Super FORTH 1.12

TATUNG

Einsoft

FOR USE WITH EINSTEIN COLOUR MICRO COMPUTER

Einstein SuperFORTH 1.12 for the Tatung Einstein

(c) 1984 Peter Amey

CONTENTS

Einstein SuperFORTH 1.12 for the Tatung Einstein

by Peter Amey

1 INTRODUCTION

1.1 FORTH is a powerful and unusual language which, although fully interactive, is compiled and very fast. It is ideally suited for control, graphics and games tasks. The popularity of FORTH, which is rapidly growing, has probably been held back in the home computer market by its need, in a full implementation, for a disc drive. In addition many of the versions available provide only a basic core of the language and provide little or no support for the special hardware features of the machines on which they run. The arrival of the Einstein with its integral disc drive and this version of Einstein SuperFORTH overcomes both problems.

1.2 This manual provides a brief introduction to FORTH. Although it is not intended to be a substitute for a good introductory text book (see bibliography), it acts as a reference manual for Einstein FORTH.

2 FORTH TUTORIAL

2.1 INTRODUCTION. The principal difficulties with learning FORTH, after using languages such as BASIC, are conceptual. The language is both interpreted and compiled and cannot be accurately defined as either high or low level. A further difficulty for the beginner is that FORTH can be intolerant of errors. Whereas the worst thing that can go wrong with a BASIC program is the appearance of the dreaded words "SYNTAX ERROR", FORTH is just as likely to respond by producing a full system crash. If the initial struggle seems too much, carry on! The rewards for learning FORTH are enormous: faster program development, faster program execution and closer control of the hardware than any language other than an assembler to name but a few.

2.2 THE FORTH DICTIONARY. The FORTH language and FORTH programs consist of a threaded nest of WORDS organised into a dictionary. If you have not had the patience to read all this before loading FORTH you can view all the words in the dictionary by typing VLIST and pressing the Enter key. Each of these words is the name of a self-contained routine capable of performing a particular function and each is defined in terms of other more primitive words in the dictionary. As an analogy consider the real life 'function' MAKE-BREAKFAST. This could be defined using more primitive functions as:-

LAY-TABLE MAKE-TEA BOIL-EGGS MAKE-TOAST

Similarly each of these functions can be defined in terms of even more primitive ones. For example MAKE-TEA could be defined as:-

FILL-KETTLE BOIL-KETTLE WARM-POT ADD-TEA ADD-WATER etc

Each of these can be further subdivided until all the actions are being described in terms of very primitive functions indeed such as OPEN-HAND, LIFT-ARM etc etc. This is exactly how FORTH works. The primitive definitions are written in machine code and perform essential functions such as printing a single character etc. Other definitions are written in terms of these primitive words. In this way most of FORTH is written in FORTH! User-written FORTH programs are merely extensions of the core dictionary until, in the threaded way outlined above, the whole program can be described by a single FORTH word. Note that FORTH is very permissive about the characters that it allows to be used as the name of words and very generous about the length of such names. Any characters with an ASCII code less than 128 decimal (no graphics!) can be used in names and up to a (user alterable) limit of 31 characters may make up a name. Thus we can write:

NUMBER_OF_ALIENS instead of the more usual (in BASIC) NA

because FORTH programs are compiled this use of long names does not have the side-effect of gobbling vast quantities of memory as it would in BASIC. The name is preserved only in its own definition; all subsequent references to it compile down to a 2 byte address.

2.3 THE INTERPRETER. The FORTH inner interpreter is responsible for taking, and making sense of, the stream of characters it receives from the keyboard or from a disc file. When it encounters a word, which it can isolate because each word in the input stream is separated by a space, it first searches the dictionary for it. If it finds it then the word is executed, that is, the function described by that word is carried out. If FORTH cannot find the word offered it attempts to convert its characters into a number using the current number base. If this succeeds then the resulting number is placed on the parameter stack (of which more later) and FORTH starts work on the next word in the input stream. If number conversion fails then FORTH rejects the word by printing it at the terminal followed by a question mark. The phrase 'using the current number base' is significant: FORTH can operate with virtually any number base athough DECIMAL and HEX are the most common.

2.4 FORTH WORDS AS FUNCTIONS. FORTH words are analagous to BASIC subroutines or, more precisely, PASCAL procedures. Once a particular word has been thoroughly tested (which FORTH's interactive nature makes easy) its internal workings may be forgotten and it may be considered as a 'black-box'. The 'black-box', or in computer jargon, 'Function' takes various arguments or parameters in and may return one or more results. A function may also have some other effect such a drawing a picture on the screen, moving a robot arm etc. Debugging a large FORTH program is relatively painless provided the lower level definitions that make it up have been thoroughly tested.

2.5 THE PARAMETER STACK. If FORTH words are going to act on input parameters and (optionally) return results then a method of passing them is required. The mechanism that performs this task is called the 'Parameter Stack'. It is called a stack because, conceptually, the values placed on it are stacked up one on another so that the most accessible is always the most recently added. A stack of plates is a useful analogy; only the top plate, the most recently added, is accessible without resorting to juggling. Adding a value to the top of the stack is called 'pushing' the value. Removing the top value from the stack is called 'popping' the value. Parameters are passed to FORTH words by placing them on the stack then executing the word. This pops the values from the stack, performs operations upon them, then pushes any results back on to the stack where they are available for access by other FORTH words. As well as providing a means of passing parameters the stack is a useful temporary store for values such as intermediate results. If you examine a typical BASIC listing you will find that a large proportion of the variables in it are only there as short-term temporary homes for values. These values would probably reside on the stack in a FORTH program and would not need to be defined as specific variables.

2.6 STACK MANIPULATION. Because of the importance of the stack and the need to put parameters in the correct order for a particular word FORTH provides a number of ways of manipulating the stack contents. Full details of these will be found in the Glossary of FORTH Words later in this manual. Examples of stack manipulation words include:

DUP SWAP DROP ROT etc

Stack manipulation is an important FORTH art. The odd ROT or SWAP here and there can eliminate the need for specifically declared variables and this can greatly speed program execution.

2.7 REVERSE POLISH NOTATION. As we have seen above FORTH words expect to find their input values or parameters on the stack. They then pop these values, process them and push any results back on to the stack. This has an important consequence which has a fundamental effect on the appearance of FORTH programs; it forces the use of Reverse Polish, or Postfix, Notation. Postfix Notation, which will be familiar to users of Hewlett Packard calculators, specifies that an operator follows its arguments. For example take the simple operator +. Using infix, or algebraic, notation we would write:

5 + 3 which would give the result 8.

Using postfix notation we would write:

5 3 +   with, of course, the same result.

In many ways this is more natural than infix notation. Consider buying 2 papers in a newsagents. The mental process runs something like this:

Daily Mail, that's 18p, Guardian, that's 23p, so the total is 41p.

We naturally use postfix notation. There are other advantages to this approach as well, in particular, it is never necessary to use brackets to define the order of calculation. For example the following infix expression:

```
( 2 + 3 ) * 9
```

can be expressed in postfix notation without brackets as:

```
9 2 3 + *     or more simply 2 3 + 9 *
```

The more complex:

```
( 2 + 3 ) * ( 4 + 5 )
```

can be similarly expressed:

```
2 3 + 4 5 + *
```

Don't be put off by postfix notation. Despite it being the most publicised consequence of using FORTH it is not a real barrier to learning it.

2.8 DEFINING WORDS.

We have learnt that FORTH consists of a dictionary of words which can be combined together to form new words and, ultimately, complete applications programs. We will now look at the method of defining new words and adding them to the dictionary. This is achieved by the use of Defining Words. These are FORTH words whose functional effect is to compile a new word and add it to the dictionary.

2.8.1 COLON DEFINITION.

The most common and important defining word is the colon (:). When FORTH encounters a colon in the input stream it creates a new dictionary entry named after the word following the colon. It then compiles all the following words into the new dictionary entry until a semicolon (;) is encountered. The semicolon ends the defining process. This is much clearer with an example:

```
: BREAKFAST   LAY-TABLE  MAKE-TEA  BOIL-EGGS  MAKE-TOAST ;
```

The most important thing to remember when making new colon definitions is that all the words refered to must already be defined. The example above would fail if, for example, MAKE-TEA had not yet been defined.

2.8.2 CONSTANT.

CONSTANT is a defining word used to give a name to a constant numerical value. There are 3 advantages to using named constant for important numbers:

a   It is faster than using a literal number.

b   It makes programs much more readable and easier to understand.

c   It makes programs much easier to modify.

CONSTANT expects a number on the stack and should be followed by the name that is going to be used to refer to that value in future. This name is added to the dictionary. Subsequent use of this new word will push its value on to the stack. For example:

```
40 CONSTANT SCREEN-WIDTH      defines a new constant.

     SCREEN-WIDTH             puts 40 on the stack.
```

2.8.3 VARIABLE.

Whereas CONSTANT gives a name to a number which will not change during execution of a program VARIABLE defines a storage location where a changing value may be temporarily stored. It is used in the following form:-

```
n VARIABLE name
```

This defines a new dictionary entry 'name' which is used to refer to a memory location where a 16 bit integer value can be stored. The contents of that address are initialised to the value n. Subsequent use of 'name' leaves the address of the storage location on the stack. Values can then be read from or written to it using other FORTH words such as ! @ and ?. The function of each of these and other memory accessing words are described in the glossary. For now it is sufficient to remember that VARIABLE gives a name to a box in which numbers may be placed.

2.8.4 ADVANCED DEFINING WORDS.

The majority of programming tasks for which FORTH is suitable could be tackled with just the defining words described above, however the power of FORTH is greatly enhanced by the more powerful defining words it possesses. It is beyond the scope of this manual to go into detail about these more advanced features but users should be aware that they exist. Included are words for defining new FORTH words in assembler mnemonics or directly in machine code and, most powerfully of all, words for defining new defining words. This powerful and conceptually difficult concept makes FORTH infinitely extensible and makes it possible to create application specific compilers using it. The most common way of defining a new defining word is to use the <BUILDS DOES> construction. This is explained in the Glossary. Examples of this method can be found in some of the source code supplied on the FORTH master disc, for example the ARRAY definitions in the file UTILITY.FIG.

Examples of assembly and machine language definitions will also be found in the supplied FORTH source files, especially GRAPHICS.FIG.

2.9 PROGRAM STRUCTURE. Now that we know how to define new words we can start to write FORTH programs, however we are limited to writing routines with no branches, decisions or loops. This severe limitation is overcome by FORTH's powerful program structuring features. Some of these have analagous BASIC constructs but most have equivalents only in highly structured languages such as Pascal.

2.9.1 DO LOOP. The FORTH DO LOOP is similar in function to the BASIC FOR NEXT loop. It permits the repetitive execution of a word or number of words a set number of times. DO LOOP is the appropriate structure to use when the number of iterations is known in advance. A DO LOOP example:

```
10 0 DO words to be executed LOOP
```

DO expects 2 numbers on the stack; top of stack is the start value of the loop counter and below this is the end value. FORTH executes the words between DO and LOOP and each time LOOP is reached it increments the loop counter by 1, if the counter is equal or greater than the end value then the loop terminates otherwise it is repeated. A DO LOOP is always executed at least once. Within a DO LOOP the value of the loop counter can be placed on the stack by the word I. For example:

```
10 0 DO I . LOOP
```

prints the numbers 0 to 9. (. is the FORTH word to print the number on the top of the stack.) If we wish the loop counter to change by a value other than 1 on each pass through the loop we can use the word +LOOP instead of LOOP. +LOOP expects a number on the stack which it uses to increment the loop counter. Thus:

```
10 0 DO I . 2 +LOOP
```

prints the even numbers between 0 and 9. Finally, if FORTH encounters the word LEAVE in a loop it sets the loop counter equal to the exit value so as to ensure that the loop is terminated next time LOOP (or +LOOP) is reached. This allows a controlled exit from a DO LOOP before its normal completion. Incidently this is often done in BASIC programs with a GOTO to a point outside the FOR NEXT loop. This practice is insecure and dangerous; the FORTH method is much to be preferred. Along with all the FORTH programming structures DO and LOOP may only be used within colon definitions - direct execution of them is not possible.

2.9.2 BOOLEAN LOGIC. Before we can consider the remaining FORTH program structures we must consider how FORTH handles Boolean or true/false logic. FORTH handles decisions by using flags to represent logically true and logically false values. These flags are merely numbers; false is represented by the value 0 and true by any non-zero value. This convention allows a logical AND to be evaluated by multiplying two flags together and a logical OR to be evaluated by adding two flags together. The FORTH comparators (=,< etc) leave logical flags on the stack and the programming structures examine the flags and make decisions based on them.

2.9.3 IF ELSE ENDIF. This structure is the basic decision making device in FORTH. IF examines the top of stack and if the value there is true the words between IF and ELSE are executed. If it is FALSE then the words between ELSE and ENDIF are executed. In either case program flow resumes after ENDIF. The ELSE part of this structure is optional. If it is not present and the top of stack is false when tested by IF then no action occurs at all and execution resumes after ENDIF. For various historical reasons THEN can be used as an alias for ENDIF.

2.9.4 BEGIN UNTIL. This construction is used when a number of words are to be repetitively executed until some condition is true. It is the appropriate structure when the number of iterations is unknown but must execute at least once. When UNTIL is reached the top of stack is tested. If it is true then execution continues with the words after UNTIL. If it is false then execution restarts after BEGIN. Care must be taken that the top of stack becomes true at some point in the loop or else it will be infinite. A common FORTH use of this structure calls ?TERMINAL (which return true if any key is pressed) just before UNTIL. This allows a loop to continue indefinitely until interrupted by the user.

2.9.5 BEGIN WHILE REPEAT. This is similar to BEGIN UNTIL except that the test is made before the loop and it is therefore possible that the loop will never be executed at all. WHILE tests the top of stack, if it is true then the words between WHILE and REPEAT are executed. If it is false then execution skips to beyond REPEAT. When REPEAT is reached execution returns to BEGIN, the test is repeated at WHILE and so on.

2.9.6 CASE ENDCASE. The CASE construct allows a multi way switch to be written. It is generally clearer and simpler than writing the same switch using a nest of IF ELSE ENDIFs. CASE is not found in most implementations of fig-FORTH but is included in Einstein FORTH. An explanation of the construction is in the Glossary and there are many examples in the accompanying files on the FORTH master disc, in particular the editors.

2.10 ACTUALLY WRITING FORTH PROGRAMS. We now have all the tools necessary to write FORTH programs. Words, perhaps involving the various loop and decision making constructions described, are combined until the entire application is described by a single word. Typing this word at the keyboard will cause

FORTH's inner interpreter to execute the program. Incurable BASIC programmers can even define this word as RUN! Definitions may be entered directly from the keyboard, in which case they are compiled and added to the dictionary as soon as Enter is pressed, or written to a disc file using the editor for later loading. The latter method is the more usual since directly entered definitions cannot be conveniently altered if they prove to be incorrect. Direct entry of definitions can, however, be convenient for setting up quick test routines or programming tools. When we use the editor to write source code it is first written to a disc file. These definitions can be called up and edited at will. Once the source code is written the entire program can be compiled into the FORTH dictionary by use of the word LOAD. This takes selected parts of the file and presents them to the FORTH inner interpreter exactly as if they had been directly typed in from the keyboard. If the definitions prove to be incorrect they can be recalled, edited, the old version forgotten (see FORGET in Glossary) and the new vesion reloaded. There are no restrictions on the layout of FORTH source code except that each word must be separated by one or more spaces. Even when entering definitions directly they may occupy any number of lines, the definition is finished only when the semicolon is reached. Comments (which are deliminated by brackets) may be freely used and, because FORTH compiles its source code, occupy no space in the dictionary. Use of the editor allows top-down programming to be practiced since it is possible to assume the existence of any desired word then go to an earlier screen and actually define it.

3 EINSTEIN FORTH 1.12 MASTER DISC

3.1 ACKNOWLEDGEMENTS. This implementation of fig-FORTH is based on the 8080 assembly language listings distributed in the public domain courtesy of the :-

FORTH Interest Group
PO Box 1105
San Carlos
CA 94070

The Tatung Einstein implementation was by Peter Amey and this, the machine-dependent portions of the code and the modifications are copyright (1983,1984). Unauthorised distribution of Einstein fig-FORTH is expressly forbidden.

3.2 COMPLIANCE WITH STANDARD fig-FORTH. This implementation provides all the features of standard fig-FORTH, as defined in the FIG implementation manual. In addition, this version incorporates a full screen editor (for both 40 and 80 column Einsteins), a FORTH 8080 assembler and a number of extensions to the basic fig-FORTH model. The principal change to the fig model is the manner in which disc access is handled. Traditional fig-FORTH uses the disc purely as pseudo ram without any file structure. This implementation uses random access Tatung/Xtal DOS files. This achieves the same purpose but enables FORTH source code to exist on a disc with other files and logically separate applications to be kept in separate files. Once a file has been accessed and opened by FORTH interaction with it is indistinguishable from traditional fig-FORTH. A number of file handling utilities and words are included in the implementation. FORTH purists who object to this file oriented approach can open a file (perhaps called SCREENS?) at the beginning of each session and thereafter ignore the fact that files are in use - all the usual virtual-memory techniques can be used. This is not recommended, however, since sensible use of files has great benefits to the user.

3.3 PROGRAM PORTABILITY. FORTH program portability is generally better than that of BASIC programs. Despite the many extra features supported by Einstein FORTH, care has been taken that the basic core of the language conforms to a recognisable standard. In the hobby market there are 3 main FORTH dialects: fig-FORTH, MVP-FORTH and FORTH79 (which is gradually being replaced by the updated FORTH83) The first 2 are very similar and should pose few portability problems. FORTH79 looks very similar but uses a number of words in a sightly different way which can be very confusing. The main aid to portability is the natural extensibility of the language. If you are trying to convert someone else's program to run on the Einstein and it uses a word that Einstein FORTH does not contain then the missing word can simply be defined and added to the dictionary. Of course, to succeed, the source program must have been adequately documented. Einstein programs will be substantially portable to other machines providing machine-specific features (sound, colour etc) are not involved. As an aid to those wishing to write portable code non-standard words and extensions to Einstein FORTH are indicated with an asterisk in the Glossary.

3.4 CHARACTER SET. FORTH makes use of square brackets which are not present in the normal Einstein character set. To provide them FORTH alters the character set on start up and provides square brackets as ASCII codes 91 & 93. Square brackets can be accessed with the left and right arrows keys (1/4 and 3/4 keys). Other characters may be redefined using the FORTH word SHAPE (see Glossary).

3.5 MASTER DISC CONTENTS. The distribution disc contains a number of files and these are described below. Many of the files are FORTH source code for extensions to the basic language core. These extensions could have been incorporated into the language core but there are considerable advantages in supplying them as extensions for optional loading. These are as follows:

a Only relevant extensions need to be loaded. This keeps the language size down. There is no need to load the Sprite handling words if you are writing a data base for example.

b The extension files serve as a demonstration of FORTH programming, especially systems level work and the use of the assembler.

c Skilled users may alter and extend the extension files to suit their own applications.

The files supplied are:

3.5.1 FORTH.COM This file contains the core of the FORTH language itself. As a COM file it can be run directly from the Xtal DOS level. A FORTH source code file may be typed on the command line thus:

```
FORTH GAME
```

This will call up the FORTH compiler/interpreter and open GAME.FIG as the current file.

3.5.2 EDITOR40.FIG This file contains source code for a full screen editor for standard 40 column Einstein machines. It must be loaded (compiled) before any source code can be written to disc files. It may be useful to save an enlarged vesion of FORTH including the editor and other extensions. Instructions for doing this will be found later in the manual.

3.5.3 EDITOR80.FIG This file serves the same purpose as EDITOR40.FIG but is intended to be used with the Einstein 80 column display card.

3.5.4 FUNKEYS.FIG Contains source code for routines to program the 16 Einstein function keys. Key definitions can be entered from the keyboard or loaded from disc files.

3.5.5 GRAPHICS.FIG Contains source for all the Einstein graphics features except Sprites. All the features familiar to users of Tatung/Xtal BASIC 4 are supported.

3.5.6 SPRITES.FIG Contains source code to enable the creation and control of Sprite characters.

3.5.7 SOUNDS.FIG The words supplied in this file allow control of Programmable Sound Generator (PSG) chip in a user-friendly manner. Further details of the functioning of this chip are to be found in pages 288-310 of the Einstein BASIC Reference Manual.

3.5.8 DOSTOOLS.FIG Contains words allowing display of the disc directory, deletion of files and the transfer of source code from one file to another.

3.5.9 UTILITY.FIG This file contains a miscellany of useful definitions including a random number generator, arrays and strings. Each screen of this file can be loaded separately as required.

3.5.10 GAME.FIG This file contains source code for a simple arcade-type game and is intended as an example of FORTH programming making use of many of the Einstein specific extensions available in this version of FORTH. Users may wish to extend this game as a learning exercise.

3.5.11 FLOAT.FIG This file contains a floating point arithmetic and trignometric function package. The first part of the file (screens 1 to 12) contains the arithmetic routines and the latter part (screens 14 to 19) the trig functions.

3.6 MAKING A WORKING COPY OF FORTH. Before experimenting with FORTH it is vital that you make a backup copy of the disc supplied. Do not even contemplate running FORTH from your master disc; should it become corrupted or physically damaged it cannot be replaced. The disc is not copy protected and may be copied using BACKUP.COM as supplied on your system master disc. Once this task has been carried out you are ready to start using FORTH. A useful first step is to make and save an enlarged version of FORTH incorporating some extensions useful for program development. Useful additions to the FORTH core for this purpose are the DOSTOOLS and the EDITOR. The procedure is defined below. Your entries from the keyboard are underlined and the Enter key should be pressed after these commands are typed in. FORTH's responses are not underlined and comments explaining what is going on are in brackets. Start by bringing your Einstein up to the DOS level and then, in answer to the 0: prompt, proceed as follows:

```
FORTH DOSTOOLS          (run FORTH, make DOSTOOLS.FIG current
                         file)

Z80A Einstein FORTH 1.12   (system sign on message)
  (c) 1984 P Amey

1 LOAD 2 LOAD 3 LOAD  (load 1st 3 screens of file)

Loading Blocks: 1 2 3 4 5 6 7       ( directory routine)
Loading Blocks: 9 10 11 12 13 14 15 ( scratchpad routine)
Loading Blocks: 17 18 19 ok         ( delete file routine)

NEWFILE EDITOR40      (change file to editor)
ok

1 LOAD                (load entire editor - screens are
                       linked)
Loading Blocks: 1 2 3 etc

: TASK ;      (define dummy word to act as a boundary between the
               FORTH dictionary and any later additions you may
               make)
```

TASK Isn't Unique ok (there is already a TASK in the dictionary - this does not matter in this case but FORTH warns you just in case)

<u>SAVE</u> (FORTH word for saving enlarged versions)

SAVE 62 BLOCKS

O:

FORTH tells you how big the new version is - in 256 byte blocks - and returns you to Xtal DOS. All that is now necessary is to type: <u>SAVE 62 BIGFORTH.COM</u>. (Any suitable name can be used.) The DOS will then save the enlarged FORTH to disc. From now on you can use BIGFORTH instead of FORTH and the Editor and Dostools will be ready for immediate use.

3.7 INPUT MODES. Einstein FORTH can accept command from the keyboard in either of two modes. On start up the default mode is screen input. This allows commands to be picked up off the screen. Nothing is read-in by FORTH until the Enter key is pressed, before that the cursor can be moved to any line at will. The input to FORTH consists of the 40 character line on which the cursor rests when the Enter key is pressed. Alternatively a line input mode can be selected with the FORTH word LINE-MODE. This allows up to 80 characters to be input at a time but the only editing permitted is to use the DEL key and retype. The full-screen editing mode can be re-selected with the word SCREEN-MODE. When in line mode the printer can be coupled to the screen with control R and uncoupled with control S. In screen mode all the control codes described in the Einstein reference card may be used.

4 NUMBER SYSTEMS

4.1 INTEGER ARITHMETIC. Standard fig-FORTH does not include routines for floating point arithmetic, but since double-length integers allow a range of +/- 2x106 and full output formatting facilities are available to position 'decimal points', integer arithmetic is more than adequate for almost every situation. Normal or single-length numbers are 16 bits long. Double-length numbers are 32 bits long. Code for floating point arithmetic is provided on the distribution disc for those who really cannot manage without it.

4.2 NUMBER BASES. fig-FORTH can operate in any number base from 2 (binary) to 36. The words HEX and DECIMAL select those number bases, but any other base can be used by storing its value in the current base in the user variable BASE. Eg.

2 BASE !

selects binary.

5 FILE HANDLING

5.1 VIRTUAL MEMORY. FORTH can be run without having a disc file open. Definitions can be entered in direct mode from the keyboard and subsequently run. Operating in this way is extremely limiting since the editor cannot be used and definitions cannot be written to or loaded from disc. To make this possible a file must be opened. This file provides a name under which disc access will be controlled by Tatung/Xtal DOS. Once a file has been opened, Einstein FORTH uses it as virtual memory, divided into 128 byte sections called 'Blocks'. The block is the fundamental unit for accessing the disc. It is these disc blocks that are accessed by the FORTH words BLOCK and BUFFER. To allow editing of source code held on disc 8 of these blocks are treated as a single unit called a 'Screen'. Screens contain 16 lines of text each 64 characters long and are accessed by the editor and by the words LIST, TRIAD etc defined below.

5.2 OPENING A FILE. A disc file is specified in one of four ways:

a By typing the name on the command line after FORTH.

b By using the FORTH word NEWFILE (see glossary).

c By using the FORTH word GETFILE (see glossary).

d By using the FORTH word FILEVAR to create a file variable. (eg. FILEVAR FRED.TXT) When the file variable is typed at the keyboard or encountered in a program it becomes the current file. (eg. typing FRED.TXT closes the current file and sets the current file to FRED.TXT)

Files can be switched at will since use of GETFILE, NEWFILE or a file variable will always close the old file before opening the new. Attempts to access screens without having a file open will result in an error message. The warning message "New File" will be displayed if a specified file cannot be found in the disc directory.

5.3 FILE EXTENSIONS. FORTH files have the default extension .FIG . This will be inserted if an extension is omitted. The filename can have an optional drive specified. If omitted the default drive will be used. The default drive can be changed with the FORTH word DRIVE (see glossary).

5.4 FILE LOCKING. As with Tatung/Xtal BASIC 4 and Tatung/Xtal DOS, files can be locked, or marked as read only, by the user. This provides some protection against accidently destroying a disc file. The word LOCK will mark the current file as read only; UNLOCK will remove the protection. Locked files are indicated with an asterisk when their name is displayed with loaded as normal and can be called up by the editor. The locking shows itself when an attempt is made to write to the file usually by the use of FLUSH, NEWFILE, GETFILE or DOS. Each of these words tries to write the contents of any updated disc buffers to disc which will not be permitted if the destination file is locked. There are two ways to escape when this error occurs: if you wish to actually carry out the write to disc then type UNLOCK then repeat the commands that caused the error; if you wish to abort the write then type EMPTY-BUFFERS.

5.5 DISC CHANGING. To ensure that files are correctly closed always FLUSH before changing discs and always use DOS (not the reset button) to return to Tatung/Xtal Dos. After a disc change use LOGON to log on (!) the new disc. This will initialise the error handling system. It is possible to corrupt disc contents if these precautions are not taken. It is especially important to use FLUSH before changing discs if anything has been written to the file in use prior to the change. This is because FORTH holds information in RAM buffers before writing it to disc. If the buffers are full, the disc is changed, and FORTH then attempts to write the buffers contents to disc then corruption will result.

5.6 DISC RANGE. The lowest screen or block number available is 1. Attempts to access screen 0 or negative screens generates an error message. High screen numbers should be used with caution since their use will rapidly fill up the disc and wasted space cannot be used by other files. For most purposes it is best to start a program at screen 1 of a file and work upwards as needed.

6 FORTH WORDS AND THEIR DESCRIPTIONS

6.1 The FORTH language is built up of 'FORTH words'. Each word is built up from other, more fundamental, words which have already been defined either by the fig-FORTH release or by the programmer. Because FORTH words operate by manipulating a stack using Reverse Polish Notation, the standard descriptive system for them is as follows:

```
        (stack before --- stack after)
```

where '---' signifies the execution of the word in question. The contents of the stack are listed as deep as necessary, separated by a space and with top-of-stack on the right.

As an example, the word + (plus) would be described as:

```
        + ( n1 n2 --- n )
```

to indicate that two 16-bit values are placed on the stack and that both are removed from the stack during execution of 'plus', which puts the total back on to the stack.

6.2 SYMBOLS USED IN THE EXPLANATIONS. The descriptions of FORTH words which follow will use the following symbols:

addr	memory address
b	8-bit byte
c	7-bit ASCII character
d	32-bit signed double integer. Most significant byte (sign) on top of stack.
ud	32-bit unsigned double integer.
f	Boolean flag. 0=false. 1=true.
ff	Boolean false flag.
tf	Boolean true flag.
n	16-bit single-length signed integer.
u	16-bit unsigned integer.
TOS	Top of stack.
*	Extensions to basic fig-FORTH.

7 VOCABULARIES

7.1 fig-FORTH allows words to be put into separate vocabularies. This version contains FORTH, EDITOR and ASSEMBLER vocabularies. Each vocabulary will be described separately, but it is important to note that the same word can be employed in each vocabulary without causing problems to FORTH. Whether or not it confuses the user is another question! Message 4 (name not unique) will be displayed as a warning even if the duplicate definition is in a different vocabulary to the original.

7.2 At any time when running FORTH, there are two vocabularies to consider: the CONTEXT vocabulary and the CURRENT vocabulary. The CONTEXT vocabulary is where dictionary searches will start, whilst the CURRENT vocabulary is that to which newly-defined words will be added. A new vocabulary is created by

```
        VOCABULARY name
```

and made into the current vocabulary by

```
        name
```

The current and context vocabularies are both set to 'name' by

```
        name DEFINITIONS
```

8 ELIMINATING WORDS

8.1 FORGET. The word FORGET will eliminate all words back to and including the specified word, starting with those words most recently defined. Only the current vocabulary is affected and it is vital that the context and current vocabularies are the same before FORGETting. The system will respond with a suitably rude message if you forget! A typical use would be:

```
FORGET MYWORD
```

8.2 Difficulties may be experienced when FORGET is used in a part of the dictionary containing many interleaved vocabularies. This fault is inherant in the design of fig-FORTH and is common to all implementations of it. In practice it does not usually cause any problems.

8.3 FENCE. User variable FENCE contains an address below which FORGET is inhibited. The sequence HERE FENCE ! protects all the words thus far defined. The sequence ANYWORD FENCE ! moves the protection to another point in the dictionary.

9 ERROR HANDLING

9.1 MESSAGES.FIG Error messages are contained in the file MESSAGES.FIG which can be user extended. On cold starting the system or using LOGON the system looks for MESSAGES.FIG on drive 0. If it is found then user variable WARNING is set to 1 and all error messages will be in text form. If the messages file is not present then WARNING is set to 0 and numeric errors are used. The selection of text or numeric errors is automatic as long as LOGON is used after a disc change. WARNING can be zeroed manually if it is desired to switch of text errors for any reason.

9.2 ERROR VECTOR. As an enhancement to the error handling system a vectored jump system has been introduced. This enables a user written error handling routine to be substituted for the inbuilt one. The user error routine is called when WARNING is set to -1. The vector is set up by

```
' NAME ERRVECT
```

where NAME is the word to be called on error. This routine must clear the stack with SP! and, if not a closed loop, end with QUIT.

Example:

```
: MYERROR ." ERROR NUMBER " . SP! QUIT ;
-1 WARNING !
' MYERROR ERRVECT
```

This will result in the message ERROR NUMBER n being displayed instead of the FORTH error messages. The standard error system can be restored by storing 1 or 0 in WARNING. The default setting in ERRVECT is ABORT. This maintains compatability with the fig standard. Tampering with the error handling routines is best left until the basic concepts of the language are mastered.

9.3 OTHER ERROR HANDLING WORDS. Other words associated with error handling are:

?COMP	If not compiling, issue error message.
?CSP	If stack pointer does not equal the value in CSP, issue error message.
?DEPTH	*(n ---) Issue error message if stack depth less than n.
?ERROR	(f n ---) Issue error message n and QUIT if f is true. Else carry on.
?EXEC	Error message if not executing.
?LOADING	Error message if not compiling.
?PAIRS	(n1 n2 ---) Error message if n1<>n2. Used to check compiled conditionals.
?STACK	Issue error message of stack out of bounds. Only checked when control returned to the terminal, so no range check during execution.
ABORT	Clear stacks and return control to terminal with sign on message.
FILE?	*(---) Issues error 10 if no filename is present in the FCB.

10 EDITOR40 AND EDITOR80

10.1 The Editor vocabulary is invoked automatically when any of the full-screen editing commands are used. It is not normally necessary to explicitly use the vocabulary word EDITOR. Any legitimate exit from the editor automatically reselects the FORTH vocabulary. The editor is used to write source code to disc screens for later LOADing. Note that definitions can be entered direct from the keyboard without the editor and in this case they are compiled as soon as Enter is pressed. This is useful for testing short words or for writing quick, once-only utilities. Where longer definitions are being written it is better to use the editor and save the source code before testing. Before the EDITOR can be used the source code for it must be LOADed. It is useful to keep 2 versions of FORTH one of which has the editor (and other utilities) permanently installed.

Screens can be selected for editing with:

EDIT	*(---) Selects the most recently accessed screen (or screen 1 if a newly opened or flushed file) for editing.
SELECT	*(n ---) Select screen n for editing.

When using the editor the keys works as follows:

Cursor keys	These function as expected. If you cursor off the top or the bottom of a screen the previous or following screen is displayed. The current cursor column is continuously displayed at the top of the screen. In 40 column versions two screen lines are used to display the standard FORTH 64 character lines.
CTRL-^	Move the cursor to the top left of the current screen.
CTRL-L	Clear the current line from the cursor to the end of the line.
INS	Switch from overwrite to insert mode. A caption will be displayed at the top of the screen to indicate this.
DEL	Delete the character to the left of the cursor. It is not possible to delete beyond the beginning of a line.
ESC	If in insert mode then switch to overwrite mode. If in overwrite mode then exit from the editor back to FORTH.

Other keys are displayed and written to the file.

Additional editing features can be accessed with the following control codes: (Note that PAD is a scratchpad area used as a temporary store for text strings)

CTRL-X	Create an eXtra blank line at the cursor line. Line 15 of the screen is lost.
CTRL-E	Erase the current line and close the gap by moving following lines up; this leaves line 15 blank. The deleted text is written to PAD.
CTRL-I	Insert the text at PAD at the cursor line. Following lines are pushed down, 15 being lost.
CTRL-P	Put contents of PAD on to current cursor line replacing the current line.
CTRL-C	Copy the current cursor line to PAD but otherwise leave it alone.
CTRL-T	Type the current contents of the PAD at the bottom of the screen
CTRL-F	(Find) Prompt for a search string and search for it beyond the cursor. If found the cursor is placed over it. If not found then a Bell sounds and the cursor is placed at the Home position. The search string is placed at PAD before the search takes place.
CTRL-A	Search Again using the search string already at PAD.
WHERE	If an error occurs when compiling, the word WHERE will list the offending screen with the cursor immediately after the word causing the error and invoke the editor. The stack contents must not be altered between the error occurring and WHERE being used.
CLEAR	(n ---) Erase the contents of screen n of the current file.
COPY	(n2 n1 ---) Copy the contents of screen n2 to screen n1 overwriting its contents. Note that this follows the usual Forth convention of source coming before destination.

11 A GLOSSARY OF FORTH WORDS

11.1 FILE ACCESSING

?LOCK	*(--- f) Returns true flag if current file is locked.
FLUSH	(---) Write all updated blocks to the current disc file then close and reopen the file. FLUSH has no effect if no file is open.
MESSAGE	(n ---) Output line n of MESSAGES.FIG. No output is generated if line n does not exist. MSG£n is output if MESSAGES.FIG is not on drive 0 or user variable WARNING contains 0.
GETFILE	*(---) Allow the input of a file name from the console and open it. The current file is closed first.
NEWFILE	*(---) As above except that no prompt is issued and the next word in the input stream is accepted as the filename. eg: NEWFILE A:FRED. NEWFILE should only be used from the keyboard, not within programs.
LOGON	*(---) Initialise the disc system and attempt to open MESSAGES.FIG on drive 0. Clears file descriptor block for the current file. Should always be used after a disc change.
DOS	*(---) FLUSH, close files and exit to Tatung/Xtal Dos. This is the equivalent of BYE or MON on most fig-FORTH systems.
.FILE	*(---) Print the current filename.
.DEF	*(---) Print the current default drive number.
FILEVAR	*(---) Used in the form FILEVAR name. Creates a word 'name' that when executed closes the current file and opens a new file (or reopens an old file) called 'name'. This is most useful within programs where you might want to be able to switch between 2 or more files under program control.
DRIVE	*(n ---) Make drive n the default drive.
SAVE	*(---) Alter boot-up parameters to reflect current dictionary size. Print dictionary size in 256 byte blocks then exit to DOS. This allows the memory image of the enlarged FORTH system to be saved to disc as a .COM file.

11.2 MANIPULATING SCREENS (NB. A file must be open)

INDEX	(n1 n2 ---) Displays the first line of screens n1 to n2. These lines should normally be comments.
LIST	(n ---) List the screen n.
LOAD	(n ---) Load and compile the text on screen n. The block numbers being loaded will be displayed during the loading process.
.LINE	(line screen ---) Print on the terminal the specified line from the screen. Trailing blanks are suppressed.
-->	(---) Continue LOADing the next screen.
TRIAD	(n ---) List 3 consecutive screens, starting with screen n. This is a useful word for creating program listings since 3 screens fit nicely on 1 page. A footer taken from line 15 of MESSAGES.FIG is displayed at the bottom of each page.
;S	Terminate loading of a disc screen.

11.3 ACCESSING THE DISC BLOCKS (NB. A file must be open)

BLOCK	(n --- addr) Check that block n is in RAM. If not, read it from disc. Leave its buffer address on TOS.
BUFFER	(n -- addr) Obtain the next memory buffer, assigning it to Block n. If the buffer is marked as updated it is written to disc. The block is not read from

the disc. The address left is the first cell within the buffer available for data storage.

EMPTY-BUFFERS	(---)	Mark all buffers as empty without writing them out to disk. Override all UPDATE flags.
UPDATE	(---)	Mark the most recently accessed block as updated so that FORTH will automatically write it out to disk if the buffer is re-used or FLUSH is executed.

11.4 SINGLE LENGTH ARITHMETIC. These words all operate on 16 bit integers.

*	(n1 n2 --- product)	Leave the signed product of 2 signed single-length numbers.
+	(n1 n2 --- sum)	Leave the sum of n1 and n2.
+-	(n1 n2 --- n3)	Apply the sign of n2 to n1 and leave as n3.
-	(n1 n2 --- diff)	Leave the difference of n1 n2.
/	(n1 n2 --- quot)	Leave the signed quotient of n1/n2.
/MOD	(n1 n2 --- rem quot)	Leave the remainder and quotient of n1/n2.
1+	(n --- n+1)	Increment TOS by one.
2+	(n --- n+2)	Increment TOS by two.
ABS	(n --- u)	Convert signed integer to absolute value as unsigned integer.
MAX	(n1 n2 --- max)	Leave the greater of n1 and n2.
MIN	(n1 n2 --- min)	Leave the smaller of n1 and n2.
MINUS	(n1 --- n2)	Leave the two's complement of n1 as n2.
MOD	(n1 n2 --- mod)	

Leave the remainder of n1/n2 with the sign of n1.

11.5 DOUBLE LENGTH ARITHMETIC. These words operate on 32 bit integers each of which takes up 2 stack locations.

D+	(d1 d2 --- dsum)	Leave the double length sum of 2 double-length numbers.
DMINUS	(d1 --- d2)	Leave the two's complement of d1 as d2.
S-->D	(n --- d)	Convert a signed single-length number to a signed double-length number.

11.6 MIXED LENGTH ARITHMETIC.

*/	(n1 n2 n3 --- n4)	Leave the answer n4=n1*n2/n3 with double-length intermediate results for greater accuracy than * and / alone would give.
*/MOD	(n1 n2 n3 --- n4 n5)	Leave the quotient n5 and the remainder n4 to the operation n1*n2/n3, again with a double-length intermediate result.
D+-	(d1 n --- d2)	Apply the sign of n to d1 to leave d2.
M*	(n1 n2 --- d)	Leave the double-length signed product of n1 and n2.
M/	(d n1 --- n2 n3)	Leave the single-length remainder n2 and single-length quotient n3 from the calculation d/n1.
M/MOD	(ud1 u2 --- u3 ud4)	Leave the unsigned remainder u2 and unsigned double-length quotient ud2 from the calculation ud1/u2.
U*	(u1 u2 --- ud3)	Leave ud3 as the unsigned product of u1 and u2.
U/	(ud1 u1 --- u2 u3)	Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

11.7 INPUT/OUTPUT

11.7.1 CHARACTER INPUT/OUTPUT

.CPU	(---) Print the computer/CPU name as set up by the implementer.
-TRAILING	(addr n1 --- addr n2) Adjust the length n1 of character string starting at addr to length n2 so as to exclude trailing blanks.
."	(---) Output a literal string terminated by a " eg. ." cccc" prints the string cccc
?TERMINAL	(--- f) Without waiting for a keypress leave a tf if a key is pressed or a ff if no key is pressed.
ASCII	*(---) Compile the ASCII code of the next character in the input stream into the dictionary as a literal if compiling or put it on to the stack if not compiling. Use of this word is intended to make source code listings more readable. eg. KEY ASCII P = IF .... instead of KEY 80 = IF ....
BCOL	(n ---) Set the backdrop colour to colour n.
BL	(--- c) Leave the ASCII code for a space.
CLS	*(---) Clear the screen and, if operating, send a form feed to the printer.
COUNT	(addr1 --- addr2 n) Examine the character string starting at addr1 which has its length stored in its first byte. Leave the start of the string itself as addr2 and its length as n. COUNT is often followed by TYPE.
CNTRL	*(---) Leave the control code equivalent of a character in the same way that ASCII leaves its ascii code.
CR	(---) Transmit a carriage return and linefeed to the selected output device and zero user variable OUT.
EMIT	(b ---) Transmit 8 bit ASCII character b to the selected output device and increment user variable OUT.
EXPECT	(addr count ---) Transfer characters from the terminal to a buffer starting at addr until a carriage return is sent or count characters have been input. One or more nulls are added to the end of the string.
GETKY	*(--- c) if key down (--- 0) if all keys up Return ASCII of any key pressed, or zero if all keys up, without waiting.
KEY	(---c) Wait for a keypress and put its ASCII code on the stack.
LINE-MODE	*(---) Set input mode to line input.
P!	(b port£ ---) Output byte b to port.
P@	(port£ --- b) Input byte b from port.
PAD	(--- addr) Put the address of the text output buffer on TOS. PAD is a fixed offset from HERE. PAD is a useful short term scratchpad for string handling.
PON	*(---) Couple the line printer device to the console output. This can be done by ^R from the keyboard but this word allows the printer to be controlled from within programs.
POFF	*(---) Turn printer off.
QUERY	(---) Transfer characters into the terminal input buffer (TIB) until either a CR is met or 80 characters have been sent. Zero IN, which is a pointer relative to TIB. When in screen mode the input to TIB the 40 characters read from

the screen line where the cursor was before CR was pressed.

SCREEN-MODE	*(---)	Set input mode to full-screen input.
SHAPE	*(b8 b7 b6 b5 b4 b3 b2 b1 b ---)	Create a new shape for character b using the 8 bit patterns b1 - b8. b1 is the top line of the character pixel grid.
SPACE	(---)	Transmit an ASCII space to the current output device.
SPACES	(n ---)	Transmit n spaces to the current output device.
TCOL	*(for bak ---)	Set text colours to for and bak.
TYPE	(addr count ---)	Type count characters from addr to the current output device.
WORD	(c ---)	Scan the input stream for the delimiter c, ignoring leading occurrences. Transfer the string to the dictionary buffer HERE, put its length in the first byte and add two blanks to the end. If user variable BLK is 0, then the input is from TIB with IN as a pointer. If BLK is non-zero then the appropriate disc memory buffer is used as the input stream.
XCR	*(---)	Send a CR & LF to the current output stream without zeroing user variable OUT.

11.7.2 SINGLE NUMBER INPUT/OUTPUT

.	(n ---)	Output the single number n to the current output device after conversion the the current base and follow it with a space.
£.	*(n ---)	Output the single number n to the current output device using base 10 without altering the value of BASE.
.R	(n1 n2 ---)	Print n1 right-aligned in a field of width n2. There is no trailing blank.

DECIMAL	(---)	Set the current base to 10 decimal.
HEX	(---)	Make 16 decimal the new base.
U.	(u ---)	Output the unsigned integer u according to the current base.

11.7.3 DOUBLE NUMBER FORMATTED OUTPUT.

Several FORTH words are concerned with converting double numbers into formatted character strings suitable for output by TYPE. These are :

<£		Start converting a double number already on the stack.
£		Produce one digit per £. If conversion is already complete, output a leading zero.
£S		Convert the rest of the number without any leading zeroes.
HOLD	(c ---)	HOLD is only valid between <£ and £> and puts character c into the output string at the point indicated by HOLD in the format mask.
SIGN	(n d ---)	Put a leading minus sign on the converted number if n is negative. n is discarded.
£>	(d addr count)	Terminate numeric conversion by dropping d and leaving the address of the character representation of the number and its length in a form suitable for TYPE.

It is important to note that conversion is right to left (or back to front) compared with the mask. For example in base hex:

```
<£ £ £ 2E HOLD £S £>
```

will produce a character string of the converted number with two digits to the right of the decimal place and without leading zeroes.

11.7.4 DOUBLE NUMBER INPUT AND UNFORMATTED OUTPUT

D.	(d ---)	Print the double number d using the current base with a trailing blank.
D.R	(d n ---)	

Print the double number d right-aligned in a field of n without a trailing space.

NUMBER	(addr --- d)	Convert a character string at addr, with its length in the first byte, to a double number on the stack using the current base. Put the position of any decimal point in user variable DPL. If conversion is invalid, issue an error message.

11.8 MANIPULATING THE PARAMETER STACK. The parameter stack is the mechanism by which values are passed to and from FORTH words. An understanding of the manipulation of the parameter stack is fundamental to mastering FORTH.

-DUP	(n --- n) if n1=0 (n --- n n) if n<>0	Duplicate the TOS only if non-zero. Most useful with IF to avoid the need for ELSE DROP if the result is false.
DEPTH	*(--- n)	Return the current depth of (number of values on) the parameter stack. Execution of this word will, of course, increase this depth by 1.
DROP	(n ---)	Discard the single number on TOS.
DUP	(n --- n n)	Duplicate the single number on TOS.
OVER	(n1 n2 --- n1 n2 n1)	Copy the second single number to TOS.
ROT	(n1 n2 n3 --- n2 n3 n1)	Rotate the top 3 numbers on the stack ending with n3 on TOS.
SP!	(---)	Clear the stack by initialising the stack pointer.
SP@	(--- addr)	Place the current address of the stack pointer on TOS.
SWAP	(n1 n2 --- n2 n1)	Swap the top 2 single numbers on the stack.
2DROP	(n1 n2 ---)	Drop the double number (or 2 single numbers) from TOS.
2DUP	(n1 n2 --- n1 n2 n1 n2)	Duplicate the double number (or 2 single numbers) on TOS.

2SWAP	*(d1 d2 --- d1 d2)	Swap the 2 double numbers on TOS.
PICK	*(n --- N)	Copy the n'th number in the stack to TOS. There is no check that the stack actually reaches the depth requested.
.S	*(---)	Print the contents of the stack (tos on the right) without affecting it. If the stack is empty the word Empty is printed.
STON STOFF	*(---)	Turn on or off a continuous display of the stack contents. This stack display is extremely useful when learning FORTH or debugging FORTH words.

11.9 MANIPULATING THE RETURN STACK. The return stack is mainly used by FORTH itself to keep track of the execution path and for loop counters, return addresses etc. It may be used as a temporary store by the programmer as long as anything placed there is removed before the end of the word or loop which put it there.

>R	(n ---)	Transfer the top single number from the parameter stack onto the top of the return stack. Balanced by R>.
I	(--- n)	Copy the current value of the loop counter to TOS within a DO LOOP.
R	(--- n)	Copy the top value on the return stack to top of the parameter stack.
R>	(--- n)	Take the top of the return stack and put it on top of the parameter stack. Balances >R.
RP!	(---)	Initialise the return stack. Mainly for use by FORTH.
RP@	(--- addr)	Place the current address of the return stack pointer on TOS.

11.10 ACCESSING THE MEMORY

!	(n addr ---)	Store single number n at addr. Can be used with a variable as in 1 WARNING !
+!	(n addr ---)	

Add n to the value at addr.

?	(addr ---) Print the value at addr in free format to the current base.
@	(addr --- n) Put the value in addr on TOS.
2!	(d addr ---) 32 bit equivalent of !
2@	(addr --- d) 32 bit equivalent of @
BLANKS	(addr count ---) Fill memory from addr with count blanks.
C!	(b addr ---) Store byte b at addr.
C@	(addr --- b) Fetch byte b from addr.
CMOVE	(from to count ---) Block move memory segment of count bytes. The move is towards high memory.
<CMOVE	*(from to count ---) As CMOVE but move is towards low memory.
ERASE	(addr count ---) Fill memory from addr with count null bytes.
FILL	(addr count b ---) Write count bytes b from addr.
MOS	*(---) Return to MOS. FORTH can be warm-started with Y or cold-started with X.
VPEEK	*(addr --- b) Return byte b from addr in the video RAM bank.
VPOKE	*(b addr ---) Store byte b at addr in the video RAM bank.

11.11 BIT LOGIC

AND	(n1 n2 --- n3) Leave the bitwise AND of n1 and n2 as n3.
OR	(n1 n2 --- n3) Leave the bitwise OR of n1 and n2 as n3.

TOGGLE	(addr b ---) Toggle the contents of addr with bit pattern b.
XOR	(n1 n2 --- n3) Leave the bitwise XOR of n1 and n2 as n3.

11.12 COMPARATORS

0<	(n--- f) Leave true flag if n less than zero, else leave a false flag.
0=	(n --- f) Leave true flag if n equal to zero, else leave false flag.
<	(n1 n2 --- f) If n1<n2 leave true flag, else false flag.
=	(n --- f) If n1=n2 leave true flag, else false flag.
>	(n1 n2 --- f) If n1>n2 leave true flag, else false flag.
U<	(u1 u2 --- f) Compare two unsigned numbers. Useful for memory addresses.

11.13 PROGRAM STRUCTURE

IF ELSE THEN	(f ---) If f is true execute code between IF and ELSE and continue after THEN. If f is false, execute code between ELSE and THEN before continuing after THEN. The ELSE part is optional. IFs can be nested to any depth (FORTH can cope even if the programmer cannot!). ENDIF may be used instead of THEN if desired.
BEGIN UNTIL	Repeatedly execute code between BEGIN and UNTIL until TOS contains a true flag, when execution resumes after the UNTIL. END may be used instead of UNTIL.
BEGIN AGAIN	Execute an endless loop between BEGIN and AGAIN. There are tricks to escape from this sort of loop and BREAK may be placed in it to allow escape with SHIFT-BREAK.
BEGIN WHILE REPEAT	Test TOS at WHILE. If true, execute code between WHILE and REPEAT and resume execution

at BEGIN. If false, skip to after REPEAT.

BREAK
*(---)
Has no effect on the program but forces a keyboard scan so that SHIFT-BREAK can be detected and acted on. It is useful as a debugging aid in large and potentially infinite loops.

DO LOOP
(n2 n1 ---)
Execute code between DO and LOOP and increment n1. Repeat the loop until n1=n2, when flow resumes after LOOP. Note that n2>n1 and that the last value of n2 during execution of the loop will be n2-1.

DO +LOOP
(n2 n1 ---)
As for DO LOOP except that the number on TOS when +LOOP is executed is added to the loop counter. This is like FOR NEXT STEP in BASIC.

LEAVE
Within a DO LOOP structure, LEAVE sets the loop counter to the exit value so that the loop terminates next time LOOP is encountered.

CASE
*(n ---)
The CASE construct enables a value on the stack to be tested and a multi-way branch to be determined from the result. Anything that can be achieved using it could equally well be done with a nest of IF....THENs but the result will be much easier to understand, modify and debug using case. The basic layout of a CASE construction is:

```
   n  CASE
    n1 =OF   words to be executed if n=n1 ENDOF
    n2 <OF   words to be executed if n<n2 ENDOF
    n3 >OF   words to be executed if n>n3 ENDOF
    n4 n5 <OF> words to be executed if n4<n<n5 ENDOF
    etc
    optional default words if all above tests fail
   ENDCASE
```

Note that the final endcase drops n from the stack. If the default case words make use of n they must DUP it first.

11.14 WORDS TO ALLOW ACCESS TO THE DICTIONARY

'
(--- addr)
Used in the form ' ccc to leave the parameter field address (PFA) of compiled word ccc or issue an error message of not found. Pronounced 'tick'. Tick is an immediate definition which means that it executes when encountered during compilation.

-FIND
(--- pfa b tf) if found
(--- ff) if not found
Accept the next space deliminated word in the input stream to HERE and search the CURRENT and CONTEXT dictionaries for it. If found leave parameter field address, byte length and true flag. Else leave just a false flag.

CFA
(pfa --- cfa)
Convert a word's pfa to its cfa.

ID.
(addr ---)
Print word's name from its name field address. If a name has been truncated by a low value in WIDTH then the name is padded out to its true length with underline symbols.

FORGET
Used in the form FORGET ccc to remove word ccc and all more-recently defined words from the dictionary. An error message is issued if the CONTEXT and CURRENT dictionaries are not the same.

LATEST
(--- addr)
Put the name field address of the top word in the CURRENT vocabulary on TOS.

LFA
(pfa --- lfa)
Convert a definition's pfa to its link field address.

NFA
(pfa --- nfa)
Convert a definition's parameter field address to its name field address.

PFA
(nfa --- pfa)
Convert the nfa of a definition to its pfa.

VLIST
List all words in the CONTEXT vocabulary. Terminated by any keypress.

11.15 WORDS TO CONTROL COMPILATION

,
(n ---)
Store n into the next free dictionary location and advance the dictionary pointer by one cell. Pronounced 'comma'.

:
Used in the form : ccc ;
Define the following text up to the next space-delimited semi-colon as a new word ccc

	to be added to the dictionary at compile time. Called a 'colon definition' this is the basic building block of a FORTH program.
<BUILDS DOES>	Used in the form : ccc <BUILDS ..... DOES> .... ; to create a new defining word ccc. ccc is used in the form ccc nnn to produce a word nnn with its compile-time behaviour determined by the code between <BUILDS and DOES> and its run-time behaviour by the code between DOES> and ; DOES> places the word's pfa on the stack.
;CODE	This is the machine code equivalent of <BUILDS DOES> and is used to create new defining words. It will only function if the Assembler is resident. Used in the form: : cccc xxxx ;CODE <assembler mnemonics> C; Instructs FORTH to stop compiling and create a new defining word cccc, then set the vocabulary to ASSEMBLER and assemble the mnemonics which follow ;CODE. xxxx must be an existing defining word such as CONSTANT. When the new defining word cccc is executed at compile time in the form: cccc nnnn it creates a word nnnn with its compile time behaviour defined by xxxx and its run-time behaviour governed by the machine code following cccc. This is advanced FORTH.
ALLOT	(n ---) Add n to the value of the dictionary pointer to reserve space within the dictionary for, say, an array.
C,	(b ---) Same as , but works for a single byte.
COMPILE	Copy the execution address of the word following COMPILE into the dictionary.
CONSTANT	(n ---) Used to define a constant in the form n CONSTANT ccc which sets constant ccc to value n. When ccc is later executed, it puts n onto TOS.
CREATE SMUDGE	Used in the form CREATE ccc Creates a dictionary header for the word ccc with the header SMUDGEd and the CFA pointing to the PFA. Mostly used to define machine-code words, after which SMUDGE must be used to complete the definition. SMUDGE must also be used before a spoiled or only partially compiled word can be FORGETed or found by a later definition.
DEFINITIONS	Sets the CURRENT vocabulary to the same as the present CONTEXT vocabulary.
EXECUTE	(addr ---) Execute the word whose cfa is on TOS. This is useful for vectored jumps. eg: VECTOR @ EXECUTE where VECTOR is a variable declared by the user.
FORTH	Make FORTH the CURRENT vocabulary. An IMMEDIATE definition. All dictionary links eventually chain back to FORTH.
HERE	(--- addr) Leave the next free dictionary address. This is the same as DPL @ . HERE is useful to find the size of an application.
IMMEDIATE	(---) Mark the most recent dictionary definition to be executed rather than compiled when encountered at compile time.
LITERAL	(n ---) If compiling, compile n into the distionary as a 16-bit literal. Intended use is to allow a literal to be calculated at compile time eg: [calculation] LITERAL
SMUDGE	Toggle the smudge bit in the name field of the definition in which it falls. Until the smudge bit has been set, FIND will not detect the word so it can neither be used, VLISTed nor FORGETed. Needed with CREATE.
TASK	A dummy word convenient for use with FORGET to discard application programs.
USER	(n ---) Make a new user variable with offset n into the user-variable area. Must be used with care to preserve existing user variables. Used in the form n USER ccc
VARIABLE	(n ---) Used in the form n VARIABLE ccc to create a variable ccc with initial value n. When ccc is later executed it puts the ADDRESS of the current value onto TOS. Note carefully the

	difference between this and the rules for CONSTANT.
VOCABULARY	VOCABULARY ccc creates a new vocabulary ccc. When ccc is later executed, it makes ccc the CURRENT vocabulary. New vocabularies should be declared IMMEDIATE.
[ ]	Used to control compile-time behaviour. [stops compilation and words up to] are then executed rather than compiled.
[COMPILE]	Forces the following word to be compiled even if it has been marked as immediate.

11.16 USER VARIABLES. User variables are the variables used by the system, but also available for use (with care) by the programmer. They reside in a particular part of the memory and are initialised by copying a reserved area of memory at the start of the Forth code when a cold start is executed. This makes it possible to make permanent changes to the the default values of the user variables which will remain in force even if COLD is executed.

BASE	(--- addr) BASE contains the current value of the base for number conversion for both input and output.
BLK	(--- addr) BLK holds the number of the block being LOADed or zero if the input stream is the keyboard.
CONTEXT	(--- addr) Pointer to the CONTEXT vocabulary.
CSP	(--- addr) Used by the system as a temporary store for the stack pointer during compilation.
CURRENT	(--- addr) Pointer to the CURRENT vocabulary.
DP	(--- addr) Dictionary pointer to the next free byte above the dictionary. Read by HERE and altered by ALLOT.
DPL	(--- addr) Contains number of digits right of decimal point during double number conversion. Otherwise holds default of -1.
FENCE	(--- addr) FORGET is trapped below the address in FENCE.
FLD	(--- addr) Unused but reserved by fig.USA for output field width.
HLD	(--- addr) Temporary store during numeric conversion.
IN	(--- addr) Pointer to current position within current input buffer, either a block or the TIB. IN is updated by WORD and zeroed by QUERY.
PREV	(n --- addr) A user variable containing the address of the most recently accessed memory buffer.
OUT	(--- addr) OUT is incremented by EMIT and zeroed by CR. It can be usefully examined and altered by the programmer. For example, a TAB function can be provided by : TAB OUT @ - SPACES ; (n---).
R£	(--- addr) R£ is used by the EDITOR to record the cursor location. At other times it is available to the programmer.
SO	(--- addr) The initial value of the stack pointer.
SCR	(--- addr) The number of the last screen to be LISTed.
STATE	(--- addr) A flag, which is non-zero to show that compilation is in progress.
TIB	(--- addr) The address of the terminal input buffer.
WARNING	(--- addr) A flag to determine the type of error message. Zero means that errors are reported by number. 1 means that error messages are obtained from the file MESSAGES.FIG. -1 means that the contents of ERRVECT (normally ABORT) will be executed on error. Advanced users may wish to change the contents of ERRVECT to provide customised error handling.
WIDTH	(--- addr) WIDTH determines the number of significant characters in dictionary names. WIDTH has an initial value of 31 and may be altered between

1 and 31. The smaller the value of WIDTH the less dictionary space is required, but the greater the chance of ambiguity.

11.17 CONSTANTS

0 1 2 3	(--- n) These small numbers have been defined as constants to recognise the frequency of their use and speed execution.
B/BUF	(--- n) Bytes per disk buffer.
B/SCR	(--- n) Blocks per screen.
C/L	(--- n) Characters per logical screen line.
FCB	(--- 5CH) A constant containing the address of the default Tatung/Xtal DOS File Descriptor Block.
FIRST	(--- addr) Place the address of the lowest-numbered buffer on TOS.
NEXT	This is the re-entry point for all definitions. It is not directly executable. All machine code definitions must end with a jump to NEXT. For convenience NEXT is defined as a constant returning 0145H.
USE	(--- addr) A variable containing the address of the next buffer to use, this being the least recently accesses one.

11.18 OPERATING SYSTEM WORDS. Because much of the FORTH language is written if FORTH there are a number of words in the dictionary which are only there as building blocks towards the useable definitions. Many of these words will be of little direct value to the programmer and their use may be fraught with danger. Some, however, are very useful at times and their Glossary definitions are included here both for completeness and as an aid to understanding the inner workings of FORTH.

!CSP	Save the stack position in CSP. Used as part of compiler security.
+BUF	(addr1 --- addr2 f) Advance the disc buffer address addr1 to the address of the next buffer addr2. Flag f is false if addr2 is the buffer presently pointed to by the variable PREV.
+ORIGIN	(n --- addr) Leave the memory address n bytes above the origin. This is used to alter boot up default values of some USER variables.
AUTOSTART	*(addr ---) Cause a jump to addr on coldstart.
BACK	(addr ---) Calculate the backward branch offset from HERE to addr and compile it into the dictionary. Used by the compiler to calculate jumps for loops and branches.
CKFILE	*(---) Used by COLD. Checks the console command line for a filename. If found the CREATEFILE is used to access it otherwise a warning is issued.
COLDJUMP	*(--- addr) A variable used to hold the cold start vector. It is altered by AUTOSTART.
CLOSE	*(--- code) Closes the file specified by the default Xtal DOS file descriptor block.
CLRFCB	*(---) Clears the default FDB with blanks.
DOSCALL	(DE C --- A) Calls Tatung/Xtal Dos, passing arguments from the stack to the Z80 registers shown and returning any message.

COLD	Reinitialise FORTH and discard any compiled words not in the basic FORTH.
CREATEFILE	*(---) Open the file specified in the default FDB if possible otherwise make the file and issue a warning (new file).
DEF	*(--- addr) A variable containing the current default drive number.
DRIVE?	*(---) Checks the filename in NAMEBUFF for a colon as the second character. This indicates the presence of a drive identifier and the FDB is adjusted to reflect this.
ENCLOSE	The text scanning primitive used by WORD.
ERRDMA	*(---) Set the disc buffer to the default DMA at 80H
ERREAD	*(--- code) Read the selected record from MESSAGES.FIG
ERRFCB	*(--- addr) A string variable used as a FDB for MESSAGES.FIG
ERRVECT	*(addr ---) Cause a jump to addr on error if WARNING is -1
EXT	*(---) Insert the file extension FIG into the FDB.
GETDEF	*(---) Interrogate Tatung/Xtal Dos and store default drive in DEF. (used by COLD).
GETNAME	*(---) Input a word from the console and move it to HERE.
INTERPRET	The outer text interpreter. INTERPRET examines the input stream. Each identified word is first sought in the CONTEXT and CURRENT dictionaries. If this search fails an attempt is made to convert it to a number using the current BASE (a double number is generated if it contains a decimal point). If this too fails an error message is generated and scanning stops.

MAKE	*(---) Makes the specified file. Error 11 is displayed if the directory is full.
MOVENAME	*(---) Moves a string from HERE to a reserved area of memory used as a buffers for filenames.
NAME-PAD	*(--- addr) Leave the address of a scratchpad used as a store for a filename during its conversion to a File Descriptor Block.
NEXTNAME	*(---) Transfers the next word in the input stream to HERE.
OPEN	*(--- dircode) Opens the file specified in the default FDB at 5C hex.
OPENERR	*(---) Look for MESSAGES.FIG on drive 0. If found then open it and set WARNING to 1. Otherwise set WARNING to 0.
PUTNAME	*(---) Transfer a filename from HERE to FDB accounting for any possible drive letter and adding the extension .FIG if an extension was not present.
QUIT	Clear the return stack, stop compiling, return control to the terminal and do not issue a message. QUIT can be used as an exit from a complex nest of loops and to suppress FORTH's normal response 'ok'.
R/W	(addr block f ---) The disc i/o primitive. If input of an unwritten disc block is requested then blanks are assumed. f is 0 for read, 1 for write. All disc i/o uses the file currently described by the default FDB.
READRAND	*(--- code) Read a random record from a file specified in the default FDB.
RESET	*(---) Resets the disc system.
SETBLOCK	*(blk ---) Insert a record number into the FDB ready for

a disc read or write.

SETDEF	*(---) Set drive in DEF as default drive.
SETERR	*(n ---) Select record number n of MESSAGES.FIG ready for ERREAD.
SETDMA	*(addr ---) Sets the address which Xtal DOS will use for a disc buffer (often called DMA) to addr.
TRAVERSE	(addr1 n --- addr2) Move across the name field of a fig.FORTH variable length name field. addr1 is the address of either the length byte of the name or its last letter. If n=1 then motion is towards high memory; if n=-1 then motion is towards low memory. addr2 is the address of the other end of the name.
WRITERAND	*(--- code) Write a random record. Error 6 displayed if the disc is physically full. The FDB must be fully set up to describe the record being written. The write will fail and an error message be isued if the current file is locked.
XFERNAME	*(---) Moves a filename to the FDB from NAME-PAD (the area of memory used as a filename scratchpad.)

12 GLOSSARY OF ADDITIONAL WORDS SUPPLIED AS SOURCE CODE

12.1 FLOATING POINT ARITHMETIC. Words to support floating point arithmetic are contained in the file FLOAT.FIG. Also in this file are trancendental functions provided by a Taylor series. The entire floating point package is written if FORTH and forms a good demonstration of the extensibility of the language. Floating point numbers are represented by a double and a single number and therefore occupy 3 stack locations. The single number is a signed exponent and is TOS when a floating point number is on the stack. The double number is a signed mantissa and is scaled so as to always lie in the range +-1.

12.1.1 Floating Point Defining Words

FVARIABLE	Used in the form f£ FVARIABLE cccc creates a storage location 6 bytes long and initialised to f£. Execution of cccc returns the address of the first byte of this storage location.
FCONSTANT	Used in the form f£ FCONSTANT cccc creates a word cccc which, when executed, places f£ in floating point format on the stack.

12.1.2 Stack Manipulation and Memory Access.

F!	(f£ addr ---) Store f£ at address addr. Warning 6 bytes are required. (See FVARIABLE)
F@	(addr --- f£) Retrieve the 6 bytes from address, interpret as a floating point number and place on the stack.
FDROP	(f£ ---) Drop the fp number f£ from the stack. FDROP actually just drops the top 3 stack contents regardless of whether it was really a fp number or not.
FDUP	(f£ --- f£ f£) Duplicate the fp number on TOS.
F2DUP	(f1£ f2£ --- f1£ f2£ f1£ f2£) Duplicate the two fp numbers on the stack.
FROT	(f1£ f2£ f3£ --- f2£ f3£ f1£) Rotate the top 3 fp numbers bringing the third to TOS.
FSWAP	(f1£ f2£ --- f2£ f1£) Swap the top two fp numbers on the stack.

12.1.3 Floating Point Number Input and Output.

S->F	(n --- f£) Convert the single number on the stack to fp format.
D->F	(d --- f£) Convert the double number on the stack to fp format.
F.	(f£ ---) Output the floating point number on the stack to the current output device in scientific format.

F£IN	(--- f£) Suspend program execution and await input from the keyboard and convert it to a fp number. Any sensible numeric input is accepted including scientific notation. In this case the exponent sign 'E' must be preceeded and followed by a space. A leading + is not permitted. Any illegal input generates an error message. Examples of valid inputs are: 123 -1.23 12.5E -3 0.0001
E	E is used to allow fp numbers to be entered directly on to the stack. It is used after a single or double number and followed by an exponent. FORTH recognises the E and converts this input to a fp number which is placed on the stack. eg: 123 E 0 <cr> ok F. <cr> 1.23 E 2 ok

12.1.4 Arithmetic and Comparators.

F*	(f1£ f2£ --- f3£) Leave the fp product of two fp numbers on the stack.
F/	(f1£ f2£ --- f3£) Leave the result of f1£ / f2£ on the stack. A check that the stack depth is adequate for this operation is made and error message results if the stack is not at least 6 entries deep.
F+	(f1£ f2£ --- f3£) Leave the sum of f1£ and f2£ on the stack.
F-	(f1£ f2£ --- f3£) Leave the difference f1£ - f2£ on the stack.
FMOD	(f1£ f2£ --- f3£) Leave the modulus of f1£ / f2£ on the stack.
FMINUS	(f£ --- -f£) Negate the fp number on TOS.
FINT	(f1£ --- f2£) Leave the integral part of f1£ still in fp format.
F<	(f1£ f2£ --- f) Leave tf if f1£ < f2£.
F<=	(f1£ f2£ --- f) Leave tf if f1£ <= f2£.
F>=	(f1£ f2£ --- f) Leave tf if f1£ >= f2£.
F>	(f1£ f2£ --- f) Leave tf if f1£ > f2£.
F<0	(f£ --- f) Leave tf if f£ is negative.

12.1.5 Floating Point Constants. A number of floating point format constants are present in the system. These were mainly implemented as an aid to writing the floating point package but are, of course, available to the user. Examples are F1, F10, F.1 etc. Each returns the number associated with its name. PI is also available when the trignometric words have been loaded.

12.1.6 Trigs and Trancendentals. The latter part of the FLOAT.FIG file contains a number of routines to calculate trig values.

SINE	(f£ --- f£1) Leave the SINE of f£ (expressed in degrees).
COSINE	(f£ --- f£1) Leave the COSINE of f£ (expressed in degrees).
TANGENT	(f£ --- f£1) Leave the TANGENT of f£ (expressed in degrees).
FSQR	(f£ --- f£1) Leave the square root of f£. An error results if f£ is negative.
ARCSIN-RAD	(f£ --- f£1) Leave the arc sine of f£ expressed in radians.
ARCSIN-DEG	(f£ --- f£1) Leave the arc sine of f£ expressed in degrees.

12.2 FUNCTION KEY PROGRAMMING. Words to handle function key programming are contained in the file FUNKEYS.FIG

CLRKEYS	(---) Set all function keys to produce null bytes.

FUNKEY	(n ---) Used in the form: n FUNKEY "string" where string is a quote delimited character string. This sets function key n to produce the string provided. n is in the range 0-15. Keys may be redefined at will and may be terminated with a c/r symbol produced by GRAPH-ENTER in which case they execute immediately. The leading quote is optional if no leading spaces are required in the string. Function key definitions may be entered directly from the keyboard or loaded from disc blocks.
1KEY	(n ---) Print the string associated with function key n.
KEYLIST	(---) List the current settings of all the function keys.

12.3 GRAPHICS EXTENSIONS. Words to exploit the Einstein Graphics are in the file GRAPHICS.FIG

PLOT	(x y ---) Set pixel at x,y to the current graphics foreground colour.
UNPLOT	(x y ---) Set pixel at x,y to the current graphics background colour.
POINT	(x y --- f) Return tf if pixel at x,y is foreground or ff if background.
DRAW	(x1 y1 x2 y2 t ---) Draw a line from x1,y1 to x2,y2 of type t. t is in the range 0-5 and produces the same line types as BASIC.
TO	(x y t ---) Draw a line of type t from the most recently drawn point to x,y.
POLY	(n x y xs ys t ---) Draw a polygon with n sides and line type t. The polygon will be centred at x,y and will fit in an ellipse with horizontal and vertical half dimensions of xs and ys.
ELLIPSE	(x y xs ys t ---) Draw an ellipse centred at x,y using line type t. The horizontal and vertical size of the ellipse is governed by xs and ys.
ORIGIN	(x y ---) Offset the graphics grid origin to a point x,y.
COLFILL	(x y ---) Fill a shape surrounding point x,y with colour. If x,y is background then the current foreground colour is used; if x,y is foreground then background colour is used.
GCOL	(for bak ---) Set the graphics colours to for (foreground) and bak (background).
SECTOR	(f n1 n2 x y xs ys t ---) This word allows part of an ellipse to be drawn. If f is true then when the sector has been drawn lines will be drawn from the centre to the perimeter. If f is false only the perimiter will be drawn. n1 is the start angle in degrees (anticlockwise from the 3 o'clock position) where drawing will commence. n2 is the angle were drawing will cease. The rest of the parameters are as for the ellipse command.
PARTPOLY	(f n1 n2 n x y xs ys t ---) This is similar to SECTOR but part of a polygon is drawn. Again f is a flag which is false if only the perimeter is to be drawn or true if lines are to be drawn from the centre of the polygon. n1 and n2 are the start and finish angles in degrees. Note that a whole number of sides is always drawn. The rest of the parameters are as for the POLY command. As well as drawing parts of polygons PARTPOLY can be used to draw a complete one at any desired angle on the screen. eg. 4 100 100 50 50 0 POLY will draw a regular polygon with 4 sides at co-ordinates 50,50. The polygon will be orientated as a diamond. By contrast: 0 45 405 4 100 100 50 50 0 PARTPOLY will draw a similar 4 sided figure but with its sides parallel to the screen edges. Note that the finish angle in this case is 360+start_angle.

12.4 SOUNDS. Words to assist with programming the sound generator are in the file SOUNDS.FIG

PSG! (b n ---)
Store byte b in register n of the PSG chip.

FREQA (n ---)
Set channel A to n Hz.

FREQB (n ---)
Set channel B to n Hz.

FREQC (n ---)
Set channel C to n Hz.

FREQN (n ---)
Set the noise frequency to n Hz.

ENABLE (b ---)
Send byte b to register 7 of the PSG chip (enable register) after adjusting it to ensure that the keyboard will not be disabled.

CHANNEL (cn bn an ct bt at ---)
The six parameters are flags which are true to turn the selected channel on or false to turn it off. cn, bn and an couple the noise generator to channels a, b and c respectively. ct, bt and at select the tone channels c, b and a respectively.

VOLA (n ---)
If n is in the range 0-15 select a fixed volume on channel A proportional to n. If n is over 15 then couple channel A to the envelope generator.

VOLB (n ---)
If n is in the range 0-15 select a fixed volume on channel B proportional to n. If n is over 15 then couple channel B to the envelope generator.

VOLC (n ---)
If n is in the range 0-15 select a fixed volume on channel C proportional to n. If n is over 15 then couple channel C to the envelope generator.

ENVELOPE (n ---)
Set the envelope period to n/100 seconds.

HAAC (f1 f2 f3 f4 ---)
Set the hold, alternate, attack and continue bits in register 13 of the PSG chip according to the flags f1 to f4 respectively. See pages 296-299 of the BASIC reference manual for details.

12.5 SPRITES. Words to allow the creation and movement of sprites are contained in the file SPRITES.FIG.

VDP! (b reg ---)
Store byte b in register reg of the VDP chip.

MAG (n ---)
n is in the range 0-3 and has the same effect as in BASIC.

SPRITE (S x y C N ---)
Sprite command works as for BASIC. X co-ordinates are in the range -33 to 255. Y co-ordinates are in range -33 to 191.

SPRITE-OFF (S ---)
Switch off sprite number S.

SPRITES-OFF (---)
Switch off all sprites.

SPRITEVAR (S C N)
A defining word used in the form:

```
S C N   SPRITEVAR   name
```

This creates a sprite that is identified by name and requires only its x,y co-ordinates on the stack to display it. Eg:

```
5  15  176  SPRITEVAR  ALIEN
( create sprite variable )

100 100 ALIEN   ( display sprite at 100,100 )
```

12.6 UTILITIES. A number of useful, general-purpose, definitions are supplied in the file UTILITY.FIG. The words supplied are:

12.6.1 SCREEN 1. RANDOM NUMBER GENERATOR.

CHOOSE (u1 --- u2)
Return a random number in the range 0 < u2 < u1.

12.6.2 SCREEN 2. MACHINE CODE CALL.

CALL (addr ---)
Make an unconditional jump to a machine code subroutine at addr. Note that FORTH will crash if register BC is not preserved.

12.6.3 SCREEN 3. ARRAY DEFINITIONS.

CARRAY A defining word to create a character or byte array. Used in the form:

```
n CARRAY name
```

creates an array called name of length n bytes. Subsequent use of name in the form:

```
n name
```

leaves the address of the n'th byte of the array. Note that there are no checks for array boundary overflow.

ARRAY Similar to CARRAY above, ARRAY creates an array of 16 bit cells suitale for the storage of integers.

STRING A defining word used in the form:

```
STRING name "quote delimited literal string"
```

Subsequent use of name leaves the address of the literal string which has its length in its first byte. Eg.

```
STRING ERRMESS "Operator brain failure"
ok     ( string now defined )

ERRMESS COUNT TYPE
Operator brain failure
ok     ( string output )
```

12.6.4 SCREEN 4. NUMERIC INPUT & ADC PORTS.

D£IN (--- n)
Wait for input from the console and convert it to a double length number which is placed on the stack. This allows for prompted numeric input. An error will result if the keyboard input cannot be converted to a legal double number.

£IN (--- n)
Wait for input from the console and convert it to$ a single length number which is placed on the stack. This allows for prompted numeric input. An error will result if the keyboard input cannot be converted to a legal number.

ADC (chan --- n)
Return value between 0 and 255 from analogue channel chan. No noise smoothing is carried out.

BTN (port --- f)
Returns tf if joystick button on ADC port is pressed. Arbitrary results will be returned if port is not 1 or 2.

12.6.5 SCREEN 5. DIRECT KEYBOARD SCAN.

1COMP (b1 --- b2)
Leave the one's complement of b1 as b2.

KSCAN (row --- col)
Directly scan the keyboard matrix row or rows selected by the bits set in row. Return a byte consisting of zeroes for unpressed keys in those rows and ones for pressed keys. This word enables multiple and continuous keypresses to be detected and is primarily of use when writing arcade games. Note that this definition uses the word PSG! which is also defined in the SOUNDS.FIG file. The keyboard matrix is laid out thus:

					COL				
		1	0	0	0	0	0	0	0
		0	1	0	0	0	0	0	0
		0	0	1	0	0	0	0	0
		0	0	0	1	0	0	0	0
		0	0	0	0	1	0	0	0
		0	0	0	0	0	1	0	0
		0	0	0	0	0	0	1	0
		0	0	0	0	0	0	0	1
		128	64	32	16	8	4	2	1
ROW									
00000001	1	ESC	SP	CR		F7	F0		
00000010	2	0	1/2	curs	−	1/4	P	O	I
00000100	4	F5	9	curs	3/4	:	;	L	K
00001000	8	F4	^	=	DEL	8	/	.	,
00010000	16	F3	1	2	3	4	5	6	7
00100000	32	F2	Q	W	E	R	T	Y	U
01000000	64	F1	A	S	D	F	G	H	J

10000000 128 FO Z X C V B N M

12.6.6 SCREEN 6. INTERNAL CLOCK WORDS

TIME=	(---) Used in the form TIME= nnnnnn where nnnnnn is a six digit ascii string equivalent to the time at which the internal clock is to be set.
GET-TIME	(---) Wait for keyboard input. Attempt to set clock to the value represented by the string entered.
.TIME	(---) Print the current time on the system clock as a six digit character string.
TIME@	(--- h m s) Read system clock and leave hours, minutes and seconds as three single length numbers on the stack.
TIME!	(h m s ---) Set the system clock from the three numbers on the stack.

12.6.7 SCREEN 7. SERIAL PORT WORDS.

BAUD	(rx tx ---) Set receive baud rate to rx and transmit rate to tx. The permitted range is 75 to 9600 baud.
SERINIT	(bits stops parity ---) Initialise serial port using characteristics on the stack. bits is from 5 to 8. stops is 1 or 2. parity is 0 for no parity, 1 for odd and 2 for even parity.
>SER	(b ---) Send byte b to the serial port. There are no checks whether the port is actually ready to snd another character.
SER>	(--- b) Accept byte b from the serial port. The value is undefined if no character is actually ready.
?SER	(--- b) Examine serial port status. ?SER returns a 3 bit number in the range 0-7. BIT 0 is set if any error is detected (parity etc) BIT 1 is set if the tx port is empty BIT 2 is set if the rx port has a character ready. Eg. To wait until the tx port is empty then send the byte on the stack try: `BEGIN ?SER 2 AND UNTIL >SER`

12.6.8 SCREEN 8. DOUBLE NUMBER EXTENSIONS.

DS*	(d n --- d) Leave the double number product of double number d and single number n.
DS/	(d n --- d) Leave the double number quotient of double number d and single divisor n.
D-	(d1 d2 --- d3) Subtract d2 from d1 and leave double length difference d3.
DO= DO< D= D< D>	These are double number equivalents of the single length number comparators provided in the basic FORTH core.

12.7 DOSTOOLS. Words in the 3 screens of this file allow access to the disc directory, transfer of screens from one file to another and deletion of the current file.

12.7.1 SCREEN 1. DIRECTORY.

DIR	(---) List the directory for the default drive.
DIR0 DIR1 DIR2 DIR3	(---) List the directory for a specific drive.

12.7.2 SCREEN 2. SCREEN SCRATCHPAD.

TOSCRATCH	(n ---) Copy screen n from the current file to a RAM-based scratchpad area.
.SCRATCH	(---) List the current scratchpad contents.
FMSCRATCH	(n ---) Copy the scratchpad contents to screen n of the current file overwriting anything already there.

Note that the current file can be changed between uses of TOSCRATCH and FMSCRATCH. This allows the transfer of screens from one file to another. If only a few lines need to be transferreit may be easier to use the editor PAD and the editor CTRL-C and CTRL-P or CTRL-I commands.

12.7.3 SCREEN 3. DELETE CURRENT FILE.

DELETE-FILE (---)
Delete the current file from the disc. Most useful if you mistype a file name which FORTH then creates for you.

13 ASSEMBLER
=========

13.1 INTRODUCTION. Einstein SuperFORTH includes a machine code assembler using Intel 8080 mnemonics. The assembler is contained in a separate vocabulary called ASSEMBLER. The assembler vocabulary can be viewed by typing ASSEMBLER VLIST. It is not usually necessary to invoke the assembler vocabulary directly since the assembler defining words switch between this and the FORTH vocabulary automatically. There is very little that can be done with the assembler that cannot be done with FORTH words but assembler will, of course, be faster. A good program development technique is, therefore, to first write in FORTH then optimise critical words in assembler.

13.2 NOTATION. Like everything else in FORTH the assembler uses Reverse Polish Notation.

ie. Instead of: MOV A,E we write: E A MOV

This follows the usual FORTH convention of source coming before destination. Jumps and loops are handled in the normal FORTH manner, by use of BEGIN UNTIL etc., rather than by jumps and labels. An example of an assembler definition using a control structure will be found at the end of this section.

13.3 DEFINING WORDS. Assembler definitions are started with the defining word CODE which is the exact equivalent to the colon in a normal FORTH definition. The definition is ended with C; which is analagous to the semi-colon at the end of a normal FORTH definition. It is essential that assembler definitions end with NEXT JMP (a jump to the FORTH inner interpreter) and that register BC is preserved.

13.4 MACROS AND LABELS. The assembler can be extended by defining macros or machine code subroutines. The format is:

MACRO name machine code ;

This creates a new assembler word (name) which can be used in assembler definitions just like the standard assembler vocabulary.

Eg. HEX MACRO IXPOP DD C, E1 C, ;

IXPOP can now be used in assembler definitions as required.

Machine code subroutines are created in the following manner:

LABEL name assembler mnemonics RET C;

Some sort of RET instruction is essential. Such a subroutine can be called from other assembler words with:

name CALL

13.5 MNEMONICS. The assembler mnemonics are listed below with their Zilog Z80 equivalents. r is any 8 bit register or M, rp is a 16 bit register pair.

13.5.1 REGISTERS.

FORTH	Z80	FORTH	Z80
PSW	AF	A	A
B	B or BC	D	D or DE
H	H or HL	M	(HL)
E	E	C	C
SP	SP	L	L

13.5.2 OPERATORS.

FORTH	Z80	FORTH	Z80
data rp LXI	LD rp,data16	data r MVI	LD r,data8
r1 r2 MOV	MOV r2,r1	addr JMP	JP addr
RET	RET	RM	RET M
RP	RET P	RPE	RET PE
RPO	RET PO	RC	RET C
RNC	RET NC	RZ	RET Z
RNZ	RET NZ	addr CALL	CALL addr
addr CM	CALL M,addr	addr CP	CALL P,addr
addr CPE	CALL PE,addr	addr CPO	CALL PO,addr
addr CC	CALL C,addr	addr CNC	CALL NC,addr
addr CZ	CALL Z,addr	addr CNZ	CALL NZ,addr
addr LDA	LD A,(addr)	addr STA	LD (addr),A
addr LHLD	LD HL,(addr)	addr SHLD	LD (addr),HL
data CPI	CP data	data ORI	OR data
data XRI	XOR data	data ANI	AND data
data SBI	SBC A,data	data SUI	SUB data
data ACI	ADC A,data	data ADI	ADD A,data
port IN	IN A,(port)	port OUT	OUT (port),A
n RST	RST n	rp DCX	DEC rp
rp INX	INC rp	r DCR	DEC r
r INR	INC r	B LDAX	LD A,(BC)
D LDAX	LD A,(DE)	B STAX	LD (BC),A
D STAX	LD (DE),A	rp PUSH	PUSH rp
rp POP	POP rp	rp DAD	ADD HL,rp
r CMP	CP r	r ORA	OR r
r XRA	XOR r	r ANA	AND r
r SBB	SBC A,r	r SUB	SUB r
r ADC	ADC A,r	r ADD	ADD A,r
CMC	CCF	STC	SCF
CMA	CPL	DAA	DAA
XCHG	EX DE,HL	XTHL	EX (SP),HL
SPHL	LD SP,HL	PCHL	JP (HL)
RAR	RRA	RAL	RLA
RRC	RRCA	RLC	RLCA
EI	EI	DI	DI
HLT	HALT	NOP	NOP

13.5.3 TESTS.

0<	leaves true flag if M flag set
PE	leaves true flag if P flag set
CS	leaves true flag if C flag set
0=	leaves true flag if Z flag set
NOT	complements flag on stack

13.5.4 STRUCTURES.

IF ELSE THEN (or ENDIF)
BEGIN AGAIN
BEGIN UNTIL
BEGIN WHILE REPEAT   all these work in a similar manner to their FORTH equivalents.

13.6 AN EXAMPLE. An assembler definition for the FORTH word FILL (addr count byte ---) which fills an area of memory with the value byte.

```
CODE FILL                  ( start and name definition )
B H MOV   C L MOV          ( get ready to preserve BC )
D POP                      ( byte in E )
B POP                      ( count in BC )
XTHL                       ( addr in HL, BC saved on stack )
BEGIN                      ( start of loop )
 E A MOV                   ( byte in A )
 A M MOV                   ( byte in addr pointed to by HL )
 H INX                     ( next address )
 B DCX                     ( decrement count )
 C A MOV   B ORA   0=      ( check if count zero )
UNTIL
B POP                      ( restore BC from stack )
NEXT JMP  C;               ( return to FORTH interpreter )
```

Incidently there is a major bug in this definition. If it is called with count=0 it will fill 64K RAM locations with byte! This could be corrected by using the BEGIN WHILE REPEAT structure instead of BEGIN UNTIL. It would be necessary to invert the count test with NOT if this structure is used. ie.

```
BEGIN ............ 0= NOT WHILE ............ REPEAT
```

14 COMMON FORTH ERRORS
====================

14.1 FORTH has minimal error trapping built in. Its nature allows the programmer to make whatever checks are necessary and make his own tradeoff between speed and robustness. Programs can be developed with checks in place and these removed later once all is well. Because of this design philosophy crashes will be more frequent that with BASIC. It is good practice to FLUSH all major amendments to code before testing in case a major crash occurs and the editing work is lost. Some common errors are detailed below.

14.1.1 VARIABLES. Remember that variables return the address where the value is stored, not the value itself. Contrast this with CONSTANT which actually returns a value.

14.1.2 STACK OVERFLOW/UNDERFLOW. Stack range errors are only detected by FORTH when control returns to the user console. If the stack empties during a long loop then execution will continue but arbitrary values will be read from the stack with unpredictable results. If the stack overflows enough to meet the dictionary then a crash will result. Such effects can be avoided by testing each word seperately and examining the stack effects with .S or by use of STON/STOFF.

14.1.3 ARRAYS. Most FORTH arrays leave an address in the same way as variables. Normally no range checks are made and writing outside the array boundary will arbitrarily alter the dictionary contents with, usually, disasterous consequences.

14.1.4 BASE. Be aware of what base you are in. Some words change the number base. This may be worth checking if a tested word suddenly misbehaves. Note that the command BASE ? or BASE @ . always give 10 because the current base is used to print the result. To find the truth use BASE @ £.

14.1.5 INFINITE LOOPS. There is no overriding BREAK function in FORTH. Infinite loops are, therefore, fatal. A break control must be explicitly programmed by the user if needed. BEGIN......AGAIN is an intentional infinite loop which must only be included in a fully debugged application. Note however that:

```
   BEGIN ............0 UNTIL and BEGIN...1 WHILE......REPEAT
```

are also infinite loops.

14.1.6 AMNESIA FAILURE. (If you can't forget a word.) If an error occurs during the compilation of a word (either directly from the keyboard or using LOAD) the word's name will be visible if VLIST is used but will not be found if FORGET is used. This is because a word's name is SMUDGED until it's definition is complete. It is this feature that allows a name to be redefined in a later definition. To remove the part compiled word type: SMUDGE FORGET name.

15 MODIFYING FORTH

15.1 Default values of the FORTH user variables can be altered by the user. The way this is is done and the location of other useful flags and pointers are described below.

15.2 Computer/processor name: The word .CPU prints the computer or processor name. The name is stored as a base 36 double integer at address 0122H. Some variations are:

```
   HEX 0 122 ! B250 124 !          gives Z80
   HEX 19 122 ! 134A 124 !         gives Z80A
   HEX 5 122 ! B320 124 !          gives 8080 etc.
```

15.3 Initial values of USER variables: the word +ORIGIN is used to access the area of memory where the default valuse of user variables are stored. A table of values and +ORIGIN offsets follows:

Offset	Present value	Function
8H	1	Fig release number
9H	1	Fig revision number
0AH	2	User revision number
0CH		Address of top word in dictionary on boot up
0EH	7FH	Backspace character
18H	1FH	Initial WIDTH
1AH	0	Initial WARNING
1CH		Initial FENCE
1EH		Initial dictionary pointer
20H		Initial vocabulary link

For example you can change the backspace character to ^D by:

```
   HEX 4 0E +ORIGIN !   <CR>
```

16 SAVING ENLARGED VERSIONS OF FORTH

16.1 To save a compiled application it is necessary to first compile the source code into the dictionary then alter the boot up parameters to reflect the changed dictionary size and lastly exit to Tatung/Xtal Dos and use SAVE to copy the application to disc as a .COM file. When run from Tatung/Xtal Dos the new program will still give the FORTH header but the range of words will be extended to include all those compiled. This method can be used either to save an enlarged version of FORTH containing additions and utilities or to create an applications program. The sequence of commands needed to save a compiled application is as follows:

```
DECIMAL
LATEST 12 +ORIGIN ! (top name field address)
HERE 28 +ORIGIN !   (initial fence)
HERE 30 +ORIGIN !   (initial dictionary pointer)
( to see how many blocks need to be saved use: )
HERE 256 / 1+ .
```

These commands have been made into a FORTH word SAVE If a new vocabulary has been declared in the application the vocabulary linkage pointers must also be updated before saving the compiled application. Use the sequence:

```
' vocabname 6 + 32 +ORIGIN !
```

16.2 AUTOSTART APPLICATIONS. A compiled application can be made to autostart by the sequence:

```
' NAME AUTOSTART
```

where NAME is the word to be executed on cold starting the system. If NAME is not a closed loop then it must end with QUIT or ABORT. Use of AUTOSTART and ERRVECT together with closed loop programs enables applications to be written in which the FORTH kernal remains invisible and protected. As an alternative to writing an application program as a closed loop it is possible to patch the directory so as to limit the range of commands available to the user. One of these must, of course, be DOS The method is as follows:

Compile a dummy word (eg : BOUNDARY ;) above the application
Redefine all the desired commands above this boundary.
Set the LFA of the dummy word to 0. eg. 0 ' BOUNDARY LFA !

This stops dictionary searches at the boundary word but allows use of all the subsequent definitions.

17 MEMORY MAP.

```
LIMIT          -----------------       EC00H
               I               I
               I Disk Buffers  I
               I               I
FIRST          -----------------       DB80H
               -----------------
               I User          I
               I      Area     I
UP             -----------------
R0             -----------------       DB48H
               I Rtn Stk       I
               I I             I
               I V         ^   I
               I     Term Buff I
RP             -----------------       TIB
S0             -----------------       DAA0H
               I Stack I       I
               I       V       I
SP             -----------------


               Spare RAM



               -----------------
               I Text Buffer   I
               -----------------       PAD
                 WORD Buffer
DP             -----------------       HERE
               I               I
               I               I
               I DICTIONARY    I
               I               I
               I               I
               -----------------

               -----------------
               I Boot Data     I
ORIGIN         -----------------       100H
```

18 DICTIONARY ENTRY STRUCTURE.

```
  MEMORY              EXPLANATION
  CELLS
  -------
->! 84H ! name field address (nfa) contains bits as follows:
^ -------         b4-b0=length, b5=smudge, b6=immediate, msb=1.
^ !  F  !)
^ -------)
^ ! R ! )         Ascii characters making up name up to
^ ------- )       limit imposed in WIDTH.
^ ! E   ! )
^ ------- )
^ !D+80H!         Last character has top bit set.
^ -------
^ ! lfa !         Link field.  Points to previous word's nfa.
^ -------
^ ! cfa !         Code field.  Points to machine code for this
^ -------         definition.  This could point 2 bytes forward to
^                 the pfa for a machine code definition or to
^                 machine code elswhere in the case of a colon
^                 definition, variable etc.
^ -------
^ ! addr! pfa     Parameter field.  Usually a list of addresses of
^ -------         other FORTH words' cfas  or actual machine code.
^ ! addr!
^ -------
^ ! addr!
^ -------
^ ! addr!        Last addr of colon definition would point to special
^ -------        definition ending word. Machine code definition would
^                have a jump to NEXT here.
^ -------
^ ! 85H !        Next word starts here
^ -------
^ !  P  !
^ -------
^ !  E  !
^ -------
^ !  T  !
^ -------
^ !  E  !
^ -------
^ !R+80H!
^ -------
--! lfa !
  -------
  ! cfa !
  -------
  ! pfa !       etc.
  -------
     !
     V
INCREASING MEMORY
```

19 FURTHER SOURCES

19.1 BIBLIOGRAPHY.

The FORTH bibliography is large and rapidly growing. The books mentioned below are those that have been seen by the author. Other quality titles may have been overlooked.

Starting FORTH
Leo Brodie

This is the definitive introduction to the language and is probably one of the best written computer books on any subject. It details the FORTH.Inc implementation which differs in a number of important and potentially confusing ways from fig.FORTH. It is nonetheless a splendid introduction to the language.

FORTH PROGRAMMING
Scanlon

This is largely a list of well commented source examples. The fig.FORTH dialect is covered. Useful.

Systems Guide to fig.FORTH
Ting

Probably the definitive reference covering the inner working of fig.FORTH. Expensive but detailed.

FORTH Encyclopaedia
Baker and Derrick

A complete programmer's manual for the language. Comparisons of fig.FORTH and FORTH79 are presented.

Threaded Interpretive Languages
Loeliger

Explains the theory of FORTH-like languages using the Z80 processor for the examples.

All About FORTH
Haydon

An extensive glossary giving examples in fig.FORTH, FORTH79 and MVP FORTH. Good value and an invaluable aid to those converting programs from other FORTH dialects. Explains the inner mechanisms by means of a FORTH inner interpreter written in FORTH.

FORTH Implementation Manual and Source Listings

These FORTH Interest Group US publications are of great value to those who whish to explore the inner workings of FORTH in great detail. They are available from a number of sources including the UK FORTH Interest Group.

19.2 FORTH INTEREST GROUP UK. Users may find it useful to join the FORTH Interest Group. Members receive a magazine and have access to a comprehensive library and reduced price books. At the time of writing (November 1984) the membership secretary is:

Roger Firth
7 Wyndham Crescent
Woodley
Reading

20 ERROR MESSAGES

20.1 If file MESSAGES.FIG is available on drive 0 then text error messages will usually be available. If not then numeric errors will be issued instead. These have the following meanings:

1	Empty Stack
2	Dictionary Full
3	Has Incorrect Address Mode
4	Isn't Unique
5	Stack Depth Inadequate
6	Disc Range Error
7	Full Stack
8	Disc Error
9	Lowest Block Available is 1
10	No File Open
11	Disc Directory Full
12	Divide by Zero
13	Negative Square Root
14	File is Locked - Read Only!
15	Einstein fig.FORTH
16	
17	Compilation Only, Use in Definition
18	Execution Only
19	Conditionals not Paired
20	Definition not Finished
21	In Protected Dictionary
22	Use Only When Loading
23	Off Current Editing Screen
24	Declare Vocabulary