program or other data files on the same disk.

If you are using cassette, 64FORTH has 16 virtual memory buffers located in high memory. You can work with reasonable sized source files in memory, and minimize the amount of time spent reading and writing data to cassette. Source screens may be read and written to cassette with or without names, as desired. When upgrading to disk, programs may be easily moved to disk from cassette.

64FORTH features four vocabularies, or groups of related words:

- FORTH
- EDITOR
- ASSEMBLER
- SYSTEM

Vocabularies are covered in detail in Chapter 5.

HOW TO USE THIS MANUAL

This manual is intended for two levels of experience. A tutorial chapter (Chapter 2) provides an introduction to FORTH programming—presenting fundamental FORTH concepts and commands. Examples lead you through the use of the Editor and several of the FORTH commands. Some familiarity with the BASIC language is assumed. An experienced FORTH programmer familiar with the Editor commands and FORTH could skip the tutorial chapter and proceed to the chapters covering the specifics of 64FORTH. The 64FORTH package gives you a machine optimized program of FORTH for the Commodore 64, and enough documentation to make all of the extras included usable. These chapters cover:

- Graphics
- Sound
- System Configuration
- Vectors
- Input and Output
- Useful Definitions and Tools
- 6502 Macro Assembler
- FORTH Glossary

If you are a complete beginner to FORTH, start with the short tutorial provided in Chapter 2. The tutorial should help you get started, however this manual is not intended to provide you with a complete, in-depth course on FORTH. If you find you need a further discussion of FORTH commands and concepts, obtain a copy of STARTING FORTH by Leo Brodie, recommended as the best tutorial manual for learning FORTH. STARTING FORTH describes a version of FORTH called FORTH-79, which is not indentical to fig (FORTH Interest Group)-FORTH/64FORTH. Refer to Appendix A of this manual covering the FORTH-79 differences while using the book.

In following the examples in the tutorial, follow the program listings exactly (including spaces). Do not be afraid to experiment with the new commands you learn. The program in the cartridge cannot be damaged by any programming error. If you need to restart the program, (for example if your screen freezes, or becomes a hopeless, confusing mess) simply press the RUN/STOP and RESTORE keys simultaneously. The 64FORTH program will start again.

Once you feel comfortable with the basic concepts presented in the tutorial, go on to the chapters on 64FORTH specifics to experiment with the magical color, sprites, and sound features. Eventually you can incorporate some of your graphic creations into an interactive game or program. Later, you will be ready to tackle the assembler included with 64FORTH.

WHAT IS THE FORTH INTEREST GROUP?

The FORTH Interest Group is an independent group of FORTH enthusiasts whose aim is to educate others and to promote FORTH. They may be contacted at:

P.O. Box 1105
San Carlos, CA 94070

They publish a newsletter called FORTH Dimensions ($15/yr) and have many other publications available.

2 LEARNING FORTH

WHAT DO YOU NEED TO USE 64FORTH?

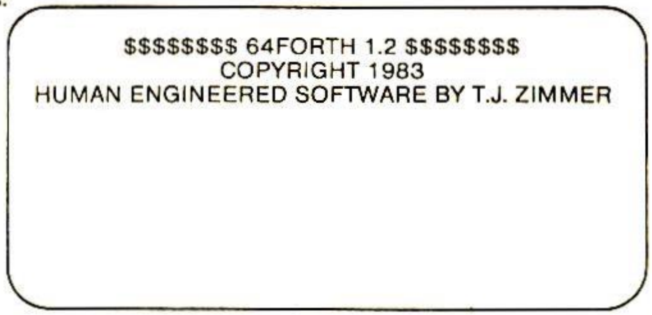
64FORTH runs on a standard Commodore 64, with either cassette or a 1541 disk drive for mass storage of your source screens.

GETTING STARTED

To start the 64FORTH program, do the following:

1. Make sure the power to the C64 and the monitor is turned off.
2. Insert the 64FORTH cartridge into the cartridge slot on the back of the C64.
3. Now you can turn on the power to the monitor, the disk drive (if applicable), and the C64.

About ten seconds will pass while 64FORTH performs a self-test before you see the 64FORTH title screen on your monitor. The computer will start up with the 64FORTH sign on message in white characters on a cyan background with a green border. 64FORTH is now operable, obediently waiting for your commands. The screen you see should look like this:



```
            $$$$$$$$ 64FORTH 1.2 $$$$$$$$
                    COPYRIGHT 1983
     HUMAN ENGINEERED SOFTWARE BY T.J. ZIMMER
```

POWER OFF

Before turning off the power to your Commodore 64, be sure to save the program you are working on (see FCLOSE for disk or WRITE for cassette). Remove any disk in your disk drive.

To exit the program, simply turn off the power to the C64 and your monitor and then remove the cartridge.

NOTE: NEVER INSERT OR REMOVE A CARTRIDGE WITH THE POWER ON.

FIRST IMPRESSIONS

Once the 64FORTH cartridge has been inserted, the FORTH operating system is running with its compiler, assembler, interpreter, and text editor.

If you press the return key (hereafter RETURN), a few times, you will notice an "ok" is printed after each RETURN is pressed. FORTH is telling you that it recognized your request to do nothing, and is waiting for your next command.

WORDS

In FORTH, a command is simply a sequence of characters called a word. A word can be one of the standard FORTH commands, or one of your own definitions, A line of commands consists of FORTH words separated by spaces. FORTH recognizes these spaces as command separators. When RETURN is pressed, FORTH "executes" or performs the line of commands from left to right. If all commands are performed without encountering an error, FORTH returns with the OK prompt to tell you its mission was accomplished. If an error is encountered at any time during execution, all processing stops. FORTH issues an error message and waits for further instructions.

To "execute" your first FORTH words, type in the following: (Your response is in boldface type, and FORTH's response follows.)

```
191 EMIT      <RETURN>
▚
ok
```

191 is the ASCII code for the character ▚ , and EMIT is the FORTH word which generates the character specified by the preceeding ASCII code. FORTH executes your command and dutifully prints the character for you.

Type the line again, this time adding the FORTH word CR before the word EMIT.

```
CR 191 EMIT      <RETURN>

▚
ok
```

CR performs a carriage return before printing out the character.

Another standard FORTH word is SPACES. With SPACES, FORTH prints out the number of blank spaces you indicate. Type in the following command and notice the position of the character after you press return:

```
20 SPACES 191 EMIT        <RETURN>
                    ▚
```

DEFINING YOUR OWN WORDS

As mentioned earlier, FORTH commands are referred to as WORDS. A set of standard, or pre-defined words form the basis of FORTH (see the Glossary) and a set of user defined words (called colon-definitions) supplement these commands. These definitions are compiled, or copied, into a "Dictionary" residing in the computer's memory. (For a further discussion on the dictionary, see Chapter 5.)

You create your own colon-definitions by entering the name of a procedure and describing the processes to be performed.

You break down the procedure into specific tasks and name each task. You can later group related tasks together producing a building block effect.

A colon-definition is preceeded by a colon, :, and followed by a semicolon, ;. Type in the following definitions which will be explained as you go along. Remember to leave spaces after the : and before the ;, and between the words.

```
: CHESS 191 EMIT ;        <RETURN>
```

The : indicates to 64FORTH that a new definition follows. **CHESS** is the name of the process that produces the character ▚ . The ; tells 64FORTH that the definition is complete. By pressing RETURN, you compile the definition into the dictionary. You should see an OK prompt if no errors were encountered.

The next definition creates an indentation using a combination of the FORTH words you learned earlier.

```
: INDENT  CR 5 SPACES ;        <RETURN>
```

INDENT names the process; **CR** causes a carriage return before the next command; and **5 SPACES** tells 64FORTH to skip five spaces each time the command INDENT is used. Type in the next definition.

```
: CHECK   INDENT   CHESS ;        <RETURN>
```

You can now combine your two new definitions to create a third definition. This definition produces a checkered square on your screen indented five spaces, although you will not see it until you execute the program.

The next command uses the FORTH words "DO . . . LOOP" to create a looping effect. (DO . . . LOOP will be discussed in detail in a later section.)

```
: CHECKERS 5 0 DO CHESS LOOP ;        <RETURN>
```

This looping effect performs the CHESS routine five times.

Now create a new definition using your words. INDENT and CHECKERS

```
: MATE   INDENT   CHECKERS ;        <RETURN>
```

This definition indents the CHECKERS routine before performing it.

The final step in your program is to name a whole procedure combining your words CHECK and MATE.

```
: COMMODORE MATE CHECK CHECK CHECK CHECK MATE
  CR ;        <RETURN>
```

Run the program by simply typing the procedure:

```
COMMODORE        <RETURN>
```

The short program you just entered was interactive, that is, you compiled your colon-definitions immediately after you entered them.

You can enter programs in two ways:

1. interactively
2. using the Editor

You can enter a word or a program directly from the keyboard to the interpreter, such as the short program you just entered. The word or program is carried out by pressing RETURN, however you cannot save it. If you want to save your program, you must edit your programs into blocks (or screens) using the Editor. This method makes fixing typing errors easier and you can save these blocks on disk or cassette.

EDITOR

The editor in 64FORTH is modeled after the editor described in the introductory book STARTING FORTH by Leo Brodie. The editor has been enhanced to provide immediate visual feedback on the results of each edit command performed, and a screen editor with insert and over-write modes has been provided. The function keys to the right of the keyboard are operable while in the editor to perform useful functions such as cursor and screen movements, and search and replace operations.

SCREENING

Before you can work with the Editor, you should understand a fundamental FORTH feature — the concept of screens or blocks. Memory storage in your C64 system can be cassette or disk based. This memory is divided up into units called "blocks." Each block holds 1 kilobyte, or 1024 characters, composed of 16 lines of 64 characters each. FORTH stores and retrieves information from mass storage a block, or a screen at a time. FORTH temporarily places these blocks into a screen buffer making sixteen of these buffers available at any one time. (See Chapter 7 for more details on the screen buffers.)

USING THE EDITOR

Using the Editor to create your programs allows you to make corrections easily and enables you to save your programs in screens.

Start the editing session by entering:

```
    1 EDIT        <RETURN>
```

where the word EDIT is preceeded by the number of the screen you want to edit. The EDITOR vocabulary is selected and the border color is green indicating the editor has been selected.

The screen splits into two sections — the top section consisting of sixteen lines is used for entering text (edit area) and the lower section consisting of six lines is used for entering edit commands (command area). If the edit area on your screen contains any unwanted characters or data, you can clear out the edit buffer in preparation for editing. Type:

```
    WIPE        <RETURN>
```

Notice two double digit numbers displayed in the lower right corner of the edit window. The rightmost number is the column position of the cursor in the edit screen, and the left number is the number of the current edit screen.

When your edit session is finished, you can leave the editor as follows. Press the RUN/STOP and RESTORE keys simultaneously, or type ABORT. Either of these operations will cause a FORTH warm start, and you will return to the normal full scrolling screen.

LINE EDITING

There are two types of editing modes: line editing and screen editing. In the line edit mode, the screen border color will be green.

I — INSERT

To enter some text, type the following at the cursor position:

```
I HELLO THERE        <RETURN>
```

'I' is the line edit command for insert. Remember to leave a space between the command and the text. Press return and you will see the text 'HELLO THERE' in the upper left corner of the screen with the cursor to the right of the E in THERE.

Enter some more text by typing:

```
I  HOW ARE YOU       <RETURN>
```

Notice that there are two spaces between the command 'I' and the word 'HOW'. And the first few letters of HELLO have disappeared on the left of the screen. The screen always tries to center itself around the cursor, and when the cursor reached the center of the screen, the screen started to scroll to the left to keep the window centered around it. This is called horizontal scrolling. This editor gives you a 40-column window into the 64-character wide FORTH screen.

To see HELLO again, press the function key <f5>. You will notice the cursor has moved to the first character position of the next line, and the word HELLO is now shown properly. Think of the <f5> key as the equivalent of the return key, but for the edit cursor, rather than the command line cursor.

T — SELECT A LINE

You can start text on any line (0 to 15) you specify. To enter the next line on the third line, type:

```
3 T      <RETURN>
```

P — PLACE

The command P places text on the cursor position line. You will learn two ways to use P:

1.	P	(text)	Places text into an insert buffer (PADI), then in current line

2. P <RETURN> **Places the contents of the last insert buffer into the current line.**

As an example, type in the following:

```
P  TO BE OR        <RETURN>
P  NOT TO BE       <RETURN>
```

Notice that the NOT TO BE typed over the TO BE OR.

U — UNDER

To avoid this, use the U command.

```
U  THAT IS THE QUESTION      <RETURN>
```

The command U places the contents of the insert buffer on the line under the current line. The two ways of using P also apply to U.

K — KILL

This command "kills" or blanks the current line. Type a K, press RETURN, and notice that the current line has been erased.

F — FIND

This command finds a string of characters. First, insert a line of text so that you will have an example to work with. Then, type TOP to move the cursor to the top of the edit area. Type F followed by a word or letters you want to correct.

```
I  TO SEEK OUT NEW LIFE AND NEW CIVILIZATIONS
     <RETURN>
TOP      <RETURN>
F NEW  <RETURN>
```

The F command puts the word new into the find buffer called PADF, and searches for the first occurence of NEW from the current cursor position in the editor screen. The cursor is positioned at the word new, waiting for a command to erase or insert, for example.

E — ERASE

The word E will erase the text in PADF just found. By pressing E and hitting RETURN, the NEW is erased. You could then use the I command to insert a new word or text. For purposes of an example, re-insert the word NEW.

```
E              <RETURN>
I  NEW         <RETURN>
```

R — REPLACE

The command R is a combination of the two commands E and I. Position the cursor to the top of the edit area (use the command TOP) and type in the following sequence of commands:

```
TOP     <RETURN>
F   NEW     <RETURN>
R   OLD     <RETURN>
F   <RETURN>
R   <RETURN>
```

The R command takes the word NEW (which is held in the find buffer, PADF) erases NEW, and inserts the word OLD (which became the contents of the insert buffer, called PADI).

Another feature of the editor uses a combination of Find and Insert. Type in the following:

```
U   A SLUG, A PEEL, A SPLIT     <RETURN>
```

Use F to find the word, a.

```
F   A     <RETURN>
```

Then use I to insert a word after A leaving two spaces after the I:

```
I   BANANA     <RETURN>
```

"A" has been placed in the find buffer, PADF, "BANANA" has been placed in the insert buffer, PADI. Type in the following and watch what happens:

```
F     <RETURN>
I     <RETURN>
F     <RETURN>
I     <RETURN>
```

SCREEN EDITING

Using the screen editor, you move the cursor around the current screen, over-type words to correct typing errors, delete characters with the DEL key, and insert spaces where needed to add text. To use the screen editor, press the <INST/DEL> key in the upper right hand corner of the keyboard along with the shift key. The screen border color should change to yellow to signal that you are in the screen edit insert mode. Now any characters you type will be inserted in the edit screen, and the cursor will move along to the right of the characters. Try it!

If you make a mistake, you can delete the characters preceeding the cursor by using the <DEL> (unshifted INST/DEL) key, or by using the cursor keys to move to the character just after the error and press the <DEL> key to start deleting there.

If you move to the middle of some of the text you have entered and start typing, you will see the text to the right of your new characters move automatically to the right to make room for the text you are typing. This is the INSERT MODE because any text you type is inserted between existing text. If you push text off the right end of the 64-character line, it will be lost.

Another mode is called the OVER-WRITE MODE, and you enter this mode by pressing the <INST> (shift-INST/DEL) key again. The border should switch to light blue. When you type characters now, your new text will replace the old text on the screen. This mode is useful when you are correcting typing errors in existing text. Press the <INST> (shift-INST/DEL) key a third time to return to the INSERT MODE.

To leave the INSERT or OVER-WRITE MODES, press <RETURN> and the border will switch to green.

FUNCTION KEYS

64FORTH provides several helpful commands to use while in the editor. First, the <f1> and <f3> keys can be used to TAB forward and TAB backwards. The <f5> functions as a <RETURN> key while you are in the screen editor. The <f7> is equivalent to typing "F <RETURN>" in the line editor. The <f7> key finds the string held in PADF. The <f8> key (shifted <f7>) allows you to easily replace strings you found with the <f7> key. The <f8> key is equivalent to R <RETURN>.

The <f7> key should only be used after the editor command F has been used at least once. And the <f8> key should only be used after the editor command R has been used at least once.

The <f6> key (shifted <f5>) opens up a section of text at the cursor position to make room for added text. The <f2> (shifted <f1>) and <f4> (shifted <f3>) keys are used to move to a higher <f2> or lower <f4> edit screen.

SAMPLE EDITING SESSION

Now that you are familiar with the use of the editor commands, try typing in the Commodore program you entered earlier, or another program. Begin by selecting a edit screen:

```
1 EDIT      <RETURN>
```

Use the command WIPE to clear the screen:

```
WIPE      <RETURN>
```

Then start typing in a program, starting with the I (insert) command, and continue adding lines with the U (under) command.

When you have finished entering the program, you can list it with the LIST command. Press the RUN/STOP and RESTORE keys together to return to the FORTH screen. Indicate the screen number you want to list.

```
1 LIST      <RETURN>
```

If you are satisfied with the screen, you can compile your definitions with the word LOAD:

```
1 LOAD      <RETURN>
```

Each time you LOAD, you increase the size of your dictionary.

To execute your program, name the procedure. In the case of the Commodore program, simply type the name of the final definition: **COMMODORE.**

After you feel comfortable with the editor commands, experiment with the additional commands listed in the summary section below. The sections following the command summary cover the FORTH concepts of the stack and arithmetic notation, and then you will learn some other FORTH words.

NOTE ON GOOD PROGRAMMING PRACTICE

The first line of your source code should describe the particular application of the block. You can enclose remarks about the program in parentheses (leaving a space between the first parentheses and the comments). Remarks serve as a reminder to you what the program does, and aid another programmer in understanding what you had intended. Remarks enclosed in parentheses are ignored by the text interpreter.

EDITOR COMMANDS

Line Editor (green border)

WORD	FORMAT	FUNCTION
T	n1 ---	Sets the edit pointer to the start of line n1.
P	--- <t>	Text <t> following space is placed into line holding the edit pointer.
U	--- <t>	Text following the space after U is placed under the current line and all lower lines are moved down.
M	n1 n2 ---	Copies current edit pointer line under line n2 in screen n1.
X	---	Deletes line containing the edit pointer and moves lower lines up one. The deleted line is held in PADI and line 15 becomes blank.
H	---	Holds the edit pointer line in PADI.
K	---	Kills (erases) the edit pointer line.
S	---	Spreads the lines at the edit pointer. All lines from the edit pointer are moved down. Line 15 is lost.
TOP	---	Move edit pointer to TOP of screen. Same as HOME key.
F	--- <t>	Find first occurence of text following 'F'. Starts at current edit pointer.
E	---	Erases as many characters going backward as the length of the last 'F' command.

D	--- <t>	Deletes the first occurence of text following the D command, searches from edit pointer until the end of the screen.
TILL	--- <t>	Deletes all text starting at edit pointer until and including the string following the command TILL. Works on current LINE only. If string is not found, no deletion occurs.
I	--- <t>	Inserts text following the command I into the edit buffer at the current position of the edit pointer. Text following that is too long for the line is lost (over 64 characters).
R	--- <t>	Replaces the string just found by 'F' with the string following the R command.
DEL	n1 ---	Deletes n1 characters before the edit pointer, and compresses the line to omit the space.
C	n1 ---	Move the cursor by the signed amount n1 characters (positive for forward move, negative for backward move). This word also redisplays the current edit window.
N	---	Move the edit pointer to the top of the next higher screen buffer. Limited by BMAX.
B	---	Move the edit pointer to the top of the previous (lower) screen buffer. Limited by BMAX.
EDIT	n1 ---	Selects the edit mode, with n1 as the screen to be edited. Moves the edit pointer to the top of the screen n1. Revectors CR and KEY to show the current edit window, and make cursor and functions keys operational.

MISCELLANEOUS EDITOR WORDS

(F)	---	Searches for text in PADF until end of screen.
(I)	---	Inserts the current contents of PADI into the edit buffer at the cursor. Text too long for the line is lost.
PADF	--- a1	Returns address of find buffer. Length = 80.
PADI	--- a1	Returns address of insert buffer. Length = 80.

PAD	--- a1	Returns address of scratchPAD area a1. Length = 80.
TEXT	c1 --- t	Accepts the text following the command TEXT into the scratchPAD area until the character with ASCII value c1.
GTEXT	a1 ---	Accepts text from the input stream until a delimiting ↑ is found, or the RETURN key is pressed. The text is placed at address a1, for a length of 64 characters.
!CUR	N1 ---	Sets the edit pointer to value n1. (n1 is limited to 0 <= n1 <= 1023.)

NOTE: In (--- <t>), the <t> symbol indicates the text is optional. Typing RETURN without any text will use the current contents of PADI, or PADF.

KEY FUNCTIONS

Cursor Keys	Active, allow scanning through the edit buffer character by character, or with auto repeat.
HOME key	Moves the cursor to the top of the edit screen.
INST	(Shift INST/DEL key). This key enables the INSERT mode and the BORDER color is changed to yellow to indicate the insert mode. All keys are inserted into the edit screen as they are typed. The DEL key is also enabled, to delete characters on the edit screen preceeding the cursor. Pressing <INST> a second time toggles to the OVER-WRITE mode indicated by a blue border. The RETURN key leaves the INSERT mode, and the BORDER color returns to green.

FUNCTION KEYS

f1	Tabs cursor right 4 characters.
f3	Tabs cursor left 4 characters.
f5	Moves cursor to beginning of next line.
f2 (shift f1)	Move edit to the next higher screen number.
f4 (shift f3)	Move edit back to the previous screen number.

f6 (shift f5)	Spreads the line at the cursor position.
f7	Finds the next occurence of search string given by the last 'F' command and leaves insert mode if not found.
f8 (shift f7)	Replaces the most recently found string with the text specified in the most recent 'R' command.

For the convenience to those of you familiar with WordStar™, a selected number of WordStar compatible cursor function keys have been included using a combination of the Commodore key (C<) and a letter key.

C< A	Left tab. Same as f3 key.
C< S	Left cursor.
C< D	Right cursor.
C< F	Right tab. Same as the f1 key.
C< G	Forward character delete.
C< E	Up one line.
C< R	Back one screen. Same as f4.
C< X	Down one line.
C< C	Next screen. Same as f2.
C< Y	Delete the line.
C< Z	Re-insert the line you just deleted with C< Y.

THE STACK

The stack is one of the important elements in FORTH. FORTH temporarily stores information before processing in what is known as a stack. You add numbers or data to the stack, and then apply various operations. When the operations are applied, numbers or data are taken from the top of the stack first. This process is called LIFO (last in, first out). You can only remove the value that is on top of the stack.

Other programming languages also use the stack, but it is normally not available to the programmer. In FORTH, you can control the stack directly.

In most computer languages information is stored permanently in memory using "variables" referred to its "variable name" (this is also possible in FORTH). Often, the data you are manipulating is transient. You can put the transient data on the stack rather than storing it in memory and giving it a name. The data on the stack is readily available and can be discarded after use.

In FORTH, each stack element consists of 16 bits, or two bytes, of information. This information can be stored in two ways:

16-bit signed number
16-bit unsigned number

Another method of storing information is known as double numbers. Double numbers consist of two stack elements, or 32-bits. You can have:

32-bit signed number
32-bit unsigned number

STACK MANIPULATION WORDS

FORTH contains several pre-defined words used to manipulate the data on the stack.

DROP	Drops the top stack item.
DUP	Duplicates the top stack item.
OVER	Duplicates the second item on the stack and puts it on top of stack. (Becomes 1st and 3rd elements of stack)
ROT	Rotates the third item to the top
SWAP	Exchanges the top two stack items.

To manipulate a double length (32-bit) number OR numbers in pairs, add a '2' to the above words:

2DROP
2DUP
2OVER
2SWAP

Examples of the use of these words can be seen in the PRIME program listing following the section on OTHER FORTH WORDS.

Put some numbers on the stack and play around with the above words and keep checking the stack to see the changes. Place numbers on the stack simply by typing them in and hitting RETURN, and you can print the stack contents with the word .S. For example:

```
1  2  3  4  5    <RETURN>
.S               <RETURN>
```

The numbers should appear on your screen in the same order.

Remember that FORTH works from the top of the stack, and the last item in is the first item out. Try some of the stack manipulation words, such as:

```
SWAP      <RETURN>
.S        <RETURN>
1  2  3  5  4             (should appear on screen)
ok
```

To clear the stack, type:

```
SP!        <RETURN>
```

POST-FIX NOTATION

FORTH uses a non-traditional method to handle arithmetic functions. First, you state the numbers of your calculation and then you state the arithmetic operation. In other words,

2 + 2 becomes 2 2 +

This is known as post-fix notation and is often referred to as reverse Polish notation. It is very similar to notation used by Hewlett Packard calculators.

FORTH also uses the standard arithmetic operators:

+	n1 n2	---	sum	adds
−	n1 n2	---	difference	subtracts
*	n1 n2	---	product	multiplies
/	n1 n2	---	quot	divides

The following table lists the standard equation and its corresponding reverse Polish notation:

5 + 4	5 4 +
90 − 4	90 4 −
60 × 8	6 8 *
21 / 7	21 7 /

Enter the equations in reverse Polish notation on the keyboard and then type a period . (called DOT) after each equation to print the results.

Other operators are included in 64FORTH. Here is a sample of additional math operators (full definitions appear in the glossary):

```
*/
*/MOD
/MOD
MOD
```

OTHER FORTH WORDS

This section covers FORTH words used to create a branching effect in your programs. These branching control structures allow you to program the computer to make decisions and to repeat tasks.

DECISIONS

You can add a decision-making process to your FORTH programs, which adds flexibility by allowing you to create logical paths to complete tasks. Several FORTH words support these conditional executions:

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

NOTE: Conditional statements may only be used within colon-definitions.

The following comparison operators perform logical comparison operations on the data stack and return boolean flags for use by the above words:

```
=        0<
-        0=
>        XOR
<        AND
         OR
```

These conditional statements make use of what is called a boolean flag. A flag is a value left on the stack as a signal to another word or definition. The value left on the stack is either a one (also called non-zero) or a zero. A one (actually any non-zero value) indicates that the condition is true; and a zero indicates that the condition is false. You do not type in the word "flag," but you leave the flag value on the stack.

IF. . . ENDIF

The IF... ENDIF statement allows you to control the sequence of the program depending upon certain conditions. The IF... ENDIF statement is preceeded by a flag, or a signal as to whether a condition is true (non-zero) or false (zero). The flags, or boolean values are left on the stack. If the condition, or boolean value is true, execution continues with the next word in the definition. If the condition is false, IF causes execution to skip to ENDIF, from which point execution will proceed. Every IF needs a corresponding ENDIF in the same definition. An example of an IF... ENDIF statement follows.

```
: TRUE?   IF ." A TRUE WAS ON THE STACK " ENDIF ;
```

To execute this example, type in the name of the definition; preceeded by a value.

```
1 TRUE?      <RETURN>
0 TRUE?      <RETURN>
```

If the value on the stack is true (that is, non-zero), the statement "A TRUE WAS ON THE STACK" is printed.

You can add another branching effect to the IF... ENDIF statement with the word ELSE. The branching to the word after ELSE takes effect if the condition is false. The following is an example of an IF... ELSE... ENDIF statement:

```
: TRUE/FALSE IF ." A TRUE " ELSE ." A FALSE "
ENDIF ." WAS ON THE STACK " ;      <RETURN>
0 TRUE/FALSE     <RETURN>
1 TRUE/FALSE     <RETURN>
```

The words after the IF are executed when the condition is true. If the condition is false, the words after the ELSE are executed. When a conditional statement completes, execution continues after the ENDIF.

NESTING

You can put an IF... ENDIF (or IF... ELSE... ENDIF) statement within another IF... ENDIF statement. This process is called nesting. However, there must be a boolean value on the top of the stack for each IF and each IF must have an ENDIF.

LOOPING

You can create control structures in your FORTH programs to perform a certain operation repetitively. These control structures are called "loops." The following FORTH words are used to create loops:

BEGIN . . . UNTIL	(loops until condition is true)
BEGIN... AGAIN	(loops forever)
BEGIN... WHILE... REPEAT	(loops while condition is true)
DO... LOOP	(loops specified number of times)

BEGIN . . . UNTIL

The BEGIN... UNTIL statement sets up an indefinite loop which continues processing until the flag condition is true. In other words, this loop repeats indefinitely UNTIL a certain condition occurs. A sample BEGIN... UNTIL statement is:

```
: COUNTER1   0 BEGIN DUP . 1+ DUP 10 = UNTIL DROP ;

COUNTER1     <RETURN>
```

If the flag remains false, the loop continues. As soon as the flag becomes true, the loop ends.

BEGIN . . . AGAIN

In this statement, the BEGIN marks a start of a sequence that may be executed forever. AGAIN causes a return to BEGIN and an example follows:

```
: COUNTFOREVER   0 BEGIN DUP . 1+ AGAIN ;

COUNTFOREVER     <RETURN>
```

All of the words between BEGIN and AGAIN will be repeated. This type of loop can only be terminated by pressing the RUN/STOP and RESTORE keys simultaneously.

BEGIN . . . WHILE . . . REPEAT

This construction performs a "test" in the middle of the loop, rather than at the end. Execution continues as long as the test, or flag is true, returning to BEGIN. When the flag becomes false, the words following REPEAT are executed. An example of this loop follows:

```
: COUNTDOWN    20 BEGIN DUP 0 > WHILE DUP . 1 -
REPEAT DROP ;

COUNTDOWN       <RETURN>
```

DO . . . LOOP

The DO . . . LOOP statement sets up a definite loop within a specified index range. You specify the number of times the loop will repeat by indicating the ending number plus one, and the beginning number before the word DO. An example of a DO . . . LOOP follows:

```
: COUNTBY2    10 0 DO I I + . LOOP ;

COUNTBY2       <RETURN>
```

The DO . . . LOOP must always be part of a definition. The ending number is called the limit (in this example, 10) and the beginning number is zero. I returns the index of the current loop counter.

DOT QUOTE ."

A handy FORTH word, ." (pronounced "dot quote") performs the same function as a PRINT statement in BASIC. You can use ." within a definition to cause some text to be typed on the screen. Type in the following and see what happens:

```
: CHEER    ." GO FORTH AND CONQUER " ;       <RETURN>

CHEER      <RETURN>
```

A blank space must follow the .". The ending quote marks the end of the text string, and it is called a "delimiter."

Try putting the above definition in a DO . . . LOOP statement:

```
: CHEERS   20 0 DO CHEER LOOP ;       <RETURN>

CHEERS      <RETURN>
```

GO FORTH AND CONQUER should appear on your screen 20 times.

For a more serious example of the above control structures, study and type in the following program to select prime numbers;

```
: PRIME   ( n1 --- ) DUP 2 / 1+ SWAP ." BEGINNING " CR
   1 DO DUP I 1 ROT
      2 DO DROP DUP I /MOD
         DUP 0= IF DROP DROP 1 LEAVE
            ELSE 1 = IF DROP 1
               ELSE DUP 0 > IF DROP 1
                  ELSE 0= IF 0 LEAVE ENDIF
                  ENDIF
               ENDIF
            ENDIF
         LOOP
   IF 4 .R ELSE DROP ENDIF
   LOOP DROP CR ." DONE " ;
```

LOAD the block to compile the definitions. Execute the program by typing in 100 PRIME and pressing RETURN. This will print the prime numbers between 1 and 99. Try other values.

TICK '

The FORTH word ' (pronounced "tick") gives you access to addresses or locations in memory of any dictionary word. Tick is used in the form:

```
' word     <RETURN>    (leaves address on the stack)
.          <RETURN>    (prints the address)
```

USING VARIABLES

If you are used to working with variables as in other programming languages, you can use two words in FORTH to create variables: VARIABLE.

You can use variables to store a certain type of information permanently rather than using the stack. To create a variable, type the word VARIABLE followed by a name or label, and preceeded by its initial value:

n VARIABLE name

When VARIABLE is executed, it creates the definition (name) with its parameter field address initialized to n. When (name) is later executed, the address of its parameter field is left on the stack. Once the address is on the stack, you can have access to the location using the FORTH words @ and ! (discussed below).

USING CONSTANTS

You can also use constants in FORTH, and they are defined like VARIABLEs. A constant is like a variable except it returns a value instead of the address of the value. And @ and ! are not required.

(value) CONSTANT (name)

STORE ! and FETCH @

The FORTH word ! (pronounced "store") takes a value and an address. The value is stored into memory at the address. The address may be the address of a variable as returned by the name of the variable.

```
(value) (address) !
(value) (variable name) !
```

The FORTH word @ (pronounced "fetch") takes the information back out of the variable. You supply the address of the information by naming the variable:

```
(variable name) @     (fetches contents of address)
.                     (prints the contents)
```

@ can also be used to retrieve information from the address you specify:

```
(address) @           (fetches contents of address)
.                     (prints the contents)
```

You can also use @ and ! with the variables pre-defined by 64FORTH. Some of the user variables are:

CONTEXT	BLK	CURRENT	D#
DP	DR#	F#	FENCE
HLD	IN	OUT	SCR
R#	TIB	VOC-LINK	

You type the name of the variable followed by a space and @, a value is left on the stack. To store a value into the variable, type the value and then the variable name followed by a space and !.

CAUTION: Changing system variables may cause a system crash. Handle with care.

3 GRAPHICS

This chapter covers the graphics capabilities of 64FORTH: changing colors and creating sprites.

COLOR

64FORTH provides you with words to control the background, border, and character colors of the Commodore 64 without having to POKE into memory using an obscure calculation. The background color is controlled as follows:

```
        n1 BGROUND        <RETURN>
```

where n1 is a number from 0 to 15, giving a range of 16 possible background colors. The colors are:

0	black	8	orange
1	white	9	brown
2	red	10	light red
3	cyan	11	gray 1
4	purple	12	gray 2
5	green	13	light green
6	blue	14	light blue
7	yellow	15	gray 3

You can control the border color similarly:

```
        n1 BORDER        <RETURN>
```

where n1 is the number of the color, 0 to 15.

Select character color by pressing the <CTRL> key, and the color key together. To select the color of the characters as displayed during the program execution, use the codes of the desired color. You can determine the value of the color key desired by typing in:

```
        KEY       <RETURN>     (will wait for key to be pressed)
        <CTRL> 1               (this places the CTRL 1 key on the data
                               stack)
        .     <RETURN>         (prints the value of the key)
        144                    (printed by the computer as the value of
                               the CTRL 1 key)
```

You can use this routine to find the ASCII value of any key. You can then EMIT the value to the screen. To make the characters black, EMIT the value:

```
        144 EMIT        <RETURN>
```

Experiment with other colors by finding the ASCII value of the color, and then EMIT the value.

SPRITES

The Commodore 64 has the capability to produce high resolution, programmable graphics called sprites. Sprites are designs which you can create in practically any shape. You can easily move the shapes, change their color, or change their size. High resolution (hires) refers to the ability to control the individual dots on a screen, and these dots are called pixels. In creating a sprite, you determine its characteristics by making each pixel visible or invisible, that is turning them on or off.

Type in the following short program to create and display a sprite, remembering to leave spaces between numbers and words. This program is presented just to get you started using sprites. When you learn to use the 64FORTH sprite editor you will have a more convenient and flexible method of creating your sprites.

```
DECIMAL         <RETURN>          (sets the number base to
                                  decimal)
SYSTEM          <RETURN>          (selects the SYSTEM)
SPBASE          <RETURN>          (initializes sprite data area)
0 -SPOBJ 64 255 FILL              (puts some date in sprite 0)
     <RETURN>
170 140 0 CXY        <RETURN>     (positions sprite 0 on screen)
1 0 COLOR         <RETURN>        (set sprite 0 color to white)
0 SHOW          <RETURN>          (displays sprite 0)
```

In the second line of the program, you entered the SYSTEM vocabulary in order to execute two system words from the sprite editor. These words are normally needed only by advanced programmers and will be described in the last section on sprites FOR THE ADVANCED PROGRAMMER.

POSITIONING A SPRITE

Two words are available for positioning your sprites on the screen. They are XY and CXY. These words have the form:

```
x#   y#   sp#   CXY
```

where x# is the x screen position in pixels, y# is the y screen position in pixels, sp# is the sprite number (0 - 7). So to move your sprite to X=200, Y=100 you enter:

```
200 100 0 CXY        <RETURN>
```

If you play around with this you will see what x and y values are on and off the screen.

The second word for positioning a sprite is XY. This word works just like CXY except that the X position must be less than 256. This word is provided for high speed movement but will not work on the right 1/6 of the screen.

EXPANDING A SPRITE

Next try changing the size of the sprite with the word eXPAND. This word has the form:

```
(x expand: 1 or 0)   (y expand: 1 or 0)   sp#   XPAND
```

The following command would make your sprite wider:

```
1 0 0 XPAND      <RETURN>   (expand sprite 0 in the x
                             direction)
```

The next command makes your sprite taller:

```
0 1 0 XPAND      <RETURN>   (expand sprite 0 in the y
                             direction)
```

To make the sprite both wider and taller, type:

```
1 1 0 XPAND      <RETURN>   (expand sprite 0 in both
                             directions)
```

To return the sprite to its original size, type:

```
0 0 0 XPAND      <RETURN>   (return sprite 0 to original size)
```

You can move the sprite to the lower left of the screen (for editing purposes described in the next section):

```
20 200 0 XY      <RETURN>   (moves sprite 0 to new X and Y)
```

USING THE SPRITE EDITOR

The sprite editor consists of a grid — 24 squares wide by 21 squares tall. Each square equals one dot, or pixel. You form a sprite by indicating which pixels are "turned on" and which are "turned off."

To see the editing screen of the sprite you created in the above program, type:

```
0 SPEDIT      <RETURN>   (selects edit screen for sprite 0)
```

The following screen should be displayed where * means that the pixel is on and a blank means that the pixel is off:

```
BUILDING          * * * * * * * * * * * * * * * * * * * * * * * *
   SPRITE         * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * * * * * * * *
```

Notice the actual sprite which you positioned in the lower left corner of the screen. This sprite represents the current contents of the sprite being edited. Each * in the edit area is one point on the sprite shown.

Use the cursor keys (up, down, left, right) to move the cursor around the edit area. Moving the cursor keys around does not affect the on/off condition of the pixel. Press any key to fill in a pixel, or press the space bar to erase a pixel. Experiment with these keys to change the shape, and when you are finished press the RETURN key. 64FORTH builds the new sprite, and the sprite shown in the left corner of the screen reflects the changes you made.

64FORTH supports up to eight sprites, numbered 0 to 7. To create a new sprite, use the word NEWSPRITE preceeded by the number of the sprite you want to create.

1 NEWSPRITE <RETURN>

You enter the sprite editor after the sprite data area is cleared. Create a new shape using the cursor keys, the space bar, and the * key. Do not use the <INST/DEL> key to back up and erase. Create shapes for space ships, animals, fruit, large letters—anything you might want to use in a program. Press RETURN when you are ready to build your sprite.

Once the sprite is built, you can

- display it
- give the sprite a position
- change its color
- make it multi-colored
- change its size
- modify it
- put it in a program
- save it and retrieve it later

DISPLAY A SPRITE

Now that you are done drawing your new sprite (sprite number 1 in this case) set its position using the word CXY (see section on positioning a sprite) and then display it on the screen by saying:

```
1 SHOW      <RETURN>   (displays sprite number 1)
```

If you do not see it, your X and Y position values may be off the screen. Type 20 140 1 CXY, and press RETURN. If you still don't see it, its color may be set to the same color as the screen. Type 1 1 COLOR and press RETURN.

COLORING A SPRITE

Change the sprite color with the 64FORTH word, COLOR. A sprite can be any of the sixteen colors available on your Commodore 64. The colors available and their corresponding numbers were covered in the previous section on color control. The word COLOR is used to select a hires single color for your sprite, and the format of the command is:

```
color#   sp#   COLOR
```

where color# is from 0 to 15 and sp# is the number of the sprite from 0 to 7.

HIDING A SPRITE

To remove a sprite from the screen display, use the 64FORTH word, HIDE. To remove the sprite on the lower left of the screen, type:

```
0 HIDE      <RETURN>   (hides sprite number 0)
```

MULTI-COLORED SPRITES

Up until now, you have been working with high resolution sprites. 64FORTH also supports sprites in a multi-color mode, although use of this mode cuts the horizontal resolution in half. This means that instead of 24 pixels across the sprite, there are only 12 pairs of pixels. Every two pixels represents one color. Multi-color mode allows you to use three different colors in one sprite. The three available colors are:

sprite color	(set with the COLOR command)
multi-color 1	(set with the MCLR1 command)
multi-color 2	(set with the MCLR2 command)

To set a sprite to the multi-color mode, use the word MULTI in the form:

```
sp#    MULTI
```

When you are in multi-color mode the two additional colors are selected using the words MCLR1 and MCLR2 in the form:

```
color#    MCLR1
color#    MCLR2
```

where the color number is in the range 0 to 15.

In multi-color mode, a pair of pixels is needed to select each color dot in the sprite. The following chart specifies the colors indicated (_ indicates a blank space on the sprite edit screen):

__ = BGROUND	_* = MCLR1
*_ = sprite COLOR	** = MCLR2

Create a new sprite, or edit a previous sprite with these color combinations in mind. Build the sprite by pressing RETURN after you have edited it. Then set its position (CXY or XY), and then display it (SHOW). Before going into multi-color mode, you must set the two colors, MCLR1 and MCLR2, using the sixteen available colors. To set a sprite to multi-color mode, type the number of the sprite followed by the word, MULTI.

To leave the multi-color mode and return to high resolution mode, enter the following command:

1 HIRES <RETURN>

SPRITE VOCABULARY

You have learned the following words for creating and displaying your sprites:

WORD	FORM
SPEDIT	sp# SPEDIT
NEWSPRITE	sp# NEWSPRITE
XY	x# y# sp# XY
CXY	x# y# sp# CXY
COLOR	color# sp# COLOR
SHOW	sp# SHOW
XPAND	xx yy sp# XPAND xx=0 or 1, yy=0 or 1
MULTI	sp# MULTI
MCLR1	color# MCLR1
MCLR2	color# MCLR2
HIRES	sp# HIRES
HIDE	sp# HIDE

PUTTING A SPRITE INTO A PROGRAM

The following short program moves sprite #1 diagonally across the screen using the words you learned in Chapter 2.

```
1 EDIT                  (enter edit screen 1)
WIPE                    (clear the screen)
I DECIMAL               (select decimal number base)
U : MOVES 200 0 DO I I 1 CXY LOOP ;
U : MOVIT 3 1 COLOR
U            1 SHOW 147 EMIT
U         BEGIN MOVES AGAIN ;
```

Then type 1 LOAD to compile the program. Press RUN/STOP RESTORE to exit the editor. Then type the new word MOVIT and watch what happens. You can stop the program by pressing RUN/STOP RESTORE.

If you use the word XY instead of CXY the sprite will move faster.

SAVING AND RETRIEVING SPRITES

The sprites you create are saved with their ASCII values, not the actual shape you created. The screens, or blocks, in 64FORTH can only contain ASCII data; not binary data. (You created your sprites using binary — on/off — data). The screens are designed to contain source text, not graphics. Therefore, screens 1 and 2 of the sprite files hold the ASCII values of your sprites. When you save a sprite, you save one or both screens containing the ASCII data of the sprites you want to save.

Two words, SPWRITE and SPREAD, allow you to write and read sprite definitions to disk or cassette. Disk operations will be covered first.

Disk

Before creating a sprite you would like to save, you must open a file to contain the sprite data. CLOSE ANY CURRENTLY OPEN FILE!! Use the word FCLOSE to close the file. You don't want to write the sprite on top of any of your source code screens. Now open the file you want to use for sprite data, as follows:

FILE SPRITEDATA <RETURN>

After creating a sprite, you can save the sprite to the file, by entering the following:

1 SPWRITE <RETURN>

This operation takes about five seconds to convert your sprite definition to the proper data statements, read the screen from disk, and transfer the data statements to the disk screen. When the command completes, type FCLOSE and wait for this operation to complete. It will take about 10 seconds.

If you get a disk error flashing on the drive, type ?D to clear the error. The error occurs because you created a new file, and read and wrote to it when there was no data in it yet.

To read sprites from a disk file, first open the file containing the sprite data definitions.

```
FILE SPRITEDATA      <RETURN>
```

Then read the definitions for the proper sprite:

```
1 SPREAD      <RETURN>
```

The sprite is read from the disk, into a screen buffer. Then the data statements are read from the screen file into the proper sprite definition area. This operation takes approximately eight seconds.

To write all of your sprites to the disk at one time, use the following definition; used in the format SAVESPRITESTODISK <filename> <RETURN>:

```
: SAVESPRITESTODISK   ( --- <text> )
    FILE
    8 0
    DO I SPWRITE LOOP
        FCLOSE ;
```

And to read the sprites back in:

```
: READSPRITESTODISK  ( --- <text> )
    FILE
    8 0
    DO I SPREAD LOOP FCLOSE ;
```

Cassette

If you are using a cassette for sprite storage, insert a cassette prepared for recording into the cassette drive. Enter the following commands:

```
1 SPWRITE      <RETURN>          (indicates saving function)
NAME SPRITEDATA     <RETURN>     (names the file)
1 WRITE     <RETURN>             (save sprite edit screen)
```

The computer responds, "PRESS RECORD AND PLAY ON TAPE". After you press the record and play buttons, the screen blanks for a while and then returns with the message, "SAVING   OK".

This writes the screen containing sprite number one to cassette. The "1 WRITE" above works for any of those sprites on screen 1 — sprites 0 to 3. If you are working with sprites 4 to 7, you will have to replace the above WRITE statement with "2 WRITE" to write the screen that holds those sprites.

To load a sprite once you have saved it, type:

```
NAME SPRITEDATA      <RETURN>
1 READ      <RETURN>
SPREAD      <RETURN>
```

The screen will blank, and then display the messages:

```
SEARCHING
FOUND
LOADING
```

It is recommended that all sprites to be used in a particular program be read or written at once. The following colon-definitions can be used to write and read all sprites on cassette format as follows: SAVESPRITES <cassette filename> <RETURN>:

```
: SAVESPRITES (  --- <text>)
   NAME
   8 0
   DO I SPWRITE LOOP
      1 2 WRITES ;
```

And to read the sprites back in:

```
: READSPRITES    (  ---  <text> )
   NAME 1 2 READS
   8 0
   DO I SPREAD LOOP ;
```

SPRITE DATA DEFINITIONS

If you would like to see the data definitions of the sprite with which you have been working, enter the editor after performing a SPWRITE operation.

```
1 EDIT      <RETURN>
```

Screens 1 and 2 of the sprite files hold the ASCII values for your sprites. Screen 1 contains the sprites 0 to 3 and screen 2 contains sprites 4 to 7. Each sprite has a four line area on the screen to hold its sprite data assignments. For example, lines 4 through 7 on screen 1 contain the ASCII data for sprite 1. You can use the editor to modify the data statements for the sprites, but use caution as errors can easily occur.

FOR ADVANCED PROGRAMMERS

A number of system words are used by the sprite editor. These words are SB, SPBASE, –SPOBJ and SBLK. See the glossary for their descriptions. Note that the constant SB is in ROM and is used by the other system words. Therefore sprites are always edited and read and saved from RAM $0C00 – $0DFF. You may put an additional 8 sprites from $0E00 – $0FFF by CMOVEing them from $0C00 and changing the sprite pointers $07F8 – $07FF (see Commodore 64 Programmers Manual for information on sprite pointers). If you want to put your sprites elsewhere in RAM or have even more of them, be careful not to over-write any memory needed by 64FORTH.

4 SOUND CONTROL

64FORTH includes a complete set of words to control the complex sound generation capability of the Commodore 64 computer.

Before selecting or adjusting any voice you need to initialize the sound system with the SINIT word as follows:

SINIT <RETURN>

This word sets up the control mechanisms provided, and automatically selects VOICE1.

The Commodore 64 uses a sound integrated circuit 6581 to generate complex waveforms with three separate voices. The sound control words provided work with whichever voice is currently selected. You select a voice with one of the following words:

VOICE1 VOICE2 VOICE3

Each voice can be one of four general waveforms, as follows:

TRIANGLE SAWTOOTH SQUARE NOISE

Each voice can be turned on and off with the following words. GATE1 turns (on) the voice, and GATE0 turns (off) the voice.

GATE1 GATE0

The volume setting affects all voices, and is set or read with the words:

VOLUME! VOLUME@

The ATTACK, DECAY, SUSTAIN, and RELEASE of each voice can be set and read with the following words:

ATTACK! ATTACK@ DECAY! DECAY@

SUSTAIN! SUSTAIN@ RELEASE! RELEASE@

The frequency of each voice is set or read with the words:

FREQ! FREQ@

Type in the following commands and you will hear a relatively low frequency tone in your television speaker, (be sure the volume is up):

```
SINIT          <RETURN>   (initialize sound)
VOICE1         <RETURN>   (select voice #1)
TRIANGLE       <RETURN>   (make it triangular)
15 VOLUME!     <RETURN>   (volume on high)
4 ATTACK!      <RETURN>   (Attack fairly fast)
0 DECAY!       <RETURN>   (Decay very fast)
8 SUSTAIN!     <RETURN>   (Sustain medium)
0 RELEASE!     <RETURN>   (Release very fast)
3000 FREQ!     <RETURN>   (set the frequency)
GATE1          <RETURN>   (turns voice on)
S!             <RETURN>   (causes all of the above to take
                          effect)
```

Wait a while and then enter:

```
GATE0          <RETURN>   (turns voice off)
S!             <RETURN>   (causes GATE0 to take effect)
```

All of the above parameters must be set before turning the voice on, but you only have to change the frequency to change the note if that is all you want to change.

The word S! must be executed following any sound word or group of sound words to modify the characteristics of a voice. S! actually causes the sound control words to take effect all at once.

Several other sound control words are available and are included in the 64FORTH glossary:

```
CURVOICE
CV
ENV3@
FILTER@
FILTER!
FMODE@
FMODE!
FSELECT@
FSELECT!
OSC3@
PWIDTH@
PWIDTH!
RESONANCE@
RESONANCE!
SYNC0
SYNC1
TEST0
TEST1
```

5 64FORTH SYSTEM CONFIGURATION

This chapter describes some of the aspects of how 64FORTH is organized and how it is similar and/or dissimilar to other FORTH systems.

STACK PICTURES

FORTH uses a form of abbreviation called a stack picture to indicate what values are passed into or out of a WORD (or command). This abbreviation provides concise information about how a word is used in a program. Below are the symbols used to complete the stack pictures in this manual.

SYMBOL	MEANING
a1, a2 ...	16-bit address
n1, n2 ...	16-bit signed number
u1, u2 ...	16-bit unsigned number
d1, d2 ...	32-bit signed double length number
b1, b2 ...	8-bit byte value
c1, c2 ...	7-bit ASCII character
f1, f2 ...	boolean flag
tf	true boolean flag — non-zero
ff	false boolean flag — zero
t1, t2 ...	ASCII text string
<sp>	the space bar
<RETURN>	the return key
<text>	a string of text follows
---	symbolic for the current word
/	separates stack parameters

Here is an example of a typical stack picture:

```
( n1/a1 --- a2 <text> )
```

n1/a1	The stack parameters required before the word executes.
---	Symbol for the word being executed.
a2	The stack contents after the word executes.
<text>	A text string which follows the word being executed in this example.

In the above example, you will notice the slash (/) between the stack value n1 and a1 signifying that two values must be placed on the data stack before the above word can be executed. In actually putting the values on the stack the slash is not used — the values are separated by a space. The number to the right of the slash will always be placed on the stack last, whereas the value on the left will be placed on the stack first. FORTH uses a LIFO (last in, first out) stack, so the last value placed on the stack will always be on top of the stack and will be the first value returned. Address a2 to the right of the three dashes (---) indicates an address a2 returned by the word after it was executed. Values n1 and a1 are not shown on the right side of the three dashes, because they were used by the word and have been discarded. The word <text> to the right of address a2 indicates a string of text which follows the word being executed, before the <RETURN> key is pressed.

MEMORY ALLOCATION

64FORTH will run on any Commodore 64 computer. The 64k bytes of computer memory is used as follows:

Amount	Memory contents	Address range
1k	Kernel variable space	($0000-$0400)
1k	Display screen space	($0400-$07FF)
1k	FORTH system space	($0800-$0BFF)
1k	Sprite storage	($0C00-$0FFF)
25k	Dictionary user space	($1000-$7400)
1k	Sound control variables	($7400-$77FF)
1k	Communications buffer	($7800-$7BFF)
1k	Screen buffer	($7C00-$7FFF)
16k	64FORTH kernel	($8000-$BFFF)
16k	Virtual mass storage	($C000-$FFFF)

You may notice an odd thing about the above memory map. The usage of the area from $D000 to $FFFF appears to conflict with the 64's I/O and KERNEL routines, which reside in this area. 64FORTH however carefully switches these routines out of memory while using this RAM, and then switches it back in before performing any I/O or KERNEL calls.

MEMORY MAP

The following table is a graphic representation of the memory allocation.

```
                      LOW END
------------------------------------------------------------ $0000
[             FORTHS DATA STACK                      ]
------------------------------------------------------------ $0060 S0
[            64 KERNAL DATA AREA                     ]
------------------------------------------------------------ $0100 TIB
[             TIB / RETURN STACK                     ]
------------------------------------------------------------ $01FF R0
[            64 KERNAL DATA AREA                     ]
------------------------------------------------------------ $0400
[          SCREEN MEMORY SPACE                       ]
------------------------------------------------------------ $0840
[         64FORTH USER VARIABLES                     ]
[            AND VECTOR TABLES                       ]
------------------------------------------------------------ $0B40
[          SPRITE DEFINITION RAM                     ]
------------------------------------------------------------ $1000 DP0
[                                                    ]
[      25K USER DICTIONARY SPACE                     ]
[                                                    ]
------------------------------------------------------------ $7400 EM
[       SOUND CONTROL EMULATOR                       ]
[              VARIABLE SPACE                        ]
------------------------------------------------------------$77FC
[     MASS STORAGE COMM BUFFER                       ]
------------------------------------------------------------ $7BFC FIRST
[       ONE FORTH SCREEN BUFFER                      ]
------------------------------------------------------------ $8000 LIMIT
[                                                    ]
[            16K FORTH KERNAL                        ]
[                                                    ]
------------------------------------------------------------ $C000
[                                                    ]
[     16K VIRTUAL SCREEN BUFFERS                     ]
[                                                    ]
------------------------------------------------------------ $FFFF
                      HIGH END
```

STACK SPACE

FORTH uses two stacks, DATA and RETURN, located in the low end of memory. The data stack is located in page zero (the first 256 bytes in memory), from about $0010 to $0060 HEX (numbers preceeded by a $ are specified in HEXADECIMAL). The data stack has room for about 40 stack entries. The return stack is located in page one as the 6502 hardware stack. This page is shared with the Terminal Input Buffer and is limited to about 60 levels of nesting.

VOCABULARIES

FORTH uses vocabularies to segment similar words into groups or classes. The 64FORTH system has four vocabularies: FORTH, EDITOR, ASSEMBLER, and SYSTEM. To select a vocabulary, simply type its name. A description of the contents of these vocabularies follows:

FORTH   This vocabulary is selected at a cold start and contains all of the words you will frequently need in application programming. The FORTH vocabulary includes sound, sprite, and color control words, in addition to standard FORTH words.

EDITOR   This vocabulary contains all of the words used specifically to edit text. This group of words is automatically selected when you enter the editor with the EDIT command. The FORTH vocabulary is automatically reselected when you leave the editor.

ASSEMBLER   This vocabulary contains all of the assembly mnemonics and addressing mode words used in the macro assembler. Assembler is selected automatically by the word CODE used whenever you start an assembly definition.

SYSTEM   This last vocabulary is used as a catch all for various system call words and words in the system not normally used in everyday programming. SYSTEM will be useful to you after you have become familiar with the FORTH system and want to proceed into more complex usage of the Commodore 64's hardware.

The words in these four vocabularies all reside in one dictionary. All definitions you create are added to the dictionary in the order they are compiled, and are not subdivided into separate vocabularies. Instead, your definitions are linked to one vocabulary.

Two user variables are used in conjunction with the vocabularies: CONTEXT and CURRENT.

CONTEXT contains a pointer to the vocabulary where dictionary searches will first begin. For example, whenever you enter the word ASSEMBLER, ASSEMBLER becomes the CONTEXT vocabulary and that vocabulary will be searched first.

CURRENT contains the variable of the vocabulary into which you are adding new word definitions. CURRENT vocabulary is normally the FORTH vocabulary. In order to link words to one of the other vocabularies, you need to use the word DEFINITIONS.

DEFINITIONS sets the current vocabulary to the context vocabulary. If you were to SOURCE (a command which displays the actual definition) the word DEFINITIONS, you would see the following:

SOURCE DEFINITIONS <RETURN>

```
: DEFINITIONS   CONTEXT @ CURRENT ! ;S
```

To add words to the ASSEMBLER vocabulary, type:

ASSEMBLER DEFINITIONS

To restore FORTH as the CURRENT vocabulary, type:

FORTH DEFINITIONS

To see a listing of all the names of the definitions in the vocabularies, use the word VLIST and press RETURN. The CONTEXT vocabulary is listed first. Pressing any key temporarily halts the scrolling and pressing any key again continues the scrolling. Hitting the RUN/STOP key ends the listing.

THE DICTIONARY

64FORTH allocates 25k of RAM memory for your dictionary space. Every new definition you compile is placed in the dictionary. 64FORTH includes several words used to manage this dictionary space: FORGET, EMPTY, DSAVE, DLOAD, ALLOT, and HERE.

FORGET deletes a definition from the dictionary along with all the entries physically following it. Use FORGET in the form:

FORGET <name of definition>

EMPTY, on the other hand, cleans out the entire dictionary by forgetting ALL of the definitions you have added. You can use this word by typing in EMPTY at the beginning of a program load.

All of your added definitions would also be "forgotten" whenever you end a 64FORTH session and turn off your computer. Using the words DSAVE and DLOAD, you can save your dictionary entries <DSAVE> and load your dictionary when you need it again <DLOAD>. Your definitions will be linked into the FORTH dictionary and you do not need to recompile them.

To save your definitions, type if you are using a cassette drive:

CASSETTE DSAVE (filename) <RETURN>

or, type the following if you are using a disk drive:

DISK DSAVE (filename) <RETURN>

64FORTH responds "SAVING (filename)" and an OK prompt appears when completed.

To load your definitons, type (for cassette):

```
CASSETTE DLOAD (filename)       <RETURN>
```

Or, for disk type:

```
DISK DLOAD (filename)       <RETURN>
```

64FORTH responds, "SEARCHING FOR (filename) LOADING". You are returned to FORTH upon completion of DLOAD.

You can increase the ALLOTed space in the dictionary by using the word ALLOT. For example,

```
0 VARIABLE SMALLARRAY 6 ALLOT
```

This would reserve an additional six bytes of space to the variable SMALLARRAY. Reserving this extra space in the dictionary is useful for creating arrays. In this case, SMALLARRAY is an array of 8 bytes (2 are automatically allocated when SMALLARRAY is created) and the address of the first byte of the array will be returned.

Another word, HERE, tells you the address of the next available dictionary location. HERE leaves the address on the stack, and . (dot) prints the address:

```
HERE .       <RETURN>
```

The dictionary makes use of an area of memory called the "pad." This pad works in the sense of being a landing pad, storing strings of text temporarily before being sent to an output device, such as a printer or monitor. The word PAD leaves the beginning address of the text output buffer (or pad) on the stack, and the . (dot) prints the address:

```
PAD .       <RETURN>
```

PAD moves as the dictionary expands, floating 66 bytes above here.

CHANGING NUMBER BASES

64FORTH can work with different number bases including decimal and hexadecimal, and you can change the number base at any time. 64FORTH starts up in the decimal base, and the word HEX converts to the hexadecimal base. Use the word DECIMAL to convert back to the decimal base. The word BASE is a user variable containing the current number base. You can create a word which selects the binary number base with the following definition:

```
DECIMAL       <RETURN>
: BINARY 2 BASE ! ;
```

SYSTEM CALLS

Commodore has built a very powerful KERNEL into the 64. In BASIC, you do not generally have access to the KERNEL because several parameters in the machine registers must be passed to the KERNEL to control the operation of the KERNEL call. In 64FORTH, however, a word

SYS (in the SYSTEM vocabulary) has been provided to allow you to set all of the machine's registers, including the status of the carry flag, before performing a system call. Once the call returns, the contents of A, X, and Y registers are placed on FORTH's stack and are available to you. This versatile word allows you to have access to all KERNEL functions from high-level FORTH definitions.

Here is the glossary entry for the SYS word:

SYS (<f1>/b1/b2/b3/a1 --- <f1>/b4/b5/b6)

The SYS command allows calling any assembly language routine. "f1" is an optional carry set/reset flag; it is returned unmodified. b1, b2, and b3 are the A, X, and Y registers respectively as sent to the assembly routine. b4, b5, and b6 are the returned values of A, X, and Y. a1 is the assembly call address. Any called routine must return with an assembly RTS instruction, and must not destroy the hardware stack contents. To those who understand Assembly language: the contents of the processor status register is returned after a SYS in the ASSEMBLER variable N.

The following is an example fo a system call:

```
HEX                     (sets the hexadecimal number base)
: MESSAGES.OFF          (turns off KERNEL message printing)
    SYSTEM              (selects the system vocabulary)
    0  0  0             (places 3 dummy values for A, X, Y)
    FF90  SYS           (calls KERNEL at address $FF90)
    3DROP ;             (clears the returned values)
DECIMAL                 (reselects the decimal number base)
```

The word, MESSAGES.OFF, will turn off all KERNEL messages for errors and warnings, etc. 64FORTH is initialized with all messages ON. The three zeros in the above definition are the mandatory A, X, and Y register values required by any assembly language call being performed. The word SYS performs the system call, and 3DROP after SYS removes the returned values of the A, X, and Y registers from the data stack.

Many KERNEL calls are provided. A complete list of these calls follows and their definitions may be found in the 64FORTH glossary at the end of this manual.

SETLFS	OPEN	SAVE	SLOAD	CHKOUT
CLALL	SETNAM	SLSN	DLSN	DTLK
SECOND	CIOUT	LISTEN	ACPTR	UNLSN
UNTLK	TKSA	TALK	CHKIN	CLOSE

WHAT IS MISSING?

64FORTH is a very complete implementation of FORTH, however, an experienced FORTH programmer will notice some functions are missing in this system. A list of the missing words and the reason they have not been included follows:

BUFFER
+BUF
PREV
USE

These words are part of the standard FORTH virtual disk interface. 64FORTH contains an improved double buffered virtual disk interface, which has been tailored to the Commodore 64's particular memory configuration.

WARNING
(ABORT)

These words are part of the error handling system in a standard fig-FORTH system. The word WARNING is used in a standard FORTH system to specify whether a disk is available for the printing of error messages. This variable is not required because 64FORTH prints error messages from ROM. The word (ABORT) is a method used in fig-FORTH to allow you to redirect the error handling to a user selected and written program, if desired. This was done primarily because fig-FORTH does not support vectored words. ERROR is a vectored word in 64FORTH, and as a result (ABORT) is not needed.

INDEX

This word prints the first line of each screen within a specified screen range. It was not included in this system due to space constraints. The following is a definition of a version of INDEX which has been optimized for the Commodore 64:

```
DECIMAL
: INDEX         ( n1/n2 --- )
   SYSTEM                  ( selects the SYSTEM vocabulary )
   D# @ 4 >                ( only index if disk file open )
   IF   1+ SWAP
      DO I 1 - 8 * 1+ POS
         PAD RECREAD DROP ?DERR
         PAD 64
         -TRAILING I 2 .R SPACE TYPE CR
      LOOP
   ELSE 2DROP ." NO DISK FILE OPEN" CR
   ENDIF ;
```

Edit this definition into a screen and save it for later use. This version of INDEX will run fairly fast, because it only reads in the first record of each screen.

USER VARIABLES

The following tables show how the user variable space is set up:

User Table Definitions

Address	Function
$0940	Start of user table
$0940-0949	Not used
$094A	Terminal input buffer TIB
$094C	WIDTH
$094E	Not used
$0950	FENCE
$0952	DP (dictionary pointer)
$0954	VOC-LINK
$0956	BLK
$0958	IN
$095A	OUT
$095C	SCR
$095E	BMAX
$0960	CONTEXT
$0962	CURRENT
$0964	STATE
$0966	BASE
$0968	DPL
$096A	Not currently used
$096C	CSP
$096E	R#
$0970	HLD

I/O Vector Table		Vector Offset	
Address	Function	Decimal	HEX
$0972	KEY	00	00
$0974	EMIT	02	02
$0976	?TERMINAL	04	04
$0978	CR	06	06
$097A	CREATE	08	08
$097C	NUMBER	10	0A
$097E	ERROR	12	0C
$0980	. (DOT)	14	0E
$0982	−FIND	16	10
$0984	, (COMMA)	18	12
$0986	C,	20	14
$0988	R/W	22	16

$098A	CTBL	Capture table address variable
$098C	F#	Logical file number for file access
$098E	D#	Device number for file access

$0990	User Vector Space	Vector Offset
$0990-$09BC	2 bytes per user vector	30 to 72

$09BE-09FE	FN	File name array for file
$0A00	EFLAG	Edits STATE flag
$0A02	DR#	Device # of current disk drive
$0A04	DCHAN	Disk drive channel for data read and write
$0A06		Not currently used
$0A08	CURVOICE	Sound current voice variable
$0A0A	MODE	Assembler mode variable

Emulator User Variables

$0A0C	ERP	Emulated return pointer
$0A0E	EIP	Emulated interpreter pointer
$0A10	'STEP	Redirectable single step variable
$0A12	'CONT	Redirectable continue variable
$0A14	IFLAG	Insert mode flag, insert/overwrite

Sprite Variables

$0A16	SPOBJ
$0A18	SPINDX
$0A1A	SPBYT
$0A1C	SCPTR

$0A1E-$0A40	Available user space

6 VECTORS

WHAT ARE THEY?

FORTH is a macro language, that is, FORTH is made up of many simple words written in assembly language. By stringing several of these smaller words together in a line, you can create more powerful words. This simple technique makes FORTH a very powerful tool in writing programs. However, one small problem arises with this method. Occasionally, a low level word needs to perform a slightly modified function, in order for a higher level word to do something a little differently. This process is known as vectoring.

As an example, suppose you had a parallel printer. The Commodore 64 has an 8-bit user port where you would connect your parallel printer. At times you may need to send all of your listings to that port, rather than to the video screen. The only problem with this is that the KERNEL does not know about your printer on the user port, and can not easily be made to talk to it. In 64FORTH this will not be a major problem because all I/O (input and output) in 64FORTH go to your own routine rather than just the routines that the Commodore 64 already knows about.

VECTORED WORDS IN 64FORTH

64FORTH includes a minimum set of words which should fill the needs of most users wishing to revector various operations in FORTH. Here is a list of all of the VECTORed words in 64FORTH, and their positions in the two vector tables: I/O and VWORDS.

POSITION		WORD
HEX	DECIMAL	
00	00	KEY
02	02	EMIT
04	04	?TERMINAL
06	06	CR
08	08	CREATE
0A	10	NUMBER
0C	12	ERROR
0E	14	. (dot)
10	16	−FIND
12	18	,
14	20	C,
16	22	R/W
18	24	CTBL

Note: User created vectored words start at an index of 30, and may consist of up 20 total vectors, two bytes each.

VECTOR CONTROL

Here is an example of how to revector the character output of a device other than the video screen. Suppose you need a printer driver for the USER port for a 7-bit printer driver, with the eighth (8th) bit used as a strobe.

```
( --- )                    ( initializes the user port )
: PINIT                    255 UPORT 2+ C! 0 UPORT C! ;
```

Next you need a word to send characters to the user port and to strobe bit 8 of the port:

```
( --- )                    ( strobes bit 8 of user port )
: STROBE8 UPORT 128 TOGGLE UPORT 128 TOGGLE ;

( n1 --- )                 ( sends character to user port )
: PORTOUT 127 AND UPORT C! STROBE8 ;
```

You have a routine to send characters to the user port. All you have to do is vector EMIT to the new character output word.

```
: TO.UPORT ' PORTOUT CFA I/O 2+ ! ;
```

The routine above takes the CFA (code field address) of the new driver word, and stores it into the thrid and fourth bytes of the I/O vector table in user memory. The instant this has been done, any further character output will go to the new driver routine. If it doesn't work, the program may hang. To restore the I/O vectors to their initial values, press the RUN/STOP and RESTORE keys together or type VRESET. Either method will restore all of the VECTORS to their initial values.

Note: The example above shows how to revector EMIT, not how to create a complete printer driver. Most printers require handshaking protocols which are beyond the scope of this manual. You will have to study the data sheets on the 6522 chip for a while before attempting such a driver.

See the section on vectors in Chapter 8 for other examples of vector control.

VECTOR WORD GLOSSARY

VECTOR	(n1 --- t) This is a new defining word used to add additional words to 64FORTH that are to be vectored. It is used as follows: 30 VECTOR <vector name> After the above is executed, any time <vector name> is executed, it will perform the routine whose CFA is in I/O + 30. You must therefore store the CFA of a valid dictionary word into it before executing <vector name>. Note: User VECTORs are from 30 to 70 decimal.

VWORDS	(--- a1) This word returns address a1 of the beginning of the initial value vector table in ROM. This data may be accessed in the same manner as the I/O table to obtain the actual routine's CFA for a given vectored word.
I/O	(--- a1) This word returns the address a1 of the beginning of the user vector table in RAM. This table is used to change the function of selected dictionary words.
VRESET	(---) This word when executed RESETS all of the vectored dictionary words to their initial values, by moving the data at VWORDS to the RAM table at I/O.

7 INPUT AND OUTPUT

In this chapter, you will learn how 64FORTH handles the input and output (I/O) of data from your Commodore 64. The input/output operations include:

- user port access
- printer output
- screen buffer control
- cassette interface
- disk interface
- file transfers

USER PORT

The Commodore 64 comes with a full 8-bit port. You can control this port using the 64FORTH word, UPORT. UPORT contains a system constant which returns the address of the data register of that port. You can use UPORT to have access to the 8 bits of external data to connect to a printer, a modem, or other hardware.

CAUTION: Study the user port section in the Commodore 64 Programmer's Reference Guide before attempting to make changes using UPORT. You could damage your computer!

The user port is initialized as an INPUT port, and you can observe any TTL (Transister Transister Logic) level as follows:

```
UPORT C@        <RETURN>
```

This places the value of the data at the port on the FORTH data stack. Once the data is on the stack, you can display or print the data (using the . command), and you can change the data. If you wish to use the user port as an OUTPUT port, you can initialize UPORT as follows:

```
255 UPORT 2+ C!        <RETURN>
```

This stores a binary value of all ones into the UPORT data direction register of the 6522 VIA chip in the computer. The VIA chip is designed to make any data bits OUTPUTS if the corresponding bits have been set in the data direction register.

To control the output data on the port:

```
n1 UPORT C!        <RETURN>
```

where n1 is a number in the range 0 to 255 decimal. This number is then sent to the user port.

PRINTER OUTPUT

64FORTH includes words to produce listing of data or programs on a serial printer. The word PRINT allows any character output from the line of commands following PRINT to be sent to the printer connected via the serial bus port. After the line has been interpreted, output returns to the video screen. A lower level word PRON in the SYSTEM vocabulary turns on the printer port, and leaves all character output going to the serial bus printer, until another word, PROFF (also in the SYSTEM vocabulary) is executed. These lower level words would be used within a program to control printer output.

PRINTING A PROGRAM LISTING

To obtain a print-out of a program listing, use the following sequence of words:

PRINT n1 LIST <RETURN>

where n1 is the number of the screen or block you want to list. You cannot print and read a screen in from the disk at the same time. You must read the screen containing the program you want to list into one of the sixteen screen buffers, and then you can print the listing. Otherwise, the serial bus will hang, and the only way to get out is to turn off the printer or turn off the computer.

Since PRINT operates on the serial bus, it is important that no other serial bus operations be performed while printing is occuring. Therefore when using PRINT to make listings of screens, the screens to be printed must be brought into memory before attempting to print. This may be done using EDIT. This problem occurs because no serial bus arbitration is built into the Commodore DOS.

RS-232 SERIAL PRINTER

You can send data to the RS-232 port by using the following lines of FORTH required to send data to an RS-232 serial printer:

```
( RS-232 serial printer output words )
DECIMAL

659 CONSTANT SCONTROL      ( serial control location )

: SOFF    SYSTEM 3 CHKOUT 5 CHKIN
          2 CLOSE 3 CLOSE 5 CLOSE ;
: SON     SYSTEM SCONTROL 4 SETNAM 2 2 0 SETLFS OPEN
                     3 3 0 SETLFS OPEN 5 5 0 SETLFS OPEN ;

: SWAIT   BEGIN 669 C@ 670 C@ = UNTIL ;

: SPRINT  ( print the command <text> following --- <text> )
          SYSTEM SON 2 CHKOUT INTERPRET SWAIT SOFF ;

: BAUD    ( n1 --- )   ( n1=baud rate, i.e. 300, 1200 ... )
          1000 1000 ROT */ 2 / 100 -
          SCONTROL 0 OVER ! 2+ ! ;
```

You must set the BAUD rate before using the SPRINT word, or you may get some very strange results.

SCREEN BUFFER CONTROL

Memory storage in your Commodore 64 system can be cassette or disk based and it is divided up into units called "blocks." FORTH stores and retrieves information from memory a block, or a screen at a time.

64FORTH places these blocks temporarily in screen buffers. Each of these screen buffers hold the contents of one block or screen (1024 bytes) of code in RAM where you can load, edit, or have access to the block. This buffer is the temporary storage between the cassette or disk and your current screen. The system transfers data to and from disk or cassette and places the data in screen buffers. You can use one screen at a time. This arrangement is known as virtual memory because it appears as if all of the mass storage area were in your RAM memory space. You have access to the amount of virtual screen buffer space currently specified with the word BMAX.

In a cassette based system the range of available screens that you can have access to is limited to the range of available virtual buffers in the system — 16 in 64FORTH. The variable used to specify the highest screen accessible is named BMAX.

If you have a disk based system, the value in BMAX will control the highest screen available in each FORTH file. For example, if you have a need for a disk file larger than 16 screens, you can increase the value of BMAX to a larger number, and you will be allowed to access screens up to this higher value.

In no case, however, should you create a file larger than 90 screens. 90 screens is the limit of the relative file system in the Commodore disk drive. Also, each time you reopen a file larger than 16 screens, you must reset the vaue of BMAX to the higher value or you will not be able to access screens above 16.

The FORTH word EMPTY-BUFFERS clears out the screen buffer area without saving the updated blocks to disk or cassette.

USING CASSETTE

When 64FORTH starts up, the system expects to work with a cassette with sixteen virtual screens available for data and editing. You do not have to tell 64FORTH that you will be using cassette. The words that control the reading and writing to and from cassette are:

READ	(n1 ---)
READS	(n1 ---)
WRITE	(n1 ---)
WRITES	(n1/n2 ---)
LOADS	(n2 ---)
CASSETTE	

In these words, n1 is the number of the screen or block to be read or written. When n2 is present, it represents the number of screens to read and write.

To save an edited program, type the number of the edit screen(s) containing the program you want to save, followed by the WRITE command. For example, to save edit screen 1, type:

```
1 WRITE      <RETURN>
```

You are then asked to press record and play on the cassette recorder. The screen will blank for few seconds while 64FORTH saves the screen(s).

To retrieve a program from cassette, type the number of the screen you want to load and the word READ.

```
1 READ      <RETURN>
```

The screen will blank during the READing process and will respond with an OK when the process is complete. Once the screen is available, you can EDIT it, LOAD it, LIST it, and then execute it.

To retrieve and compile a block or screen in one step from cassette, use the LOADS command:

```
n1 LOADS      <RETURN>
```

The command retrieves a series of screens from cassette. When complete your program is ready to run. Each screen is read into buffer number one and a 1 LOAD is performed.

FILES

If you have used BASIC files, using LOAD and SAVE, you will find FORTH VIRTUAL files to be very different. But in a very short time you won't know how you ever did without them.

First, to give you some basis for comparison, the BASIC method of file operation will be covered. When you LOAD a file under BASIC, the entire file is pulled in from disk, into main computer memory. Under BASIC, there is somewhat less than 40k bytes of memory available for programs, and no file larger than 40k can be loaded or run. While this may seem like a lot of memory, there are many applications which would like to have more than that, not the least of which is word processing. While there are several word processors for the Commodore 64, none allow manipulation of files larger than can be held in memory.

FORTH contains a word named BLOCK, which it uses and which you may use in your programs, to perform the interface between the computer and your disk drive. As mentioned above, a block in FORTH is 1024 bytes or characters. Blocks are numbered sequentially from one to "n," with "n" being the highest block number contained within the current file. The FORTH word BLOCK is passed the number of the block of characters you wish to access, and it returns the address of the first character of that block.

For example, suppose you wanted the 2097th character of a file. To obtain that character, you must perform a block operation that will cause block 3 to be read in from disk, since block three contains characters 2048 to 3071. Each time you want to get a character, it may be in a different block. Therefore, it would seem difficult to remember what block contains each character desired but in FORTH it is not difficult at all. Here is a simple command sequence which will obtain a numbered character in the range 0 to 65535:

```
( n1 --- c1 )
: GET.CHAR.N
     B/BUF /MOD 1+ BLOCK + C@ ;
```

And here is a word that will place a character into a disk file:

```
( c1/n1 --- )
: PUT.CHAR.N
     B/BUF /MOD 1+ BLOCK + C! UPDATE ;
```

You may have noticed a couple of differences in this definition from the one above. The second definition uses the C! operator, which stores a character into memory; and the word UPDATE, which tells FORTH that you have modified a block (you stored a character into it). UPDATE also writes the block back to disk when it is convenient for the system to do so. In 64FORTH this write operation will occur at one of two times: either when the file is closed, or when the virtual buffer holding the data is needed by another block that you requested.

It is important to realize that the write operation is deferred to a later time, and it is therefore not proper to turn off the computer or disk drive until an FCLOSE has been performed. Note also that when you open another file, a close is automatically performed, and data will not be lost.

This method automatically reads and writes to disk as needed, without your having to worry about whether a particular block of data is already in memory.

64FORTH adds one additional structure to the FORTH block interface, specifying that the blocks are within a file, and that BLOCK can only have access to the blocks within the currently open file. To allow BLOCK to access the blocks of your program, the file containing them must be open. This is done with the FILE command. When you enter FILE <filename> <RETURN>, you are not loading in a file from disk as in BASIC. You are simply telling FORTH that if a block of data is requested, it should get the block from the file you have just opened. An ideal example of this is entering the EDITOR with the EDIT command. This issues a block command to the system requesting the block you want to edit, and if a file is open, the block will be read in from disk.

USING A DISK DRIVE

64FORTH differs from other FORTH systems significantly in the area of disk operation. Most FORTH systems provide only a block interface to mass storage, bypassing the operating system file structure. This method requires a separate disk to be used for FORTH, and prevents interaction between FORTH and other operating system files. 64FORTH, on the other hand, runs under the Commodore file system, allowing you to have several files on a disk. These files can contain separate sets of FORTH blocks, along with BASIC program files, and other files.

Before a file on disk can be opened, the disk drive must be connected and turned on, and a formatted disk inserted. If you want to format a new disk, the following command line will initialize the disk and erase all existing data on it:

CMD N:(disk name),XX   <RETURN>

The text following CMD is sent to the disk as a command to "NEW" (using the Commodore system command, NEW) the inserted disk with the name (disk name) and the disk ID# (XX). Any disk command can be sent to the disk drive with the CMD word. The sequence CMD I <RETURN> may be used to initialize a new disk but should never be used while a file is open on the disk.

OPENING FILES

To save a file to disk, you must open the file before you begin to edit it with the editor. Use the word FILE, which opens a disk file, followed by a file name. When you open a disk file, you open the disk.

FILE (new or existing file name)   <RETURN>

The disk drive will activate, and the file will be opened.

Note: If you do not have a disk drive, do not use the disk FILE command. The system will hang because the Commodore KERNEL tries to talk to a device that is not present. Certain other disk related words may also cause trouble when they are used without a disk drive connected to the system, such as ?D, DISK, ↑ FILE, and (FILE).

If you have just created a new file, you start running into some interesting peculiarities of the Commodore disk operating system. If you try to EDIT a screen in a file that does not yet exist, the system issues a disk error because there was not data to read. EDIT the file created with the word FILE:

1 EDIT   <RETURN>

The drive will activate, and return shortly with the empty screen of the editor. The light on the disk drive will blink signifying a disk error. If you now type:

?D   <RETURN>

the system will clear the error, and type out the error message that no data was found.

For a new file, type in the following command to clear the current screen:

WIPE   <RETURN>

You have now cleared the screen buffer you are editing to blank spaces and you could begin editing. But first, write the screen to disk to extend the file to a size large enough to hold it. Enter the following command:

```
FLUSH      <RETURN>
```

The disk starts again, writing out your screen #1 to the disk. If you get another disk error, type ?D and press RETURN to clear the error. Errors are reported when you access a relative record that does not exist, and when you wrote a non-existent relative record. These errors are normal for the Commodore disk operating system, and can be cleared with ?D and ignored. Now you can start editing. To save the completed screen, use the word FCLOSE. FCLOSE performs a FLUSH and then CLOSES the file.

Or you could switch to another file by entering:

```
FILE (next file name)      <RETURN>
```

The currently open file will be checked for updated screens, and these will be written to the currently open file if required. The current file will then be closed, and the next file will be opened. This process may take several minutes if many screens in the current file are updated and need to be written to disk.

LOADING AN EXISTING FILE

To retrieve an existing file from disk, type the following:

```
FILE (existing file name)      <RETURN>
1 EDIT                         <RETURN>
```

To run the program, it must first be compiled. The THRU word will load a range of screens as follows:

```
1 4 THRU                       <RETURN>
```

In this example screens 1 through 4 are loaded. You can then execute the program by entering the name of the appropriate procedure followed by return.

IF YOU FORGET TO OPEN A FILE

Occasionally, you may forget to open a file before editing. In this situation, the system has no open file in which to put the contents of your memory, and you would not be able to write to disk.

If, once having realized your error, you perform a FILE command to open a file, your data will be lost. The word FILE wipes out the contents of the virtual memory buffers before opening a new file.

To illustrate why the system works this way, suppose you had two files: FILEA containing a game you have written, and FILEB containing some graphics routines. If you edit screen one of FILEA, and make a modification to that screen, then switch to FILEB, the graphics routines, there is a good chance that the modified screen from FILEA will be written to FILEB by accident. This would be disasterous for your programming, and would soon cause you to stop using the system. So, the primary rule to remember when you are about to edit a routine you want to save, is to OPEN A FILE FIRST, THEN START TO EDIT.

However, in the interest of assisting you in recovering from the above situation, a word has been provided which will open a file without clearing out memory. This word is (FILE), and should only be used with great care, since it is possible to read a screen from one file, and write it to another. If you create a program without first opening a file, you can open a file to write the screen out to, as follows:

```
(FILE)   <file>     <RETURN>
```

Then the data can be written out to it as follows:

```
FCLOSE      <RETURN>
```

All updated screens will be written to the file before it is closed. REMEMBER HOWEVER THAT ANY DATA ALREADY IN THE FILE WILL BE OVERWRITTEN!!!

TRANSFERRING FILES

Two lower level words have been provided to facilitate the transfer of screens between disk files, or between cassette and disk: (FILE) and (FCLOSE). The primary difference between these words and their non-parenthesis versions, is that these words do not clear out memory in the process of closing or opening the file specified. Here is an example of how a screen might be read in from cassette to disk:

```
1 READ                <RETURN>
(FILE) CASSTODISK     <RETURN>
FCLOSE                <RETURN>
```

In this example, one screen was read into screen buffer #1, and then the disk file CASSTODISK was created, or opened. The FLCOSE word then closed the file, automatically FLUSHing any updated screens to disk. One thing to note here is that READ and READS automatically UPDATE the screen after it has been read in from cassette.

The (FCLOSE) word is similarly useful when transferring screens from one disk file to another. There is a utility source screen included, which allows the transfer of up to 16 screens from one disk file to another on the same or a different disk.

Two final words are provided which allow your programs to manipulate files: (↑FILE) and ↑FILE. These two words require you to pass them an address of a string, and a count of the length of the string. The string is then opened as a file on disk. (↑FILE) is a lower level word, which does not clear out memory before opening the file. ↑FILE performs a full FCLOSE before opening the new file. Here is an example of how ↑FILE might be used:

```
: DATA.FILE.OPEN ↑ DATA↑  ↑ FILE ;
```

Here the ↑ symbol is used to specify an in-line string, called DATA, which is the name of the out data file. ↑ returns the address and count of the in-line string when executed, and these parameters are used by ↑ FILE to open the file DATA on the current disk drive. Later this file can be closed with FCLOSE, or another file can be opened.

LOADING MULTIPLE SCREENS

The word FLOAD opens a file from the disk drive and loads screen 1. Screen 1 is then expected to load the rest of the file.

FLOAD can be used within a file to load other files. However, the files loaded from within a file using FLOAD cannot themselves perform an FLOAD. In other words, the FLOAD command is not nestable.

For example, suppose you had a file called TEST which you would like to load three other files called TEST1, TEST2, and TEST3. The following statements can be placed in screen 1 of the file TEST to cause it to load these other files:

```
FLOAD TEST1↑
FLOAD TEST2↑
FLOAD TEST3↑
```

FLOAD may not be used in a colon-definition.

When FLOAD is executed from within a file during a load, the current file is saved during the FLOAD operation. When FLOAD completes, operation will revert to the file which contained the FLOAD.

FLOAD can also be executed from the keyboard. When FLOAD completes, it closes the files and returns to the keyboard.

The advantage in using this method is that you can break up your program into smaller files. This type of file system allows you to modularize your program by placing small programs in separate files.

8 USEFUL DEFINITIONS AND TOOLS

DEFINITIONS

This section provides several useful definitions you might need in experimenting with your C-64. These defined words are not in 64FORTH, but may be entered into an edit screen and saved to cassette or disk.

Sprites

First, some extra sprite words:

```
DECIMAL
        ( turn on all sprites, white across screen )
: VIEW ( - - - )
        8 0
        DO I 40 * 30 + 100 I CXY
                1 I COLOR I SHOW
        LOOP ;

        ( turn all sprites off )
: RID ( - - - )
        8 0
        DO I HIDE LOOP ;

        ( place all sprites on, around sprite edit area )
: EDVIEW ( - - - )
        4 0
        DO 40 I 1 + 50 * I CXY 1 COLOR I SHOW
        LOOP
        8 4
        DO 294 I 3 - 50 * I CXY 1 COLOR I SHOW LOOP ;

        ( reset the position of sprite n1 to 0.0 )
: SPRESET ( N1 - - - )
        0 0 ROT CXY ;
```

Dumb Terminal

In addition to the definitions for the RS-232 serial printer words in the Printer Output section, here are some words you can use to make a simple dumb terminal:

```
HEX
        ( scan serial port for a character waiting )
: S?TERMINAL ( - - - c1 )
        SYSTEM 2 CHKIN ?TERMINAL 5 CHKIN ;

        ( send character to serial port )
: SEMIT ( c1 - - - )
        SYSTEM 2 CHKOUT EMIT 3 CHKOUT ;
        (a simple dumb terminal program—can be aborted by
         pressing CTRL D )
    : TERM ( - - - )
        SYSTEM
        SCONTROL 2+ @ 0=
        IF 300 BAUD         ( default to 300 baud )
        ENDIF               ( if no other baud set )
        SON 3 CHKOUT 5 CHKIN
        BEGIN ? TERMINAL -DUP
            IF DUP 4 = ( CTRL D will abort )
                IF SOFF ." ABORTING" DROP QUIT
                ENDIF SEMIT
            ENDIF
            S?TERMINAL -DUP
            IF EMIT ENDIF
        AGAIN ;
```

Vectoring

This group of definitions allows you to control the vectoring capability of 64FORTH at a higher level, in a simpler fashion

```
    DECIMAL
        ( check a1, to assure it is the PFA of a vectored word )
    : ?VECTOR ( a1 - - - a1 )
        DUP CFA @ LIT KEY @ -
        IF CR HERE ID. ." NOT A VECTORED WORD" SP! QUIT
        ENDIF ;

        ( redirect vectored word <text1> to perform function
         specified by <text2> )
    : VECTOR! ( - - - ) ( VECTOR! <text 1> <text 2> )
        -FIND 0= 0 ?ERROR DROP ?VECTOR C@ I/O +
        -FIND 0= 0 ?ERROR DROP CFA
        STATE @
        IF [COMPILE] LITERAL [COMPILE] LITERAL
           COMPILE !
        ELSE SWAP !
        ENDIF ; IMMEDIATE

        ( reset vectored word <text1> to its default function )
    : RESETVECTOR ( — ) ( RESETVECTOR <text1> )
        -FIND 0= 0 ?ERROR DROP
        C@ DUP VWORDS + @ SWAP I/O + SWAP
        STATE @
        IF [COMPILE] LITERAL [COMPILE] LITERAL
           COMPILE !
        ELSE SWAP !
        ENDIF ; IMMEDIATE
```

Now that you have good vector control, you can put it to use. Suppose you have an RS-232 printer, which needs a linefeed to follow each carriage return. The following set of words will redefine CR to accomplish this new function.

```
DECIMAL
        ( the new definition of CR )
: CRLF ( --- )
        13 EMIT 10 EMIT 0 OUT ! ;

        ( here is a word to revector CR to CRLF )
: LFON ( --- )
        VECTOR! CR CRLF ;

        ( and a word to restore the vector to its original function )
: LFOFF ( --- )
        RESETVECTOR CR ;
```

Directory

If you have a disk drive, you can use this utility to get a directory from the disk in the drive. However, this is not a perfect directory program and empty directory entries are likely to show up as garbage.

```
DECIMAL
        ( read in n1 characters from serial bus, and throw them away )
: ACPTRS ( n1 --- )
        0
        DO SYSTEM ACPTR DROP
        LOOP ;

        ( close a disk channel, n1 )
: DCLOSE ( n1 --- )
        SYSTEM 15 AND 224 OR DR # @ LISTEN
        SECOND UNLSN ;

        ( print the file type of value n1 )
: .FTYPE ( n1 --- )
        DUP 132 = IF ."   REL FORTH FILE" ENDIF
        DUP 130 = IF ."   PRG" ENDIF
            129 = IF ."   SEQ" ENDIF ;
        ( and finally the directory word itself )
  : DIR ( --- ) SYSTEM CMD I
        DCHAN @ 1+ 240 OR DLSN ↑ $ ↑STYPE 13 CIOUT UNLSN
              ?DERR
        DCHAN @ 1+ DTLK 142 ACPTRS CR 4 SPACES 18 0
        DO ACPTR DUP 128 <
              IF EMIT ELSE DROP ENDIF
        LOOP ." ," 6 0
        DO ACPTR 127 AND EMIT LOOP CR CR
        ." FILENAME     SECTORS TYPE" CR 88 ACPTRS
        BEGIN ACPTR DUP ASCII G 128 + -
        WHILE DUP 0= IF 28 ACPTRS DROP
              IF    CR 2 ACPTRS SPACE 15 0
                    DO ACPTR 127 AND EMIT
                    LOOP 9 ACPTRS
                    ACPTR ACPTR SWAP 256 * + 3 .R SPACE
              .FTYPE ENDIF 1+ DUP 8 =
              IF DROP 0 ACPTR DROP ELSE 3 ACPTRS ENDIF
        REPEAT DROP UNTLK DCHAN @ 1+ DCLOSE CR ;
```

Copying Screens

This next utility is also for disk drive owners. It allows you to copy screens from one file to another, either on the same disk or on another disk with one disk drive. There is a limit of sixteen screens that can be moved at once, since that is the number of screen buffers in this system.

```
DECIMAL
      ( copy screens from one file to another, possibly on another
        disk drive; all input is prompted )
: FCOPY ( --- )
CR ." SOURCE FILE          ->" QUERY
BL WORD HERE DUP C@ 1+ FN    SWAP CMOVE
CR ." DESTINATION FILE ->" QUERY
BL WORD HERE DUP C@ 1+ FN 20 + SWAP CMOVE
FN COUNT ↑ FILE
CR ." FIRST SCREEN  ->" QUERY INTERPRET
CR ." LAST SCREEN  ->" QUERY INTERPRET
DO I BLOCK UPDATE
LOOP (FCLOSE) CR
." SWAP DISKS IF REQUIRED AND PRESS ANY KEY"
KEY DROP CMD I ↑
FN 20 + COUNT ( ↑FILE) FCLOSE ;
```

Time

```
HEX ( --- d1 ) ( returns a double number time )
      : ?TIME 0 0 0 FFDE SYS >R 100 * + R> ;
```

Auto-Repeat

This definition causes all keys to repeat when held down.

```
( --- )
( Toggle the auto-repeat switch in the system )
( WARNING: May also cause some keybounce )

DECIMAL
: AUTO .REPEAT 650 128 TOGGLE ;
```

Sound Extensions

The following definitions control some of the sound features:

```
DECIMAL
: RANDOM VOICE3 20000 FREQ! NOISE
      VOLUME@ 128  AND VOLUME! GATE1 S! ;
      ( Turn on random generator )
: RND ( --- c ) OSC3@ ; ( Returns value between 0 and 255 )
: BEEP VOICE2 FREQ! SQUARE 15 VOLUME! 2 ATTACK!
      2048 PWIDTH! 0 DECAY! 15 SUSTAIN! 0 RELEASE! GATE1 S!
      200 0 DO LOOP GATE0 0 FREQ! S! ;
: BEEPS 20 0 DO 8000 BEEP LOOP ;
```

FORTH-79 Extensions

Here are some sample extensions to make 64FORTH more compatible with FORTH-79.

```
DECIMAL
( n1 --- n2 )
: 1- 1 - ;
: 2- 2 - ;
: 2* 2 * ;
: 2/ 2 / ;

( Returns the limit value in a DO . . . LOOP )
: I' R> R> R SWAP >R SWAP >R SWAP > R ;

(Returns index value of DO . . . LOOP outside of current
DO . . . LOOP )
( Can only be used with nested DO . . . LOOPs )
: J R> R> R SWAP >R SWAP >R SWAP >R ;

( Clears the screen )
PAGE 147 EMIT ;

( n1/n2 --- )
( Prints n1 unsigned in a field of n2 characters )
: U.R 0 SWAP D.R ;

( Record length/number of records --- )
( n1/n2 --- <text> ) (compiling)
( n1 --- a1 ) (executing)
: ARRAY <BUILDS OVER , * ALLOT
      DOES> DUP @ ROT * + + 2+ ;
```

When compiling, an array with name <text> is created with n2 records each having a length of n1 bytes. At execution time, when <text> is executed, address a1 (the first byte of record n1) is returned.

Miscellaneous Extensions

```
( d1/d2 --- d1/d2/d1 )
( Duplicate double number d1 over top of double number d1 )
: 2 OVER >R >R 2DUP R> R> 2SWAP ;

( --- n1 ) ( Return the amount of free memory )
: FREE EM @ HERE - ;

( --- n1 )
HEX
( Read the system status byte )
: STATUS 0 0 0 FFB7 SYSTEM SYS 2DROP ;

( Print name of current vocabulary )
(                                  )
: .VOC CONTEXT @ 4 - NFA ID. ;

( Print the value of the current base in decimal )
: .BASE BASE @ DUP DECIMAL . BASE ! ;
```

TOOLS

This section describes some of the useful FORTH words to help you in your programming efforts.

Debugger

64FORTH includes a very useful debugger that allows you to TRACE and STEP through high level FORTH definitions. Here is an example of a TRACE:

```
TRACE HEX <RETURN>
```

The computer displays the following:

```
CLIT    >>  16
BASE    >>  16 2406
!       >> EMPTY

OK
```

The actual definition of HEX is as follows:

```
: HEX 16 BASE ! ;
```

The word CLIT stands for character literal; the value 16 is less than 256 and as a result it was compiled as a character literal. Numbers in FORTH definitions cannot be directly interpreted at run-time, so FORTH places a word before them called LIT or CLIT. LIT or CLIT then picks up the number following and places it on the data stack. The numbers to the right of the >> symbols represent the status of the data stack after each word in the definition has been executed. You will also notice the word EMPTY after the final >> symbol indicating the data stack was empty. Almost any high level definition can be traced in this way, however some words will need to have parameters passed to them before your trace can be performed. The following is an example of such a word:

```
3 4 TRACE * <RETURN>
```

This operation traces the multiplication of two numbers: 3 and 4. The computer prints the result:

```
U*      >>    12    0
DROP    >>    12
```

The definition of * is:

```
: * U* DROP ;
```

The numbers 3 and 4 were placed on the stack before the word TRACE was entered. Then TRACE was entered followed by *—the word you want to trace. The result is the execution of U*, which multiplies the two numbers together and returns the double number result, 12. The high word of the double number result is not needed, so it is discarded with the following DROP. The final result 12 is left on the data stack. You can print it out with the . (dot) command:

```
.          <RETURN>   (entered by you)
12                    (printed by the computer)
```

You can interrupt a TRACE at any time by pressing the space bar. You can then list the definition one line at a time with the STEP command, or continue the trace with the CONT command:

```
TRACE VLIST <RETURN>

Press the space bar after a few seconds.

STEP    <RETURN>   (causes a single step execution)
STEP    <RETURN>   (step again)

CONT    <RETURN>   (resumes tracing continuously)

Press the space bar again after a few seconds.
```

If you were to trace all of the definition for VLIST, it would take several hours to complete. Tracing actually runs the definition you are tracing, but it is much slower than actual execution. Clear the stack with the SP! word:

```
SP!     <RETURN>
```

If you want to start a trace of a definition in the single step mode rather than a continuous listing, you can select a word to trace without actually starting the trace with the EMULATE word as follows:

```
EMULATE DECIMAL    <RETURN>
```

The computer will return with the OK prompt, and wait for further instructions. You can now use STEP or CONT to actually perform the emulation or trace.

Decompiler

A decompiler has been included in the 64FORTH to help you learn how a FORTH system is designed. The word SOURCE followed by any high-level word decompiles and prints out the words used to make up the definition. An example follows:

```
        SOURCE   /   <RETURN>
```

Prints: : / /MOD SWAP DROP ;S

This shows you that the word / is created by using the /MOD operator, and discarding the second stack entry. You can dissect this definition further by decompiling /MOD:

```
        SOURCE / MOD  <RETURN>
```

Prints: : /MOD >R S–>D R> M/ ;S

/MOD is created with a word M/, which you can SOURCE:

```
        SOURCE M/  <RETURN>
```

Prints: : M/ OVER >R >R DABS R ABS U/ R> XOR +–
        SWAP R> +– SWAP ;S

If you want more information, each of the words appearing above is a standard fig-FORTH word appearing in the Glossary.

Dump Contents of Memory

64FORTH includes a DUMP utility used in the following way:

```
        <address> <count> DUMP <RETURN>
```

This word displays memory contents starting at the address specified for the number of bytes in length <count>. Each line will have eight bytes preceeded by the address of the first byte in the line.

9 ASSEMBLER

This chapter is a reprint of the documentation which accompanied W. Ragsdale's 6502 assembler as published in FORTH DIMENSIONS Volume III, Number 5. This chapter is intended for programmers already familiar with assembly programming.

6502 FORTH ASSEMBLER by W. Ragsdale

Introduction

This article should further polarize the attitudes of those outside the growing community of FORTH users. Some will be fascinated by a label-less, macro assembler whose source code is only 96 lines long! Others will be repelled by reverse Polish syntax and the absence of labels.

The author immodestly claims that this is the best FORTH assembler ever distributed. It is the only assembler that detects all errors in op-code generation and conditional structuring. It is released to the public domain as a defense mechanism. Three good 6502 assemblers were submitted to the FORTH Interest Group but each had some lack. Rather than merge and edit for publication, the author chose to publish his with all the submitted features plus several more.

Imagine having an assembler in 1300 bytes of object code with:

1. User macros (like IF, UNTIL,) definable at any time.
2. Literal values expressed in any numeric base, alterable at any time.
3. Expressions using any resident computation capability.
4. Nested control structures without labels with error control.
5. Assembler source itself in a portable high level language.

Overview

FORTH is provided with a machine language assembler to create execution procedures that would be time inefficient, if written as colon-definitions. It is intended that "code" be written similarly to high level, for clarity of expression. Functions may be written first in high level, tested, and then re-coded into assembly, with a minimum of restructuring.

The Assembly Process

Code assembly consists of interpreting with the ASSEMBLER vocabulary as CONTEXT. Thus each word in the input stream will be matched according to the FORTH practice of searching CONTEXT first, and then CURRENT.

```
ASSEMBLER      (now CONTEXT)
FORTH          (chained to ASSEMBLER)
user's         (CURRENT if one exists)
FORTH          (chained to user's vocabulary)
try for literal number
else, do error abort.
```

The above sequence is the usual action of FORTH's text interpreter, which remains in control during assembly.

During assembly of CODE definitions, FORTH continues interpretation of each word encountered in the input stream (not in the compile mode). These assembler words specify operands, address modes, and op-codes. At the conclusion of the CODE definition, a final error check verifies correct completion by "unsmudging" the definition's name, to make it available for dictionary searches.

Run-Time, Assembly-Time

One must be careful to understand at what time a particular word definition executes. During assembly, each assembler word interpreted executes. Its function at that instant is called 'assembling' or assembly-time'. This function may involve op-code generation, address calculation, mode selection, and so forth.

The later execution of the generated code is called 'run-time'. This distinction is particularly important with the conditionals. At assembly time each such word (that is, IF, UNTIL, BEGIN, etc.) itself 'runs' to produce machine code which will later execute at what is labeled 'run-time' when its named code definition is used.

An Example

As a practical example, here's a simple call to the system monitor (KIM-I only), via the NMI address vector (using the BRK op-code).

```
CODE MON (exit to monitor)
      BRK, NEXT JMP, END-CODE
```

The word CODE is first encountered and executed by FORTH. CODE builds the following name "MON" into a dictionary header and calls ASSEMBLER as the CONTEXT vocabulary.

The "(" is next found in the FORTH and executed to skip until ")". This method skips over comments. Note that the name after CODE and the ")" after "(" must be on the same text line.

Op-Codes

BRK, is next found in the assembler as the op-code. When BRK, executes, it assembles the byte value 00 (zero) into the dictionary as the op-code for "break to monitor" via "NMI".

Many assembler word's names end in ",". The significance of this is:

1. The comma shows the conclusion of a logical grouping that would be one line of classical assembly source code.
2. "," compiles into the dictionary; thus a comma implies the point at which code is generated.
3. The "," distinguishes op-codes from possible HEX numbers ADC and ADD.

Next

FORTH executes your word definitions under control of the address interpreter, named NEXT. This short code routine moves execution from one definition, to the next. At the end of your code definition, you must return control to NEXT or else to code which returns to NEXT.

Return of Control

Most 6502 systems can resume execution after a break, since the monitor (KIM-1 only) saves the CPU register contents. Therefore, we must return control to FORTH after a return from the monitor. NEXT is a constant that supplies the machine address of FORTH's address interpreter ($8115 for 64FORTH). Here it is the operand for JMP,. As JMP, executes, it assembles a machine code jump to the address of NEXT from the assembly time stack value.

Security

Numerous tests are made within the assembler for user errors:

1. All parameters used in CODE definitions must be removed.
2. Conditionals must be properly nested and paired.
3. Address modes and operands must be allowed for the op-codes.

These tests are accomplished by checking the stack position (in CSP) at the creation of the definition name and comparing it with the position at END-CODE. Legality of address modes and operands is insured by means of a bit mask associated with each operand.

Remember that if an error occurs during assembly, END-CODE never executes. The result is that the "smudged" condition of the definition name remains in the "smudged" condition and will not be found during dictionary searches.

The user should be aware that one error not trapped is referencing a definition in the wrong vocabulary:

i.e., 0= of ASSEMBLER when you want
0= of FORTH

Summary (KIM-1 only)

The object code of our example is:

```
3059  83  4D  4F  CE     CODE MON
305D  4D  30             link field
305F  61  30             code field
3061  00                 BRK
3062  4C  42  02         JMP NEXT
```

Op-Codes, revisited

The bulk of the assembler consists of dictionary entries for each op-code. The 6502 one mode op-codes are:

BRK,	CLC,	CLD,	CLI,	CLV,
DEX,	DEY,	INX,	INY,	NOP,
PHA,	PHP,	PLA,	PLP,	RTI,
RTS,	SEC,	SED,	SEI,	TAX,
TAY,	TSX,	TXS,	TXA,	

When any of these are executed, the corresponding op-code byte is assembled into the dictionary.

The multi-mode op-codes are:

ADC,	AND,	CMP,	EOR,	LDA,
ORA,	SBC,	STA,	ASL,	DEC,
INC,	LSR,	ROL,	ROR,	STX,
CPX,	CPY,	LDX,	LDY,	STY,
JSR,	JMP,	BIT,		

These usually take an operand, which must already be on the stack. An address mode may also be specified. If none is given the op-code uses z-page or absolute addressing. The address modes are described by:

SYMBOL	MODE	OPERAND
.A	accumulator	none
#	immediate	8 bits only
,X	indexed X	z-page or absolute
,Y	indexed Y	z-page or absolute
X)	indexed indirect X	z-page only
)Y	indirect indexed Y	z-page only
)	indirect	absolute only
none	memory	z-page or absolute

Examples

Here are examples of FORTH vs. a conventional assembler. Note that the operand comes first, followed by any mode modifier, and then the op-code mnemonic. This makes best use of the stack at assembly time. Also, each assembler word is set off by blanks, as is required for all FORTH source text.

```
FORTH ASSEMBLER                          CONVENTIONAL ASSEMBLER
.A            ROL,                       ROL   A
1 #           LDY,                       LDY   #1
DATA ,X       STA,                       STA   DATA,X
DATA ,Y       CMP,                       CMP   DATA,Y
06 X)         ADC,                       ADC   (06,X)
POINT )Y      STA,                       STA   (POINT),Y
VECTOR )      JMP,                       JMP   (VECTOR)
```

(.A distinguishes from the HEX number 0A)

The word DATA and VECTOR specify machine addresses. In the case of " 06)X ADC, " the operand memory address $0006 was given directly. This is occasionally done if the usage of a value does not justify devoting the dictionary space to a symbolic value.

6502 Conventions

Stack Addressing

The data stack is located in z-page usually addressed by "Z-PAGE,X". The stack starts near $009E (64FORTH is at $0060) and grows downward. The X index register is the data stack pointer. Thus, incrementing X by two removes a data stack value; decrementing X twice makes room for one new data stack value.

Sixteen-bit values are placed on the stack according to the 6502 convention; the low byte is at low memory, with the high byte following. This allows "indexed,indirect X" directly off a stack value.

The bottom and second stack values are referenced often enough that the support words BOT and SEC are included. Using:

```
BOT  LDA,   assembles LDA (0,X) and
SEC  ADC,   assembles ADC (2,X)
```

BOT leaves 0 on the stack and sets the address mode to ,X. SEC leaves 2 on the stack also setting the address mode to ,X.

Here is a pictorial representation of the stack in z-page:

```
[                                ]
- - - - - - - - - - - - - - - - - -
[            sec high            ]
[            sec low             ]
- - - - - - - - - - - - - - - - - -
[            bot high            ]
[            bot low             ]  <-- X offset above $0000
- - - - - - - - - - - - - - - - - -
```

Here is an example of code to "or" to the accumulator four bytes on the stack:

```
BOT         LDA,                    LDA (O,X)
BOT 1+      ORA,                    ORA (1,X)
SEC         ORA,                    ORA (2,X)
SEC 1+      ORA,                    ORA (3,X)
```

To obtain the 14-th byte on the stack:

```
BOT 13 +    LDA,                    LDA (13,X)
```

Return Stack

The FORTH Return Stack is :ocated in the 6502 machine stack in page 1. It starts at $01FE and builds downward. No lower bound is set or check as Page 1 has sufficient capacity for all (non-recursive) applications.

By 6502 convention the CPU's register points to the next free byte below the bottom of the return stack. The byte order follows the convention of low significance byte at the lower address.

Return stack values may be obtained by: PLA, PLA, which will pull the low byte and then the high byte from the return stack. To operate on arbitrary bytes, the method is:

1. Save X in XSAVE.
2. Execute TSX, to bring the S register to X.
3. Use RP) to address the lowest byte of the return stack. Offset the value to address higher bytes. (Address mode is automatically set to ,X)
4. Restore X from XSAVE.

As an example, this definition non-destructively tests that the second item on the return stack (also the machine stack) is zero.

```
CODE IS-IT      ( zero ? )
     XSAVE      STX,        ( save current value of X register )
                TSX,        ( setup for return stack access )
     RP) 2+     LDA,
     RP) 3 +    ORA,
0=   IF,        INY,        ( if zero bump Y to one )
     ENDIF,
     TYA,
     PHA,                   ( save result on stack )
     XSAVE      LDX,        ( restore return stack pointer )
     PUSH       JMP,        ( go push a boolean from stack )
     END-CODE               ( terminate the CODE definition )
```

```
                        [                                ]
                        [          Return Stack          ]
                        ----------------------------------
                        [           high byte            ] second
RP) = $0101,X --->      [           low byte             ] item
                        ----------------------------------
                        [           high byte            ] bottom
                        [           low byte             ] item
                        ----------------------------------
               S ---> [           free byte            ]
```

FORTH Registers

Several FORTH registers are available only at the assembly level and have been given names that return their memory addresses. They are:

IP	Address of the Interpretive Pointer, Specifying the next FORTH address which will be interpreted by next.
W	Address of the Interpretive Pointer, specifying the next definition just interpreted by NEXT.
UP	User Pointer containing address of the base of the user area.
N	A utility area in z-page from N-1 through N+7.

CPU Registers

When FORTH execution leaves NEXT to execute a CODE definition, the following conventions apply:

1. The Y index register is zero. It may be freely used.
2. The Z index register defines the low byte of the bottom data stack item relative to machine address $0000.
3. The CPU stack pointer S points one byte below the bottom return stack item. Executing PLA, will pull this byte to the accumulator.
4. The accumulator may be freely used.
5. The processor is in binary mode and must be returned in that mode.

XSAVE

XSAVE is a byte buffer in z-page, for temporary storage of the X register. Typical usage, with a call which will change X, is:

```
CODE DEMO
      XSAVE      STX,         ( save current value of X )
      USER'S     JSR,         ( Go to a user's routine )
      XSAVE      LDX,         ( restore value of X register )
      NEXT       JMP,         ( return to FORTH )
      END-CODE                ( terminate the CODE definition )
```

N

When absolute memory registers are required, use the 'N Area' in the base (zero) page. These registers may be used as pointers for indexed/indirect addressing or for temporary values. As an example of use, see CMOVE in the "fig MODEL" installation manual.

The assembler word N returns the base address (64FORTH=$0068). The N area spans 9 bytes, from N−1 to N+7. Conventionally, N−1 holds one byte and N, N+2, N+4, N+6 are pairs which may hold 16 bit values. See SETUP for help on moving values to the N area.

It is very important to note that many FORTH procedures use N. Thus, N may only be used within a single code definition. Never expect that a value will remain there, outside a single definition.

```
CODE DEMO       HEX
     6 #        LDA,
     N 1 -      STA,         ( setup a counter )
BEGIN,
     8001       BIT,         ( tickle a port in KIM-1 )
     N 1 -      DEC,         ( decrement the counter )
0=   UNTIL,                  ( loop till counter = zero )
     NEXT       JMP,         ( return to FORTH )
     END-CODE                ( complete the definition )
```

SETUP

Often we wish to move stack values to the N area. The subroutine SETUP has been provided for this purpose. Upon entering SETUP the accumulator specifies the quantity of 16-bit values to be moved to the N area. That is, A may be 1, 2, 3, or 4, only:

```
     3 #        LDA,         ( setup to move three values )
     SETUP      JSR,         ( move 3 16 bit values to N area )

stack before    N after      stack after
     H high                         H
     G low                    bot-G
     F          F
     E          E
     D          D
sec-> C         C
     B          B
bot-> A      N-->A
```

Control Flow

FORTH discards the usual convention of assembler labels. Instead, two replacements are used. First, each FORTH definition name is permanently included in the dictionary, This allows procedures to be located and executed by name at any time as well as compiled within other definitions.

Secondly, within a code definition, executing flow is controlled by label-less branching according to "structured programming". This method is identical to the form used in colon-definitions. Branch calculations are done at assembly time by temporary stack values placed by the control words:

```
BEGIN,    UNTIL,    IF,    ELSE,    ENDIF,
```

(THEN, is used in some assemblers in place of ENDIF,)

Here again, the assembler words end with a comma to indicate that code is being produced and to clearly differentiate from the high-level form.

One major difference occurs! High-level flow is controlled by run-time boolean values on the data stack. Assembly flow is instead controlled by processor status bits. The programmer must indicate which status bit to test, just before a conditional branching word (IF, and UNTIL,).

Examples are:

```
        PORT        LDA,
0=      IF,                          ( read port, if equal to zero do )
                 <function A>        ( <function A> )
        ENDIF,
        PORT        LDA,
0= NOT  IF,                          ( read port, if not equal to zero )
                 <function A>        ( do <function A> )
        ENDIF,
```

The conditional specifiers for 6502 are:

CS	test carry set	C=1 in processor status
CS NOT	test carry clear	C=0
0<	byte less than zero	N=1
0< NOT	test positive	N=0
0=	equal to zero	Z=1
0= NOT	test not equal zero	Z=0
OVS	overflow set	V=1 (added to 64FORTH)
OVS NOT	overflow clear	V=0 (added to 64FORTH)

Conditional Looping

A conditional loop is formed at assembler level by placing the portion to be repeated between BEGIN, and UNTIL,:

```
        6 #         LDA,
        N           STA,          ( define loop counter in N )
   BEGIN,
        PORT        DEC,          ( repeated action )
        N           DEC,
0= UNTIL ,                        ( N reaches zero )
```

First, the byte at address N is loaded with the value 6. The beginning of the loop is marked (at assembly time) by BEGIN,. Memory at PORT is decremented, then the loop counter N is decremented. Of course, the CPU updates its status register as N is decremented. Finally, a test for Z=1 is made; if N hasn't reached zero, execution returns to BEGIN,. When N reaches zero (after executing PORT DEC, 6 times) execution continues ahead after UNTIL,. Note that BEGIN, generates no machine code, but is only an assembly time locator.

Conditional Execution

Paths of execution may be chosen at assembly in a similar fashion as done in colon-definitions. In this case, the branch is chosen based on a processor status condition code.

```
        PORT      LDA,
  0=    IF,
             <function A>    ( executed if PORT is zero )
        ENDIF,
                             ( then continue on with rest )
```

In this example, the accumulator is loaded from PORT. The zero status is tested if set (Z=1). If so, the code (for zero set) is executed. Whether the zero status is set or not, execution will resume at ENDIF,.

The conditional branching also allows a specific action for the false case. Here we see the addition of the ELSE, part.

```
        PORT      LDA,
  0=    IF,
             <function A>    ( executed if PORT is zero )
        ELSE,
             <function B>    ( executed if PORT is not zero )
        ENDIF,
                             ( then continue on with rest )
```

The test of PORT will select one of two execution paths, before resuming execution after ENDIF,. The next example increments N based on bit D7 of PORT:

```
        PORT      LDA,
  0<    IF,
               N  DEC,       ( if D7=1, decrement N )
        ELSE,
               N  INC,       ( if D7=0, increment N )
        ENDIF,
                             ( continue ahead )
```

Conditional Nesting

Conditionals may be nested according to the conventions of structured programming. That is, each conditional sequence begun (IF, BEGIN,) must be terminated (ENDIF, UNTIL,) before the next earlier conditional is terminated. An ELSE, must pair with the immediately preceeding IF,.

```
     BEGIN,        <code always executed>
           CS IF,  <code if carry set>
                   ELSE,    <code if carry clear>
                   ENDIF,
          0= NOT UNTIL, ( loop till condition flag is non-zero )
               <code that continues onward>
```

Next is an error that the assembler security will reveal.

```
     BEGIN,         PORT LDA,
           0=    IF,  BOT INC,
                    0= UNTIL,  ENDIF,
```

The UNTIL, will not complete the pending BEGIN, since the immediately preceeding IF, is not completed. An error trap will occur at UNTIL, saying "conditionals not paired".

Return of Control, revisited

When concluding a code definition, several common stack manipulations often are needed. These functions are already in the nucleus, so we may share their use just by knowing their return points. Each of these returns control to NEXT.

POP	Remove one 16-bit stack value.
POPTWO	Remove two 16-bit stack values.
PUSH	Add two bytes to the data stack.
PUT	Write two bytes to the data stack, over the present bottom of the stack.

Our next example complements a byte in memory. The bytes' address is on the stack when INVERT is executed.

```
     CODE  INVERT       ( a memory byte )  HEX
           BOT X)       LDA,   ( fetch byte addressed by stack )
           FF #         EOR    ( complement the accumulator )
           BOT X)       STA,   ( replace result in memory )
           POP          JMP,   ( discard pointer from stack )
           END-CODE                   ( and return to next )
```

A new stack value may result from a code definition. We could program placing it on the stack by:

```
     CODE  ONE          ( put 1 on the stack )
                        DEX,
                        DEX,   ( make room on the data stack )
           1 #          LDA,   ( get a 1 in accumulator )
           BOT          STA,   ( store low byte )
           BOT 1+       STA,   ( high byte stored from Y since=zero )
           NEXT         JMP,   ( return to FORTH )
           END-CODE
```

A simpler version could use PUSH:

```
CODE  ONE
      1 #          LDA,
                   PHA,    ( push low byte to machine stack )
                   TYA,    ( clear accumulator, high byte=zero )
      PUSH         JMP,    ( go push to data stack )
      END-CODE
```

The convention for PUSH and PUT is:

1. push the low byte onto the machine stack.
2. leave the high byte in accumulator.
3. jump to PUSH or PUT.

PUSH will place the two bytes as the new bottom of the data stack. PUT will over-write the present bottom of the stack with the two bytes. Failure to push exactly one byte on the machine stack will disrupt execution upon usage!!

Fooling Security

Occasionally we wish to generate unstructured code. To accomplish this, we can control the assembly time security checks for our purpose. First, we must note the parameters utilized by the control structures at assembly time. The notation below is taken from the assembly glossary. The --- indicates assembly time execution, and separate input stack values from the output stack values of the words execution.

```
BEGIN,     ==>                        --- addrB 1
UNTIL,     ==>    addrB 1       cc    ---

IF,        ==>                  cc    --- addrI 2
ELSE,      ==>    addrI 2             --- addrE 2
ENDIF,     ==>    addrI 2             ---
             or   addrE 2             ---
```

The address values indicate the machine location of the corresponding 'B'EGIN, 'I'F, or 'E'LSE,. cc represents the condition code to select the processor status bit referenced. The digit 1 or 2 is tested for conditional pairing.

The general method of security control is to drop off the check digit and manipulate the addresses at assembly time. The security against errors is less, but the programmer is usually paying intense attention to detail during this effort.

To generate the equivalent of the high level:

```
BEGIN  <a>  WHILE  <b>  REPEAT
```

We write in assembly:

```
BEGIN,  DROP    ( the check digit 1, leaving addrB )
       <a>
       CS IF,   ( leaves addrl and digit 2 )
           <b>
           ROT  ( bring addrB to bottom )
           JMP, ( to addrB of BEGIN, )
       ENDIF,   ( complete false forward branch from IF, )
```

It is essential to write the assembly time stack on paper, and run through the assembly steps, to be sure that the check digits are dropped and re-inserted at the correct points and addresses are correctly available.

NOTE: The ASSEMBLER glossary is included in the main glossary at the end of this manual.

APPENDIX A FORTH-79 DIFFERENCES

64FORTH is a fig-FORTH implementation of the FORTH language. The differences between 64FORTH and FORTH-79 will be covered here on a page by page basis from STARTING FORTH to help you understand and adjust to the differences.

Page in STARTNG FORTH	Comments
12, 13	Change the definition of margin to: `: MARGIN CR 10 SPACES ;` The Commodore 64 has a 40 column screen. The above change will improve the appearance of the demo.
60	The screens in 64FORTH are numbered 1 to 16.
68	Remember that lines in 64FORTH longer than 40 characters will wrap around. 64FORTH still has 64 character lines internally.
77	S The editor S command is not included, due to space restriction. Use the F (find) command. In 64FORTH, the S command is used for Spread a line, which makes room at the current edit line for additional text to be inserted.
91	0> This operator is not supported. Use 0 >. (leave a space).
101	?DUP Not supported. Use −DUP. ABORT" Not supported. Use the following colon-definition: `IF ." ERROR MESSAGE " SP! QUIT ENDIF`
107	1− 2− 2* 2/ Not supported Use: 1 − 2 − 2 * 2 / (Leave a space between the numbers and the operands.)
110	I' and J are not supported. You can define them as follows: `: I' R> R> R SWAP >R SWAP >R ;` `: J R> R> R> SWAP >R SWAP >R SWAP >R ;`
143	PAGE Not supported. You can define it as follows: `DECIMAL : PAGE 147 EMIT ;` U.R Not supported. A definition follows: `: U.R 0 SWAP D.R ;`

153 2* and 2/ not usable. Leave a space between the number and the operator: 2 * and 2 / .

161 /LOOP Not needed in 64FORTH.

164 DOUBLE NUMBER DELIMITERS. 64FORTH only recognizes the decimal point "." as a double number delimiter.

173 Only D+ and D.R supported; for DNEGATE use DMINUS.

174 Only M* and M/ supplied, but use the definitions in the fig-FORTH installation manual for these words.

183 VARIABLE The definition of VARIABLE used in 64FORTH is the fig-FORTH definition, which requires an initial value to be on the stack before creating the variable. For example,

```
12 VARIABLE DATE
```

This will create a variable with an initial value of 12.

193 2VARIABLE, 2CONSTANT, 2@, and 2! are not supplied in 64FORTH.

207 CREATE The word functions differently in 64FORTH/fig-FORTH than in this book. Use the following to create the definition of limits as shown in the book:

```
220 VARIABLE LIMITS 340 , 170 , 100 , 190 ,
```

The rule of thumb is to use VARIABLE in place of CREATE for definitions which do not have DOES> in them. If the definition is of the form:

```
CREATE xxxx DOES> xxxx
```

then use: <BUILDS xxxx DOES> xxxx

This conforms to the fig-FORTH usage.

220 VARIABLE LIMITS 340 , 170 , 100 , 190 ,

The rule of thumb is to use VARIABLE in place of CREATE for definitions which do not have DOES> in them. If the definition is of the form:

CREATE xxxx DOES> xxxx

then use: <BUILDS xxxx DOES> xxxx

This conforms to the fig-FORTH usage.

216 FIND and EXECUTE 64FORTH uses the fig-FORTH word –FIND in place of FIND. In fig-FORTH the word execute must receive the code field address (CFA)

instead of the parameter field address (PFA). Change the example on this page as follows:

```
' GREET CFA EXECUTE        <RETURN>
```

FORTH responds with:

```
HELLO I SPEAK FORTH ok
```

217 VECTORED EXECUTION The techniques will work on 64FORTH with the modification that the addresses obtained with ' (tick) are converted to code field addresses (CFA) by the use of CFA. As an example, line 6 would read:

```
' HELLO CFA 'ALOHA !
```

SAY The definition of SAY in 64FORTH is:

```
: SAY [COMPILE] ' CFA 'ALOHA ! ;
```

There are two changes here. The word ' (tick) is used, and because in fig-FORTH it is immediate, it must be compiled by the [COMPILE] word. The second change is the use of CFA to prepare the address for EXECUTE.

219 NUMBER This definition is vectored in 64FORTH. To revector NUMBER as shown on this page, in 64FORTH you must say:

```
DECIMAL ' (number) I/O 10 + !
```

64FORTH uses a table, where each entry in the table does not have to have a header. Again due to space restrictions.

['] This word is not supported in 64FORTH. Functionally it is the same as ' (tick).

220 NAME LENGTHS 64FORTH supports full names up to 31 characters in length.

230 EXIT In 64FORTH, use ;S.

232 RELOAD Not needed; all code is in ROM.

233, 237 H In 64FORTH use DP.

235 'S In 64FORTH use SP@.

239 OPERATOR Not needed in 64FORTH.

240 >IN Use IN in 64FORTH.
OFFSET Not needed in the cassette version.

243 ASSEMBLER A full 6502 Macro Assembler built into 64FORTH.

245 LOCATE Not supported in 64FORTH.

255-257 UPDATE, FLUSH, SAVE-BUFFERS, EMPTY-BUFFERS, BUFFER These words not used with cassette.

259 LABEL In 64FORTH, change the definition to:

```
: LABEL 8 * ' "LABEL" 3 + + 8 TYPE SPACE ;
```

64FORTH does not support ['] and the word ' serves the same function in a definition.

261 >TYPE Not needed in 64FORTH. Use TYPE.

266 MOVE,<CMOVE Not in 64FORTH.

272 H Use DP.

281 −TEXT In 64FORTH, use (MATCH).

291 VARIABLE, CREATE To create the STARTING FORTH type of definition for VARIABLE, use:

```
: VARIABLE <BUILDS 2 ALLOT DOES> ;
```

To create the 64FORTH/fig-FORTH definition for VARIABLE, define it as follows:

```
: VARIABLE <BUILDS , DOES> ;
```

The fig-FORTH word CREATE is use only for creating CODE word headers.

292 DEFINING WORDS The definition of a DEFINING WORD in the book must be changed to:

```
: DEFINING-WORD <BUILDS (compile time action)
                DOES> (run-time action) ;
```

The example for CONSTANT is then:

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

297 ARRAY The definition of ARRAY must be changed to:

```
: ARRAY <BUILDS OVER , * ALLOT
    DOES> DUP @ ROT * + + 2+ ;
```

313 DOES> For most purposes, 64FORTH is the same as FORTH-79. For more advanced programmers, see the document FORTH-79 STANDARD CONVERSION from the FORTH INTEREST GROUP.

332 JOB, 1FIELD, 2FIELD These must change to account for the different CREATE. Use:

```
20 VARIABLE JOB     24 ,
00 VARIABLE 1FIELD 30 ,
30 VARIABLE 2FIELD 12 ,
```

339 SIMPLE FILES In screen 240 change the definitions as follows:

```
00 VARIABLE SURNAME 16 ,
16 VARIABLE GIVEN      12 ,
28 VARIABLE JOB        24 ,
52 VARIABLE PHONE      12 ,
```

FREE Change the definition of FREE as follows:

```
: FREE 1 MAXRECS 0
    DO I #RECORD ! RECORD C@ 33 <
        IF 0= LEAVE ENDIF
    LOOP
    IF ." FILE FULL " QUIT ENDIF ;
```

339 ' Prefix all occurences of ' with the word [COMPILE], for example:

```
: CHANGE [COMPILE] ' PUT ;
```

347 DENSITY, THETA, STRING Prefix all of the words when defined with a zero (0).

APPENDIX B ERROR MESSAGES

Here is a list of error numbers and messages used by 64FORTH:

ERROR #		ERROR MESSAGE
DEC	HEX	
0	0	64FORTH doesn't know this word.
1	1	THE DATA STACK IS ALREADY EMPTY.
2	2	OUT OF USER MEMORY.
3	3	HAS BAD ADDRESSING MODE
8	8	SCREEN BLOCK RANGE ERROR. You asked for an invalid screen number.
17	11	USE WHILE COMPILING ONLY. This word cannot be used while executing.
18	12	USE DURING EXECUTION ONLY. This word cannot be used while compiling.
19	13	CONDITIONALS NOT PAIRED. Match your IF — ELSE — ENDIFs, etc.
20	14	THIS DEFINITION IS NOT FINISHED. You started a conditional without completing it. IE: BEGIN missing UNTIL.
21	15	THIS WORD IN A PROTECTED DICTIONARY. You can't forget anything below FENCE.
22	16	USE ONLY WHEN LOADING.
23	17	EDIT POINTER IS OFF CURRENT EDIT SCREEN.
24	18	DECLARE YOUR VOCABULARY. Specify the VOCABULARY in which you wish to perform the operation, that is FORTH, EDITOR, ASSEMBLER, or SYSTEM.

The KERNEL has several I/O errors that can occur, and these are printed out as follows.

```
I/O ERROR #5
```

The above example indicates an attempt to talk to a device which is not in the system, such as a disk or printer. The following is a full list of other errors that can be printed by the KERNEL. These message numbers are always printed in DECIMAL.

I/O ERROR #	COMMENT
0	ROUTINE TERMINATED BY STOP KEY.
1	TOO MANY OPEN FILES.
2	FILE ALREADY OPEN.
3	FILE NOT OPEN.
4	FILE NOT FOUND.
5	DEVICE NOT PRESENT.
6	FILE IS NOT AN INPUT FILE.
7	FILE IS NOT AN OUTPUT FILE.
8	FILE NAME IS MISSING.
9	ILLEGAL DEVICE NUMBER.

WHAT DO I DO WHEN IT CRASHES?

64FORTH provides a much more powerful programming environment than BASIC. Unfortunately it also places a higher level of responsibility on you, the programmer, than BASIC. There are many ways to cause FORTH to GO AWAY! When this happens, there are several ways to recover. If the crash is relatively minor, you can press RUN/STOP RESTORE, keys together and FORTH will sign back on as if nothing has happened. There are however, some cases when this will not work. If this happens, turn off the power, and then turn it back on after a few seconds, and 64FORTH will restart. It is advisable to save any large program to disk or cassette tape prior to attempting to execute it. This may save you a lot of time later.

APPENDIX C TERM GLOSSARY

address	a number which identifies a location in a computer memory. Addresses usually expressed in hexadecimal.
binary	number base 2.
block	division of memory containing up to 1024 characters of source text.
branching	interrupting the normal flow of execution, depending on conditions in effect at the time of execution.
buffer	temporary storage area for data.
cfa	code field address. The address of a dictionary entry's code pointer field.
compile	generate a FORTH dictionary entry in computer memory from your source text. See execute.
constant	a value that has a name. Value is stored in memory and usually does not change.
decimal	number base 10.
dictionary	list of words and definitions, with both pre-defined (system) words and user-defined words.
double length numbers	integers within the range of −2 billion to +2 billion.
execute	to perform the operations specified in a definition of a word once compiled.
fetch	to retrieve a value from a given memory location.
flag	value stored in memory which serves as a signal as to whether a condition is true or false.
hexadecimal	number base 16.
input stream	the stream of data or text to be read by the interpreter.
literal	a number that appears inside a definition.
nesting	placing a branching structure within an outer branching structure.
octal	number base 8.
pad	region of memory used as a "scratch pad" area to hold text for immediate processing.

parameter field	an area of the dictionary entry containing the contents of the definition. For example, for a colon definition, the list of addresses of words to be executed in turn when the definition is executed.
parameter stack	region of memory which serves a common ground between various operations to pass arguments (numbers, flags) from one operation to another.
pfa	parameter field address. The address of the first cell in a dictionary entry's parameter field.
pointer	a location in memory where a number can be stored as a reference to something else.
post-fix notation	method of writing mathematical formulas with the operands first and the operator last. e.g. 3 2 + for 3 + 2. Also known as reverse Polish notation.
return stack	region of memory distinct from the parameter stack used to hold return addresses.
run-time code	a routine, compiled in memory which specifies what happens when a member of a given class of words is executed.
single length numbers	integers within the range of −32768 to +32767.
source text	written out from a definition or definitions as opposed to the compiled form entered into the dictionary.
stack	a region in memory controlled by storing or removing manipulated data in a last in first out (LIFO) operation. See parameter stack and return stack.
two's complement	for any number, the number of equal absolute value but opposite sign. For example, the two's complement of 5 is −5.
unsigned number	a number assumed to be positive.
user variable	one of a set of variables provided by 64FORTH, whose values are unique for each task.

variable	a location in memory which has a name and in which values are frequently stored and fetched.
vectored execution	the method of specifying code to be executed by providing not the address of the code itself but the address of a location which contains the address of the code. This location is often called the "vector." The vector can be reset to point to some other piece of code.
virtual memory	the treatment of mass storage (such as the disk) as though it were resident memory.
word	a defined dictionary entry.

64FORTH GLOSSARY

This glossary contains all the word definitions used in 64FORTH including the FORTH, SYSTEM, EDITOR, and ASSEMBLER vocabularies. The definitions are listed in order of their ASCII sort.

Each word is followed by its vocabulary designation and the format, or stack picture. The three dashes (- - -) indicate the WORD, or execution point. Any values left on the stack or "returned" are listed to the right of the stack picture.

The symbols include:

a, a1, a2...	16-bit address
n, n1, n2...	16-bit signed number
u, u1, u2...	16-bit unsigned number
d, d1, d2...	32-bit signed double length number
b, b1, b2...	8-bit byte value
c, c1, c2...	7-bit ASCII character
f, f1, f2...	boolean flag
tf	true flag – non-zero
ff	false flag – zero
t, t1, t2...	ASCII text string
<text>	a string of text follows
- - -	symbolic for the current word
/	separates stack values (used only in stack picture; do not use in actual definition)

Word	Vocabulary / Format / Definition
!	FORTH (n1/a1 - - -) Stores the value of n (16 bits) at address a1. Pronounced "store."
!CSP	FORTH Saves the stack position in CSP. Used as a part of the compiler security.
!CUR	EDITOR (n1 - - -) Moves edit cursor to character position n1 in current edit screen if n1 is in the range 0 to B/BUF-1. !CUR will only change the current cursor position if n1 is within the valid range. Causes the current edit window around the cursor to be viewed in the upper 16 lines of the display.
#	FORTH (d1 - - - d2) Generates from a double number d1, the next ASCII character placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between <# and #>. See #S.

#	ASSEMBLER (---) Specify 'immediate' addressing mode for the next op-code generated.
#LAG	EDITOR (--- a1/n1) An editor primitive, which returns the address of the cursor as a1, and the number of characters remaining on the line following the cursor as n1.
#LEAD	EDITOR (--- a1/n1) An editor primitive, which returns the address of the cursor line as a1, and the number of characters before the cursor on the line as n1.
#LOCATE	EDITOR (--- n1/n2) An editor primitive which returns the cursor line as n1 and the character position on the line as n2.
#S	FORTH (--- d1/d2) Generates ASCII text in the text output buffer, by the use of #, until a zero double number n2 results. Used between <# and #>.
#>	FORTH (d --- a count) Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.
'	FORTH (--- a1) Used in the form: ' nnnn Leaves the parameter field address of dictionary word nnnn. As a compiler directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "tick."
'CONT	SYSTEM (--- a1) Returns the address of the execution variable used to control the CONT action. The word CONT gets vectored to an error message if you attempt to TRACE a code word. Normally, 'CONT contains the CFA of (CONT).
'STEP	FORTH (--- a1) Returns the address of the execution variable used to control the STEP action. The word STEP gets vectored to an error message if you attempt to TRACE a code word. Normally, 'STEP contains the CFA of (STEP).
(	FORTH Used in the form: (cccc) Encloses explanatory comments which will be ignored by the computer. Delimited by a right parenthesis on the

	same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required. Pronounced "left paren."
(+LOOP)	FORTH (n - - -) The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.
(.")	FORTH The run-time procedure, compiled by ." which transmits the following on-line text to the screen or other output device. See ."
(;CODE)	FORTH The run-time procedure, compiled by ;CODE that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.
(CONT)	SYSTEM (- - -) Performs a continuous emulation through the currently selected TRACE word. This is the low level operation to which CONT gets vectored when a high level FORTH word is TRACED. This word is not normally used by the user at the keyboard.
(DO)	FORTH The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO.
(F)	EDITOR (- - -) This is a low level editor primitive, which searches for the next occurence of the text in the PADF buffer. Not normally used by the user, but could be useful if the editor was being expanded with additional commands.
(FCLOSE)	FORTH (- - -) This is a low level word used by FCLOSE, which closes the currently open disk file without emptying the memory buffer contents. Useful mostly for transferring screens from cassette to disk, and when copying screens from one file to another with the utility FCOPY.
(FILE)	FORTH (- - - t) This is a low level word used by FILE, which opens the file specified by text 't'. (FILE) does not empty the screens out of memory before opening the new file. This word is used mostly when copying screens from cassette to disk, or when copying screens from one disk file to another.
(FIND)	FORTH (a1 a2 - - - pfa b tf) (ok) (a1 a2 - - - ff) (bad)

Searches the dictionary starting at the name field address a2, matching to the text at address a1. Returns the parameter field address, length in bytes of name field, and boolean true for a good match. If no match is found, only a boolean false is left.

(I)	EDITOR (---) A low level editor primitive, which inserts the character string held in PADI, into the next text screen at the current location specified by the contents of R#. Potentially useful for future expansion of the editor.
(LINE)	FORTH (n1 n2 --- a count) Converts the line number n1 and the screen n2 to the disk buffers address containing the data. A count of 64 indicates the full line text length.
(LOOP)	FORTH The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.
(MATCH)	FORTH (a1/a2/n1 --- f1) An assembly language MATCH primitive. Given the addresses of strings of characters (a1 and a2), and a match length of characters (n1), returns boolean f1 as true if the strings are the same character sequence. Otherwise, returns a boolean false.
(NUMBER)	FORTH (d1 a1 --- d2 a2) Converts the ASCII text beginning at a1+1 using the current numeric BASE. The new value is accumulated into double number d1, being left as d2. a2 is the address of the first unconvertible digit. Used by NUMBER.
(R)	EDITOR (---) A lower level editor word, which replaces the contents of the current cursor line with the current contents of the insert buffer PADI.
(STEP)	SYSTEM (---) Performs a single step through the currently selected TRACE word. This is the function to which STEP is vectored, when a high level word is being traced. Not normally used by the user.
(↑FILE)	FORTH (a1/n1 ---) Similar to (FILE), but requires an address a1 of the text string for the file to open, and n1 the length of the name. The file is opened, but memory is not cleared before opening the file. See also ↑FILE (FILE) FILE.

)	ASSEMBLER (---)	Specify 'indirect' addressing for the next op-code generated. Used only on the JMP instructions.
)Y	ASSEMBLER (---)	Specify 'indirect' indexed Y' addressing mode for the next op-code generated.
,	FORTH (n ---)	Stores n into the next available dictionary memory cell, advancing the dictionary pointer. Pronounced "comma."
,X	ASSEMBLER (---)	Specify 'indexed' X addressing mode for the next op-code generated.
,Y	ASSEMBLER (---)	Specify 'indexed Y' addressing mode for the next op-code generated.
.	FORTH (n ---)	Prints a number from a signed 16-bit two's complement value, converted according to the numeric BASE. Followed by one blank space. Pronounced "dot." This is a vectored word.
."	FORTH	Used in the form: ." cccc" Compiles an in-line string cccc (delimited by the trailing quotation mark) with an execution procedure to send the text to the selected output device. If executed outside a definition, ." will immediately print the text until the closing ". The maximum number of characters is 64. See (.") Pronounced "dot quote."
.LINE	FORTH (n1/n2 ---)	Prints on the terminal device, a line of text from the disk or cassette by its line and screen numbers. Trailing blanks are suppressed.
.R	FORTH (n1 n2 ---)	Prints the number n1 right aligned in a field whose width is n2. No following blank is printed.
.S	FORTH (---)	Prints the current contents of the data stack on the screen in the current base, without destroying its contents.
*	FORTH (n1 n2 --- prod)	Leaves the signed product of two signed numbers on the stack. Use . (dot) to print the result.
*/	FORTH (n1 n2 n3 --- n4)	Leaves the ratio n4=n1*n2/n3 where all are assigned

	numbers. Retention of an intermediate 31-bit product permits greater accuracy than would be available with the sequence: n1 n2 * n3/
*/MOD	FORTH (n1 n2 n3 --- n4 n5) Leaves the quotient n5 and remainder n4 of the operation n1*n2/n3 on the stack. A 31-bit intermediate product is used as for */.
+	FORTH (n1 n2 --- sum) Leaves the sum of n1 + n2 on the stack.
+!	FORTH (n a ---) Adds n to the value at the address, a. Pronounced "plus-store."
+–	FORTH (n1 n2 --- n3) Applies the sign of n2 to n1, which is left as n3.
+LOOP	FORTH (n1 ---) (run-time) (a n2 ---) (compile-time) Used in a colon definition in the form: DO...n1+LOOP. At run-time +LOOP selectively controls branching back to the corresponding DO based on n1, the loop index and the loop limit. The signed increment n1 is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or greater than the limit (n1=0), or until the new index is equal to or less than the limit (n1=0). Upon exiting the loop, the parameters are discarded and the execution continues ahead.
+ORIGIN	FORTH (n --- a) Leaves the memory address relative to n to the origin parameter area. N is the minimum address unit, either byte or word. This definition is used to have access to or modify the boot-up parameters at the origin area.
+SCR	EDITOR (n1 ---) Adds the signed value n1 to the current editor screen number, and clips the result to within the currently specified valid screen range, 1 to BMAX.
–	FORTH (n1 n2 --- diff) Leaves the difference of n1 minus n2 on the stack.
-->	FORTH Continues interpretation with the next disk screen. Pronounced "next-screen."

–DUP	FORTH (n1 – – – n1) (if zero) (n1 – – – n1 n1) (non-zero) Reproduces n1 only if it is non-zero. Usually this is used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.
–FIND	FORTH (– – – pfa b tf) (found) (– – – ff) (not found) Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then the CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length in bytes, and a boolean true is left. Otherwise, only a boolean false is left. This is a vectored word.
–MOVE	EDITOR (a1/n1 – – –) Moves c/l characters of text from address a1 to line n1 of current edit screen. The screen is updated.
–SPOBJ	SYSTEM (n1 – – – n1/a1) Converts the sprite number n1 to the address a1 of the data definition area for that sprite. Also leaves a copy of the sprite number on the stack as n1. This is a low level sprite manipulation word, normally used only by the system.
–TRAILING	FORTH (a n1 – – – a n2) Suppresses the output of trailing blanks, from a text string that starts at address a, by adjusting the character count n1. I.e., the characters at +n1 to a +n2 are blanks.
/	FORTH (n1 n2 – – – quot) Leaves the signed quotient of n1/n2.
/CMD	FORTH (n1 – – – t) Sends the text string 't' to channel n1 on the serial bus. This is a lower level word than CMD, which always sends a command string to the disk drive.
/MOD	FORTH (n1 n2 – – – rem quot) Leaves the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.
0.0	SYSTEM (– – – d1) A space saving constant, which returns the double length (32-bit) value zero, used in several system calls.
0,1,2,3	FORTH (– – – n) These small numbers are used so often that it is recommended that you define them by name in the dictionary as constants.

0<	FORTH (n --- f) Leaves a true flag if the number is less than zero (negative); otherwise leaves a false flag.
0<	ASSEMBLER (--- n1) (assembling) Specify that the immediately following conditional will branch based on the processor status bit being negative (N=1), i.e., less than zero. The flag n1 is left at assembly time; there is no run-time effect on the stack.
0=	FORTH (n --- f) Leaves a true flag if the number is equal to zero; otherwise leaves a false flag.
0=	ASSEMBLER (--- n1) (assembling) Specify that the immediately following conditional will branch based on the processor status bit being equal to zero (Z=1). The flag n1 is left at assembly time; there is no run-time effect on the stack.
0BRANCH	FORTH (f ---) The run-time procedure to branch conditionally. If f is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL, and WHILE.
1+	FORTH (n1 --- n2) Increments n1 by 1.
2+	FORTH (n1 --- n2) Leaves n1 incremented by 2.
2DROP	FORTH (n1/n2 ---) or (d1 ---) Drops the top two single length elements (16-bits) from the data stack, or one double length (32-bits) element from the stack. Pronounced "two drop."
2DUP	FORTH (n1/n2 --- n1/n2/n1/n2) Duplicates the top two single length elements on the data stack. Pronounced "two dup."
2SWAP	FORTH (n1/n2/n3/n4 --- n3/n4/n1/n2) Exchanges the order of the top two pairs of elements on the data stack.
3DROP	FORTH (n1/n2/n3 ---) A code-saving word, which drops three arguments from the data stack. Used by several system call words.
:	FORTH Used in the form called a colon-definition `: cccc ... ;` Creates a dictionary entry defining cccc as equivalent to the following sequence of FORTH word definitions '...' until

the next ';' or 'CODE'. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled. The : must be followed by a space, and the ; must be preceeded by a space.

;	FORTH Terminates a colon-definition and stops further compilation. Compiles the run-time ;S. The ; must be preceeded by a space in a colon-definition.
;CODE	FORTH (---) Used to conclude a colon-definition in the form: : (name)... ;CODE (assembly code) END-CODE. Stops compilation and terminates a new defining word (name). Sets the CONTEXT vocabulary to ASSEMBLER. assembling to machine code the following mnemonics. An existing defining word must exist in (name) prior to ;CODE. When (name) later executes in the form: (name) (namex), the definition (namex) will be created with its execution procedure given by the machine code following (name). That is, when (namex) is executed, the address interpreter jumps to the code following ;CODE in (name).
;S	FORTH Stops interpretation of a screen. ;S is also the run-time word compiled at the end of a colon-definition which returns execution to the calling procedure.
<	FORTH (n1 n2 --- f) Leaves a true flag if n1 is less than n2; otherwise leaves a false flag.
<#	FORTH Begins the number conversion for formatting using the words: <# # #S #>. The conversion is done on a double number producing text at PAD.
<BUILDS	FORTH Used within a colon-definition: : cccc<BUILDS ... DOES>.. ; Each time cccc is executed, <BUILDS defines a new word with a high level execution procedure. Executing cccc in the form: cccc nnnn uses <BUILDS to create a dictionary entry for nnnn. When nnnn is later executed, it has the address of its parameter

area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time procedures to be written in high level rather than in assembler code (as required by ;CODE).

<↑>	FORTH (--- a1/n1) A low level primitive compiled in-line when a string is built in-line in a colon-definition. This word returns the address of the string and the count of its length on the data stack. Not used from the keyboard.
=	FORTH (n1 n2 --- f) Leaves a true flag if n1=n2; otherwise leaves a false flag.
>	FORTH (n1 n2 --- f) Leaves a true flag if n1 is greater than n2; otherwise leaves a false flag.
>=	ASSEMBLER (--- n1) (assembling) Specify that the immediately following conditional will branch based on the processor status bit carry being set (see =1). The flag n1 is left at assembly time. There is no run-time effect on the stack.
>BUF	SYSTEM (a1/n1 ---) This is one of the buffer deblocking primitives. Its function is to send the 1k block of data at address a1, to the screen buffer specified by n1, in upper memory. A modulus operation is performed on the values n1, to determine into which buffer the data is to be placed. There are currently 16 virtual buffers in high memory, and a modulus 16 operations is performed to determine the buffer location for use. If the buffer already contains data from a different block (as in block 1 and block 17, which both modulus to screen buffer 1), then the contents of the old screen 1 in this case must be written out to disk before the new screen, 17, can be moved to the buffer. This operation is done by GBLOCK, and must be performed before >BUF is executed. See also BUF> GBLOCK.
>R	FORTH (n ---) Removes a number from the computation stack and places as the most accessible on the return stack. Use should be balanced with R> in the same definition.
>RAM	SYSTEM (---) This is a system primitive, which uncovers the ram under the KERNEL and I/O from $D000 to $FFFF. Virtual buffer operations can then be performed to this area. This word is only used by the system, and MUST NOT be used from the keyboard, since NO I/O can be performed while the

	KERNEL is switched out. The logical reverse of this word is RAM>, which restores the KERNEL and I/O to the memory map.
?	FORTH (a - - -) Prints the value contained at the address in free format according to the current base.
?COMP	FORTH Issues an error message if not compiling.
?CSP	FORTH Issues an error message if stack position differs from value saved in CSP.
?D	FORTH (- - -) This is a short word used to determine status of the disk, and to clear any error condition that might occur while disk operations are performed.
?DERR	FORTH (- - -) This word is very similar to ?D, but only prints a message to the terminal if there was a disk error. If no error has occurred, then operation continues after ?DERR. If an error condition was detected, execution is aborted.
?ERROR	FORTH (f n - - -) Issues an error message number, if the boolean flag is true.
?EXEC	FORTH Issues an error message if not executing.
?LOADING	FORTH Issues an error message if not loading.
?STACK	FORTH Issues an error message if the stack is out of bounds.
?TERMINAL	FORTH (- - - n1) A modified version of the fig-FORTH word of the same name. If a key is pressed, the value of the key pressed n1 is returned. If no key was pressed, zero is returned. This is a vectored word.
@	FORTH (a - - - n) Leaves the 16-bit contents of the address, a. Pronounced "fetch."
ABORT	FORTH Clears the stacks and enters the execution state. Returns control to the operator's terminal, printing a message appropriate to the installation.

ABS	FORTH (n --- u) Leaves the absolute value of n as u.
ACPTR	SYSTEM (--- n1) A KERNEL system call, which accepts one character from serial bus.
AGAIN	FORTH (a n ---) (compiling) Used in a colon-definition in the form: BEGIN . . . AGAIN At run-time, AGAIN forces execution to return to the corresponding BEGIN. There is no effect on the stack. Execution can be terminated by pressing the RUN/STOP and RESTORE keys simultaneously.
ALLOT	FORTH (n ---) Adds the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-origin memory. N is with regard to computer address type (byte or word).
AND	FORTH (n1 n2 --- n3) Leaves the bitwise logical AND of n1 and n2 as n3.
ANDMASK	SYSTEM (n1/a1 ---) This is a SPRITE system primitive, which takes a bit number n1 in the range 0 to 7, and clears that bit in the byte contained at address a1. Example "4 500 ANDMASK" resets sprite register bit 4 and the value in address 500 with 11101111 binary, and stores the value back in location 500. This word is useful for manipulating the various sprite register bits.
ASCII	FORTH (--- t n1) Accepts the word following ASCII, and returns the ASCII value of the first character of the word on the data stack.
ASSEMBLER	FORTH Makes ASSEMBLER the context vocabulary. It will be searched first when the input stream is interpreted.
ATTACK@	FORTH (--- n1) Returns the attack register value for the currently selected voice, in the range from 0 to 15. See S!
ATTACK!	FORTH (n1 ---) Sets the attack register value for the currently selected voice, in the range 0 to 15. See S!
B	EDITOR (---) An editor operation, which causes the edit to be moved to the previous screen. Pronounced "back."

B/BUF	FORTH (--- n1) A constant which returns the number of bytes per FORTH buffer. A value of 1024 in this system.
B/SCR	FORTH (--- n1) A constant which returns the number of blocks per FORTH buffer. A value of 1 (one) in this system.
BACK	FORTH (a ---) Calculates the backward branch offset from HERE to address 2, and compiles into the next available dictionary memory address.
BASE	FORTH (--- a) A user variable containing the current number base used for input and output conversion.
BDG	SYSTEM (n1 ---) This is another system sprite primitive. It draws a horizontal line on the screen as a line specified by n1, 26 characters long, to make the top and bottom of the borders of the sprite editor. This is not a user word.

BEGIN	FORTH (--- a n) (compiling)
	Occurs in a colon-definition form:
	BEGIN...UNTIL
	BEGIN...AGAIN
	BEGIN...WHILE...REPEAT
	At run-time, BEGIN marks the start of a sequence that may be executed repeatedly. It serves as a return point from the corresponding UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs. At compile-time BEGIN leaves its return address and n for compiler error checking.

BEGIN,	ASSEMBLER (--- a1/n1) (assembling) (---) (run-time) Occurs in a code definition in the form: BEGIN,...n2 UNTIL, At run-time, BEGIN, leaves the dictionary pointer address a1 and the value n1−1 for later testing of conditional pairing by UNTIL,.
BGROUND	FORTH (n1 ---) Selects background color for the screen, using the standard values for Commodore colors in the range 0 to 15.
BL	FORTH (--- c) A constant that leaves the ASCII value for "blank."

BLANKS	FORTH (a count ---) Fills an area of memory beginning at address a with blanks.
BLK	FORTH (--- a) A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.
BLKREAD	SYSTEM (a1/n1 ---) The disk storage primitive which reads block n1 into address a1 from the currently open RELATIVE file. A disk error will be reported when reading or writing the first time to a newly created screen in a disk file. This error should be ignored, as it is a pecularity of the Commodore Disk Operating System.
BLKTBL	SYSTEM (--- a1) This is a system array variable, which points to the beginning of the table indicating the screens that are currently in the memory buffers. This array is 32 decimal bytes long.
BLKWRITE	SYSTEM (a1/n1 ---) This is a disk block primitive, which writes block n1 to disk from address a1, to the currently open RELATIVE disk file. A disk error will be reported the first time a write is performed to a previously non-existent screen. However, this error should be ignored if it is a non-existent record error.
BLOCK	FORTH (n1 --- a1) This word is a FORTH mass storage primitive, which returns the address a1 of screen n1. If a disk file is open, the screen will be read into memory if required, before returning the address. The range of screens accessible is limited by the range 1 through the value specified in variable BMAX. BMAX is initialized to 16. As a result, no disk file can be larger than 16k, unless you change the value in BMAX to some number greater than 16. Only 16 continuous screens can be held in memory at one time, so additional time overhead will occur if you frequently access screens which are exactly 16 apart, that is, screens 1 and 17, or 2 and 18, etc. Once a screen is read in from mass storage, it remains in memory until it is over-written by a screen 16 or higher or 16 lower than it, or en EMPTY-BUFFERS is performed. When no disk file is open, the 16 screen buffers perform virtual mass storage operations, giving the illusion of a very fast disk with 16k of storage.

BMAX	FORTH (--- a1) The FORTH user variable which specifies the highest screen that will be accessible with BLOCK operations. This variable is initialized to 16, equal to the number of screen buffers available.
BMOV	EDITOR (a1 ---) An editor primitive, (that moves the contents of PAD for C/L characters to address a1) if PAD contains text other than a null in the first character position (pad+1), otherwise, nothing is done.
BORDER	FORTH (n1 ---) Selects the screen BORDER color. N1 is a value between 0 and 15, for the colors as shown on page 139 of the Commodore 64 Users Guide.
BOT	ASSEMBLER (--- n1) Used during code assembly in the form: BOT LDA, or BOT X) STA, Addresses the bottom of the data stack (containing the low byte) by selecting the ,X mode and leaving n1=0, at assembly time. This value of n1 may be modified to another byte offset into the data stack. Must be followed by a multi-mode op-code mnemonic.
BRANCH	FORTH The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer to IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.
BUF#	SYSTEM (--- a1) A system variable not currently used.
BUF>	SYSTEM (a1/n1 ---) Moves the data for screen n1 from the virtual screen buffer to address a1, used by GBLOCK to perform SCREEN DEBLOCKING.
BUFADD	SYSTEM (n1 --- a1) This word calculates the address of the buffer which should hold the screen specified as n1, and returns the buffer address as a1.
C	EDITOR (n1 ---) An editor primitive, used to move the cursor a RELATIVE amount within the current edit screen. The signed value n1 is added to the current cursor location, and if the value is within the current screen, the cursor location is updated to the new value. Otherwise the cursor location is unchanged.

C@	FORTH (a --- b) Leaves the 8-bit contents of memory address. On word addressing computers, further specification is needed regarding byte addressing.
C/L	FORTH Characters per line.
C!	FORTH (b a ---) Stores 8 bits at address specified. On word addressing computers, further specification is necessary regarding byte addressing.
C,	FORTH (b ---) Stores 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This is only available on byte addressing computers.
CASSETTE	SYSTEM (---) Selects the cassette for all further mass storage operations. This is the default mode, so it is not normally necessary to switch to CASSETTE. When a disk file has been opened and then closed, operation is automatically switched back to cassette.
CFA	FORTH (pfa --- cfa) Converts the parameter field address of a definition to its code field address.
CHRIN	SYSTEM (--- n1) A KERNEL system call, which obtains a line of characters from the currently open character input channel, normally the keyboard. This routine waits for a line of input to be entered terminated by a carriage return; after a line of text has been entered, each call to CHRIN will return one character of the line entered. Note: CHRIN allows the use of the Commodore screen editor.
CHKIN	FORTH (n1 ---) A KERNEL system call, which redirects character input to the channel specified as n1. The channel n1 must already be open.
CHKOUT	SYSTEM (n1 ---) A KERNEL system call, which redirects character output to the channel specified as n1. The channel n1 must already be opened.
CIOUT	SYSTEM (n1 ---) A KERNEL system call, which sends a character n1 out of the serial bus.

CLALL	SYSTEM (- - -) A SYSTEM primitive, used to clear all channels, in the Commodore KERNEL to their default values. See the Commodore Programmers Reference Manual.
CLOSE	SYSTEM (n1 - - -) A KERNEL system call, which closes the channel n1 specified to the Commodore operating system.
CMD	FORTH (- - - t) A utility word, used to send command lines to the disk. The sequence " CMD I (return) " would cause the disk to be initialized. This word can be used in a definition, or on a line with multiple commands. But when this is done, the command string must be terminated with an up arrow " ↑ " as shown in the following example: : INIT CMD ↑ ; This definition initializes the current disk drive, and must not be used while a disk file is open.
CMOVE	FORTH (from to count - - -) Moves the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding toward high memory. Further specification is necessary on word processing computers.
CND	SYSTEM (a1 - - - a2) This is an emulator primitive, which converts the address of a selected group of emulated words from their real code field address of the emulation word that is their equivalent. Not a user word.
CODE	FORTH (- - - t) A defining word used in the form: CODE (name) ...END-CODE To create a dictionary entry for (name) in the CURRENT vocabulary. Name's code field contains the address of the parameter field. When (name) is later executed, the machine code in this parameter field will execute. The CONTEXT vocabulary is made ASSEMBLER, to make available op-code mnemonics.
COLD	FORTH (- - -) The COLD start procedure adjusts the dictionary pointer to the minimum standard and restarts via ABORT. May be called from the terminal to remove application programs and restart. See EMPTY.
COLOR	FORTH (n1/n2 - - -) Sets the color n1 for sprite number n2. The color numbers are the same as shown on page 139 of the Commodore 64 Users Manual.

COMBUF	SYSTEM (--- a1) This is the buffer used to send and receive screen data to and from the disk drive. This buffer is 1024 bytes long, and is in NON-PAGED memory.
COMPILE	FORTH (---) When the word containing COMPILE executes the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter has already done).
CONSTANT	FORTH (n ---) A defining word used in the form: n CONSTANT cccc to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.
CONT	FORTH (---) This is a vectored word which causes all of a definition to be traced after a trace has been interpreted, or an EMULATE command has been issued. As the trace scrolls up the screen, it can be interrupted at any time, by pressing any character key. To continue after interruption, simply enter CONT followed by a carriage return.
CONTEXT	FORTH (--- a) A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.
COPY	EDITOR (n1/n2 ---) Copies screen n1 to screen n2, thereby creating a duplicate screen.
COUNT	FORTH (a1 --- a2 n) Leaves the byte address a2 and byte count n of a message text beginning at address a1. It is presumed that the first byte at a1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.
CPU	ASSEMBLER (n1 ---) (compiling assembler) An assembler defining word used to create assembler mnemonics that have only one addressing mode: EA CPU NOP, CPU creates the word NOP, with its op-code EA as a parameter. When NOP is later executed, it assembles EA as a one byte op-code.

CR	FORTH Transmits a carriage return and line feed to the selected output device. This is a vectored word.
CREATE	FORTH A defining word used in the form: CREATE cccc by such words as CODE and CONSTANT to create a dictionary header for a FORTH definition. The code field contains the address of the word's parameter field. The new word is created in the CURRENT vocabulary. This is a vectored word.
CS	ASSEMBLER (--- n1) (assembling) Specify that the immediately following conditional will branch based on if the processor carry is set (C=1). The flag n1 is left at assembly time; there is no run-time effect on the stack.
CSP	FORTH (--- a) A user variable temporarily storing the stack pointer position for compilation error checking.
CTBL	FORTH (--- a1) 64FORTH supports CAPTURED function keys, through the use of a capture table. This is a SYSTEM variable, which returns the address a1 of the current capture table. The table consists of a set of entries, each consisting of one byte, the value of the character to be captured, and a two byte address of the function to be performed when the character is entered. Here is an example entry: 0 VARIABLE TESTTABLE -2 ALLOT 3, C, ' . S CFA , 0 C, The table just created, contains the value 3, (CTRL C), and the CFA of the ".S" definition. The table is terminated with a zero byte following the last entry. Now whenever a CTRL C is pressed, the contents of the data stack will be printed out. NOTE: The key capture functions of 64FORTH will only work when the value in SYSTEM variable EFLAG is one (1). The Commodore line input routine is used when EFLAG is zero (0) and all function control keys are filtered out. You must use the fig-FORTH input routine when using key capture with EFLAG set to one.
CUR	SYSTEM (n1/n2 ---) This is another sprite system primitive, but one which might be useful to some people. It allows you to specify the location on the screen where the cursor will be

	positioned, by performing the system call which positions the cursor. The parameters are n1=the X position, and n2=the Y position for the new cursor location.
CURBLK	SYSTEM (--- n1) Returns the non-updated screen number of the screen most recently addressed with BLOCK. Used by the virtual disk interface.
CURRENT	FORTH (--- a1) Leaves the address of the variable that specifies the vocabulary into which you are adding new word definitions.
CURVOICE	FORTH (--- a1) A user variable, which holds the number of the current voice on which the sound control words are working. This is used to simplify and reduce the stack operations that have to be done when working with the complex sound system in the SID chip.
CV	SYSTEM (--- a1) Returns the base address of the currently selected voice of the sound system.
CXY	FORTH (n1/n2/n3 ---) This is the sprite position control word, which allows any sprite n3 to be placed at any address on the screen at n1=x and n2=y. This slower version of XY allows full screen placement of a sprite. See XY.
D	EDITOR (--- (text)) Deletes the first occurence of (text) following the D command and then searches from the edit cursor until the end of the screen. (text) is optional, and if omitted, will cause D to delete the same string as used previously.
D#	FORTH (--- a1) A user variable, which holds the device number used for the READ and WRITE operators. Defaults to one (1), CASSETTE.
D+	FORTH (d1 d2 --- sum) Leaves the double number sum of two double numbers.
D+-	FORTH (d1 n --- d2) Applies the sign of n to the double number d1, leaving it as d2.
D.	FORTH (d ---) Prints a signed double number from a 32-bit two's complement value. The high-order 16 bits are most accessible on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced "D dot."

D.R	FORTH (d n ---) Prints a signed double number d right aligned in a field n characters wide.
DABS	FORTH (d --- ud) Leaves the absolute value ud of a double number.
DECAY@	FORTH (--- n1) Return the decay register value for the currently selected voice, in the range 0 to 15. See S!
DECAY!	FORTH (n1 ---) Set the decay register value for the currently selected voice in the SID chip to value n1, in the range 0 to 15. See S!
DECIMAL	FORTH (---) Set the numeric conversion BASE for decimal input-output.
DEFINITIONS	FORTH (---) Used in the form: cccc DEFINITIONS Sets the current vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.
DEL	EDITOR (n1 ---) An editor primitive, which deletes n1 characters before the cursor; clips at the beginning of the screen, to prevent deleting off the screen. Text to the right of the cursor on the current line will be moved left to fill the space of the deleted characters.
DEPTH	FORTH (--- n1) Returns the current depth of the data stack as value n1. The actual depth of the stack is increased by one element.
DIGIT	FORTH (c n1 --- n2 tf) (c n1 --- ff) Converts the ASCII character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.
DISK	SYSTEM (---) This word selects the disk device for further mass storage operations by setting variable D# to the device specified by variable DR#, which is the actual drive number plus 8 (eight). This word is performed automatically by "FILE" operations.

DLITERAL	FORTH (d --- d) (executing) (d --- d) (compiling) If compiling, compiles a stack double number into a literal. Later execution of this definition containing the literal will push it to the stack. If executing, the number remains on the stack.
DLOAD	FORTH (--- t) Allows you to load your pre-compiled FORTH dictionary from cassette or disk. Used as follows: DLOAD <t> <RETURN> will load a pre-compiled dictionary from the cassette file named <t>. To load from disk, preceed the word DLOAD with the word DISK as follows: DISK DLOAD <t> <RETURN> Any disk file currently open should be closed before performing DLOAD.
DLSN	SYSTEM (n1 ---) Sends a secondary address to the disk, after telling the disk to LISTEN. Pronounced "disklisten."
DMINUS	FORTH (d1 --- d2) Converts d1 to its double number two's complement.
DO	FORTH (n1 --- n2) (execute) (a n ---) (compile) Occurs in a colon-definition in form: DO...LOOP DO...+LOOP At run-time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value of n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop "I" will copy the current value of the index to the stack. See I, LOOP, +LOOP, LEAVE. When compiling within the colon-definition, DO compiles (DO), leaves the following address a and n for later error checking.
DOES>	FORTH A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part

executes, it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the FORTH assembler, multi-dimensional arrays, and compiler generation.

DP	FORTH (--- a) A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.
DPL	FORTH (--- a) A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used to hold output column location of a decimal point, in user generated formatting. The default value on single number input is −1.
DR#	FORTH (--- a1) A user variable which holds the device number of the currently active disk drive. Defaults to drive 0, device 8, and may be changed with the word DRIVE, and checked with the word DRIVE?.
DRIVE?	FORTH (--- n1) Returns the drive number of the currently active disk drive: defaults to device 8 drive 0. See also DRIVE and DR#.
DRIVE	FORTH (n1 ---) Selects drive n1 as the currently active disk drive, by setting user variable DR# to the drive number plus 8. See also DRIVE? and DR#.
DROP	FORTH (n ---) Drops the number from the stack.
DSAVE	FORTH (--- t) Saves your compiled dictionary to cassette or disk. Used as follows: `DSAVE <t>    <RETURN>` will save a pre-compiled dictionary from the cassette file named <t>. To save from disk, preceed the word DSAVE with the word DISK as follows: `DISK DSAVE <t>    <RETURN>` Any disk file currently open should be closed before performing DSAVE.
DTLK	SYSTEM (n1 ---) Sends a secondary address n1 after telling the disk to talk. Pronounced "disk talk."

DTYPE	SYSTEM (n1/a1/n2 ---) Sends the character string at address a1, for length n2 to channel n1 in the serial bus. Pronounced "disk type."
DUMP	FORTH (a1/n1 ---) The contents of memory starting at address a1 for a length of n1 bytes is displayed on the screen, in the current number base, with 8 bytes per line, preceeded by the address of the first byte in the line.
DUP	FORTH (n --- n n) Duplicates the value on the stack.
E	EDITOR (---) An editor word, used after an "F" command, to ERASE the text just found. Deletes characters for the length of the text in the find buffer.
E>R	SYSTEM One of a group of EMULATION words, used by the DEBUGGER. Not a user word.
EDIT	FORTH (n1 ---) Enters the split screen editor, on screen n1. The EDITOR is selected, and the editor capture table is enabled. See the EDITOR documentation, for further information on editor commands.
EFLAG	FORTH (--- a1) A user variable, which specifies if the edit mode is selected. If this variable is not zero, EXPECT obtains characters from the KERNEL one character at a time, and no Commodore screen editor functions are available. If the variable contains zero (0), characters are obtained from the KERNEL with the line read routine, and all Commodore screen edit functions are operational. This is the normal mode when the editor is not selected.
EI	SYSTEM One of a group of EMULATION words, used by the DEBUGGER. Not a user word.
EIP	SYSTEM One of a group of EMULATION words, used by the DEBUGGER. Not a user word.
EIP+	SYSTEM One of a group of EMULATION words, used by the DEBUGGER. Not a user word.
ELSE	FORTH (a1 n1 --- a2 n2) (compiling) Occurs within a colon-definition in the form: IF...ELSE...ENDIF

At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after ENDIF. It has no stack effect. At compile-time, ELSE emplaces BRANCH reserving a branch offset and leaves the address a2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from a1 to HERE and storing at a1.

ELSE,	ASSEMBLER (--- a1/n1) (assembling) (---) (run-time) Occurs within a code definition in the form: n2 IF, (true part) ELSE, (false part) ENDIF, At run-time, if the condition code specified by n2 is false, execution skips to the machine code following ELSE,. At assembly time ELSE, assembles a forward jump to just after ENDIF, and resolves a pending forward branch from IF,. The values n1=n2 are used for error checking of conditional pairing.
EM	FORTH (--- a1) A SYSTEM variable, which holds an address one byte higher than the highest user address available to the FORTH dictionary. Changing this address will cause both FORTH and the Commodore KERNEL to think the memory size has changed. The contents of EM must not be increased, however, as 64FORTH uses all higher memory locations.
EMIT	FORTH (c1 ---) Sends the character c1 to the selected output device. OUT is incremented for each character output. The output device is determined by the most recent use of the SYSTEM call CHKOUT. Normally characters are sent to the video screen. This is a vectored word.
EMPTY	FORTH (---) Cleans out the dictionary to the COLD condition. Similar to FORGET, but forgets ALL user added words.
EMPTY-BUFFERS	FORTH Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to the disk. This is also an initialization procedure before first use of the disk.
EMULATE	FORTH (--- t) Selects the word following EMULATE as the word to be traced by the select debugger. Revectors 'CONT and 'STEP to (CONT) and (STEP) if the word is a high level word; otherwise vectors them to an error message routine.

ENCLOSE	FORTH (a c - - -) (a1 n1 n2 n3) The text scanning primitive used by WORD. From the text address a1 and an ASCII delimiting character c is determined the byte offset to the first non-delimiter character n1, the offset to the first character delimiter after the text n2, and the offset to the first character not included. This procedure will not proceed past an ASCII 'null', treating it as an unconditional delimiter.
END-CODE	ASSEMBLER An error check word marking the end of a CODE definition. Successful execution to and including END-CODE will unSMUDGE the most recent CURRENT vocabulary definition, making it available for execution. END-CODE also exits the ASSEMBLER making CONTEXT the same as CURRENT. This word previously was named C;.
ENDIF	FORTH (a n - - -) (compile) Occurs in a colon-definition in the form: IF...ELSE...ENDIF IF...ENDIF At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. See also IF and ELSE.
ENDIF,	ASSEMBLER (a1/n1 - - -) (assembly-time) (- - -) Occurs in a code definition in the form: n2 IF, (true part) ELSE, (false part) ENDIF, At run-time ENDIF, marks the conclusion of a conditional structure. Execution of either the true part or the false part resumes the following ENDIF,. When assembling, address a1 and n1=2 are used to resolve the pending forward branch to ENDIF,.
ENV3@	FORTH (- - - n1) Returns the value of the oscillator #3 ENVELOPE generator, in the range 0 to 255.
ER>	SYSTEM One of a group of EMULATION words, used by the DEBUGGER. Not a user word.
ERASE	FORTH (a n - - -) Clears a region of memory to zero from address 'a' over 'n' addresses.
ERP	SYSTEM One of a group of EMULATION words, used by the DEBUGGER. Not a user word.

ERP!	SYSTEM One of a group of EMULATION words, used by the DEBUGGER. Not a user word.
ERR	SYSTEM (---) This word prints the error message CODE DEFINITION, when TRACE or EMULATE are used on a CODE WORD. This prevents a crash, if an attempt was actually made to trace these words.
ERROR	FORTH (line --- in blk) Executes error notification and re-start of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING is −0, n is just printed as a message number (non disk installation). If WARNING is −1, the definition (ABORT) is executed which executes the system ABORT. 64FORTH saves the contents of IN and BLK to assist in determining the location of the error. Final action is execution of QUIT. This is a vectored word.
EXECUTE	FORTH (a count) Executes the definition whose field address is on the stack. The code field address is also called the compilation address.
EXPECT	FORTH (a1/n1 ---) Same as fig-FORTH, but uses the Commodore line read routine, which can only be terminated by a carriage return at the end of the line. If a fig standard version of EXPECT is required, this can be obtained by setting EFLAG to value of 1 (one) which causes alternate section of EXPECT code to be used.
F	EDITOR (--- (text)) An editor word which finds the first occurence of (text) following an "F" command. Starts at the current cursor location. (text) is optional, and "F (return)" will find the next occurence of the last text searched for.
F#	FORTH (--- a1) A user variable, which holds the current file number for all READ and WRITE operations. Defaults to one (1) for CASSETTE operation.
FCLOSE	FORTH (---) Closes the current disk file and clears all screen buffers to blanks. If you need to close a file without clearing the screen buffers, use (FCLOSE).

FENCE	FORTH (--- a) A user variable containing an address below which FORGETting is trapped. To FORGET below this point, the user must alter the contents of FENCE.
FILE	FORTH (--- t) Closes any file currently open, and opens the file specified by text 't' on the currently specified disk drive. Any updated screen in memory will be flushed to the previously open file before it is closed. A user variable called BMAX contains the highest numbered screen that can be accessed. If you wish to manipulate a disk file larger than 16k bytes, you will have to change the value of BMAX to a larger number. The largest single file that may be created is 90k bytes. This is due to a limitation of the Commodore DOS.
FILL	FORTH (a quan b ---) Fills memory at the address with the specified quantity of bytes b.
FILTER@	FORTH (--- n1) Returns the filter center/cutoff frequency. See S!
FILTER!	FORTH (n1 ---) Sets the filter center/cutoff frequency to the value specified as n1, in the range 0 to 2047. See S!
FIRST	FORTH (--- n) A constant that leaves the address of the first (lowest) block buffer.
FLOAD	FORTH (--- <t>) Opens file <t> on current disk drive and loads screen 1. Screen 1 is expected to load the rest of the file.
FLUSH	FORTH (---) This system uses the intelligent version of flush. After a screen is written to disk, it is left in memory also, to prevent a screen reread, although its UPDATE bit is cleared.
FMODE@	FORTH (--- n1) Returns the filter mode register value, indicating the type of filter currently selected. See FMODE! for further information about the value returned as n1. See S!
FMODE!	FORTH (n1 ---) Sets the filter mode register value to select the type of filtering desired. Each of the low 4 bits of the value n1 are a type of filter action as follows:

	low-pass=01, band-pass=02, high-pass=04 and voice #3 off=08 Any of the first three filter types can be selected, or ORed together, to form a more complex filter action. The last bit, VOICE #3 OFF, is used to remove voice #3 from the output without turning it off internally. This allows oscillator #3 to be used to modulate the other voices, without its output appearing directly in the output of the sound chip. See S!
FN	FORTH (--- a1) A user array, which holds the name of the current file. This array is 65 characters long.
FORGET	FORTH Executed in the form: FORGET cccc Deletes definition named cccc from the dictionary with all entries physically following it. In 64FORTH an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same. See FENCE.
FREQ@	FORTH (--- n1) Returns the frequency value from the currently selected voice. This is the compliment of FREQ! See S!
FREQ!	FORTH (n1 ---) Sets the frequency value for the currently selected voice. This is the compliment of FREQ@. n1 is in the range 0 to 65, 535. See S!
FSELECT@	FORTH (--- n1) Returns the filter select register, which holds the value n1 --- the voices specified to be filtered. The bit values for each voice are as follows: voice#1=01, voice#2=02, voice#3=04, external=08 These values can be ORed together to create n1. See S!.
FSELECT!	FORTH (n1 ---) Sets the filter select register, to cause the filter to be effective on the voices specified by the bits of value n1. The bit values for each voice are as follows: voice#1=01, voice#2=02, voice#3=04, external=08 Any or all of these values can be ORed together to create the value n1 passed to FSELECT! See S!.
FSET	SYSTEM (---) A low level file control word which performs a SETLFS system call with the current values of D# and F#. Used by READ and WRITE.

GATE0	FORTH (- - -) Disables the currently selected voice, that is, turns it OFF. See S!.
GATE1	FORTH (- - -) Enables the currently selected voice, that is, turns it ON. See S!
GBLOCK	SYSTEM (n1 - - -) This is the main disk interface word in this version of FORTH. It performs the screen buffer deblocking, and reads the requested screen n1 into the user disk buffer from disk of the virtual screen buffers in high memory. It also takes care of writing any needed screens to disk as required.
GTEXT	EDITOR (- - -) An editor primitive, accepts text from the input stream, until a delimiting "↑" is encountered, or the (RETURN) key is pressed. The text is placed at address a1, for a length of 65 characters.
H	EDITOR (- - -) An editor word, which causes a copy of the line currently containing the cursor to be moved to the insert buffer PADI. The text can later be reinserted with "U", or "I", or "P".
H.	FORTH (n1 - - -) Prints the value n1 to the current output device in the HEXADECIMAL number base and in an unsigned format.
HERE	FORTH (- - - a) Leaves the address of the next available dictionary location.
HEX	FORTH Sets the numeric conversion base to sixteen (hexadecimal).
HIDE	FORTH (n1 - - -) Causes the sprite specified by n1 to be hidden from the screen view. See also SHOW.
HIRES	FORTH (n1 - - -) Causes the sprite specified by n1 to be set in high resolution mode. This is the default mode, where each sprite has a foreground and a background color. See also MULTI, for use of the multi-color sprite mode. Pronounced "hi rez."
HLD	FORTH (- - - a) A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD	FORTH (c - - -) Used between <# and #> to insert an ASCII character into a pictured numeric output string. e.g. 2E HOLD will place a decimal point.
I	EDITOR (- - - (text)) An editor word, causes (text) to be inserted at the current location of the cursor, with all text following the cursor being moved over to make room. Text on the cursor line too long for the line after the insert will be LOST. (text) is optional, and use of "I" without following text, causes the current contents of PADI the insert buffer to be inserted.
I	FORTH (- - - n) Copies the index of the current DO . . . LOOP counter to the computation stack.
I/O	FORTH (- - - a1) A user array, containing the CFA's of the VECTORED words in 64FORTH. See the section on VECTORED WORDS, for further information.
ID.	FORTH (a - - -) Print a definition's name from its name field address.
IF	FORTH (f - - -) (run-time) (- - - a n) (compile) Occurs in a colon-definition form: IF (tp) . . . ENDIF IF (tp) . . . ELSE (fp) . . . ENDIF At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead through the true part. If f is false (zero), execution skips until just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional; if missing, false execution skips to after ENDIF.
IF,	ASSEMBLER (b1 - - - a1/n1) (assembly time) Occurs within a code definition, in the following form: n1 IF, (true part) ELSE, (false part) ENDIF, At run-time, IF branches based on the condition code n1<0<or 0= or CS or OVS). If the specified processor status is true, execution continues ahead. Otherwise branching occurs just after ELSE, or ENDIF, when ELSE, is not present. At ELSE, execution resumes at the corresponding ENDIF,. When assembling, IF, creates an unresolved forward branch based on the condition code n1, and leaves address a1 and n1=2 for resolution of the branch by the corresponding ELSE, or ENDIF,. Conditionals may be nested.
IMMEDIATE	FORTH Marks the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled, that is, the precedence bit in its

	header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an IMMEDIATE definition by preceeding it with [COMPILE].
IN	FORTH (--- a) A user variable containing the byte offset within the current input text buffer (terminal or disk) from which the next text will be accepted. WORD uses and moves the value of IN.
INDEX	ASSEMBLER (--- a1) (assembling) An array used within the assembler, which holds bit patterns of allowable addressing modes.
INTERPRET	FORTH The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disk) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT, it is converted to a number according to the current base. That also failing, an error message echoing the name with a "?" will be given. Text input will be taken according to the convention for the word. If a decimal point is found as a part of a number, a double number value will be left. The decimal point has no other purpose than to force this action. See NUMBER.
IP	ASSEMBLER (--- a1) (assembling) Used in a code definition in the form: IP STA, or IP) Y LDA, A constant which leaves the address of the pointer to the next FORTH execution address in a colon-definition to be interpreted at assembly time. At run-time, NEXT moves IP ahead within a colon-definition. Therefore, IP points just after the execution address being interpreted. If an in-line data structure has been compiled (i.e. a character string), indexing ahead by IP can access this data: IP STA, or IP) Y LDA, Loads the third byte ahead in the colon-definition being interpreted.
K	EDITOR (---) An editor word, which erases or "KILLS" the current cursor line. The line is filled with blanks, but no text compression occurs.
KEY	FORTH (--- n1) Returns value n1, the ASCII key value of the key pressed on the keyboard. Before returning with the key value, the

	key pressed is checked to see if it is in the key capture table. If it is, the associated function is executed. and another key is obtained from the keyboard. This is a vectored word.
L#	EDITOR (--- n1) An editor primitive, which returns the line number of the line which currently contains the cursor.
LATEST	FORTH (--- a) Leaves the name field address of the topmost word in the CURRENT vocabulary.
LEAVE	FORTH Forces termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until LOOP or +LOOP is encountered.
LFA	FORTH (pfa --- lfa) Converts the parameter field address of a dictionary definition to its link address.
LIMIT	FORTH (--- n) A constant leaving the address just above the highest memory available for a disk or cassette buffer. Usually this is the highest memory system.
LINE	FORTH (n1 --- a1) Returns address a1 of line n1 in current edit screen as specified by variable SCR.
LIST	FORTH (n ---) Displays the ASCII text of screen n on the selected output device. SCR contains the screen number during and after this process.
LISTEN	SYSTEM (n1 ---) Sends a command to cause device address n1 to listen for data on the serial bus.
LIT	FORTH (--- n) Within a colon-definition, LIT is automatically compiled before each 16 bit literal number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack.
LITERAL	FORTH (n ---) (compiling) At compile-time, the LITERAL compiles the stack value n as a 16-bit literal. This definition is immediate so that it will execute during a colon-definition. The intended use is: : xxx (calculate) LITERAL ;

	Compilation is suspended for the compile time calculation of a value. Compilation is resumed and LITERAL compiles this value. At run-time, the value will be pushed onto the stack.
LOAD	FORTH (n - - -) Begins interpretation of screen n. Loading will terminate at the end of screen or at ;S. See ;S and - ->
LOADS	FORTH (n1 - - -) A mass storage word, used to load from cassette a sequence of screens into screen butter one, and then a one (1) load is performed. This is mainly a hold over from VICFORTH, and was used to minimize the number of buffers required to load a program from cassette.
LOOP	FORTH (a n - - -) (compiling) Occurs in a colon-definition in form: `DO . . . LOOP` At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. A branch back to DO occurs when the index equals or exceeds the limit. At that time, the parameters are discarded and execution continues ahead. At compile time, LOOP compiles (LOOP) and uses address a to calculate an offset to DO. n is used for error testing.
M	EDITOR (n1/n2 - - -) An editor line MOVE word used to move the line currently containing the cursor to screen n1, under line n2.
M*	FORTH (n1 n2 - - - d) A mixed magnitude math operation which leaves the double number signed product of two signed numbers.
M/	FORTH (d n1 - - - n2 n3) A mixed magnitude math operator which leaves the signed remainder of n2 and signed quotient n3 from a double number dividend and divisor n1. The remainder takes its sign from the dividend.
M/CPU	ASSEMBLER (n1/n2 - - -) (compiling assembler) An assembler defining word used to create assembler mnemonics that have multiple address modes `HEX 1C6E 60 M/CPU ADC,` M/CPU creates the word ADC, with two parameters. When ADC later executes, it uses these parameters, along with stack values and the contents of MODE to calculate and assemble the correct op-code operands.

M/MOD	FORTH (ud1 u2 --- u3 ud4) An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend u1 and single divisor u2.
MASK	SYSTEM (--- a1) An array, which contains the values used when converting from bit number to the mask value for that bit, used by ORMASK and ANDMASK. This array contains 8 values ranging from 01 to 80, as follows: 01, 02, 04, 08, 10, 20, 40, 80.
MASK?	SYSTEM (n1/a1/f1 ---) This word allows you to specify a boolean flag to select which of the operations "ORMASK" and "ANDMASK" are done. If f1 is zero (0), an "ANDMASK" is performed. And if f1 is non-zero, an "ORMASK" is done. The additional parameters n1 and a1, are passed on to the appropriate function. Used by the XPAND word in determining the expansion of a particular sprite.
MAX	FORTH (n1 n2 --- max) Leaves the greatest of two numbers.
MCLR1	FORTH (n1 ---) Selects color n1 as the multi-color number 1.
MCLR2	FORTH (n1 ---) Selects color n1 as the multi-color number 2.
MEM	ASSEMBLER Used with the assembler to set MODE to the default value for direct memory addressing, z-page.
MESSAGE	FORTH (n1 ---) Prints the error message number n1 to the current output device. 64FORTH prints the error messages from ROM memory, and not from disk.
MIN	FORTH (n1 n2 --- min) Leaves the smaller of two numbers.
MINUS	FORTH (n1 --- n2) Leaves the two's complement of a number.
MOD	FORTH (n1 n2 --- mod) Leaves the remainder of n1/n2, with the same sign as n1.
MODE	ASSEMBLER (--- a1) A user variable used within the assembler, which holds a flag indicating the addressing mode of the op-code being generated.

MSREG	SYSTEM (--- a1) A system constant which returns the address of the memory area that is a copy of the SID chip sound register. This array is maintained to allow both reading and writing to the SID chip registers.
MULTI	FORTH (n1 ---) Selects multi-color mode for sprite n1. Don't forget to set MCLR1 and MCLR2 first.
N	ASSEMBLER (--- a1) (assembling) Used in a code-definition in the form: N1 – STA, or n2 =)Y ADC, A constant which leaves the address of a 9 byte workspace in z-page. Within a single code definition, free use may be made over the range N-1 through N-7.
NAME	FORTH (--- (text)) Accepts the string of text following, delimited by an up arrow " ↑ ", into the FN buffer, and performs a system call to the SETNAM KERNEL routine.
NEXT	ASSEMBLER (--- a1) (assembling) A constant which leaves the machine address of the FORTH address interpreter. All code definitions must return execution to NEXT, or code that returns to NEXT (that is, PUSH, PUT, POP, POPTWO).
NEXTMP	ASSEMBLER (--- a1) (assembling) An extra variable used in the 64FORTH kernel, which removes the need to prevent colon-definitions from landing on a page boundary. The 6502 has a nasty habit of crashing if you attempt to jump indirect through an address which is sitting on a page boundary. Most FORTH systems modify the compiler to test and assure that this condition cannot happen. 64FORTH has been modified to keep it from being sensitive to this occurence, by modifying the assembly code for next.
NEWSPRITE	FORTH (n1 ---) Enters the sprite editor, with sprite n1 cleared. You now use the cursor movement keys and the star (*) symbol to draw your sprite on the screen within the box shown. Pairs of symbols in the form "**", "_*", "*_", or "__", are used to select the specific color in multi-color mode.
NFA	FORTH (pfa --- nfa) Converts the parameter field address of a definition to its name field address.
NOISE	FORTH (---) Sets the currently selected voice to generate noise. This is

	one of four basic signals the SID chip can generate. See also TRIANGLE, SAWTOOTH, SQUARE. See S!
NOT	ASSEMBLER (n1 --- n2) (assembly-time) When assembling, reverse the condition code n1 for the following conditional, returning n2 the complemented condition code. For example: 0 NOT IF, (true part) ENDIF, will branch based on "not equal to zero."
NUMBER	FORTH (a --- d) Converts a character string left at address a with a preceeding count to a signed double number using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL, but no other effect occurs. If numeric conversion is not possible, an error message will be given. This is a vectored word.
OPEN	SYSTEM (---) A KERNEL system call, which opens a channel as previously specified with the SETNAM and SETLFS commands.
OR	FORTH (n1 n2 --- or) Leaves the bit-wise logical or of two 16-bit values.
ORMASK	SYSTEM (n1/a1 ---) Sets the bit specified by bit number n1 in the byte contained in address a1. Example: "4 500 ORMASK" OR the contents of address 500 with 00010000 binary, and replace the result in address 500. Used to manipulate the bits of the various sprite registers.
OSC3@	FORTH (--- n1) Returns the output of the waveform for oscillator #3 as a number n1 in the range 0 to 255.
OUT	FORTH (--- a) A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formatting.
OVER	FORTH (n1 n2 --- n1 n2 n1) Copies the second stack value, placing it as the new top.
OVS	ASSEMBLER (--- n1) (assembling) Specify that the immediately following conditional branch based on the processor overflow flag is set (V=1). The flag n1 is left at assembly time; there is no run-time effect on the stack.
PAD	FORTH (--- a) Leaves the address of the text output buffer, which is a fixed offset above HERE.

PADF	EDITOR (--- a1) An editor buffer, 65 characters long used to hold the strings of text searched for: the find buffer.
PADI	EDITOR (--- a1) An editor buffer 65 characters long used to hold the strings of text inserted: the insert buffer.
PFA	FORTH (nfa --- pfa) Converts the name field address of a compiled definition to its parameter field address.
PICK	FORTH (n1 --- n2) Picks element n1 from the data stack and leaves it as value n2. One (1) PICK is equivalent to DUP, and two (2) PICK is equivalent to OVER, etc.
POP	ASSEMBLER (--- a1) (assembling) (n1 ---) (run-time) A constant which leaves (during assembly) the machine address of the return point which, at run-time, will pop a 16 bit value from the data stack and continue interpretation.
POPTWO	ASSEMBLER (--- a1) (assembling) (n1/n2 ---) (run-time) A constant which leaves (during assembly) the machine address of the return point, which, at run-time, will pop two 16-bit values from the data stack and continue interpretation.
POS	SYSTEM (n1 ---) Sets the position of the relative file pointer to record n1. 64FORTH uses 129 byte records.
PRINT	FORTH (--- (text)) Sends all character output from the following command line as specified by (text) to the serial bus printer.
PROFF	FORTH (---) Restores character output to the video screen, and turns off the serial bus printer channel.
PRON	FORTH (---) Opens the serial bus printer channel, and directs all character output to that channel. If no printer is connected, the computer may HANG.
PTRADD	SYSTEM (n1 --- a1) A system primitive, which accepts the value n1, a screen and returns the address a1 which is the address inside of the BLKTBL array which specifies which screens are currently in memory.

PUSH	ASSEMBLER (--- a1) (assembling) (--- n1) (leaving) A constant which leaves (during assembly) the machine address of the return point which, at run-time, will add the accumulator (as high byte) and the bottom machine stack byte (as low byte) to the data stack.
PUT	ASSEMBLER (--- a1) (assembling) (n1 --- n2) (run-time) A constant which leaves (during assembly) the machine address of the return point which, at run-time, will write the accumulator (as high byte) and the bottom machine stack byte (as low byte) over the existing data stack 16-bit value n1.
PWIDTH@	FORTH (--- n1) Returns the PULSEwidth register value for voice #1, in the range 0 to 4095. See S!
PWIDTH!	FORTH (n1 ---) Sets the PULSEwidth register value for voice #1, in the range 0 to 4095. See S!
QUERY	FORTH Inputs 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.
QUIT	FORTH Clears the return stack. Stops compilation, and returns control to the operator's terminal. No message is given.
R	EDITOR (--- (text)) An editor command, replaces the string just found with the "F" command with the (text) following "R" command. (text) is optional, and if omitted, will cause a replace with the last text used in the replace command.
R	FORTH (--- n) Copies the top of the return stack to the computation stack.
R#	FORTH (--- a) A user variable which may contain the location of an editing cursor, or other file related functions.
R/W	FORTH (a blk f ---) The fig-FORTH standard disk read-write linkage. A (address) specifies the source or destination block buffer; blk is the sequential number of the referenced block; and f

	is a flag for f=0 write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs any error checking. This is a vectored word.
R>	FORTH (--- n) Removes the top value from the return stack and leaves it on the computation stack. See >R and R.
RAM>	SYSTEM (---) Selects the KERNEL and I/O back into the system, pronounced RAM FROM. See also >RAM.
RBYT	SYSTEM (---) A system primitive, used to read a byte of the sprite currently selected and about to be edited. A "*" is shown on the screen for each bit of the byte read, which was non-zero. Probably not a useful user word.
READ	FORTH (n1 ---) Used to read a screen from cassette, into screen number n1. Will read screen from cassette with names, by specifying a name with the NAME command prior to READ. The variables D# and F# are used to determine which device and file number are used to read from. Therefore, it is possible to read files from disk.
READS	FORTH (n1/n2 ---) Used to READ a group of screens from cassette, into screen n1 for a count of n2 screens. Cannot be made to read screens from disk, since each file must have a unique name.
RECREAD	SYSTEM (a1 --- a2) Reads a record from the currently open disk file, into address a1, and returns address a2, which is address a1 plus length of the normal read. Caution is recommended in the use of this word. Normally a POS is performed before a RECREAD.
RECWRITE	SYSTEM (a1/n1 ---) Writes a record to the currently open disk file, from address a1, for length n1 bytes, and terminates the write with a carriage return (13 decimal). Caution is recommended in the use of this word; normally a POS is performed before a RECWRITE. n1 is always 128 in 64FORTH.
REGREAD	SYSTEM A low level primitive, used by the sound system. Not a user word.

REGSET	SYSTEM A low level primitive, used by the sound system. Not a user word.
RELEASE@	FORTH (--- n1) Reads the release register value for the currently selected voice. See S!
RELEASE!	FORTH (n1 ---) Sets the release register value for the currently selected voice. See S!
REPEAT	FORTH (a n ---) (compiling) Used with a colon-definition in the form: BEGIN...WHILE...REPEAT At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN. At compile-time, REPEAT compiles BRANCH and the offset from HERE to address n is used for error testing.
RESONANCE@	FORTH (--- n1) Returns the current value of the filter, RESONANCE register setting, in the range 0 to 15.
RESONANCE!	FORTH (n1 ---) Sets the RESONANCE register value for the filter, lower values reduce filter effect, and higher values increase filter effect. N1 is in the range 0 to 15.
RP!	FORTH A computer dependent procedure to initialize the return stack pointer.
RP)	ASSEMBLER (--- n1) (assembly-time) Used in the code definition in the form: RP) LDA, or RP) 3 + STA, Addresses the bottom byte of the return stack (containing the low byte) by selecting ,X mode and leaving n1=$0101. N1 may be modified to another byte offset. Before operating on the return stack the X register must be saved in XSAVE and TSX, must be executed; before returning to NEXT, the X register must be restored.
S	EDITOR (---) An editor word, causes all lines from the current cursor line down to be SPREAD down, by one line. Line 15 will be lost, and the current cursor line will become blank.
S!	SYSTEM (---) A sound system primitive, which moves the contents of the memory copy of the SID chip, to the actual SID chip registers. This word must be executed whenever a change

	is made to the SID chip registers. It is not done automatically, because there would then be a heavy speed penalty for each sound word executed. As a result, this word is used after a group of changes are performed to cause them to all have effect at once.
S->D	FORTH (n d) Sign extend a single number to form a double number.
SAVE	SYSTEM (a1/n1 ---) Saves the data or program starting at address a1, for a length of n1 bytes, to the currently open mass storage device.
SAWTOOTH	FORTH (---) Sets the current voice to generate a SAWTOOTH waveform. One of a group of four words, which select the basic waveform for a selected voice. See also NOISE, SQUARE, TRIANGLE. See S!
SB	SYSTEM (---) A system constant, #30 hex in this system, which specifies the 64 byte block where the sprite data definitions start; $40 hex=64 decimal, and $30 times $40 is $0C00, the start of the sprite definition area.
SBASE	SYSTEM (--- a1) A system constant which returns the base address a1 of the SID chip.
SBLK	SYSTEM (--- a1) This is a system constant which returns the address a1 of an array which contains the sprite data block pointer bytes. This array is located at $07F8, just beyond the end of the display screen.
SCPTR	SYSTEM (--- a1) A system variable, used in the creation of a sprite definition. Not a user word.
SCR	FORTH (--- a) A user variable containing the screen number most recently referenced by LIST.
SCRN?	SYSTEM (n1 --- n2) A system primitive, which converts the sprite number n1 to the mass storage screen which contains, or will contain, the data statements for that sprite. Each sprite is assigned a particular location on a mass storage screen. Eight (8) sprites can be stored in two screens. Sprites 0 to 3 are always located in screen one (1), and sprites 4 through 7 are always located in screen two (2). Each sprite is

	assigned four (4) lines of its first screen to hold the ASCII decimal values associated with that sprite. Sprite 0 is located on line 0 through 3 of screen 1, sprite 1 is located on lines 4 through 7 of screen 1, and so on. See also SPWRITE and SPREAD.
SEC	ASSEMBLER (--- n1) (assembling) Identical to BOT, except that n1=2. Addressing the low byte of the second 16-bit data stack value (third byte on the low data stack).
SECOND	SYSTEM (n1 ---) A KERNEL system call, which sends the secondary address on the serial bus, normally sent after the LISTEN command. See also SLSN and LISTEN.
SETLFS	SYSTEM (n1/n2/n3 ---) A KERNAL system call, sets the logical file with file number n1, device number n2, and secondary command n3 into the KERNEL operating system.
SETNAM	SYSTEM (a1/n1 ---) A KERNAL system call, used to set the system filename to the string at address a1, for length n1 characters.
SHOW	FORTH (n1 ---) Turns on the sprite specified by n1 and makes it visible on the screen. N1 is in the range 0 to 7.
SIGN	FORTH (n d --- d) Stores an ASCII "–" sign just before a converted numeric output string in the text output buffer when n is negative. N is discarded but double number d is maintained. Must be used between <# and #>.
SINIT	SYSTEM (---) This is the sound initialization word. It must be executed before any other operations can be done to the sound system.
SLOAD	SYSTEM (a1 ---) A KERNEL system call, which loads the file specified by currently selected file name from the currently selected device into memory starting at address a1. This word is used after performing a NAME and SETLFS operation.
SLSN	SYSTEM (n1 ---) A system primitive, which commands the disk drive to listen with a secondary address of n1. Pronounced "secondary listen."
SMOVE	FORTH (a1/a2 --- a1/a2) This word moves 40 characters from address a1 to

	address a2, and modifies the date during the move, from ASCII to the Commodore equivalent value to show properly on the screen. Address a1 and a2 are returned unmodified. Address a2 is normally on the Commodore 64's screen, in the range 1024 to 2040, and address a1 is normally somewhere within the FORTH screen buffer. When this word is used in a DO LOOP, a full 24 line screen can be moved from anywhere in memory, to the screen in less than 50 milli-seconds. Pronounced "screen move."
SMUDGE	FORTH Used during word definition to toggle the "smudge bit" in a definitions' name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error.
SOURCE	FORTH (- - -) Decompiles the word specified by the text following SOURCE, to its sequence of high level steps.
SP@	FORTH (- - - a) Returns the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ ... would type 2 2 1).
SP!	FORTH Clears the stack.
SPACE	FORTH Transmits an ASCII blank to the output device.
SPACES	FORTH (n - - -) Transmits n ASCII blanks to the output device.
SPBASE	SYSTEM (- - -) Initializes the array SBLK to contain the proper pointers to the sprite data definition areas. Automatically executed when a new sprite is defined, or when a sprite is read from mass storage.
SPBIT	SYSTEM (- - -) A system sprite word used to read bytes from the screen in the process of building a sprite data definition. Not a user word.
SPGET	SYSTEM (- - -) A system sprite word used to read bytes from the screen in the process of building a sprite data definition. Not a user word.
SPRITE	SYSTEM (- - -) Builds the sprite data definition from the characters on the

	screen in the sprite definition area within the sprite editor border. This word is automatically executed by the sprite editor when the <RETURN> key is pressed upon completion of a sprite edit.
SPBYT	SYSTEM (--- a1) This is a system variable used in the creation of a sprite data definition after a sprite edit has been completed. Not a user word.
SPEDIT	FORTH (n1 ---) Enters the sprite edit mode with sprite n1. Shows the current form of the selected sprite. SPEDIT is terminated by pressing <RETURN>.
SPINDX	SYSTEM (--- a1) This is a system variable used in the creation of a sprite data definition after a sprite edit has been completed. Not a user word.
SPOBJ	SYSTEM (--- a1) This is a system variable used in the creation of a sprite data definition, after a sprite edit has been completed. Not a user word.
SPREAD	FORTH (n1 ---) Reads sprite n1 from the FORTH screen 1 or 2, into the sprite data area for the sprite specified. Each sprite has a designated area where it may be written to or read from. Sprites 0 to 3 are on screen 1, and sprites 4 through 7 are on screen 2. Each sprite has four lines, and its data is maintained in ASCII decimal in the screen. See SPWRITE.
SPWRITE	FORTH (n1 ---) Writes sprite n1 out of the FORTH screen, either screen 1 or 2, from the sprite data area where it may be written to or read from. Sprites 0 to 3 are on screen 1, and sprites 4 through 7 are on screen 2. Each sprite has four lines, and its data is maintained in ASCII decimal in the screen. See SPREAD.
SQUARE	FORTH (---) Sets the currently selected voice to generate a square wave. This is one of several types of waveforms that can be generated. See also TRIANGLE, SAWTOOTH, and NOISE. See S!
STATE	FORTH (--- a) A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

STEP	FORTH (---) Performs a TRACE single step, after a trace has been stopped, or a word has been selected for emulation with the EMULATE word.
STYPE	SYSTEM (a1/n1 ---) Sends n1 characters from address a1, out the serial bus. STYPE must be used after telling a device to listen. This is a low level primitive used by the system.
SUSTAIN@	FORTH (--- n1) Returns the value of the sustain register for the currently selected voice. See S!
SUSTAIN!	FORTH (n1 ---) Sets the sustain register value for the currently selected voice. See S!
SWAP	FORTH (n1 n2 --- n2 n1) Exchanges the top two values on the stack.
SYNC0	FORTH (---) When this word is executed, oscillator #1 is not hard synchronized with voice #3, and each oscillator can be programmed independently. This is the normal condition. See S!
SYNC1	FORTH (---) When this word is executed, oscillator #1 will be hard synchronized with voice #3. See S!
SYS	SYSTEM ((f1)/n1/n2/n3/a1 --- (f1)/n4/n5/n6) A SYSTEM primitive used to allow calling assembly language subroutines. Calls routine at address a1, passing registers A, X, and Y, as n1, n2, and n3. The flag (f1) is optional, and if included, sets the state of the carry flag to clear for a zero value, or sets for non-zero values. Register values A, X, and Y, are returned as n4, n5, and n6, after the subroutine has completed. Flag (f1) is returned unmodified if included. The contents of the processor status register is returned after a SYS in the ASSEMBLER variable N.
T	EDITOR (n1 ---) An editor word, moves the cursor to the beginning of line n1 on the current edit screen. N1 is in the range 0 to 15.
TAB	FORTH (---) Tabs to the next tab position on the screen. Used by VLIST and several other words requiring output formatting.
TALK	SYSTEM (n1 ---) A KERNEL system call, which commands a device in the range of 0 to 31 to TALK on the serial bus.

TESTO	FORTH (- - -) A sound control word, which can be used to synchronize oscillator #1 to external events. If this word is executed, oscillator #1 will be enabled—the normal condition. Normally used for testing. See S!
TEST1	FORTH (- - -) A sound control word, which can be used to synchronize oscillator #1 to external events. If this word is executed, oscillator #1 will stop output, until a TESTO is executed. Normally used for testing. See S!
TEXT	FORTH (c1 - - -) Accepts text from the input stream delimited by character c1. The text accepted is placed at PAD.
TKSA	SYSTEM (n1 - - -) A KERNEL system call, which sends byte value n1 in the range 0 to 31 as a secondary address command on the serial bus. This command is normally used after a TALK command to a device.
THRU	FORTH (n1/n2 - - -) Loads screen n1 THRU screen n2. If a disk file is open, the screens will automatically be read in from disk. If used on a cassette based system, the screens must be READ into the screen buffers before the THRU word can be used.
TIB	FORTH (- - - a) A user variable containing the addresses of the terminal input buffer.
TOGGLE	FORTH (a b - - -) Complement the contents of address a, by the bit pattern b.
TOP	EDITOR (- - -) Sends the cursor to the top left corner of the screen—the home position.
TRACE	FORTH (- - - t) Lists the lines of a definition of the word following TRACE. Performs the trace in the continuous mode, with vertical scrolling and stack output at each step.
TRAVERSE	FORTH (a1 n - - - a2) Moves across the name field of the fig-FORTH variable length name field. a1 is the address of either length byte or the last letter. If n=1, the motion is toward high memory; if n=−1, the motion is toward low memory. The a2 resulting is the address of the other end of the name.

TRIAD	FORTH (scr ---) Displays on the selected output device the three screens which include that numbered screen, beginning with a screen evenly divisible by three. Output is suitable for source text records. The screens to be printed must already be in memory.
TRIANGLE	FORTH (---) Sets the currently selected voice to generate a triangle wave. This is one of several types of waveforms that can be generated. See also SQUARE, NOISE, SAWTOOTH. See S
TYPE	FORTH (a count ---) Transmits count characters from address a, to the selected output device.
U	EDITOR (--- (text)) An editor word which places the (text) following U under the current cursor line. (text) is optional.
U*	FORTH (u1 u2 --- ud) Leaves the unsigned double number product of two unsigned numbers.
U.	FORTH (u1 ---) Displays the unsigned number in the current BASE on the screen, with one trailing blank following the free formatted number.
U/	FORTH (ud u1 --- u2 u3) Leaves the unsigned remainder of u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.
U<	FORTH (u1 u2 --- <f) Compares the two unsigned numbers and leaves a flag representing the truth of the statement u1 < u2 on the stack.
UNLSN	SYSTEM (---) A KERNEL system call, which tells all devices on the serial bus which were previously listening, to stop listening. Pronounced "unlisten."
UNTALK	SYSTEM (---) A KERNEL system call, which tells all devices on the serial bus to stop talking if they were currently talking, or to stop sending data on the serial bus.
UNTIL	FORTH (f ---) (run-time) (a n ---) (compile) Occurs within a colon-definition in the form: BEGIN...UNTIL

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

At compile-time, UNTIL compiles (0BRANCH) and an offset from HERE to address n is used for error testing.

UNTIL,	ASSEMBLER (a1/n1/n2 ---) (assembling) (---) (run-time) Occurs in a code definition in the form: BEGIN....n2 UNTIL, At run-time, UNTIL, controls the conditional branching back to BEGIN,. If the processor status bit specified by condition code n=2 is false, execution returns to BEGIN,; otherwise execution continues ahead. At assembly time, UNTIL, assembles a conditional relative branch to address a1, based on condition code n2. The number n1=1, and is used for error checking.
UP	ASSEMBLER (--- a1) (assembling) Used in a code definition in the form: UP LDA, or UP) Y STA, A constant leaving at assembly time the address of the pointer to the base of the user area. For example: HEX 12 # LDY, UP) Y, LDA Loads the low byte of the sixth user variable, DP.
UPDATE	FORTH Marks the most recently referenced block as altered. The block will subsequently be transferred automatically to disk should its buffer be required for storage of a different block.
UPORT	FORTH (--- a1) Returns the address of the user port in the Commodore 64 computer. Address a1 is the data port address.
USER	FORTH (n ---) A defining word used in the form: n USER cccc which creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.
V	EDITOR (---) Causes a VIEW of the current edit screen, around the currently specified cursor location.

VARIABLE	FORTH A defining word used in the form: n VARIABLE cccc When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch (@) or store (!) may have access to this location.
VDG	SYSTEM (n1 ---) A system sprite editor primitive used to draw a vertical line on the screen at column n1. Not a user word.
VECTOR	FORTH (n1 --- (text)) A defining word used in the form: n1 VECTOR (text) to create a word (text) which when executed will itself execute the contents of the n1 vectored routine in the I/O table. User created vectors must start at value 30 decimal, and must not be higher than 50. The vector position in the I/O table must be initialized before the new vector is executed or a crash will result.
VLIST	FORTH Lists the names of the definitions in the context vocabulary. The RUN/STOP key will end the listing. Hitting any key will stop the scrolling and any key will continue the scrolling.
VOC-LINK	FORTH (--- a) A user variable containing the address of a field in the definition of most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETting through multiple vocabularies.
VOCABULARY	FORTH A defining word used in the form: VOCABULARY cccc to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed. In fig-FORTH cccc will be chained to include all definitions of the vocabulary in which cccc is itself defined. All vocabularies ultimately chain to FORTH. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.

VOICE1	FORTH (---) Sets the current voice to be manipulated by the sound control words to VOICE1, by setting the value of the CURVOICE system variable to the offset into the SID sound register area. See S!
VOICE2	FORTH (---) Sets the current voice to be manipulated by the sound control words to VOICE2, by setting the value of the CURVOICE system variable to the offset into the SID sound register area. See S!
VOICE3	FORTH (---) Sets the current voice to be manipulated by the sound control words to VOICE3, by setting the value of the CURVOICE system variable to the offset into the SID sound register area. See S!
VOLUME@	FORTH (--- n1) Fetches the current value of the volume register in the SID sound generator chip. See S!
VOLUME!	FORTH (n1 ---) Sets the current value of the volume register to value n1 in the range 0 to 15 in the SID sound generator chip. See S!
VRESET	FORTH (---) Resets the I/O vector table, from the VWORDS vector table, to restore the state of the system I/O vectors. This word is executed by ABORT. The value of D# and F# are also reset by this word.
VWORDS	FORTH (--- a1) This word returns the address a1 of the beginning of the initial value vector table in ROM. See the section VECTORED WORDS for further information.
W	ASSEMBLER (--- a1) (assembling) Used in a code definition in the form: W 1+ STA, or W 1 – JMP, or W) Y ADC, A constant which leaves at assembly time the address of the pointer to the code field (execution address) of the FORTH dictionary word being executed. Indexing relative to W can yield any byte in the definition parameter field, for example: 3 # LDY, fetches the first byte of the parameter field.
WFORM	SYSTEM A defining word used to create many of the sound control words. Not a user word.

WIDTH	FORTH (--- a) In fig-FORTH, a user variable containing the maximum number of letters saved in the compilation of a definitions' name. It must be 1 through 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits.
WIPE	EDITOR (---) An editor word that causes the current editor screen to be cleared to blanks.
WORD	FORTH (c ---) Reads the next text characters from the input stream being interpreted until a delimiter c is found; storing the packed character string beginning at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or more blanks. The leading occurence of c is ignored. If BLK is zero, text is taken from the terminal input buffer; otherwise from the block stored in BLK. See BLK, IN.
WRITE	FORTH (n1 ---) Writes screen n1 to the current device as specified by D# and F#. The written screen will be sent to cassette by default.
WRITES	FORTH (n1/n2 ---) Screens n1 through n1 + n2 will be written to cassette.
X	EDITOR (---) An editor word which deletes the current cursor line, placing it in PADI—the insert buffer. All lower lines are moved up to fill the space. The text may be put back into the edit screen, with U <RETURN>, after moving the cursor to the desired line.
X)	ASSEMBLER Specify 'indexed indirect X' addressing mode for the next op-code generated.
XPAND	FORTH (f1/f2/n1 ---) A sprite control word, which allows sprite n1 to be expanded in either vertical or horizontal directions, or both. The boolean flag f1 is the X direction expand flag, and flag f2 is the Y direction expand flag. Example "1 0 3 XPAND" will expand sprite 3 in the X direction, but not the Y direction.
XSAVE	ASSEMBLER (--- a1) (assembling) Used in a code definition in the form: XSAVE STX, or XSAVE LDX,

	A constant which leaves the address at assembly time of a temporary buffer for saving the X register. Since the X register indexes to the data stack in z-page, it must be saved and restored when used for other purposes.
XOR	FORTH (n1/n2 - - -) Leaves the bitwise logical exclusive-or of the two values.
XY	FORTH (n1/n2/n3 - - -) Moves sprite n3 to screen position x=n1 and y=n2. This is the fast version which only has access to the first 255 X locations on the screen. Before XY can be used, the word CXY must have been used at least once to initialize the most significant bit of the sprite position. n1 is in the range 0 to 255, and n2 is in the range 0 to 159. n3 is the sprite number and must be in the range 0 to 7.
[	FORTH Used in a colon-definition in the form: : xxx [words] more ; Leaves the compile mode. The words after [are executed, not compiled. This allows calculation or compilation exceptions before resuming compilation with]. Called "left-bracket."
[COMPILE]	FORTH Used in a colon-definition in the form: : xxx [COMPILE] FORTH ; [COMPILE] will force the compilation of an immediate definition that would otherwise execute during compilation. The above example will select the FORTH vocabulary when xxx executes, rather than at compile time.
]	FORTH Resumes compilation to the completion of a colon-definition. See [. Called "right bracket."
↑	FORTH (- - - a1/n1) (text) This word allows text strings to be included into colon-definitions, in the form: : (name) ... ↑ text string ↑ ... ; Here the string "text string" will be compiled into definition (name), and when executed the address a1 of the string, and the length of the string n1 will be passed on the data stack to the words following, for further manipulation. This word can also be used outside of a definition, and will simply leave the address and count of the string of the text following on the data stack, with the string up above the PAD area. This word is often used with the system words DTYPE and STYPE, to send strings to the disk or serial bus.

↑FILE	FORTH (a1/n1 ---) This word opens the file name string at address a1, for character length n1, after first closing any file that is currently open. The text string should not be placed at HERE, because HERE is used for other processing. The variable FN is recommended as the optimum location for file name storage.

Human Engineered Software
150 North Hill
Brisbane, CA 94005
Telephone 415-468-4111
Telex: 278968