48/80 FORTH

FOR 48K AND 80K ZX SPECTRUM

INTRODUCTION AND TECHNICAL MANUAL

48/80 FORTH

from

EAST LONDON ROBOTICS LIMITED

48/80 FORTH

INDEX

PREFACE

ABOUT THIS MANUAL

If you are new to Forth, it is ESSENTIAL that you refer to one of the books recommended in Appendix 5. This manual is primarily a formal definition of the words in the Forth vocabulary. It is not a text book.

We have, however, written an extensive introduction for beginners. This is of necessity over-simplified and in-complete, but it is hoped that it will provide enough of an insight into the language for you to understand most of the word-definitions in the Glossary, and to get started on writing some programs.

ABOUT THE TAPE

Each tape includes both a 48k version and an 80k version of 48/80 Forth. On one side of the tape the 48k version precedes the 80k version. On the other side they are the other way round.

The order of programs on each side of the tape is therefore as follows:

SIDE 1	SIDE 2
48k Forth Compiler	80k Forth Compiler
48k Forth Editor	80k Forth Editor
80k Forth Compiler	48k Forth Compiler
80k Forth Editor	48k Forth Editor

To load either version of the compiler, type LOAD ""

The Editor can only be loaded when the compiler is already running. To load the Editor, type

LOADSCREENS EDITOR (No inverted commas needed)

When you have loaded the compiler, you should get the sign-on message as follows:

48/80 FORTH v 1.1 (48k VERSION)
or 48/80 FORTH v 1.1 (80k VERSION)

followed by the copyright message and a flashing C cursor. If you press enter, the system will respond with OK.

USE OF THE SPECTRUM KEYBOARD

The cursor types which are needed for Spectrum 48/80 Forth are C,L,G and E. The E cursor should only be used for square brackets **[** and **]**. (Keywords will not appear in E mode).

All Forth words must be typed in letter by letter.

USE OF THE ZX PRINTER

The ZX Printer can be selected by pressing Caps Shift and 1. If you do this, all output to the screen will also be printed out on the Printer.

To get back to screen output only, (i.e. to turn the printer off), press Caps Shift and 1 again.

BREAKING INTO THE PROGRAM

The normal Break function is available using Caps Shift and Space. Note, however, that the ability to break into your own Forth programs is dependent on your own programming - if you write an endless loop with no break-in provisions, you may have to pull the plug and reload.

The Break function will return you to BASIC in certain circumstances. This will happen in particular when

- you are saving to tape
- you are loading from tape
- you are using the ZX Printer

If you are in BASIC and wish to return to Forth, enter one of the following:

RANDOMISE USR 24576	This performs a "cold start", deleting any additions you have made during the session.
RANDOMISE USR 24580	This gives a "warm start", retaining any additions you have made during the session.

Note: 80k users must ensure that page 1 is selected for these cold and warm starts.

THE 80k SPECTRUM

The 80k version of 48/80 FORTH will work only on a Spectrum fitted with the SP80 64k paged memory expansion produced by East London Robotics Ltd

BEGINNERS INTRODUCTION

THE WORD, THE VOCABULARY AND THE DICTIONARY

Any form of program instruction in FORTH is called a "word". This includes not only such words as DO and THEN, but also punctuation marks such as : ' ! and ;.

These "words" are grouped into a series of "vocabularies". You may have one or more vocabularies in your FORTH system. The name of the vocabulary in your FORTH compiler is "FORTH". Your editor uses a different vocabulary, called "EDITOR". You, as the user, will eventually wish to form your own specialised vocabularies suited to your own particular purposes.

All the words in all the vocabularies which have been loaded onto your Sectrum are held in the "dictionary", which is a specially reserved area of RAM. The words of several different vocabularies may be intermingled. But if you specify which vocabulary you are using, your system will only find the words in that vocabulary, and the vocabularies to which it is linked.

The FORTH compiler (whose vocabulary is called FORTH) has enough words for you to be able to write almost any kind of program.

FORTH is unique in that it allows you to add to the dictionary by defining your own words. The initial FORTH vocabulary is extensive enough to let you define your own words purely in terms of other words which have already been defined.

Not only can you define new words in terms of the words already supplied by the compiler; you can also define further words in terms of the new words you have defined.

FORTH, BASIC AND MACHINE CODE

FORTH is, so to speak, half way between BASIC and machine code. FORTH programs, therefore, do not read like English. The meaning of a simple BASIC program can often be worked out by someone who has never heard of computers. To translate ("interpret") a BASIC program, a very large amount of work has to be done by the microprocessor and the interpretation routines in ROM before the command itself can be carried out. BASIC programs are therefore slow.

Forth is much much closer to machine code than BASIC. So to program successfully, you will have to go some way towards understanding what goes on inside the computer, and learn to think in terms of numbers-logic, rather than in the English-type grammar of a BASIC program. More of this later.

Meanwhile, it is most important to gain some idea of what goes on when you define a new word.

DEFINING A NEW WORD

The word " : " (colon) tells the system that you are about to define a new word.

The colon is always followed by the name of the word to be defined. The name is followed by the definition itself.

The whole definition is terminated with " ; " (semi). If you forget this semi-colon, the system will not know when the definition ends, and will attempt to include whatever comes next. This can cause the system to crash, and you will have to reload.

All FORTH words must be separated by one or more spaces.

When you have written your definition, beginning with : and ending with ; you must press Enter. ONCE YOU HAVE PRESSED ENTER YOU CANNOT GO BACK AND EDIT THE DEFINITION.

This is because the computer has already turned (or tried to turn) your definition into machine code. When you enter a BASIC line with a line number, the actual text of the line is stored in memory. FORTH does not remember the actual text of the line. It simply stores the "meaning", i.e. the compiled code, next to the new word name in the dictionary.

The editor allows you to retain the actual text of your definitions. More of this later.

FORTH has no syntax checking to stop you entering garbage. Words are translated immediately into machine code. This means that the system is not difficult to crash if you enter garbage.

If you define a word incorrectly, it may still be entered in the dictionary, but because it makes no sense, it cannot be used. It will appear on the dictionary listing, but will produce an error report if you attempt to use it.

To remove a word from the dictionary, type FORGET and the name of the word. If the definition is garbage, this may not work. In these circumstances type SMUDGE and try again. (See SMUDGE and TOGGLE in the Glossary).

RUN-TIME AND COMPILE-TIME

It should now be clear that the FORTH system operates in two different modes; called run-time and compile time. At run-time, the system executes the words in the program. At compile-time, the system converts words within a definition into code which can later be used at run-time.

Certain words can only be used for compiling (i.e. within colon-definitions. These include the looping words DO and LOOP (which are similar to BASIC's FOR...NEXT)

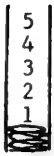
THE PARAMETER STACK

If you ask the FORTH system to store temporarily a few numbers, it will, in most cases, push them onto the parameter stack. The stack is, so to speak, a spring-loaded ping-pong ball box. The numbers (ping-pong balls) are pushed inside the box. This means that the most accessible one is always the last one to be pushed in.

So if you type

1 2 3 4 5 and Enter

the stack will look like this:



So if you ask the computer to print out the numbers on the stack, you will get them out in reverse order.

The word " . " (called DOT) removes the number at the top of the stack and prints it on the screen. So if you type

1 2 3 4 5 . . . . . and Enter

you will get 5 4 3 2 1 OK

At this point the stack will be empty again.

MATH OPERATIONS

All arithmetic in FORTH is performed on the numbers on the stack. Even if the numbers are stored elsewhere in memory, they must first be brought to the stack before they are operated on.

This means that the programmer must ensure that the numbers are already on the stack before he asks the system to do something with them (such as add, subtract, multiply etc.)

So Forth uses "Reverse Polish" notation, also known as "post-fix" notation. Instead of writing:

3 + 4

we write 3 4 +

The Forth word " + " adds the two numbers on the top of the stack and leaves the answer in their place on top of the stack. (If you want to print the answer, you must put a . (dot) after the +. Don't forget the space).

When programming in FORTH, you must always be aware of what is happening to the stack. If you look at the formal definitions of the word + - * and / in the Glossary, you will find they are all defined in terms of what is happening to the stack.

STACK MANIPULATION

The initial Forth vocabulary includes some words which enable you to change the position of numbers on the stack. Examples of these are:

DROP This simply erases the top number from the stack.
SWAP This swaps the positions of the top two numbers.
DUP This adds a duplicate of the top number to the stack.
OVER This adds a duplicate of the second number to the top of the stack.
ROT Rotates the top three values. (i.e. the third becomes the first)

Both DUP and OVER create a new number on the top of the stack, so all the other numbers already on the stack are pushed down one place.

NO FLOATING POINT ARITHMETIC

All numbers in Spectrum BASIC are held to 9 or 10 significant digits, even such simple ones as 1 or 2. This is very expensive in processing time.

Your Forth system deals with whole numbers only. If you need floating point arithmetic, then you must define the procedures yourself.

Furthermore, numbers entered ordinarily will be held in only two bytes. This allows a maximum range of -32768 to +32767. (You should be aware that one of the 16 bits in the two bytes is used to indicate whether a number is negative or positive. Such 15+1 bit numbers are called "signed" numbers, the 16th bit being the sign.)

Thus if you enter a number larger than 32767, only part of the binary bit-pattern will be stored, so when you enter . you will get back a totally different number.

Certain words allow you to use the normal two bytes for unsigned numbers. 16 bits allow a range from Ø to 65535. See the Glossary from U. to U<

Other words allow you to use four bytes to hold a number, giving a range 2147483647 to -2147483648. See the Glossary from D+ to D.R

Unless you define your own number-handling routines, Forth can manipulate whole numbers only. So the calculation 5/3 , written

```
5 3 /
```

will leave the answer 1.

But if you use the word /MOD instead of just / it will leave the result (1) on top of the stack, and the remainder (2) second on the stack. Try

```
5 3 /MOD . .
```

MOD will leave the remainder only. So

```
5 3 MOD .
```

will give the answer 2 only.

The reason that FORTH leaves the number handling systems so open is because different applications require totally different levels of accuracy. If you are using FORTH for a financial application, you are highly unlikely every to need more than two places of decimals. If you are writing games, you are probably better off with whole numbers, and no decimals will be used. If your application is scientific, you may need extreme accuracy.

LOOPS

You can usually only create a loop within a colon-definition.

The simplest type of loop uses the words DO and LOOP. The syntax is, however, very different from the BASIC "FOR.....NEXT" loop.

The limits of the loop must first be put on the stack, with the highest limit first. Next comes DO, followed by whatever you want to happen each time the program loops. Finally type LOOP. For example:

```
:  LOOPER 10 Ø DO ." I'M LOOPING" LOOP ;
```

(The word ." means - print the following string up to the next " . Note that there must be a space after ." as ." is a word.)

The name of your new word is "LOOPER". If you now type in LOOPER, the Spectrum will print "I'M LOOPING" ten times.

What actually happens in this loop is as follows:
a) On reaching the word DO, the program transfers the two values on the parameter stack to another stack (called the Return Stack). As soon as LOOP is reached, the lower value on the return stack is incremented by 1, and compared with the limit value. If they are not equal, (i.e the limit has not yet been reached) then the program will jump back to just after the word DO. As soon as the limit is reached, however, the program will pass LOOP and carry on to what is next.

The word +LOOP enables you to loop in steps other than 1. For example:

```
: STEPLOOP 1ØØ Ø DO ." ONE IN TEN" 10 +LOOP ;
```

This will add 10 to the loop index each time.

The DO loop is called a "definite" loop, because its upper limit is specified within the program. With careful use of the stack, however, the user can specify how many times the loop should be performed without having to redefine the word each time. For example:

```
:  LOOPER Ø DO ." THIS MANY TIMES " LOOP ;
```

Then type

```
5 LOOPER
```

As the limit comes first, the 5 you have entered will be found by DO in its correct position on the stack.

I

The word I is extremely useful in looping. It transfers the top value from the return stack to the parameter stack without deleting it from the return stack. You can thus use the loop index as a variable. For example:

```
: SCREENFILL 22527 16384 DO 255 I ! LOOP ;
```

(The word ! means "store in the address held at the top of the stack the number held second on the stack")

This word SCREENFILL will fill in all the bits in the Spectrum display file.

For more complex loops, see BEGIN AGAIN REPEAT and WHILE in the Glossary. These words allow an "indefinite" loop, which repeats itself until or while a certain condition is met.

CONDITIONALS

Consider the following conditional colon-definition:

```
: TEST Ø < IF ."  THE NUMBER IS NEGATIVE " ELSE ." THE NUMBER
IS POSITIVE " ENDIF ;
```

If you now type

```
5 TEST
```

or

```
-5 TEST
```

the computer will tell you whether the number is positive or negative.

What actually happens in this program?

First your 5 (or -5) goes on the parameter stack.
Then Ø goes on the parameter stack.
Then the two numbers are compared to see whether the first number (5) is less than the second number.
If it is, (i.e. if it is true), then a "true" flag is left at the top of the stack. A true flag is any number greater than Ø.
IF looks at the flag at the top of the stack. If it is true, then execution continues from immediately after IF. If it is false (i.e.Ø) then execution jumps to immediately after the ELSE.
ENDIF (which can be replaced by THEN) terminates the conditional structure.

VARIABLES, CONSTANTS AND ARRAYS

Variables, constants and arrays are all dictionary definitions and are held in the dictionary just like a colon-definition. Each one has the effect of reserving bytes for holding the appropriate numbers.

The difference between a variable and an array lies in what happens when you "call" it, rather than in the form in which it is held in memory.

A constant called, for example, C, with a value of say 365, is entered as follows:

```
365 CONSTANT C
```

No colons or semi-colons must be used.
If you then type

```
2 C + .
```

C will be executed. Execution of a constant means that the value of the constant is pushed directly onto the stack.

A variable called, for example, V, with an initial value of say 10, is defined as follows:

```
10 VARIABLE V
```

But if you then execute V, you will not get the number 10 on the stack. You will instead get the address of the two bytes (called a cell) where the variable is stored. To get the actual value of V, you must use the word @ (pronounced "fetch"), which fetches the number stored at the address currently held on the stack.

So if you type

```
V @ .
```

you will get the number 10.

To change the variable you must likewise use the word ! (pronounced "store") which "pokes" the second number held on the stack into the address held at the top of the stack. For example

```
20 V !
```

will change V to 20.

Arrays (defined in BASIC using the DIM statement) are created in Forth by reserving extra bytes in the dictionary immediately after aVARIABLE definition. This is done by using the word ALLOT. For example

```
20 ALLOT
```

will reserve 20 free bytes in excess of those already reserved by VARIABLE. (enough for 10 more values of V). To store or fetch these values, a little simple arithmetic must be put into your instructions. For example:

```
30 V 2 + !
```

will store the value 30 in the next free cell in the array.

FORTH AND LINE NUMBERS

Once a Forth program has been compiled it has no line numbers. If you are used to BASIC only, this may startle you as you can no longer use GOTO or GOSUB.

But of course each word you have defined is in itself a GOSUB routine, and to call that routine, you simply type in the word.

Moreover, you can call a long and complex program with a single word, and you can use this word within a DO... LOOP structure or a BEGIN ... UNTIL structure without difficulty.

Forth's use of colon-definitions forces you to structure your programs in a much more logical and efficient way than BASIC. Too many programmers start writing their sub-routines in BASIC before they have decided what the general structure of the program will be, and even before they have decided what they want the program to do. A Forth programmer, writing say a stock-control program, would start at the other end of the problem. The last word to be compiled will be STOCKCONTROL, but this will be the first one to be defined by the programmer. He will define it with words he has not yet precisely defined, such as GOODSIN, GOODSOUT, BALANCE and so on. He will proceed to define these lower level words until he reaches a level where he is using only those words which are already in the vocabulary.

This is also the process one ought to follow when writing a program in any other language.

Line numbers do, however, exist in Forth. An essential part of a Forth system is the editor. This allows you to store "source code" in the original form;- i.e. in the same format as when you typed it in.

Source code is stored in "Screens" of 1k bytes each, made up of 16 lines of 64 bytes. When you list a screen (see the section on the EDITOR in this manual) you will see a series of colon-definitions and program instructions in a comprehensible form. These can be edited, added to or deleted.

Code held in screens will not execute until it has been compiled. To compile the words in screen 3, for example, you must type

```
3 LOAD
```

This will compile the words defined in screen three and put them into the dictionary. Once they are in the dictionary, the code on the screen can be deleted.

SAVING AND LOADING

You can both load and save either;-

a) The contents of the screens, using the word SAVESCREENS or LOADSCREENS.
or
b) The contents of the dictionary, using the words SAVETAPE or LOADTAPE.

Don't forget that the contents of the screens will not execute until you have compiled them with LOAD.

FIELDS

In browsing through the Glossary you will frequently come across references to the PFA (Parameter Field Address) and the NFA (Name Field Address). This is a brief explanation of the meaning of these terms.

A dictionary definition is held in memory as follows:

It consists of four distinct areas known as "fields".

1. The Name Field. The first byte of this area holds the number of characters in the name in the lowest 5 bytes, thus limiting the length of any Forth name to 31 characters maximum. The sixth bit is 1 when SMUDGEd, thus preventing a match by (FIND). The seventh bit is called the "precedence bit, and is set to 1 when the word is "immediate". "Immediate" means that if the word is used within a colon definition, it will be executed and not compiled. The eighth bit is always set to 1. The rest of the name field is made up of the ASCII characters of the name of the word.

2. The name field is followed by the link field. This is a field of two bytes, which contains the start address of the previous definition in the dictionary. Using this address, the system can search through the dictionary very rapidly. Note that the dictionary is searched backwards, so if you have two definitions of the same name, the one most recently defined will be found first and used, and earlier ones will be ignored.

3. Next comes the code field. This is also two bytes only, and contains the "code-pointer".
The code pointer points to the run-time code which is appropriate for this particular definition.
If the definition is a variable, or a machine code definition, then the run time code will push the first address of the parameter field (pfa) onto the stack.
If the definition is a constant, the run-time code will push the contents of the first two bytes of the parameter field onto the stack.
If it is a colon definition, the code pointer will point to code which jumps one by one through each address held in the parameter field.

4 The last field in a dictionary definition is the parameter field.
In the case of variables, arrays and constants, this will hold the variable or constant numbers.
In the case of a colon definition, however, the parameter field will hold the addresses of each word used in the definition. The code-pointer points to code which executes the code held at each one of these addresses in turn.

48/80 EXTENSIONS TO STANDARD FIG FORTH

This section describes the additional words included in 48/80 FORTH which enable you to take full advantage of the Spectrum's capabilities. These words cover the colour, graphics, sound and tape handling facilities which you will need to give the same range of capabilities as are available when writing programs in Spectrum BASIC.

GRAPHICS

Several words are available which are very similar to the equivalent BASIC statements.

```
n INK
n PAPER
```

where n is a number between Ø and 9 and is equivalent to a Spectrum colour.

```
n BORDER
```

where n is a number between Ø and 7.

A further facility included in 48/80 FORTH is that a colour can be typed in directly, instead of using a number. i.e.

```
BLACK INK WHITE PAPER
```

is equivalent to

```
Ø INK 7 PAPER
```

The following expressions are equivalent to the BASIC statements:

```
n FLASH
n BRIGHT
n INVERSE
n GOVER
```

GOVER is equivalent to the BASIC OVER statement, and has been renamed so that it does not clash with the standard FORTH word OVER. n is Ø to turn the appropriate function off, and non-zero to turn it on.

Note that all these words affect both the temporary and permanent Spectrum colour attributes. (See pages 194 and 195 in the Spectrum manual). This means that the effect of these words will last until changed by the use of the word again.

These four words perform the same functions as the Spectrum statements;

```
x y POINT
x y PLOT
x y DRAW
x y z CIRCLE
```

but unlike the Spectrum, DRAW statement x and y are actual screen coordinates, rather than displacements. DRAW draws a line from the last plotted position (or end of the last line drawn) to the coordinates give. So to draw a square in the centre of the screen you could type:

100 60 PLOT 100 116 DRAW 156 116 DRAW 156 60 DRAW
100 60 DRAW and Enter.

z is the radius of the circle. POINT returns Ø on the stack if the pixel at x y is PAPER colour, else it returns 1 on the stack if it is INK colour.

n1 n2 AT
positions the cursor at n1 column and n2 line.

n1 TAB
positions the cursor at cclumn number n1, modulo 32, either on the present line if that does not involve back-spacing or else on the next line.

CLS
This is similar to the BASIC statement, clearing the screen, setting it to the present PAPER colour, and placing the cursor at the top left hand corner of the screen.

n1 n2 n3 n4 n5 n6 n7 n8 nchar UDG
This enables user defined graphics characters to be created. nchar lies between Ø and 15 and is equivalent to the graphics shift characters A to P. n1 is the bottom byte of the 8 X 8 pixel UDG, and n8 is the top byte. So to define graphics-shifted A as a square box, tyupe:-

HEX FF 81 81 81 81 81 81 FF Ø UDG (Note that the word HEX changes the number base from 10 to 16)

SOUND

n1 n2 BEEP

This words allows sounds to be produced. The smaller n1 is, the higher the frequency which will be produced, and the larger n2 is, the longer the duration of the note. More precisely the value of n1 can be calculated from the following equation:

$$n1 = \left(\frac{3500000}{8 \times fout} \right) - 30$$

where fout is the required frequency. Note that the value of 3500000 is the frequency of the Spectrum system clock, i.e. 3.5 MHz. So to produce middle C (= 261 Hz) n1 will be 1646.

n2 is equal to the no. of cycles in the BEEP, so to produce 1 second of middle C there will be 261 cycles , so n2 = 261. i.e.

1646 261 BEEP

By using BEEP within a loop it is possible to produce real arcade games types of sounds. Enter these two lines as an example:

```
: SOUND1  3ØØ 1Ø DO I 2Ø BEEP 4 +LOOP ;
; SOUND2  6 1 DO SOUND1 LOOP ;
```

and then type SOUND2

TIMING WORDS

There are 3 words which help you to implement timing functions. They operate on the Spectrum FRAMES system variable (see page 175 of the Spectrum Manual)

TIMEØ

sets the FRAMES variable back to zero.

TIME@

fetches the contents of FRAMES, i.e. the elapsed time since the Spectrum was switched on, or since TIMEØ was executed, and leaves it as a double number on top of the stack. This is the elapsed time in 50ths of a second, or 60ths in the USA.

TIME.

prints the elapsed time in seconds to Ø.Ø2 of a second.

Note that the FRAMES system variable is not incremented during operation of the ZX Printer, or during loading and saving cassette operations, or during BEEP operation.

FREE MEMORY SPACE

The available space which is free for your own word definitions can be found by using the word

FREE

which will leave the available space on the top of the stack. You may wish to define a word which will actually print this information for you, such as this example:

```
: FREE.  FREE CR U.  ." Bytes free "  CR ;
```

Then by typing FREE, you should get the following response:

```
31568 Bytes free.          (for 80k version)
```

CASSETTE HANDLING WORDS

Four words are included which allow you to save and load data from cassette. They will all be dealt with here, although two of them (LOADSCREENS and SAVESCREENS) will usually only be used with the Editor, which is dealt with in a separate section of the manual.

SAVETAPE

address length SAVETAPE ccccc

where address is the starting address and length is the number of bytes of code to be saved, to a tape with a name (not more than 10 characters, just as in BASIC) following the SAVETAPE command in the position of the ccccc. Note that no quotes are required round the name, also that there is no "Start tape and press any

The output will start as soon as you press enter. The most important use of this is to save an extended version of 48/80 FORTH, including your own words which have been added to the top of the dictionary. Note that this copy is for your own use only and is not for resale or hiring purposes, as this would infringe the copyright.

Before you can save the extended dictionary you will need to change the cold start parameters, as is shown below. Type in the following lines:

```
FORTH DEFINITIONS DECIMAL
LATEST 12  +ORIGIN !
HERE 28 +ORIGIN !
HERE 30 +ORIGIN !
HERE  FENCE !
'  FORTH 6 +  32  +ORIGIN  !
```

Then type the following, but before pressing enter at the end of the line, start the tape running with the cassette recorder in record mode (and do not use your master 48/80 FORTH tape!)

```
Ø  +ORIGIN  DUP  HERE  SWAP  -  SAVETAPE  NEWFORTH (or whatever you
                                                    wish to call it)
```

and you will record a new extended version of your 48/80 FORTH.

Change FORTH for the name of the appropriate vocabulary if you are using a different vocabulary. An example is to be found on screen #5 of the editor supplied.

LOADTAPE

The syntax for this word is

```
addr   LOADTAPE ccccc
```

where ccccc is the name of the bytes to be loaded from tape. This is not normally a very useful word, and finds most use when transferring screens (see the section on the Editor for a discussion of screens) between the 80k and 48k versions of FORTH and vice versa.

LOADSCREENS

This word enables edited source code prepared using the FORTH editor to be saved on cassette. The syntax for this word is

```
Ist-screen-no   No-of-screens  SAVESCREENS cccc
```

where screen-nos refers to the Editor screen numbers. So to save screens 7 and 8 to tape with a name FORTHTOOLS (for example) type

```
7   2  SAVESCREENS   FORTHTOOLS
```

and again start the tape before pressing the enter key, just as for the SAVETAPE word.

LOADSCREENS

The syntax for this word is

LOADSCREENS ccccc

and is used to load a previously saved screen or screens of source code from tape. So to load the source code for FORTHTOOLS, type

LOADSCREENS FORTHTOOLS

Some general comments on using the tape cassette routines are in order. Firstly, error message #10 will occur if the name is longer than ten characters, and control will then be passed back to the Forth operating system, just as for any other error message. If you press the break key during tape operations, then you will be returned to BASIC. If this happens type

RANDOMIZE USR 24580

to perform a WARM start. You will not lose any of the words you have defined. (This is also the case when pressing the Break key during ZX Printer operation, and during BEEP). If you mistype the name of the tape you wish to load, and have already pressed the enter key before you realise your mistake, then you will have to press the break key to exit from the tape routines. Perform a warm start as above to retrieve the situation.

THE EDITOR

LOADING THE EDITOR

The EDITOR is located on your tape immediately after both the 48k and the 80k compilers. Once you have loaded 48/80 FORTH you can load the EDITOR by typing

LOADSCREENS EDITOR and enter.

Once the editor is loaded successfully, the program will respond with OK.

The source text of the EDITOR is now held in the screens. It cannot be used until it is compiled into the dictionary. To do this, type

1 LOAD and enter.

The EDITOR words will now be compiled. Do not worry about error message 4, which will occur for the words R and I, which have different definitions in FORTH and the EDITOR.

Now type EDITOR. This will select the new vocabulary called EDITOR. Then type VLIST. You will see that you have a whole new list of words in the dictionary.

FORTH SCREENS

Forth organises all mass storage on a system of screens. A screen consists of 1024 bytes, or characters, and is divided into 16 lines of 64 characters each. This does not correspond to the screen format of the Spectrum, but this is not important.

Note that since the screens are an arrangement of "virtual memory" (i.e. they are loaded into RAM complete, and are thus treated by the system as though they were part of RAM) you can have any number of them stored on tape. You are limited to having 16 screens in RAM with the 48k FORTH and 32 screens with 80k FORTH at any one time.

The screens are numbered Ø to 15 (48k FORTH) and Ø to 31 (80k version). Screen number Ø is usually reserved for comments, and cannot be used to load source code.

SELECTING A SCREEN AND INPUTTING TEXT

To start an editing session, the user types EDITOR to select the appropriate vocabulary.

The screen to be edited is then selected using either

n LIST (List screen n and select it for editing) or
n CLEAR (Clear screen n and select it for editing)

To input new text to screen n after LIST or CLEAR the P (PUT) command is used. i.e.

```
Ø  P  THIS IS HOW
1  P  TO INPUT TEXT
2  P  TO LINES Ø, 1 AND 2 OF THE SELECTED SCREEN.
```

LINE EDITING

During this description of the EDITOR, reference is made to PAD. This is a text buffer which may hold a line of text used by or saved with a line-editing command, or a text string to be found or deleted by a string editing command.

PAD can be used to transfer a line from one screen to another as well as to perform edit operations within a sincle screen.

Line Editor Commands

n H Hold line n at PAD (Used by system more often than by user)
n D Delete line n but hold it in PAD. Line 15 becomes blank as line n+1 to line 15 move up one line.
n T Type line n and save it in PAD
n R Replace line n with the text in PAD
n I Insert the text from PAD at line n, moving the old line n and following lines down. Line 15 is lost.
n E Erase line n with blanks.
n S Spread at line n. n and subsequent lines move down one line. Line n becomes blank. Line 15 is lost.

CURSOR CONTROL AND STRING EDITING

The screen of text being edited resides in a buffer area of storage. The editing cursor is a variable holding an offset into this buffer area. Commands are provided for the user to position the cursor, either directly, or by searching for a string of buffer text, and to insert or delete text from the current cursor position.

Commands to position the cursor

TOP Position the cursor at the top of the screen.
n M Move the cursor by a signed amount n and print the cursor line. The position of the cursor on its line is shown by _ (underline).

Commands to position the cursor (cont)

F text Search forward from the current cursor position until string "text" is found. The cursor is left at the end of the text string, and the cursor line is printed. If the string is not found an error message is given and the cursor is repositioned at the top of the screen.

B Used after F to back up cursor by the length of the most recent text.

N Find the next occurence of the string found by an F command.

X text Find and delete the string "text".
C text Copy in text to the cursor line at the cursor position.
TILL text Delete on the cursor line from the cursor until the end of the text string "text".

NOTE: Typing C with no text will copy a null into the text at the cursor position. This will abruptly stop later compiling! To delete this error, type TOP X and Enter.

n LIST List screen n and select it for editing.
n CLEAR Clear screen n with blanks and select it for editing.
n1 n2 COPY Copy screen n1 to screen n2.
L List the current screen, The cursor is relisted after the screen listing to show the current cursor position.
FLUSH Used at the end of an editing session to ensure that all entries and updates of text have been transferred from the buffer area to the mass storage area.

FUTURE PLANS

Work is already in progress to extend 48/80 FORTH. In future it is planned to offer a set of Forth software tools including a Z80 Assembler, a decompiler and several other useful aids for the Forth programmer.

It is also planned to add additional words to enable operation with the Microdrive and Interface 1, and our own Trickstick.

ACKNOWLEDGEMENTS

Acknowledgement is duly made to the Forth Interest Group, P.O. Box 1105, San Carlos, Ca 94070 for parts of this compiler and manual.

APPENDIX 1

GLOSSARY

This glossary contains all the word definitions in 48/80 FORTH. The words are presented in the order of their ASCII sort.

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "---" indicate the execution point. Any parameters left on the stack are listed. In this notation the top of the stack is to the right.

The symbols include:

addr	memory address
b	8 bit byte (i.e. high 8 bits zero)
c	7 bit ASCII character (high 9 bits zero)
d	32 bit signed double integer, most significant portion with sign on top of stack.
f	boolean flag. Zero = false, non-zero = true.
ff	boolean false flag
n	16 bit signed integer
u	16 bit unsigned integer
tf	boolean true flag

The capital letters to the right of some definitions show definition characteristics as follows:

C	word may only be used within a colon-definition.
E	Intended for execution only.
P	has precedence bit set. Will execute even when compiling.
U	a user variable.

48/80 FORTH uses standard FIG-FORTH, and the definitions in this glossary are those specified by the FORTH Interest Group.

!

n addr ---
Store 16 bits of n at address.
Pronounced " store ".

!CSP

Save the stack position in CSP. Used as part of the compiler security.

#

d1 --- d2
Generate from a double number d1, the next ASCII character which is paced in the output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between <# and #>.
See #S

#>

d --- addr count
Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.

#BUF

--- n
A constant returning the number of disk buffers allocated.

#S

d1 --- d2
Generates ASCII text in the text output buffer, by use of #, until a zero double number results. Used between # and # .

'

--- addr P
Used in the form:
' nnnn
Leaves the parameter field address of dictionary word nnnn. As a compilation directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT an appropriate error message is given. Pronounced " tick ".

(

P
Used in the form:
(cccc)
Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon definition. A blank after the leading parenthesis is required.

(.")

C
The run-time procedure, compiled by ." which transmits the following in-line text to the selected output device. See ."

(;CODE)

C
The run-time procedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.

(+LOOP) n --- C
The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.

(ABORT)
Executes after an error when WARNING is -1. This word normally executes ABORT but may be altered (with care) to a user's alternative procedure.

(DO) C
The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO.

(FIND) addr1 addr2 --- pfa b tf (OK)
addr1 addr2 --- ff (bad)
Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address (pfa), length byte of name field and boolean true for a good match. If no match is found only a boolean false is left.

(LINE) n1 n2 --- addr count
Convert the line no. n1 and the screen no. n2 to the disk buffer address containing the data. A count of 64 indicates the full line text length.

(LOOP) C
The run-time procedure compiled by LOOP which increments the index and tests for loop completion. See LOOP.

(NUMBER) d1 addr1 --- d2 addr2
Convert the ASCII text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.

* n1 n2 --- prod
Leave the signed product of two unsigned numbers.

*/ n1 n2 n3 --- n4
Leave the ratio n4 = n1 * n2/n3 where all are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence n1 n2 * n3 /

*/MOD n1 n2 n3 --- n4 n5
Leave the quotient n5 and remainder n4 of the operation n1*n2/n3. A 31 bit intermediate product is used as for */.

+ nl n2 --- sum
Leave the sum of nl + n2.

+! n addr ---
Add n to the value at the address. Pronounced " plus-store ".

+- nl n2 --- n3
Apply the sign of n2 to nl, which is left as n3.

+BUF addr1 --- addr2 f
Advance the disk-buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by variable PREV.

+LOOP nl --- (run) P,C
addr n2 --- (compile)
Used in a colon definition in the form:
DO .... nl +LOOP
At run time, +LOOP selectively controls branching back to the corresponding DO based on nl, the loop index and the loop limit. The signed increment nl is added to the index and the total compared to the limit (nl 0), or until the new index is equal to or less than the limit (nl 0). Upon exiting the loop the parameters are discarded and execution continues ahead.
At compile time, +LOOP compiles the run time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. n2 is used for compile time error checking.

+ORIGIN n --- addr
Leave the memory address relative by n to the origin parameter area. n is the minimum address unit, either byte or word. This definition is used to access or modify the boot-up parameters at the origin area.

, n ---
Store into the next available dictionary memory cell, advancing the dictionary and pointer. (comma)

- nl n2 --- diff
Leave the difference of nl-n2.

--> P
Continue interpretation with the next disk screen. Pronounced " next-screen ".

-DUP nl --- nl (if zero)
nl --- nl nl (if non-zero)
Reproduce nl only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

-FIND --- pfa b tf (found)
--- ff (not found)
Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte and a boolean true is left. Otherwise only a boolean false is left.

-TRAILING addr n1--- addr n2
Adjusts the character count of a text string beginning address to suppress the ouput of trailing blanks. i.e. the characters at addr+n1 to addr+n2 are blanks.

n ---
Print a number from a signed 16 bit two's complement value, converted according to the numeric BASE. A trailing blank follows. Pronounced DOT.

." Used in the form P
." cccc"
Compiles an in-line string cccc (delimited by the trailing ") with an execution procedure to transmit the text to the selected output device. If executed outside a definition, ." will immediately print the text until the final " . The maximum number of characters may be an installation dependant value. See (.")

.CPU This word prints out the version of memory for which the FORTH is configured, either 48k or 80k.

.LINE line scr ---
Print on the terminal device a line of text from the disk by its line and screen number. Trailing blanks are suppressed.

.R n1 n2 ---
Print the number n1 right-aligned in a field whose width is n2. No following blank is printed.

/ n1 n2 --- quot
Leave the signed quotient of n1/n2

/MOD n1 n2 --- rem quot
Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 --- n
These small numbers are used so often that it is attractive to define them by name in the dictionary as constants.

0< n --- f
Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

0=
n --- f
Leave a true flag if the number is equal to zero, otherwise leave a false flag.

0BRANCH
f --- C
The run-time procedure to conditionally branch. If f is a false (zero) the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL and WHILE.

1+
n1 --- n2
Increment n1 by 1

2+
n1 --- n2
Leave n1 incremented by 2

2!
nlow nhigh addr ---
32 bit store. nhigh is stored at addr. nlow is stored at addr+2.

2@
addr --- nlow nhigh
32 bit fetch. nhigh is fetched from addr. nlow is fetched from addr+2.

2DUP
n2 n1 --- n2 n1 n2 n1
Duplicate the top two values on the stack. Equivalent to OVER OVER.

P,E
Used in the form called a colon-definition:
: cccc .... ;
Creates a dictionary entry defining cccc as equivalent to the following sequence of FORTH word definitions "..." until the next " ; " or " ;CODE". The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the the CONTEXT VOCABULARY IS SET TO THE CURRENT vocabulary and that words with the precedence bit set are executed rather than compiled.

;
P,C
Terminate a colon definition and stop further compilation. Compiles the run-time ;S.

;CODE
P,C
Used in the form:
: cccc .... ;CODE
Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following mnemonics.
When cccc later executes in the form:
cccc nnnn
the word nnnn will be created with its execution procedure given by the machine code following cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S P

Stop interpretation of a screen. ;S is also the run-time word compiled at the end of a colon definition, which returns execution to the calling procedure.

< nl n2 ---- f

Leave a true flag if nl is less than n2, otherwise leave a false flag.

<#

Set up for pictured numeric output formatting using the words:

```
<# # #S SIGN #>
```

The conversion is done on a double number producing text at PAD.

<BUILDS C

Used within a colon definition:

```
: cccc <BUILDS ....
      ....DOES> .... +
```

Each time cccc is executed, <BUILDS defines a new word with a high level execution procedure. Executing cccc in the form :

```
cccc nnnn
```

uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES in cccc. <BUILDS and DOES> allow run-time procedures to be written in high level rather than assembler code (as required by ;CODE)

= nl n2 ---- f

Leave a true flag if nl=n2. Otherwise leave a false flag.

> nl n2 --- f

Leave a true flag if nl is greater than n2. Otherwise leave a false flag.

>R n --- C

Remove a number from the computation stack and place it as the most accessible on the return stack. Use should be balanced with R> in the same definition.

? addr ---

Print the value contained at the address in free format according to the current BASE.

?COMP

Issue an error message if not compiling.

?CSP

Issue an error message if stack position differs from value saved in CSP.

?ERROR f n ---

Issue an error message number n if the boolean flag is true.

?EXEC
Issue an error message if not executing.

?LOADING Issue an error message if not loading.

?PAIRS n1 n2 ---
Issue an error message if n1 does not equal n2. The message indicates the compiled conditionals do not match.

?STACK Issue an error message if the stack is out of bounds. This definition may be implementation dependant.

?TERMINAL --- f
Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is implementation dependant.

@ addr --- n
Leave the 16n bit contents of address.

ABORT Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.

ABS n --- u
Leave the absolute value of n as u

AGAIN addr --- (compiling) P,C
Used in a colon definition in the form
BEGIN....AGAIN
At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R DROP is executed one level below).
At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile time error checking.

ALLOT n ---
Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-origin memory. n is with regard to computer address type (byte or word).

AND n1 n2 --- n3
Leave the bitwise logical and of n1 and n2 as n3.

AT n1 n2 ---
Positions the print position at n1 column and n2 line.

B/BUF --- n
This constant leaves the number of bytes per disk-buffer, the byte count read from disk by BLOCK.

B/SCR --- n
This constant leaves the number of blocks per editing screen. By convention an editing screen is 1024 bytes organised as 16 lines of 64 characters each.

BACK addr ---
Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address.

BASE --- addr U
A user variable containing the current number base used for input and output conversion.

BEEP n1 n2 ---
Sounds n2 cycles of frequency given approximately by the expression

fout = (437500/n1)-30Hz

BEGIN --- addr (compiling) P
Occurs in a colon-definition in the form:

```
BEGIN....UNTIL
BEGIN....AGAIN
BEGIN....WHILE....REPEAT
```

At run time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding UNTIL, AGAIN or REPEAT.
When executing UNTIL, a return to BEGIN will occur if the top of the stack is false: for AGAIN and REPEAT a return to BEGIN always occurs.
At compile time BEGIN leaves its return address and n for compiler error checking.

BL --- c
A constant that leaves the ASCII value for blank.

BLACK --- Ø
Leaves the numerical value of BLACK colour on the stack for use by PAPER, INK and BORDER.

BLANKS addr count ---
Fill an area of memory beginning at address with blanks.

BLK --- addr U
A user variable containing the block number being interpreted. If zero, input is taken from the terminal input buffer.

BLOCK

n --- addr
Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disk to whichever buffer was least recently written.
If the block occupying that buffer has been marked as updated, it is rewritten to disk before block n is read into the buffer.
See also BUFFER R/W UPDATE and FLUSH.

BLUE

--- 1
leaves the numerical value of BLUE on the stack for use by PAPER, INK and BORDER.

BORDER

n ---
Sets the BORDER colour to n, where n lies between Ø and 7.

BRANCH C

The run-time procuedure to unconditionally branch.
An in-line offset is added to the interpretive pointer IP to branch ahead or back.
BRANCH is compiled by ELSE, AGAIN and REPEAT.

BRIGHT

n ---
Turns permanent BRIGHT attribute on if n is non-zero.
Turns it off if n is zero.

BUFFER

n --- addr
obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marke as updated it is written to the disk. The block is not read from disk.
The address left is the first memory cell within the buffer for data storage.

BYE

Return to BASIC

C! b addr ---
Store 8 bits of b at address.

C, b ---
Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer.

C/L --- n
Constant leaving the number of characters per line. Used by the editor.

C@ addr --- b
Leave the 8 bit contents of memory address.

CFA pfa --- cfa
Convert the parameter field address of a definition to its code field address.

CIRCLE n1 n2 n3 ---
Draws a circle of radius n3 centred on coordinates n1 n2.

CLS

Clears screen and sets cursor or print position to top left hand corner of screen.

CMOVE from to count ---
Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding towards high memory.

CMOVE2 from to count ---
Move bytes to or from FORTH screens (i.e. page 2 of SP80 memory). Normally this will be used to access screens. The switching between page 1 and page 2 is done automatically. Otherwise the action is just as for CMOVE. 80k version only.

COLD

The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.

COMPILE C

When the word containing COMPILE executes, the execution address of the word following compile is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address(which the interpreter already does).

CONSTANT

n ---
A defining word used in the form
n CONSTANT cccc
to create word cccc with its parameter field containing n. When cccc is later executed, it will push the value of n onto the stack.

CONTEXT

--- addr U
A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

COUNT

addr1 --- addr2 n
Leave the byte address 2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at address 1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.

CR

Transmit a carriage return and line feed to the selected output device.

CREATE

A defining word used in the form
CREATE cccc
by such words as CODE and CONSTANT to create a dictionary header for a FORTH definition. The code field contains the address of the word's parameter field. The new word is created in the CURRENT vocabulary.

CSP

--- addr
A user variable temporarily storing the stack pointer position for compilation error checking.

CURRENT

--- addr
A user variable containing a pointer to the current vocabulary.

CYAN

--- 5
Leaves the numerical value of CYAN on the stack for use by PAPER, INK and BORDER.

D+

d1 d2 --- dsum
Leave the double number sum of two numbers.

D+-

d1 n --- d2
Apply the sign of n to d1 leaving it as d2.

D.

d ---
Print a signed double number from a 32 bit two's complement value. The high order 16 bits are the most accessible on the stack. Conversion is performed according to BASE. A blank follows. Pronounced D dot.

D.R
d n ---
Print a signed double number d right aligned in a field n characters wide.

DABS
d --- ud
Leave the absolute value ud of a double number.

DECIMAL
Set the numeric conversion BASE for decimal input/output.

DEFINITIONS
Used in the form
cccc DEFINITIONS
Set the current vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.

DIGIT
c n1 --- n2 tf (ok)
c n1 --- fff (bad)
Converts the ASCII character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid only a false flag is left.

DLIST
List the names of the dictionary entries in the CONTEXT vocabulary.

DLITERAL
d --- d (executing) P
d --- (compiling)
If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it onto the stack. If executing, the number will remain on the stack.

DMINUS
d1 --- d2
Convert d1 to its double-number two's complement.

DO
n1 n2 --- (execute)
addr n --- (compile)
Occurs in a colon definition in the form
DO..... LOOP
DO..... +LOOP
At run time, DO begins a sequence with repetitive exectuion controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching loop the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop, 'I' will copy the current value of the loop index to the stack. See I, LOOP, +LOOP LEAVE.
When compiling within the colon-definition, DO compiles (DO) , leaves the following address addr and n for later error checking.

DOES >

A word which defines the run-time action within a high level defining word. DOES alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES .
Used in combination with BUILDS. When the DOES part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents.
Typical uses include the FORTH assembler, multi-dimensional arrays, and compiler generation.

DP U

A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.

DPL --- addr U

A user variable containing the number of digits to the right of the decimal point on double integer input. It may also be used to hold output column location of a decimal point, in user generated formating. The default value on single number input is -1.

DRAW n1 n2 ---

Draws a line from the last plotted or drawn pixel to coordinates n1 and n2.

DROP n ---

Drop the number from the stack.

DUP n --- n n

Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2 (compiling) P.C.

Occurs within a colon-definition in the form:

```
IF ... ELSE ... ENDIF
```

At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect.
At compile time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.

EMIT c ---

Transmit ASCII character c to the selected output device. OUT is incremented for each character output.

EMPTY-BUFFERS

Marcks all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to the disk. This is also an initialisation procedure before first use of the disk.

ENCLOSE addrl c --- addrl n1 n2 n3
The text scanning primitive used by WORD. From the text address addrl and an ASCII delimiting character c, is determined the byte offset to the first non-delimiter character n1, the offset to the first delimiter after the text n2, and the offset to the first character not included. This procedure will not process past an ASCII 'null', treating it as an unconditional delimiter.

END P.C.
This is an 'alias' or duplicate definition for UNTIL.

ENDIF addr n --- (compile) P.C.
Occurs in a colon definition in the form
IF ... ENDIF
IF ... ELSE ... ENDIF
At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. Se also IF and ELSE.
At compile time ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error testing.

ERASE addr n ---
Clear a region of memory to zero from addr over n addresses.

ERROR line --- in blk
Executes error notification and restart of system. WARNING is first examined. If 1, the text of line n relative to screen 4 is printed. This line number may be positive or negative, and beyond screen 4. If WARNING = Ø, n is just printed as a message number. If WARNING is -1, the definition (ABORT) is executed which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). The contents of IN and BLK are saved to assist in determining the location of the error. Final action is execution of QUIT.

EXECUTE addr ---
Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXPECT addr count ---
Transfer characters from the terminal to address until a 'return' or the count of characters have been received. One or more nulls are added to the end of the text.

FENCE --- addr
A user variable containing address below which FORGETting is trapped. To FORGET below this point the user must alter the contents of FENCE.

FILL addr quan b ---
Fill memory at address with specified quantity of bytes b.

FIRST --- n
A constant that leaves the address of the first (lowest) block buffer.

FLD --- addr U
A user variable for control of number output field width. Presently unused.

FLASH n ---
Sets permanent FLASH attribute on if n is non-zero, and turns it off if n is zero.

FLUSH
Write all updated disk-buffers to disk. Shjould be used after editing.

FORGET E
Executed in the form
FORGET cccc
Deletes definition named cccc and all successive entries from the dictionary. An error message will occur if the CURRENT and CONTEXT vocabularies are not the same.

FORTH P
The anme of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined new user definitions bocom a part of FORTH. FORTH is immediate so it will execute during the creation of a colon definition, to select this vocabulary at compile-time.

FREE --- n
Leaves the available free dictionary space on the stack.

GOVER n ---
Sets permanent OVER attribute on if n is non-zero, and off if n is zero.

GREEN --- 4
Leaves the numerical value of GREEN on the stack for use by PAPER INK and BORDER.

HEADER Used in the form
HEADER cccc
where cccc is the name given to a file saved to or loaded from cassette. Issues an error message if cccc is greater than 10 characters. Used by LOADTAPE, SAVETAPE, LOADSCREENS and SAVESCREENS.

HERE --- addr
Leave the address of the next available dictionary location.

HEX
Set the numeric conversion base to 16 (hexadecimal)

HLD --- addr
A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD c ---
Used between # and # to insert an ASCII character into pictured nummeric output string. e.g. 2E HOLD will place a decimal point.

I --- n C
Used within a DO loop to copy the loop index to the stack. See R.

ID. addr ---
Print a definition's name from its name field address.

IF f --- (run time) P,C
--- addr n (compile)
Occurs in a colon-definition in the form
IF (true part) ... ENDIF
IF (true part) .... ELSE (false part) ... ENDIF
At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead through the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional and if missing execution skips to just after ENDIF. At compile time, IF compiles OBRANCH and reserves space for an offset at addr. addr and n are used for resolution of the offset and error testing.

IMMEDIATE
Mark the most recently made definition so that when encountered at compile time it will be executed rather than compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceding it with COMPILE .

IN --- addr
A user variable containing the byte offset within the current input text buffer (terminal or disk) from which the next imput will be accepted. WORD uses and moves the value of IN.

INDEX from to ---
Print the first line of each screen over the range from to. This is used to view the comment lines of an area of text on disc screens.

INK nl ---
Sets permanent INK colour to nl, where nl is from Ø (black) to 9.

INTERPRET
The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disk) depending on STATE. If the word name cannot be found aftyer a search of CONTEXT and then CURRENT it is converted to a number according to the current base. That also failing, an error message echoing the name with a "?" will be given. Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose than to force this action. See NUMBER.

INVERSE n ---
Turns permanent INVERSE attribute on if n is non-zero, and off if n is zero.

J --- n
Used within a DO loop to copy the second loop index to the stack. Usually only useful for a nested DO loop within the same word.

K --- n
Used within a DO loop to copy the third loop index to the stack. Used for nested DO loop within a single definition, as J.

KEY --- c
Leave the ASCII value of the next terminal key struck

LATEST --- addr
Leave the name field address of the topmost word in the current vocabulary.

LEAVE
Force termination of a DO loop at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged and execution proceeds normally until LOOP or +LOOP is encountered.

LFA pfa --- lfa
Convert the parameter field address of a dictionary definition to its link field address.

LIMIT --- n
A constant leaving the address just above the highest memory available for a disk buffer. Usually this is the highest system memory.

LIST n ---
Display the ASCII text of screen n on the selected output device. SCR contains the screen number during and after this process.

LIT --- n C

Within a colon definition LIT is automatically compiled before each 16 bit leteral number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack.

LITERAL n --- (compiling) P,C

If compiling, then compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon-definition. The intended use is

```
: xxx (calcualted) LITERAL ;
```

Compilation is suspended for the compile time calcualtion of a value. Compilation is resumed and LITERAL compiles this value.

LOAD n ---

Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S See ;S and --❯ .

LOADSCREENS

Occurs in the form

```
LOADSCREENS cccc
```

Loads bytes from tape with name cccc into screens memory. It will load screens back to the screen numbers from which they were saved. See SAVESCREENS.

LOADTAPE addr ---

Occurs in the form

```
LOADTAPE cccc
```

Loads bytes from tape starting at address and of length taken from header. If addr is Ø then loads length no of bytes starting at the address given in the tape header. See LTAPE.

LOOP addr n --- (compiling)

Occurs in a colon definition in the form:

```
DO ... LOOP
```

At run-time LOOP selectively controls branching back to the corresponding DO based opn the loop index and the limit. The loop index is encremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit. At that time the parameters are discarded and execution continues ahead.
At compile time LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error testing.

LTAPE addr ---

This loads data from tape using the parameters set up in a 34 byte block with start address addr using the normal Spectrum format.

LTAPE2 addr ---
As LTAPE but operates on page 2 of RAM. (80k version only)

M* n1 n2 --- d
A mixed magnitude math operation which leaves the double number signed product of two signed numbers.

M/ d n1 --- n2 n3
A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3 from a double number dividend and divisor n1. The remainder takes its sign from the dividend.

M/MOD ud1 u2 --- u3 ud4
An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3 from a double dividend ud1 and single divisor u2.

MAGENTA --- 3
Leave the numerical value of MAGENTA on the stack for use by PAPER, INK and BORDER.

MAX n1 n2 --- MAX
Leave the greater of the two numbers.

MESSAGE n ---
Print on the selected output device the text line n relative to screen 4. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero the message will simply be printed as a number.

MIN n1 n2 --- min
Leave the smallest of two numbers.

MINUS n1 --- n2
Leave the two's complement of a number.

MOD n1 n2 --- mod
Leave the remainder of n1/n2 with the same sign as n1

NFA pfa --- nfa
Convert the parameter field address of a definition to its name field address.

NOOP
A FORTH'no-operation'.

NUMBER addr --- d
Convert a character string left at addr with a preceeding count to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given DPL but no other effect occurs. If numeric conversion is not possible an error message is given.

OFFSET --- addr U
A user variable which may contain a block offset to disk drives. The contents of OFFSET is added to the stack number by BLOCK. See BLOCK MESSAGE.

OR n1 n2 --- or
Leave the bit-wise logical "or" of two 16 bit values.

OUT --- addr
A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formatting.

OVER n1 n2 --- n1 n2 n1
Copy the second stack value, placing it as the new top.

P! b port# ---
Outputs byte b to port #.

P@ port# --- b
Inputs byte b from port#.

PAD --- addr
Leave the address of the text output buffer, which is a fixed offset above HERE.

PAPER n ---
Set permanent PAPER colour to n where n = Ø (black) to 9.

PFA nfa --- pfa
Convert the name field address of a compiled definition to its parameter field address.

PLOT n1 n2 ---
Plots pixel at coordinates n1 n2.

POINT n1 n2 --- n3
Returns 1 if pixel at coordinates n1 n2 is INK. Otherwise returns Ø

PREV --- addr
A variable containing the address of the disk buffer most recently referenced. The UPDATE command marks this buffer to be later written to disk.

QUERY
Input 80 characters of text (or until a "return") from the operator's terminal. Text is positioned at the address contained in TIP with IN set to Ø.

QUIT
Clear the return stack, stop compilation and return control to the operators terminal. No message is given.

R --- n
Copy the top of the return stack to the computation stack.

R#
--- addr U
A user variable which may contain the location of an editing cursor or other file related function.

R/W
addr blk f ---
The disk read-write linkage. Addr specifies the source or destination block buffer. Blk is the sequential number of the referenced block, and f is a flag for write (f=Ø) and read (f=1). R/W determines the location on mass storage, performs the read/write, and performs any error checking.

R>
--- n
Remove the top value from the return stack and leave it on the computation stack.
See R and R.

RØ
--- addr U
A user variable containing the intial location of the return stack. Pronounce Rzero.
See RP!

RED
--- 2
Leaves the number value of RED on the stack for use by PAPER, INK and BORDER.

REPEAT
addr n --- (compiling) P,C
Used within a colon definition in the forms
BEGIN....WHILE....REPEAT
At run time REPEAT forces an unconditional branch back to just after the corresponding BEGIN.
At compile time REPEAT compiles BRANCH and the offset from HERE to addr. N is used for error testing.

ROT
n1 n2 n3 --- n2 n3 n1
Rotate the three values on the stack, bringing the third to the top.

RP!
A computer dependant procedure to initialise the return stack pointer from user variable RØ.

RP@
--- addr
Leaves the current value in the return stack pointer register.

S->D
n --- d
Sign extend a single number to form a double number.

TIME.
Prints elapsed time in seconds since the Spectrum was switched on or the timer was reset with TIMEØ.

TIME@
--- d
Leaves elapsed time in 50th of a second increments as a double number on the stack.

TIMEØ
Resets the Spectrum timer to Ø

TOGGLE
addr b ---
Complement the contents of addr by the bit pattern b.

TRAVERSE
addr1 n --- addr2
Move across a variable length name field. addr1 is the address of either the length byte or the last letter. If n=1 the motion is towards high memory. If n = -1 the motion is towards low memory. The addr2 resulting is the address of the other end of name.

TRIAD
scr ---
Display on the selected output device the three screens which include that numbered screen, beginning with a screen evenly divisible by three. Output is suitable for source text records

TYPE
addr count ---
Transmit count characters from addr to the selected output device.

U.
u ---
Print a number from an unsigned 16 bit value, converted according to the numeric BASE. A trailing blank follows.

U*
u1 u2 --- ud
Leave the unsigned double number product of two unsigned numbers.

U/
ud u1 --- u2 u3
Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

U<
u1 u2 --- f
Leave the boolean value of an unsigned less-than comparison. Leaves f = 1 for u1 u2; otherwise leaves Ø. This function must be used when comparing memory addresses.

UDG
n1 n2 n3 n4 n5 n6 n7 n8 n9 ---
Creates a user defined graphic character n9 where n9 lies between Ø and 15 and is equivalent to graphics shifted A to P. n1 is the bottom byte and n8 is the top byte of the 8 X 8 pixels making up the character.

UNTIL

f --- (run time)
addr n --- (compile)
Occurs within a colon definition in the form
BEGIN ... UNTIL
At run time UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false execution returns to just after BEGIN; if true, execution continues ahead.
At compile time UNTIL compiles (OBRANCH) and an offset from HERE to addr. n is used for error tests.

UPDATE

Marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disc should its buffer be required for storage of a different block.

USE

--- addr
A variable containing the address of the block buffer to use next as the least recently written.

USER

n ---
A defining word used in the form
n USER cccc
which creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.

VARIABLE

A defining word used in the form E
n VARIABLE cccc
When VARIABLE is executed it creates the definition cccc with its parameter field initialised to n. Whn cccc is later executed, the address of its parameter field (containing n) is left on the stack so that a fetch or store may access this location.

VOC-LINK

--- addr U
A user variable containing the address in the field of a definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETting through multiple vocabularies.

VOCABULARY E

A defining word used in the form
VOCABULARY cccc
to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence cccc DEFINITIONS will also make cccc the CURRENT vocabulary into which new defintions are placed. cccc will also be so chained as to include all definitions of the vocabulary in which cccc is itself defined. All vocabularies ultimately chain to FORTH. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.

VLIST
List the names of the definitions in the context vocabulary. "BREAK" will terminate the listing.

WARM
Perform a warm start.

WARNING
--- addr
A user variable containing a value controlling messages. If = 1 then screen 4 is is the base location for messages. If = Ø then messages will be presented by number. If = -1 then execute (ABORT) for a user specified procedure. See MESSAGE and ERROR.

WHILE
f --- (run time) P.C.
addr1 n --- addr1 nl addr2 n2
Occurs in a colon definition in the form
BEGIN ... WHILE (true part) ... REPEAT
At run time WHILE selects conditional execution based on boolean flag f. If f is true (non-zero) WHILE continues exectuion of the true part through to REPEAT, which then branches back to BEGIN. If f is false (zero) execution skips to just after the REPEAT exitting the structure. At compile time WHILE emplaces (OBRANCH) and leaves addr2 of the reserved offset. The stack values will be resolved by REPEAT.

WHITE
--- 7
Leaves the numerical value of WHITE on the stack for use by PAPER, INK and BORDER.

WIDTH
c ---
A user variable containing the maximum number of characters saved in the compilation of a definition's name. It must be 1 throguh 31 with a default value of 31. The name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits.

WORD
c ---
Read the text characters from the input stream being interpreted until a delimiter c is found, storing the packed character string beginning at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or mor blanks. Leading occurrances of c are ignored. If BLK is zero text is taken from the terminal input buffer, otherwise from the disk block stored in BLK. See BLK and IN.

XOR
nl n2 --- xor
Leave the bitwise logical exclusive-or of two values

YELLOW
--- 6
Leaves the numerical value of YELLOW on the stack for use by PAPER, INK and BORDER.

[P
Used in a colon definition in the form
: xxx [words] more ;
Suspend compilation. The words after [are executed, not compiled. This allows calculation or compilation exceptions before resuming compilation with]. See LITERAL.]

[COMPILE] P.C

Used in a colon definition in the form

```
: xxx [COMPILE] FORTH ;
```

COMPILE will force the compilation of an immediate definition that would otherwise execute during compilation. The above example select the FORTH vocabulary when xxx executes, rather than at compile time.

] Resume compilation to the completion of a colon defintion. See [

APPENDIX 2

EDITOR
GLOSSARY

TEXT c ---
Accept following text to PAD. c is text delimiter.

LINE n --- addr
Leave address of line n of current screen. Thjis address will be in the disk buffer area.

WHERE nl n2 ---
n2 is the block no., nl is offset into block. If an error is found in the source when loading from disk, the recovery routine ERROR leaves these values on the stack to help the user to locate the error. WHERE uses these to print the screen and line nos. and a picture of where the arror occurred.

R# --- addr
A user variable which contains the offset of the editing cursor from the start of the screen.

#LOCATE --- nl n2
From the cursor position determine the line-no n2 and the offset into the line nl.

#LEAD --- line-address offset-to-cursor

#LAG --- cursor-address count-after-cursor-till EOL

-MOVE addr line-no ---
Move a line of text from addr to line of current screen.

H n ---
Hold numbered line at PAD.

E n ---
Erase line n with blanks.

S n ---
Spread. Lines n and following move down. n becomes blank.

D n ---
Delete line n but hold in PAD.

M n ---
Move cursor by a signed amount and print its line.

T n ---
Type line n and save in PAD

L ---
List the current screen.

R
n ---
Replace line n with the text in PAD.

P
n ---
Put the following text on line n

I
n ---
Spread at line n and insert text from PAD.

TOP

Position editing cursor at top of screen.

CLEAR
n ---
Clear screen n. Can be used to select screen n for editing.

FLUSH

Write all updated buffers to disk. This has been modified to cope with an error in the Micropolis CPM disk drivers.

COPY
n1 n2 ---
Copy screen n1 to screen n2.

-TEXT
addr1 count addr2 --- f
True if strings exactly match.

MATCH
cursor-addr bytes-left-till-EOL str-count
--- tf cursor-advance-till-end-of-matching-text
--- ff bytes-left-till-EOL
Match the string at str-addr with all strings on the cursor line forward from the cursor. The arguments left allow the cursor R# to be updated either to the end of the matching text or to the start of the next line.

1LINE
--- f
Scan the cursor line for a match to PAD text. Return flag and update the cursor R# to the end of the matching text, or to the start of the next line if no match is found.

FIND

Search for a match to the string at PAD, from the cursor position till the end of the screen. If no match is found issue an error message and reposition the cursor at the top of the screen.

DELETE
n ---
Delete n characters prior to the cursor.

N

Find the next occurrence of PAD text.

F ---
Input following text to PAD and search for match from cursor position till end of screen.

B ---
Backup cursor by text in PAD.

X ---
Delete next occurrence of following text.

TILL ---
Delete on cursor line from cursor to end of following text.

C ---
Spread at cursor and copy the following text into the cursor line.

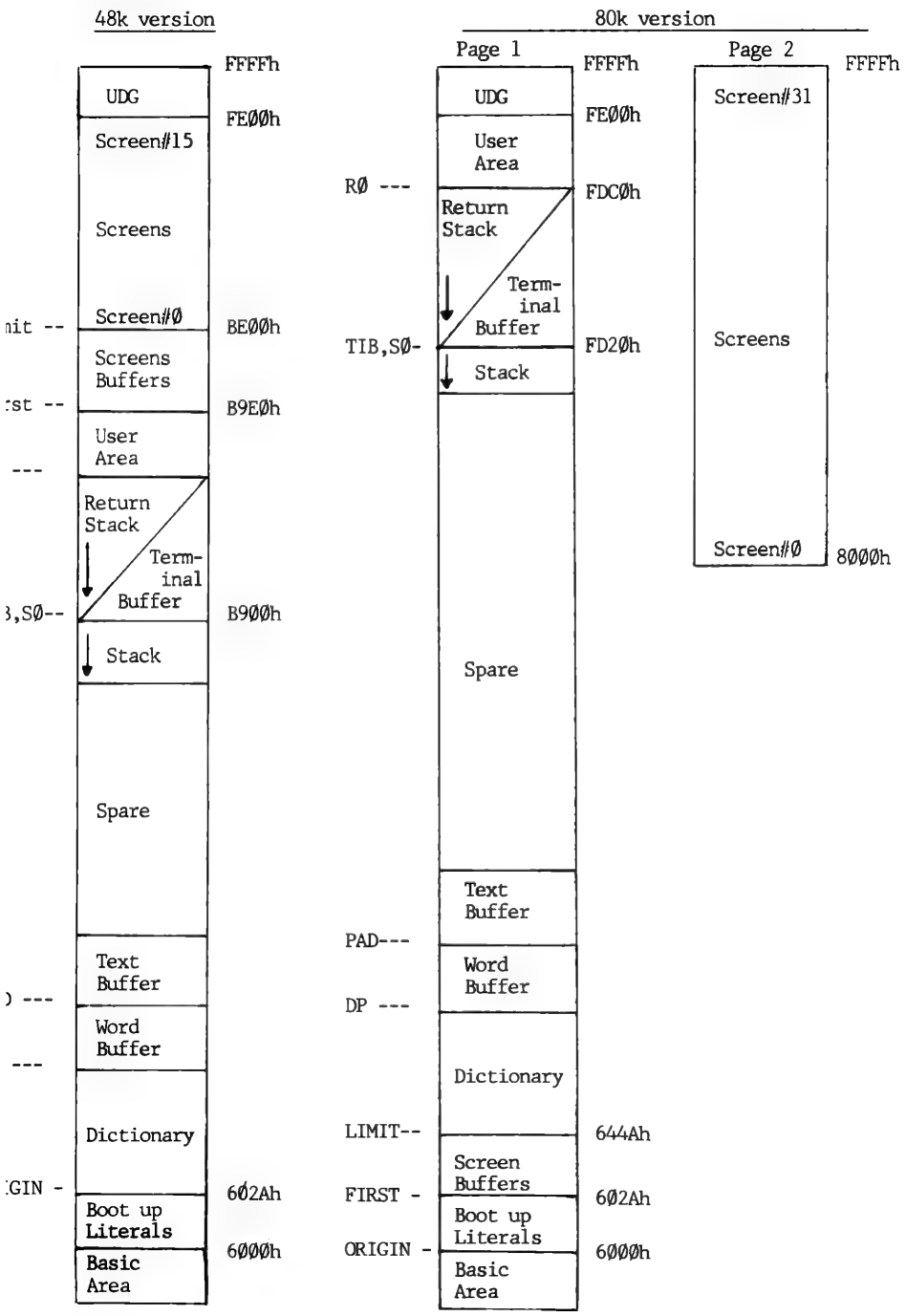
APPENDIX 3

ERROR MESSAGES

MSG #Ø	Word not in dictionary
MSG #1	Stack empty.
MSG #2	Dictionary full
MSG #3	Has incorrect address mode
MSG #4	Word not unique
MSG #6	Screen number out of range
MSG #7	Stack full
MSG #1Ø	Tape name too long
MSG #17	Compilation only. Use in a definition
MSG #18	Execution only
MSG #19	Conditionals not paired
MSG #2Ø	Definition not finished
MSG #21	In protected dictionary
MSG #22	Use only when loading
MSG #23	Off current editing screen
MSG #24	Declare vocabulary

APPENDIX 4

MEMORY MAPS



APPENDIX 5

BIBLIOGRAPHY

This is not intended to provide a definitive list of books on Forth. There are many books available covering all aspects of the language and catering for the beginner and advanced programmer. Some of them are based on a Forth implementation for a specific computer, but those in this list contain material of a general nature and can therefore be recommended to all.

Many magazines carry articles and/or programs in Forth. In particular the August 1980 issue of BYTE should be mentioned, which devoted most of that issue to Forth. Also the Forth Interest Group UK publishes "FORTHWRITE" the journal of the group, six times a year. Serious Forth programmers will wish to join. The address is

Roger Firth, membership secretary,
24 Western Ave,
Woodley,
Reading

Here is a list of recommended books.

Starting Forth	Leo Brodie	Prentice Hall
Using Forth	Leo Brodie	
Forth Programming	Leo J Scanlon	Howard Samms
Complete Forth	Alan Winfield	Sigma Tech Press
Invitation to Forth	H. Katzan	Petrocelli Books
Discover Forth	Thom Hogan	McGraw Hill
Threaded Interpretive Languages	R.F. Loeliger	McGraw Hill
Forth Encyclopedia	Mitch Derick	Mountain View Pass
Forth Theory and Practice	De-Grandis Harrison	Acorn-Soft
Introduction to Forth	Ken Knecht	Howard Samms

APPENDIX 6

REGISTER
USAGE

If you intend to write machine code definitions using CREATE or ;CODE it is important to know how the Z80 registers are used. The following table shows this.

FORTH	Z80	Forth Preservation Rules
IP	BC	Should be preserved across Forth words.
W	DE	Sometimes output from 'NEXT' may be altered before jumping to NEXT. Input only when DPUSH called.
SP	SP	Should be used only as data stack across Forth words. May be used within Forth words if restored before NEXT.
	HL	Never output from NEXT. Input only when HPUSH called.

The start of the inner interpreter code looks like this:

		48k address	80k address
DPUSH :	PUSH DE	602Ah	644Ah
HPUSH :	PUSH HL	602Bh	644Bh
NEXT :	etc	602Ch	644Ch

This gives you enough information to write machine code definitions of words.

APPENDIX 7

COLD/WARM START PARAMETERS

This appendix gives the source code for the first 34 bytes of FORTH, starting at address 6000h, and explains why various locations are stored with certain values, before saving an extended dictionary.

```
Address

6000        NOP
6001        JMP     COLD        ; jump to cold start
6004        NOP
6005        JMP     WARM        ; jump to warm start
6008        DEFB    REL         ; release no.
6009        DEFB    REV         ; revision no.
600A        DEFB    USRVER      ; user version no.
600B        DEFB    ØEH         ; implementation attributes
600C        DEFW    TASK-7      ; topmost word in Forth vocab
600E        DEFW    ØCH         ; backspace character
6Ø1Ø        DEFW    INITRØ      ; initial user pointer
6012        DEFW    INITSØ      ; initial address for SØ
6014        DEFW    INITRØ      ; initial address for RØ
6016        DEFW    INITSØ      ; initial address for TIB
6018        DEFW    IFH         ; initial WIDTH
601A        DEFW    Ø           ; initial WARNING
601C        DEFW    INITDP      ; initial FENCE
601E        DEFW    INITDP      ; initial DP
6020        DEFW    FORTH+8     ; initial VOC-LINK
```