FLOATING POINT FORTH

USER GUIDE

SPECTRUM FLOATING POINT FORTH

USER MANUAL

Contents

Floating-Point Forth Program
Mike Hampson © 1983
User Manual by: F.A. Vachha 1983
Published by C.P. Software, 17 Orchard Lane, Prestwood, Gt. Missenden, Bucks., HP16 0NN, England.

1. WHAT IT'S ALL ABOUT

1.1 FORTH is the most modern of popular computing languages. It was created in the early 1970s by Charles Moore and Elizabeth Rather at the National Radio Astronomy Observatory, USA. "FORTH" is not an abbreviation or a mnemonic (it's actually the trademark of FORTH Inc, California) — it just sounds good!

1.2 Technically speaking FORTH is a stack-orientated language which uses reverse Polish notation. It has won over a large following because of its speed, economical use of computer memory and the way in which it uses simple building blocks to construct complex structures. FORTH is intermediate in speed and compactness between BASIC (slow, space inefficient) and machine code (fast, compact)

1.3 Floating Point Forth (also called FP50) is an implementation of the language for the 48K ZX Spectrum Microcomputer. A little explanation is called for at this stage — isn't the Spectrum a BASIC computer? If so, how can it use FORTH?

The answer is that the Central Processing Unit (CPU) of the Spectrum, which is effectively the only part of the Spectrum which "thinks", is a ZX80A microprocessor chip — it understands neither BASIC, FORTH nor anything else except (its own) machine code. When switched on, the CPU automatically starts executing machine code commands which reside in the ROM. These instructions allow the computer to accept, edit and aexecute programs in BASIC — all BASIC commands are 'interpreted' by the CPU (using the ROM program as reference) and executed step by step.

In conclusion, then, the machine code program in ROM *supports* the BASIC handling capability of the Spectrum. You need not concern yourself at all with the machine code program operation if you want to write and run BASIC programs.

In a similar vein, FP50 is *supported* by a BASIC and a machine code program — these, together with some data, are what you load into the Spectrum at the start. These reside in RAM, we thought having to change your Sinclair ROM chip to get FORTH was rather drastic. As before, you need not concern yourself with their operation. They are "user-transparent" — ie, you need not even know they exist (except in special cases, say when you BREAK out of the program) in order to program effectively in FORTH.

Well then — the Spectrum is no less A FORTH computer than a BASIC computer. It's just that the BASIC-handling system is in ROM while the FORTH-handling system (ie FP50) loads into RAM.

1.4 Why use FORTH?

Programs in FORTH can run up to one hundred times faster than those in BASIC — FORTH brings you right to the boundaries of machine code speed. Arcade games come alive in FORTH — as do all the other program applications for which BASIC is just not good enough.

Programs in FORTH are much more compact — for example, an adventure in FORTH could probably have thrice as many locations as one in BASIC (with instant keyboard response as a bonus!).

Programming in FORTH is far more instructive and educational than it is in BASIC. FORTH is at least as easy to learn as BASIC; however it lends itself to "structured" programming which BASIC does not. Many computer education courses have switched from the traditional BASIC — base to FORTH. While FP50 is user friendly, using it gives discipline to your programming (BASIC certainly does not) — this discipline is very useful if you decide to learn other high level languages like PASCAL, COBOL or FORTRAN. Further, as FORTH is all about stack manipulation (explanations later) it is essentially similar to machine code. Mastery of FORTH is hence a very useful stepping stone to the lower level languages like assembler and machine code itself.

Lastly, because it's fun!

1.5 FP50 contains all the structures of Forth 79 (the industry standard). This means that once you understand FP50 you will be able to use most other versions of Forth.

Note, however, that software written in Forth 79 (or in FIG Forth, the next most common version) may need some conversion to run on FP50.

There is a lot of FORTH software available. Currently, there is at least one FORTH home microcomputer available in the UK, and program listings for it can be converted to FP50 and run on the Spectrum.

Note that the special feature of FP50, normally available only on expensive commercial packages, is its ability to handle floating point numbers (as opposed to integers only). Other versions of FORTH available for the Spectrum can handle numbers from -32767 to +32767 with an accuracy of zero decimal places. FP50, however, can handle numbers from -2E127 to + 2E127 (ie-1.7E38 to +1.7E38) with a maximum accuracy of nine decimal place

SUMMARY OF CHAPTER 1

+ FORTH is a new(ish), fast, compact structured language.
+ FP50 allows you to run near — standard Forth programs on your Spectrum, by providing a Forth operating system which replaces BASIC.

2. FORTH FUNDAMENTALS

2.1 If you are not familiar with FORTH, it is strongly recommended that you read this section before loading the cassette. Because FORTH operates on principles quite different from those of BASIC, it is unlikely you will make the computer do anything productive in FORTH until you have grasped the elements of this new language.
If, however, you like to learn the hard way, load the program using LOAD" " and start. When you finally manage to crash the program, refer to Section 3.3 to get you restarted. Once your appetite for exploring the unknown has been whetted, return to this section, and do things the right way around.

2.2 There are two concepts fundamental to the understanding of any FORTH dialect — the concept of the WORD and that of the STACK. Before going any further let us explain what these are.

2.3 THE WORD
Everything you enter into the computer is a WORD, irrespective of whether it is a command, number, symbol, name or variable.
The principle of FORTH programming is that you teach the computer new words to enable it to perform the tasks that you want.
To begin, the computer starts off with a number of words it knows already (you load these into the computer as "data" — see 3.1). For a list of all the words initially in its vocabulary, refer to Appendix A and Appendix B.
Now while these words are very useful, you cannot do very much with them alone. If you want to write a program that will play chess (or space invaders!) you will not find an existing word that will do this. What you will have to do is teach the computer new words, defined in terms of words it already knows (you can define a new word ONLY in terms of words previously defined). A single new word can combine the effect of five or ten or twenty existing FORTH words. Further, you can define even newer words based on the new words, and so on. In the end, your whole program will be just one WORD, defined in terms of many other words (all of which are defined in the dictionary).

2.4 It is vital that you grasp the concept of the WORD in FORTH. A comparison with BASIC might help you to do this:-
A program in BASIC *does* consist of words, such as LET, FOR, NEXT, STEP, GOTO etc. But your program has to be constructed *directly* from these words: you cannot define your own words from them (if you try, you will get an error message).

A program in FORTH can consist of a mixture of original words, and words you have defined yourself. This gives you far more flexibility.

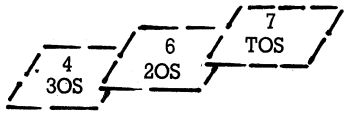
The closest that BASIC can come to this is by using subroutines. If in your BASIC program you wanted a command to paint your screen with alternate black and white squares, you could do this by writing a subroutine (starting at 2000, say) and then using the command GOSUB 2000 at the appropriate stage of the main program. In a sense, "GOSUB 2000" is a word that YOU have defined, one that will accomplish the specific task you designed it to do. In FORTH you would call the word something appropriate (like PATTERN) and define it in the way described later. Every time the computer met the word PATTERN it would paint the pattern on the screen.

2.5 THE STACK

A fascinating characteristic of FORTH is the stack. The stack contains numbers. All the numbers used for calculations, as well as the results of these calculations, are stored on the stack.

Visualise the stack as a pile of cards with numbers written on them. The last card you put on the pile is always the first card that you take off. The operation of the stack can hence be described as LIFO — Last In, First Out. Each time you put a new number on the stack, the number which previously was at the top of the stack is now second on the stack.

Throughout the manual, the term TOS is used to represent the item on the top of the stack, 2OS the next item on the stack, 3OS the third item on the stack (counting from the top), etc. This diagram should help you picture the stack.



That was simple, wasn't it?

2.6 The numbers you use on the stack can be integers (3, -5 decimal point numbers (3.8, -0.7) or be in scientific notatio (+1.7E19, 2.8E-5).

The range of permitted numbers and their format are exactl the same as in BASIC (ie, top limit about +1.7E38, bottom limi about -1.7E38).

2.7 At this stage it should be pointed out that Section 2.5 actually oversimplifies the situation: FP50 really uses three stacks, not just one. They are the data/user stack, the return stack and the calculator stack. But do not worry — the only stack you need to know about at this stage is the first one, and whenever the word 'stack' is used alone it refers to this one. The other two stacks are described later, in Chapter 8.

SUMMARY OF CHAPTER 2

+ Two important FORTH concepts are those of the WORD (any command, number, symbol etc) and the STACK (a LIFO operating pack of cards, which can contain numbers only)
+ TOS represents the item on the top of the stack
  2OS represents the item immediately below TOS
  3OS represents the item immediately below 2OS etc.

3. GETTING STARTED

3.1 The professionally duplicated cassette supplied with this manual contains a copy of FP50 on each side. To begin, restart the Spectrum by executing RANDOMISE USR ∅ (this simulates a power-off) and then enter LOAD" "
FP50 loads automatically, in 3 parts (A program (fforth) bytes (data) and bytes again (routines)). Loading takes about 1 min 45 seconds — once it is complete a picture materialises, a short tune plays and a prompt appears at the foot of the screen.
The prompt is ***>, followed by a flashing cursor (L in lower case, C in Caps mode — you can switch between these two in the usual way).
This signifies that you are in the command mode of operation — FP50 is waiting for you to tell it what to do next. If you wish to use the editor, refer to chapter 7 at this stage.
(most words resident in the FP50 dictionary must be entered in capitals. See Appendix A and Appendix B for exceptions).

3.2 You are now almost ready to begin. There are just a couple of rules of syntax you need to know about — they will help prevent you misunderstanding or wrongly implementing instructions given later on.

a. Between each work there MUST be a separator. There are two permitted separators — one is the ENTER key and the other is the SPACE key. They have almost exactly the same effect — use whichever one you prefer.
The only difference between SPACE and ENTER is that the latter not only separates words, but also gets the computer to "compile" (translate into simple instructions) all words entered so far. The SPACE key only separates words.

b. Be careful about spaces between words when entering examples from the manual into the computer. 90% of errors arise from missed spaces. Spaces in the manual examples are shown by a blank area between words. There is (unfortunately!) a FORTH word called SPACE: this is always spelt out with 5 letters in the manual. (See 3.7).

c. Do not use quotes within quotes or embedded brackets.

d. Commands in upper case will not be understood if entered in lower case, and vice versa.

e. Do not use any of the Spectrum keyboard tokens (like SYMBOL SHIFT and Y to give you AND) — FP50 recognises none of these. Type in everything in full.

3.3 WHEN THINGS GO WRONG
If and when things go wrong and the program crashes, you

will be returned to BASIC and an error message will appear at the foot of the screen.
Sources of errors could be using of invalid colours, performing operations that are arithmetically impossible, replying 'N' to the screen prompt "Scroll?" using BREAK etc.
To get back into FP50, enter either GOTO 4051 or GOTO 3 (with the latter you get the screen picture and tune again). You will NOT lose any words you have already defined.

3.4 USING FP50

Let us now begin. You should have, as stated in 3.1, a "***>" at the bottom of your screen, followed by a flashing cursor. If you have been playing with the program and have 'lost' this message, input a semicolon followed by pressing ENTER ***> will reappear.

3.5 We shall now put some numbers on the stack. Let us first put the number 7 on the stack. We do this by pressing the key 7, the space key (or ENTER — see 3.2a) the semicolon key and then ENTER.
This looks like:
***>7 ;
The semicolon marks the end of an instruction, and, provided the instruction is valid, semicolon followed by ENTER will first execute the instruction and then return you to command mode.
We have put the number 7 on the top of the stack (ie, TOS=7) If you were starting from scratch, the stack would previously have been empty — there would be only 1 item on the stack now. Hence the computer displays:
Stack:1 ———OK (had it been the 5th item, Stack:5 would have shown) and then the command mode prompt:
***>
Note — no spaces should be entered between digits/decimal points/—//exponent signs: if space is entered the computer will assume you are trying to input two distinct numbers.
Let us put more items on the stack.
***>3.7 ; (a decimal, use the . as in BASIC)
***>1E-3 ; (scientific notation)
***>-.05 : (negative)
***>4 5 11 ; (three numbers at once — this leaves TOS=11, 2OS = 5 and 3OS=4. You can enter any amount of numbers like this.
Note — no spaces between the numbers and decimal points or between the numbers and symbols.
Notice that the computer displays as a memo the number of items on the stack every time you enter a number. This is

useful feedback: FP50 can cope with a maximum of 300 stack items at one time (or 150 if the editor ED50 is in operation). Overflow is taboo and will cause an error. Do not worry if your stack becomes big — it is very easy to remove 'garbage' from it, as you will see later.

3.6 Now let us see how to get the computer to take items off the stack and print them on the screen. With the space key, ENTER key and the ';' were all markers of a sort, they were not Forth words (see **). You shall now meet your first FORTH word, the dot (the fullstop on the Spectrum keyboard, the same one you used as a decimal point in the examples above)

. makes the computer print out TOS, ie, the last number you put in. So try ***>. ; followed by the ENTER key, as usual — did you remember to use the space between the two symbols?

The computer will now have printed out the number on the top of the stack (11, if you've followed these instructions exactly). Not only does it print this number, but it also 'forgets' it — the stack becomes shorter, as the memo — message indicates.

3.7 Now try taking more numbers off the stack, either one by one or several at a time. You will see them coming off in the opposite order to that in which you entered them — LAST IN, FIRST OUT, remember?

***>3 8 10 201 ; (to put 3, 8, 10, 201 on the stack) and then you enter

***>. . . . ;

The computer will print out 2011083, which is in the order you expected (TOS first, so 201 discarded to leave 10 as TOS, etc) — but why no spaces?

Well, the spaces you input between the dots in the command served only to separate the commands (if you had omitted one or more of them, you would have got the message — not known.)

To separate the output; use the FORTH word SPACE (a five letter word, to be keyed in — do NOT use the space key!) Now try

***>3 8 10 201 ;

followed by

***>. SPACE .SPACE . ;

and you will get the output in a more usable form.

3.8 If you continue taking items off the stack you will soon reach a stage when there are none left. Do *not* continue to take items off the stack after this. Though the computer will permit this (showing the size of the stack as -1. -2. -3 etc) you are

encroaching on the VARIABLES area and will corrupt it. Instead, put some items back on, even if they are dummies.

3.9 You will have noticed that after you have ENTERed your instruction next to the ***> ,indicator, they disappear, the message 'Compiling — please wait.' appears, and the instruction is repeated step by step, before being executed. This indicates normal operation of FP50.

If a message of the '—not known' type appears, you have input a word that does not appear in FP50's dictionary. Either you have not yet defined a word you intended to (see Chapter 5) or you misspelt a word or omitted a separator between two words, or between a number and a word.

If this happens, do not be perturbed. Simply retype in a corrected form, the characters shown before the '—not known' flag. Do NOT retype the entire command.

For example, ENTER

***>5 6 7; [omitting the space between the 7 and the ;] The computer will show "7;- not known. Continue definition". Now you should ENTER only 7 ; whereupon the Spectrum will complete the operation, display an OK message and, as usual, return you to command mode. Had you entered the whole 5 6 7 ; in reply to the continue definition prompt, the computer would have implemented ***>5 6 5 6 7 ; which is not what you wanted.

3.10 Note that you will get the "continue definition:" prompt whenever an instruction is incomplete, and not only when the cause of the incompleteness was an undecipherable word (as was 7; in 3.9). Remember that ALL instructions MUST be terminated by the FORTH word ";". This explains why the method is 3.4 (return to command mode) words.

3.11 USING A PRINTER

If a printer is connected to your Spectrum, the following three commands enable you to use it [All are in lower case]

***>z ;	Accomplishes a screen COPY
***>pron ;	Switches the printer on — all output is now directed to the printer and *not* the screen,
***>proff ;	Switches the printer off (ie, restores normality)

Interestingly enough, these three commands are not really FORTH words they are part of the FP50 operating system although they could have been defined as such. The only real difference this makes to you is that you cannot make the computer 'forget' them which you can do with FORTH words. It is not advisable for you to use pron, proff, z or ; as the names of FORTH word that you define, while these names

could be used it may lead to confusion.

SUMMER OF CHAPTER 3

+ ***> represents command mode.
+ Use a space of ENTER to separate words — this is essential.
+ Use a ; followed by pressing ENTER to implement an instruction
+ If the instruction is a number, or a sequence of numbers, they will be entered in order on to the stack (ex ***>5 ;)
+ ***>. ; prints by the last item on the stack. You can print a number of items by using . . . (with spaces between)
+ The word SPACE prints a blank space between successive numbers taken off the stack — its effect is completely different from the space key.
+ If a ' — not known' message appears, retype correctly the specified word (s) and not the whole instruction.
+ The command z, pron and proff accomplish COPY, Printer On and Printer Off respectively.

4. THE NEXT STEP — CALCULATIONS

4.1 One of the first things you did when learning BASIC was to use the computer as a calculator — doing things like PRINT 2+3* 4/5, etc. Not only did this give you confidence but it also made you familiar with the mathematical syntax rules of the computer. Let us do the same with FORTH and your FP50.

4.2 If you know a little about FORTH you probably know all implementations of its use. Reverse Polish Notation (RPN). RPN is reputedly difficult but in fact quite easy. If Hewlett Packard (all of whose calculators use RPN) had priced themselves down to the Japanese level, we would all be quite familiar with it.

RPN uses the concept of numbers first, then operators (the mathematical operators are +, -, ×, /, $\sqrt{\ }$, SIN etc etc.). On most calculators, you would add 2 and 3 together by pressing 2, + 3, followed by = or ENTER.

On an RPN calculator, to accomplish the same task you would do 2 3 + (followed by = or ENTER, depending on the calculator). If you think about it, RPN is very logical — it mimics what you do when adding two numbers. You don't think of the first (2), then perform some magical addition, and then think of the second (3) and then the answer.

What you do is to think of first one number, then the second, and then perform addition on them. Which is exactly the order RPN handles it.

4.3 Let us now use FP50 to perform some calculations. ENTER ***> 2 3 + . ; (note, if stack: Ø before calculation, then stack: Ø after) and the answer 5 duly appears. It is important to understand exactly what the computer did just now. In sequence, it is

2 (i) Put 2 on top of the stack (so TOS = 2)
3 (ii) Put 3 on top of the stack (so TOS = 3, 2OS = 2)
+ (iii) Took 3 off the stack, and then 2, and then added them together putting the answer, 5, on top of the stack.
. (iv) Took 5 off from the top of the stack and printed it on the screen.

Note that the stack has not changed as a result of this instruction whatever we put on it, we have taken off.

Had we instead done ***>2 3 + ; we would have, of course, left 5 on top of the stack.

This is confirmed by the stack memo, see 3.5.

4.4 Note that numbers already on the stack can be operated on in a very similar way. ENTER.

***> 7 6 5 4 3 2 1 ; which puts 7 numbers on the stack (TOS=1) Now ENTER

***> + + + + + + . ; the answer 28 is displayed (and the

stack length went down by seven).
What the first + did was take TOS (=1) to 2OS (=2), remove both of them from the stack, and replace them by their sum 3. Now this 3 (=TOS) is added to the 3 you entered (which was 3OS, but is now 2OS because you took two numbers off the stack and only put one back on), etc etc.
1 + 2 = 3, 3 + 3 = 6, 6 + 4 = 10, 10 + 5 = 15, 15 + 6 = 21 and 21 + 7 = 28.
Of course, it's simpler to think of it as just 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28.
In this case (because the operation was addition) the order in which the operations were done is not important — but it could be, so make sure you have understood this.

4.5 Of course, + is not the only mathematical operation available on FP5∅. For a complete list of all mathematical operations (as well as conditionals and other commands) refer to Appendices *A and B*.
Here are some more examples of FP5∅ working as a calculator. If any of the operations are unfamiliar look them up in Appendix A.
In each case, the BASIC equivalent is given
***> 7 -5 + . ; is the same as PRINT 7 + (-5) ie 2
***> 7 15 — . ; is the same as PRINT 7 - 15 ie -8
***> -11 60 * . ; is the same as PRINT (-11) * 60 ie -660
***> 80 7 / . ; is the same as PRINT 80/7 ie 11.428571
***> 2 5 ↑ . ; is the same as PRINT 2 ↑ 5 ie 32
***> 1.8 COSR . ; is the same as PRINT COS 1.8 ie -0.22720209
***> 2 SQR . ; is the same as PRINT SQR 2 ie 1.4142136
***> -2.9 INT . ; is the same as PRINT INT (-2.9) ie -3
***> -2.9 ABS . ; is the same as PRINT ABS (-2.9) ie 2.9
***> 0.38 LN . ; is the same as PRINT LN (0.38) ie -0.96758403
***> 20 EXP . ; is the same as PRINT EXP (20) ie 4.8516519E+8
***> 1 ATNR . ; is the same as PRINT ATN (1) ie 0.78539816

4.6 Let us now perform some compound calculations, involving more than one operation. Again, consult 5. If in any doubt as to the order in which the operators act on the operands (numbers).
***>5 20 4 */ . ; (Try to predict the answer before trying it out) As * precedes / , it is the first to operate 20S(=20) is hence multiplied with TOS (=4) to give 80. 20 and 4 vanish from the stock and 80 is now TOS, with 5 as 20S. The division is now performed. 20S(=5) is divided by TOS(=80), 5 and 80 vanish, and the result, 0.0625, is made TOS. Lastly, the answer is removed from the stack and is printed out.

Follow the same reasoning with ***>7 5 40 — / . ; and deduce that the correct answer is —0.2 ie 7/(5-40).
Let us and try and compute the following in FORTH (2+3)*7 ie 57 ie 35.
Looking at the brackets, the addition must be performed first. One would do this by 2 3 + ; if it were the only task this leaves 5 as TOS. Now 5 is to be multiplied by 7 , and the order in which this is done is irrelevant (because* is commutative). Hence
***>7 2 3 + * ; will put 35 as TOS
***> 2 3 + 7 * ; will do exactly the same
Use . ; to print out TOS
Say however, we wished to compute ((2 + 3) * 7)/4 which is 35/4 ie 8.75.
***>4 7 2 3 + * / . ; does not work, because after performing the + and * the computer is effectively left with 4 35 / . ; This gives 4/35 and not 35/4
[Remember / divides 20S by TOS and not vice versa]
In order to compute ((2+3)*7)/4 we should hence do ***> 7 2 3 + * 4 / . ; this prints out 8.75, which is correct! (2 3 + 7 * 4/is also correct)
As a final exercise, let us compute ((3 4) / (5-6) 8 which is 7/-1 8 or -7 8 or -56. Now 3 4 5 6 — leaves 7 as 20S and -1 as TOS
Since these 2 have to be divided, and the result multiplied, by 8, the following instruction does the trick.
***>3 4 + 5 6 — / 8 * . ;
Note that the numbers have not changed their relative positions — they were in the 3, 4, 5, 6, 8 "sequence" in the question too. So when working out the FORTH RPN equivalent of a computation, leave the numbers in the same order — its just the operation signs ("+", "—" etc) that move around.

4.7 Later on you are going to see that FP50 includes an integer arithmetic system as well as a floating point system. Of course floating point can do everything that Integer can (and much more besides) but Integer is much faster. Refer to Chapter 6 for details on integer mode operation, and to Appendix B for all integer commands.
There is no advantage to using integer arithmetic in the calculator mode (where speed is of little consequence), but if you wish to, use only pure integers (from 0 to 65535) and utilise the %+, %—, %* and %/commands.

SUMMARY OF CHAPTER 4

+ FORTH uses reverse polish notation for computations: This can be summarised as numbers first, operators later.

5. PROGRAMMING IN FORTH

5.1 The last two chapters introduced you to FP Forth, RPN, stack handling and the use of the machine for computations. It is very important that you should have mastered them before continuing with this chapter.

5.2 The purpose of this chapter is to take you through the steps of FORTH programming, and not to introduce you to the predefined words in the dictionary. These can be found in Appendix A (and B for integer FORTH words, explained in the next chapter), together with a description of operation and an example. This chapter will enable you to use the appendices effectively. Each time a new word is mentioned, look up its meaning in Appendix A and try it out.

5.3 EXAMINING THE DICTIONARY

To see all the words in the dictionary, enter VLIST; from command mode. The first FORTH word, STKSWP, appears, together with a flashing square.

This square means "press Y to continue" — you will meet it again. Press the Y key and all the FORTH words defined so far will scroll past. Well, not all of them, actually — some of them [like pron, proff, ;, SPACE, S, Z, UNTIL, BEGIN etc] are not literally defined in FORTH and hence do not appear. All words you define yourself will be added to this Vocabulary List, so VLIST is a very useful function to use.

5.4 DEFINING YOUR OWN WORDS

As was stated in 2.3, FORTH programming is all about defining your own words. For example, the game (see 8.1) at the end of the tape really comprises *seven* FORTH words, defined in terms of the original FORTH words.

To define your own FORTH words, proceed as follows:

a. From the command mode, type a colon followed by a space. Now name your word. Use any amount of characters — only the first six characters count. The name can comprise letters, numbers and symbols (but not keyboard tokens like CHR$, TAB etc) — all you have to ensure is that the name could not be mistaken for a number. So A56+ is a suitable name, 2 - 5 is not.

As an example, type : TEST (and press ENTER) from command mode. The word will appear at the top left of the screen along with its compilation address (see Appendix C for a memory map if you are interested — you need not do so). As you define more and more words, the address to which they are compiled increases. Once this reaches 65ØØØ, define no more words — your machine is full. (This is, in practice, very unlikely to occur).

Now enter the definition of the word TEST. The definition

must be in terms of FORTH words the machine already knows (just the same as while entering direct commands). Since we do not know much FORTH yet, it will probably be simple. Try entering, as the definition

6 4 3 2 + ... ; (press ENTER)

[As you know, this will put 6,4 and 3 + 2 on the stack (with TOS = 3 + 2) and then print them out in order ie. 546, leaving the stack as it first was.]

The word TEST will have been added to your computer's Vocabulary (use VLIST; to verify this). Every time you enter TEST, the computer will print 546. (You can use TEST in the definition of other new words too). Check this by trying ***>TEST ;

and then defining TEST 2 by

***> : TEST 2 9. TEST Ø . ;

Now try TEST 2

***> TEST 2 ;

The computer prints out 9546Ø — see how it has used TEST in performing TEST 2.

This is how one builds up a FORTH program — words on words on words, until in the end the entire program is but a word.

While defining words, you can do the following:-

1). Use a double space between 2 words so that the second one starts on a new line — this keeps the screen tidy. Note this option does not work on ED50 (Ch. 7).

2). Place comments within brackets — these will be ignored during execution. Comments can obtain spaces, quotes, punctuation signs etc.

Note that you cannot use as word names the names of existing words (both original FORTH words and ones defined by you) — the message 'Word Already Used' appears. It is first necessary to forget the word — see 5.5 below.

5.5 DELETING WORDS

You can delete words by entering, from command mode, f (for Forget) followed by a space and then the name of the word. The forget instruction will cause the computer to delete the named word *and* all the ones defined since that one. This can be a great disadvantage at times, especially if you wish to *change* the definition of a word defined some time ago, since which there have been many new definitions. FP5Ø will not let you directly redefine the word, but first forgetting it would cause all the subsequently defined words to be deleted as well. The solution to this problem is to use the editor ED50 (see 7.6) — it enables you to directly redefine.

Note that the Spectrum's initial words are *not* protected against forget — so be careful. Words in the dictionary list called by VLIST appear in the order in which they were defined, so forgetting (say) ?DUP which is about half way down will delete half the original FORTH words (including VLIST, unfortunately) — you will probably need to load again.

5.6 SAVing Words

You can save FORTH words by making a complete new copy of FORTH. From command mode enter a (lower case) 'S' press ENTER. Then, when prompted, enter a file name, maximum 9 letters, no spaces.

The program is saved in 3 blocks and at the start of each block you will have to press enter. Once saving is over, you are required to VERIFY. If this is OK you are returned to command mode — otherwise you should BREAK and GO TO 2010 to save again.

5.7 USER DEFINABLE GRAPHICS

There are a total of 117 UDGs available. 21 of these are Spectrum's standard UDGs from the G cursor. As well as these 21, the entire set of 96 keyboard characters are redefinable, and these 96 are saved and loaded with the FORTH whereas the other 21 are lost.

From command mode enter a lower case d (for define) and press enter. The existing characters are displayed and you are asked to enter the label (ie. name) of the character you wish to change. Enter it from the keyboard, using the G cursor if necessary. You will then be asked for eight lines of definition, and each line should consist of exactly eight ones and zeros, indicating black and white in the definition. You are then returned to command mode. Don't forget you can print out the UDG if you have a printer — see 3.11. NOTE, input errors can result in a BASIC error — to return to command mode enter GOTO 4.

5.8 THE ARITHMETIC WORDS

The precise operation of these words can be found from Appendix A:

+ - * / ↑ 1+ 1- 2+ 2- NEGATE ABS SGN INT SQR RND SINR COSR TANR ASNR ACSR ATNR LN EXP.

You should be familiar with the operation of most of these from Chapter 3.

5.9 THE LOGIC WORDS

The precise operation of these words can be found from Appendix A:

>< = >=< = <> Ø> Ø< Ø= MAX MIN AND OR NOT.

(The <> consists of a < and then a >, not the single Spectrum token). Note that >, <, =, >=, <= and >= all take

2OS as the first operand and TOS as the second. The two items disappear from the stack and are replaced by the result ie. ∅ if the statement is false and 1 if it is true. These are very useful with words like IF, WHILE and UNTIL.
The zero comparisons simply save typing out a space. If TOS = 5, for example, ∅< . ; will print out ∅ (false, because 5 < ∅ is false). AND, OR and NOT are very similar to their BASIC equivalents.

5.10 STACK MANIPULATING WORDS

The precise operation of these words can be found from Appendix A:
DROP DUP ?DUP DEPTH SWAP OVER PICK ROT ROLL.
All these words allow you to manipulate the stack effectively.

5.11 VARIABLES

There are 23 variables available in FP Forth. Initially they have names A, B, C . . . . X, Y, Z excluding I, J and K. All the variable names are words — check this using VLIST. You can hence rename any variable by defining a new word whose meaning is that variable. To rename say B as SCORE, use : SCORE B ;
Storing values in variables is done using '!', which should be read 'store'. For example, to set Z to -43 use -43 Z ! ;
Note that variables do not need to be declared as in Spectrum BASIC — they are all initially assigned and set to zero.
The reason why I, J and K are not available is that they are exclusively for use in loop control — refer to 5.13.
To read a variable onto the stack, use the command @, which should be read 'fetch'. Hence to set TOS to the value of E use E @; or to print out the value of E use E @ . ;
As an alternative to @ . you can use '?', which stands for read. This prints out the value of the variable — for example, E ? ;
Like the M+ key on a calculator there is an increment word, +! (no space between them). To increment the variable Q with 235, enter -235 Q+!;

5.12 CONSTANTS

Constants are words which put numbers on the stack. To define the word PI as 3.141592654, enter
: PI 3.141592654 ;
While there are similarities between constants and variables (see 5.11) there is no limit to the number of constants. The words which operate on variables, are +! and ? and @, will not work on constants.

5.13 LOOP WORDS

Just as in BASIC where loop controls is available using FOR. . .TO. . .NEXT, FORTH provides loop control using DO. . . .LOOP. DO takes TOS as the first looping value and 2OS as the

first illegal value (not as the last legal value as in BASIC). Hence FOR n = 1 to 100 becomes 101 1 DO.
Next is replaced by LOOP, and needs no operand. For example,

```
50 1 DO I . LOOP ;
```

will print out numbers from 1 to 49 (without spaces between them). The word I (for index) after DO puts the current top value on top of the stack.
This (and only this) sort of loop is nestable to any extent — I will always put the current loop value of the innermost loop on to the stack. Now try the following (look up EMIT in Appendix A)

```
256 32 DO I EMIT LOOP ;
```

This prints out the character set.
So far we have only dealt with steps of 1. If a step other than 1 is to be used, +LOOP and -LOOP should be used. They make I (the index, or loop control variable) increase or decrease by TOS instead of by 1. In these cases, TOS should be positive. Try the following (looking up definitions of words you do not yet know)

```
51 0 DO I . CR 5 LOOP
1000 1 DO I . FIELD I LOOP ;
0.1 0.9 DO I . Ø.1 -LOOP ;
31 127 DO I EMIT 1 -LOOP ;
```

Here the steps were 5, I (!), -0.1 and -1 respectively.
DOs and LOOPs should match, and (just as in BASIC) intersecting, non-nested loops are strictly taboo.
When using nested loops, I Innermost loop index, J Second to innermost loop index and K Third to innermost loop index, provided that the loops exist. These index names are not interchangeable, and their use is reserved to loops (they cannot act as ordinary variables — see 5.11). Try

```
8 1 DO 8 1 DO I J * . SPACE LOOP CR LOOP ;
```

LEAVE sets the index value of the innermost loop to the loops limit — as this is the first illegal value rather than the last legal one, control exists from the loop immediately LOOP is executed.
EXITLP, which can only be used within loops, causes the computer to abandon the current command.
Note that Integer loops are much faster to execute — see Chapter 6.
There is another type of loop, different from the DO. . .LOOP one. This is the BEGIN. . .UNTIL loop.
The operations between BEGIN and UNTIL are executed continuously, till UNTIL finds a true value on TOS. Try

BEGIN 127 EMIT INKEY 13 = UNTIL ;
This will print CHR$ (127) (ie. "©") till you input the character with code 13 (ie. the enter key). This is because = will put 1 on the stack only when INKEY has returned 13, and 1 is what UNTIL is waiting for (any number <> Ø on the top of stack counts as true).
A variation of this is BEGIN. . . .WHILE. . . .REPEAT. The operations between BEGIN and REPEAT are executed continuously so long as WHILE finds true values (ie. Ø) as TOS. Once WHILE finds a false value, control jumps out of the loop to the word after repeat. Try
8192 BEGIN DUP 1 > WHILE DUP . FIELD 2 / REPEAT DROP ;
This continues halving 8192 until the result is not >1 (hence the last number printed is 2, not 1).

5.14 BRANCHING WORDS

Like the IF. . .THEN available in BASIC, an IF. . .ELSE. . .THEN control is available in FORTH. As usual, it looks for a true or false on the stack (preceding the IF). Between IF and ELSE come all the words to be executed if TOS was true. Between ELSE and THEN come all the words to execute if TOS was false. Once the computer has executed one of the word sets, it executes the words following THEN. Try

```
5 5     IF 444 . ELSE 333 . THEN ;
5 5     IF 444 . ELSE 333 . THEN ;
```

Note that IF, THEN and ELSE must be used together. IFs are not nestable.
As with UNTIL (see 5.13) anything other than zero counts as true.

5.15 SOUND WORDS

The commands are BEEP and BLEEP, defined in Appendix A. They both take 2OS as duration and TOS as pitch. Try

```
13 Ø DO Ø.25 I BEEP LOOP ;
2ØØ Ø %DO 10 %I BLEEP %LOOP ;
```

The % sign prefixing the words indicates integer operation, which is much faster than FP. Refer to Chapter 6 for details.

5.16 PRINTING WORDS

The precise operation of these words can be found from Appendix A.
SPACE SPACES EMIT . FIELD CR CLS AT TAB
Note that to print a string, use . "the string you want to print"
Spaces, brackets and colour control characters are allowed.

5.17 COLOUR WORDS

The precise operation of these words can be found from Appendix A.
INK PAPER FLASH BRIGHT INVERSE BORDER PROVER

ATTR
PROVER is like a BASIC OVER, the prefix PR has been used to distinguish it from the stack manipulating word OVER.

5.18 PLOTTING WORDS

The precise operation of these high resolution graphics words can be found from Appendix A.
PLOT CIRCLE DRAW POINT
All except POINT reset the colour controls of 5.17.

5.19 INPUT WORDS

The precise operation of these keyboard words can be found from Appendix A.
INKEY KEY QUERY WORD PAD >IN WAIT
It is interesting to contrast INKEY, KEY and QUERY.

5.20 EXIT WORDS

The precise operation of these words can be found from Appendix A.
EXIT QUIT ABORT

5.21 GETTING IT ALL TOGETHER

By now (if you have been following the suggested approach) you will be familiar with a great number of FORTH words, all of which are defined in Appendix A. There are some more FORTH words (which appear on VLISTing) that have not yet been explained — words like STKSWP. This is intentional. You will only need to use such words if you are an advanced programmer, familiar with low level language concepts. If this is the case, refer to 8.2 for a detailed explanation.
As for the rest, you are now well enough equipped to go out and write marvellous FORTH programs. If this seems rather remote or improbable, look at, for example, the second routine in 5.15, which produced a delightful zap. Why not define a word, ZAP, with that routine (or a development of it) as its definition? The trick is building up your own words to do increasingly long and sophisticated things. Why not define a word RND() [or any name you choose] which will produce a pure integer random number in the range 1 to TOS. One way is by: RND() RND* INT 1 %+ ; Let us say you now do 5 RND(), and say the RND word yields 0.613. Now INT (5 * 0.613) + 1 = 4, which is what RND() will put on TOS. Its range is, of course 1 to 5 in this case.
Define a word LOG to give logarithms to the base 10, by : LOG LN 10 LN / ;
[Because LOG X LN X / LN 10]
Or draw a pair of axes by defining
: AXES CLS Ø Ø PLOT Ø 175 DRAW Ø 88 PLOT 255 Ø DRAW ; Each time you enter AXES ; you will get them drawn for you. Let us take this figure. Define CURVE which draws a sine

wave (magnitude = TOS) : CURVE 256 ∅ %DO %I 88 PLOT ∅ %I 256 / 360 *SIND 3 PICK * DRAW %LOOP DROP ;
See what 30 CURVE ; produces.
Now combine CURVE with AXES by defining a new word, WAVE
: WAVW AXES 80 CURVE WAIT ;
On a different tack, if you would like to use the BREAK key within a long/endless loop, define the word BREAK? by
: BREAK? INKEY 32 %= IF ABORT ELSE THEN ;
Now insert BREAK? within any long loop.
If you are not convinced that FORTH is worth all the effort, define and then run : STRIPES 16384 6144 51 FILL ;
Then see how long it takes you to do the same in BASIC . . .
Once you have defined a couple of dozen of your own, useful words [and SAVEd then — see 5.6], ideas for programs and applications will come naturally to you. Use the FORTH editor (see Chapter 7) to find out how the FORTH game (see 8.1) actually works.

6. INTEGER FORTH

6.1 FP5Ø's special feature is its ability to handle floating point (ie FP) numbers and functions — numbers with decimal digits and in scientific notation.

However, FP5Ø is also includes a comprehensive integer arithmetic system, for the simple reason that integer operations can be performed much much faster than floating point ones, (integer numbers also occupy less memory space). The speed differential is tremendous, as can be seen from the following timings for 1000 operations:

	JUPITER ACE FORTH (Integer)	SPECTRUM BASIC	FP5Ø (Floating Pt)	FP5Ø (Integer)
Empty Loop	0.12	4.2	2.8	0.05
print a number	7.5	18	10	2.6
Add	0.45	7.5	0.7	0.23
Multiply	0.9	7.5	1.0	0.5

The figures are rather convincing, aren't they? The speed improvement of Integer FP5Ø over normal FP5Ø ranged from 2x to 56x . . . . Hence to get the maximum speed out of your FORTH programs try to get them to work in integers only.

6.2 To use integer FORTH, all numbers must be PURE integers (ie. stored in integer format).

A PURE integer is a whole number in the range of Ø to 65535. Further it must never have been derived from anything other than PURE integers. To illustrate what I mean, entering 7ØØØØ 2 / leaves 35000 on the stack in floating point form and not in integer form. This is because one of its 'antecedents', 70000, was not a PURE integer. Similarly, 10 2.8 X does not leave a PURE integer on the stack — 28 is in FP form.

In circumstances like this, the INT command can be used to convert the FP format number into integer format, provided TOS is a number between 0 and 65535.9.

Note that while floating point operations will work both on numbers in floating point and in integer formats, integer operations will work ONLY on PURE integers (ie. numbers in an integer format) — using them on other numbers will produce garbage.

6.3 To see how to put a number in integer format on the stack, simply enter

```
7 ;
```

as usual — 7 is now TOS, and is in integer format. (Note you would not have got it in integer format had it been 70,000 or 7.7).

6.4 Many of the FP operators have integer FORTH equivalents. The usual connection between the two is that the integer operator is the FP operator, with a % sign stuck immediately in front of it (no spaces).
For example, integer addition is % and integer division is %/. Note that you can read Integer for '%'.
The integer equivalent of . (ie. print TOS) is of course %. Try the following examples (does not work ➔ answer is wrong):

7 %. ;	works.
2 3.5 * %. ;	does *not* work (the 7 is not a pure integer).
2 3.5 %* %. ;	does *not* work (the 3.5 is not a pure integer).
2 3.5 INT %. ;	works.
7.7 %. ;	does not work.
7000 %. ;	does not work.
7000 %. ;	works — and puts a comma in. This is another refinement of integer FORTH.
7 3 %/ . ;	works — (had %. been used, it would have been in integer form), note that the result is 2 in FP form.

6.5 The complete list of Integer FORTH words can be found in Appendix B. Most of them are exactly similar in operation to their FP (ie. without the % sign) equivalents, and are hence not described at length. They all work only on pure integers, and are much faster in operation than their FP equivalents. It is hence sound programming practice to use integer FORTH wherever possible.
Some integer FORTH words have no FP equivalents. These are %MOD and %XOR, and Appendix B explains them fully.

6.6 THE INTEGER DO LOOP
As can be seen from the table in 6.1, there is considerable advantage in using integer loops. Their operation is similar to FP loops (see 5.13) except that no integer equivalent of -LOOP is available, loop control variables J and K cannot be used (hence no nested integer loops are possible — the only integer loop control variable is %I) and EXIT%L (not %EXITLP) is the equivalent of EXITLP.
For example, try

```
1001 0 %DO 21 0 AT %1 %. %LOOP ;
```

Fast wasn't it?
The Integer DO loop words are %DO, %LOOP, %+LOOP, %LEAVE, EXIT%L and %I.

SUMMARY OF CHAPTER 6

+ Wherever possible use Integer FORTH, because it is much faster
+ Prefix most FP words by % to get their Integer FORTH equivalents — refer Appendix B for details.
+ Integer FORTH words can be used only on PURE integers — whole numbers from ∅ to 65535 which have no non-PURE integer antecedents, INT can be useful in many cases.

7. THE FORTH EDITOR

7.1 Immediately after FP Forth (3 sections, as described in 3.1) there is another program on the cassette with which you have been supplied. This program, (called ED5Ø) is a FORTH editor.

7.2 ED5Ø provides an alternative operating system to FP5Ø. It allows you to edit, correct and test programs with ease, and includes a full screen editor. If you do not understand what this signifies, the examples which follow will explain everything.

7.3 To operate in ED5Ø, proceed as follows:-
a. Get to the FP5Ø command mode, ***> (You are already "in" FP5Ø, and you can have defined words already). Enter a (lower case) e and press enter.
b. Load ED5Ø, which as stated before is positioned immediately after their third part of FP5Ø.
c. The program will autostart. All the words you have defined and all the parameters/pointers set up will have been automatically transferred between FP5Ø and ED5Ø.

7.4 The ED5Ø command mode prompt is ***>. If at a later stage you wish to return to FP5Ø from ED5Ø (eg. to SAVE the program), type in the word ret when you are in command mode, and press enter. Now start playing the program cassette from the start (ie. the beginning of FP5Ø). FP5Ø will now load and autostart, returning you to ***>.
Note that (again) all necessary information is transferred automatically to FP5Ø. Further, only the first part of FP5Ø loads when returning from ED5Ø. This is normal.

7.5 ED5Ø can do almost everything that FP5Ø can. Its memory map is slightly different, and the maximum number of stack items is now 150 instead of 300. You cannot SAVE directly from ED5Ø (which is why you should return to FP5Ø at the end of editing) — the s command no longer operates. You cannot define your own characters/graphics from within ED5Ø either. When defining words you cannot use a double space to start a new line.
In all other respects, ED5Ø can do whatever FP5Ø can ie. define words, manipulate the stack, run programs etc.

7.6 To understand the extra power of ED5Ø, get into command mode # # #>. Define a word, say TEST, by entering
: TEST 5 Ø DO 127 EMIT LOOP ;
Try TEST ; to check that it works (it should print out 5 copyright symbols, without spaces).
Now define TEST2 by entering
: TEST2 TEST CR 1 BRIGHT TEST CR Ø BRIGHT TEST ;
and try out TEST2 ;

which should give the TEST pattern thrice, with the middle one bright.

Now redefine TEST, say by entering

: TEST 10 0 DO 1 . LOOP ;

Try out both TEST and TEST 2 — you will see that the definition of TEST has been changed, and TEST 2 now uses the new TEST result to operate on. FP5∅ could not do this without first using FORGET (which deletes *all* words defined after the desired word — see 5.5).

Note that when you are redefining an existing word within ED5∅, you get the message "You are redefining an existing word — press y if this is OK."

If you want to redefine, make sure you press a *lower case* y. Switch to lower case if you have been in upper. Entering n will allow you to escape the redefinition mode.

7.7 The most powerful feature of ED5∅ is the full screen editor. This enables you to change the definitions of existing FORTH words without having to retype them entirely (a very painful process if they are very long).

To use the screen editor, enter a (lower case) e from command mode. You will be asked to name the word to be edited. As an example, try TEST (typing in words which are not in the VLISTed dictionary will simply return you to > command mode).

You will see the FORTH definition of TEST appear on the screen, with a flashing cursor at the start of the definition. The cursor can be moved around the screen using CAPS SHIFT and keys 5, 6, 7, 8 (the usual 'arrow' Keys), for left, down, up and right respectively.

Type over whatever you wish to change. Use the SPACE key to delete (ie. print spaces over) — the cursor will move on automatically in each case. All keys have autorepeat.

Once you have completed editing, press the ENTER key. You will receive the same 'redefining existing word' message as described in 7.6 — respond accordingly. You will then be returned to the command mode.

If you wish to abort editing, press SYMBOL SHIFT and 'Q'. This will quit the edit mode and return you to the command mode, with no change made to the word upon which you were operating.

The only restrictions on the redefined word are that all control characters within print strings are lost and that there must be only one semicolon on the screen. Everything on the screen is sent to the compiler routine exactly as if you had typed it in as a definition directly.

Some words of caution — FORGET must not be used after

editing, except on words defined since you last edited. So if the last operations you performed on ED5∅ were defining TEST, TEST2, editing TEST, defining TEST2, TEST 4 you can FORGET TEST2 or TEST 4 but not TEST. Also, do not try to edit any of the original FORTH words — they contain machine code and cannot be edited.

7.8 The screen editor is a powerful debugging tool and should enable you to get many of your abandoned FORTH programs into good working order!

7.9 If you manage to BREAK out of ED5∅ or get an error message, the way to restart is by using GOTO 6∅. On no account should you use RUN.

SUMMARY OF CHAPTER 7

+ It is possible to switch from FP5∅ to ED5∅ and back using e and ret respectively.
+ ED5∅ allows you to use a screen editor, which is a powerful debugging tool. FP5∅ does not have this feature.
+ ED5∅ is otherwise similar in operation to FP5∅, with a few differences (SAVEing, character definition, memory map, maximum stack size).
+ GOTO 6∅ enables ED5∅ to be restarted, if needed.

8. MISCELLANEOUS TOPICS

8.1 FORTH GAME

After FP5Ø (3 parts) and ED5Ø (1 part) on your FP FORTH cassette, a FORTH game has been recorded. It is called GAME and loads in 3 parts. It was written using FP Forth and shows what can be achieved with it.

Reset your Spectrum and load the game using the command LOAD " ". When the command mode prompt ★★★> is obtained, enter GAME ; Alternatively, first use VLIST and try to see what new words have been added to the FP5Ø dictionary to enable it to play GAME. Remember, GAME is a word itself, defined in terms of other words, which are defined in terms of ... and so on .

The object of the game is to turn over all the squares to yellow side up as they are at the beginning.

First enter skill level, from 1 to 9 (9 is hardest). I suggest starting at 1. The computer now inverts a number of squares.

To change a square back to yellow you must enter the co-ordinates of that square, letter first (a — z) then number (1 to 18, single digit numbers to be prefixed by aØ). The problem is that not only does that squarechange colour but the eight around it do the same...

Try a Ø1 a few times to get the idea. Once you have done it you are returned to FORTH command mode — for another game, just enter GAME ; again.

If you are interested in how the game works (ie. if you want to see the "program" — or, more correctly, the word definitions) break out of command mode using CAPS SHIFT and '6', and load ED5Ø (see Chapter 7). Use the screen editor option to examine the definitions of these FORTH words (the only new ones defined)

SCREEN TURN 9 INP3 RAND SKILL FINI? and GAME.

Try and work out how the program works ... The definition of GAME,

: GAME SKILL BEGIN INP3 TURN 9 FINI? UNTIL WAIT ;

is really quite elegant — far more structured than any BASIC program could be.

8.2 LOW LEVEL PROGRAMMING

You should read this section only if you are familiar with low level (ie. assembler or code) programming. It shows how to access and manipulate addresses, control the registers and execute machine code programs from within FP5Ø.

Note that a memory map has been provided in Appendix C.

a. %AND %OR — These are full 16 bit operations, operating on pure integers (16 bit unsigned numbers) and

%XOR giving pure integer results.

b. Fetching and Storing

All memory addresses must be pure integers. @ (Fetch) and ! (Store) can be prefixed by nothing (5 byte floating point), by % (16 bit unsigned integer), by a C (character or byte) or by a P (port). Fetch reads in a value from address TOS and puts it on the stack. Store sends stack item 20S to address TOS. Stores and fetches available are @ %@ C@ P@ ! %! C! P!

c. Reading and Incrementing

?	does @ . Pronounced read.
%?	does %@%.
C?	does C@%.
%+!	(integer increment) increases the pure integer found in 2 bytes at address TOS by 20S (pure integer). 65535 %+! will effect a decrement, and so on.
+!	(increment) increases the floating-point number found in 5 bytes at address TOS by 20S.

d. General Commands

All the following are pure integer operand/result functions.

FILL	fills 20S (minimum 2) bytes with value TOS starting at address 30S.
ERASE	fills TOS (minimum 2) bytes with value zero starting at address 20S.
DELETE	fills TOS (minimum 2) bytes with value 32 (ascii blank) starting at address 20S.
CHOVE	copies TOS single byte numbers from address 30S
%MOVE	to destination 20S. %MOVE does the same as CMOVE
MOVE	for 2 byte numbers, ie. twice as many bytes. MOVE does the same as CMOVE for 5 byte numbers, ie. 5 times as many bytes. The source and destination blocks must not overlap if the destination is higher in memory than the source.
TYPE	emits TOS characters found at address 20S, eg. 5Ø5Ø 3ØØ TYPE ;
C DUMP	prints out the values of TOS bytes found at address 20S.
%DUMP	prints out the values of TOS 2-byte numbers found at address 20S.
DUMP	prints out the values of TOS 5-byte

numbers found at address 20S.

FIND — gives the compilation address of the next word without executing that next word. Use only to find words which appear in the VLIST. A simple apostrophe can be used as an abbreviation for FIND. As an example, FIND TEST . ; would print out the compilation address of TEST.

wrdsch — searches in the dictionary for the word whose name is stored in six bytes at 232 64. It returns the address of the dictionary entry (or zero for word not alone) in the BC register pair. Each dictionary entry consists of 6 bytes of word name follow by 2 bytes of compilation address.

flgtst — ORs together all bits of TOS and sets the Z flag if the result is zero. Used by IF, WHILE and UNTIL, therefore these read zero as false and all non-zero values as true.

EXECUTE — causes a jump to address TOS.

EXPECT — inputs up to TOS characters from the keyboard and puts them at address 20S. Input can be terminated early by pressing enter, in which case the value 13 IS stored.

—TRAIL — (—TRAILING) takes 20S as the address of a string in memory and TOS as its length including trailing space. Leaves 2) S unchanged — adjusts TOS to exclude all trailing spaces.

COUNT — For reading through a table of bytes. Reads a byte from address TOS and puts it on top of the stack. The address is retained as 20S, and is incremented to point at the next byte.

Multiple stack control

Three stacks are in operation on the FP5Ø FORTH IMPLEMENTATION. These are the return stack, the data stack and the calculator stack. The return stack is pointed to the SP register pair in the Z8ØA CPU, and is used by PUSH, POP, CALL and RET. FORTH uses it for storing return address and looping variables. It has 2-byte data items. The data stack, meanwhile, is pointed to by the SP register pair after the execution of STKSWP. Re-execution restores normality. It is the FORTH stack discussed

thoughout this manual, and has 6-byte data items (5-byte floating-point plus 1 dummy byte or 2-byte pure integer plus 3 zero bytes and 1 dummy byte). The calculator stack is operated by the Sinclair ROM, and is used for floating-point calculations on 5-byte items.

STKSWP	swap SP between data and return stacks.
R>	transfer pure integer from return stack to data stack (SP pointing to return stack).
R@	copy one pure integer from top of return stack to data stack (SP pointing to return stack).
SPtoCS	transfer 6-byte item from stack pointed to by SP to 5-byte position on calculator stack.
2 to CS	transfer two items from data stack to calculator stack. (SP pointing to return stack).
CStoD	transfer one item from calculator stack to data stack (SP pointing to return stack).

f. System Variables
System variables used are: DF—CC for print position, 23681 for WORD displacement within the PAD, 23728 to store the stack pointer not currently in SP, 23662 for temporary storage.

g. Entering machine code
As if it was a FORTH word, anywhere in a command or definition, enter
mc nl n2 n3 ... nm end
where nl, n2...nm are machine code bytes (in decimal). You can use ENTER instead of the spaces, but NOT the double-separator start-a-new-line trick.

h Run time routines
The runtime routines which appear in VLIST are number (to stack a number) prstrg (used by ."string") and wrdsch (to search for words). Do not try to use them.

8.3 Further reading on FORTH

This manual cannot be a comprehensive guide to FORTH. It is intended to be a helpful and hint-filled introduction, and no more. To find out more about FORTH, I recommend you refer to any of the following books:

1. Starting FORTH, by L. Brodie (Prentice Hall, ISBN 013-842922-7)
2. Introduction to FORTH, by K. Knecht (Howard W Sons, ISBN &-672-21842-9)
3. The Complete FORTH, by A. Winfield (Sigma Technical, ISBN 0905-104-22-6)

Borrowing a Jupiter Ace manual from a friend could be helpful too. While not quite as good as the Spectrum manual, it is certainly well written.

If you are interested in combining machine code with FORTH (see 8.2), the best book to read is

4. Programming the Z80, by Rodney Zaks (Sybex, ISBN 0-89588-094-6).

If you are interested in using the Spectrum ROM routines, calling them from within FP5∅ etc. necessary reading is:

5. The Complete Spectrum ROM Disassembly, by Dr Ian Logan & Dr Frank O'Hara (Melbourne House, ISBN 0-86759-117-X).

SUMMER OF CHAPTER 8

+ **Play the FORTH game and then (using the editor) understand how it works.**
+ **Low level programming can be accomplished on FP Forth.**
+ **If you are fascinated by FORTH, further reading is recommended.**

APPENDIX A — FP FORTH WORDS

FLOATING POINT COMMAND	Description of operation	BEFORE STACK	AFTER STACK
.	Prints TOS on screen/printer	X,Y,5,	X,Y
+	Adds 2OS to TOS and puts result on stack	X,3,5,	X,8
-	Subtracts TOS from 2OS and puts result on stack	X,3,5,	X,-2
*	Multiplies 2OS by TOS and puts result on stack	X,3,5,	X,15
/	Divides 2OS by TOS and puts result on stack	X,3,5,	X,0.6
|	Raises 2OS to the power of TOS and puts result on stack	X,3,5,	X,243
>	Puts 1 on stack if 2OS>TOS; zero otherwise	X,3,5	X,0
<	Puts 1 on stack if 2OS<TOS; zero otherwise	X,3,5	X,1
=	Puts 1 on stack if 2OS=TOS; zero otherwise	X,3,5	X,0
>=	Puts 1 on stack if 2OS>=TOS; zero otherwise	X,4,4	X,1
<=	Puts 1 on stack if 2OS<=TOS; zero otherwise	X,5,4	X,0
<>	Puts 1 on stack if 2OS<>TOS; zero otherwise	X,5,4	X,1
!	'STORE' When used after a variable, will set the value of the variable to TOS. Ex: if TOS is 4, A!; will set variable A to value 4	X,Y,4	X,Y
(a)	'FETCH' When used after a variable it will set TOS to the value of the variable. If B=0.8, then B (a); will set TOS to 0.8	X,Y	X,Y,0.8
?	'READ' This prints out the value of the variable it is used after. So if C=-21, C? prints out -21	X,Y,Z	X,Y,Z
+!	'INCREMENT' This adds TOS to the variable it is used after. If TOS=4 and A=7,A+!; will set A to 11	X,Y,4	X,Y
'	This puts the address of the top of the routines compilation area on TOS. Hence if it is 44089, ';;; prints out 44089	X,Y,	X,Y,44089
0>	Puts 1 on stack if TOS >0, zero otherwise	X,Y,4	X,Y,1
0<	Puts 1 on stack if TOS <0, zero otherwise	X,Y,4	X,Y,0
0=	Puts 1 on stack if TOS =0, zero otherwise	X,Y,4	X,Y,0
1+	Adds one to TOS and puts the answer on TOS	X,Y,4	X,Y,5
2+	Adds two to TOS and puts the answer on TOS	X,Y,4	X,Y,6
2-	Subtracts two from TOS and puts the answer on TOS	X,Y,4	X,Y,2
2toCS	Transfers two items from the data stack to the calculator stack (SP pointing at the return stack)	X,Y,Z	X
79-STA	Prints out a message identifying FP50 as being based on FORTH 79 standard	X,Y,Z	X,Y,Z
ABORT	This clears the stack (makes stack length zero irrespective of whether it was originally +ve or -ve) and returns you to command mode	X,Y,Z	—

FLOATING POINT COMMAND	DESCRIPTION OF OPERATION	"BEFORE" STACK	"AFTER" STACK
ABS	Removes the -ve sign (if present) from TOS	X,Y,-3.1	X,Y,3.1
ACSD	Replaces TOS by its $\cos^{-1}$, in degrees	X,Y,0.5	X,Y,60
ACSR	Replaces TOs by its $\cos^{-1}$, in radians	X,Y,0.5	X,Y,1.0471976
AND	If both 2OS and TOS were >∅, replaces them with TOS, otherwise with ∅	X,2,-1	X,7
AT	Moves the print position to line 2OS and column TOS	X,Y,Z	X
ATTR	Reads the colour attributes of the character square on line 2OS, column TOS. Hence if 2OS = 7, TOS = 3 and the colour attribute of (7,3) is 56, ATTR will make TOS = 56. The co-ordinates must be integers but it does not matter if one or both of them is 'out of range' — the Spectrum will return the ATTR value of the closest existing square so 100 100 ATTR will do the same as 23 31 ATTR	X,7,3	X,56
ASND	Replaces TOS by its $\sin^{-1}$, in degrees		
ASNR	Replaces TOS by its $\sin^{-1}$, in radians		
ATND	Replaces TOS by its $\tan^{-1}$, in degrees		
ATNR	Replaces TOS by its $\tan^{-1}$, in radians		
BEEP	Produces a BASIC-type BEEP of duration 2OS seconds and pitch TOS	X,Y,Z	X
BLEEP	Produces a note of duration 2OS units and pitch TOS-2OS and TOS must be pure integers. Duration units vary with pitch. Useful for machine code style sound effects	X,Y,Z	X
BRIGHT	Sets the bright control to TOS (should br ∅ to 1). Use before Print commands	X,Y,Z	X,Y
C@	Refer to 8.2		
C	Refer to 8.2		
C?	Refer to 8.2		
CDUMP	Refer to 8.2		
CIRCLE	Draws a circle centred at 3OS, 2OS) and having radius TOS. Works with pure integers too	A,X,Y,Z	A
CLS	Clears the screen and moves the PRINT AT position to the top left hand corner	X,Y,Z	X,Y,Z
CMOVE	Refer to 8.2		
COSD	Replaces TOS degrees by its cosine	X,Y,60	X,Y,0.5
COSR	Replaces TOS radians by its cosine	X,Y,2	X,Y,-0.4164684
COUNT	Refer to 8.2		
CR	Carriage return — moves the PRINT AT position to the start of the next line	X,Y,Z	X,Y,Z
CStoD	Refer to 8.2		
DELETE	Refer to 8.2		
DEPTH	Gives the number of items on the stack before DEPTH was put there. So if Z was the 7th item on stack:-	X,Y,Z	X,Y,Z,7
DO	Loop control command — see 5.13		
DRAW	Draws a straight line from the last plotted point to a point whose displacement is (2OS, TOS) pixels. Works with pure integers too	X,Y,Z	X

DROP	Discards TOS	X,Y,Z	X,Y
DUMP	Refer to 8.2		
DUP	Copies TOS so it appears twice on the stack	X,Y,Z	X,Y,Z,Z
?DUP	Performs DUP provided TOS is non-zero. If it is zero does nothing	X,Y,0	X,Y,0
EMIT	Prints out CHR$ (TOS), provided TOS is a pure integer. If TOS is greater than 255 it is reduced modulo 256. Non-pure integer TOS's will be misinterpreted.	X,Y,Z	X,Y
ERASE	Refer to 8.2		
EXECUT	Refer to 8.2		
EXIT	Makes the computer abandon the current word/command. Do not use within DO loops — see EXITLP	X,Y,Z	X,Y,Z
EXITLP	EXIT for use within DO loops. See 5.13	X,Y,Z	X,Y,Z
ESP	Replaces TOS by e to the power TOS	X,Y,1	X,Y,2.7182818
EXPECT	Refer to 8.2		
FIELD	Moves the print position into the next 16 character field, like a comma in a BASIC list	X,Y,Z	X,Y,Z
FILL	Refer to 8.2		
FIND	Refer to 8.2		
FLASH	Sets the FLASH control to TOS (which should be 0 or 1). Use before print command	X,Y,Z	X,Y
flgtst	Refer to 8.2		
I	Innermost loop control index — see 5.13		
>IN	This puts the address of the system variable which stores the word displacement onto the stack. >IN C@ gives the number of characters WORD has read so far (ie. since the last QUERY)	X,Y,Z	X,Y,Z,23681
INK	Sets the ink control to TOS (which should be ∅ to 8). Use before print commands	X,Y,Z	X,Y
INKEY	Puts the ASCII value of the currently pressed key on to the stack, or 255 if no key is being pressed. So, if "A" is pressed	X,Y,Z	X,Y,Z,65
INT	Replaces TOS by its pure integer equivalent, if possible. If not, or if TOS was already a pure integer, no change. See Chapter 6	X,Y,3.8	X,Y,3
INVERS	Sets the inverse control to TOS (should be ∅ or 1). Use before print commands	X,Y,Z	X,Y
J	Second to innermost loop control index — see 5.13		
K	Third to innermost loop control index — see 5.13		
KEY	Waits for you to press a key and then puts its ASCII value on stack. So if no key is pressed	X,Y,Z	X,Y,Z
LEAVE	Sets the index value of the innermost loop (ie. I) to the loop's limit value — see 5.13.	X,Y,Z	X,Y,Z
LN	Replaces TOS by its natural logarithm	X,Y,10	X,Y,2.3025851
LOOP	Like the NEXT command in BASIC — see 5.13	X,Y,Z	X,Y,Z
+LOOP	Makes the loop "step" = TOS (TOS should be positive)	X,Y,Z	X,Y
—LOOP	Makes the loop "step" = -TOS (TOS should be positive)	X,Y,Z	X,Y
MAX	Takes the top 2 stack items off the stack and replaces only the larger one	X,3,7	X,7
MIN	Takes the top 2 stack items off the stack and replaces only the smaller one	X,7,2	X,2
MOVE	Refer to 8.2	X,3,7	X,3

NEGATE	negates TOS, ie. multiplies it by -1	X,Y,6 X,Y,-6	X,Y,-6 X,Y,6
NOT	Takes TOS and replaces it with 1 if it was 0, and with 0 if it was non-zero	X,Y,O X,Y,-2	X,Y,1 X,Y,O
number	Run-time routine — see 8.2		
OR	Takes off the top two stack items and replaces them with 1 if 2OS was non-zero or with TOS if 2OS was ∅	X,3,-2 X,∅,3 X,∅,∅	X,1 X,3 X,∅
OVER	Puts a copy of 2OS on the top of the stack	X,Y,Z	X,Y,Z,Y
P@	Refer to 8.2		
P!	Refer to 8.2		
PAD	Puts the address of the note pad area of memory (onto which characters are put by QUERY and read by WORD) onto the stack	X,Y	X,Y,32883
PAPER	Sets the PAPER control to TOS (should be ∅ — 8). Use before print commands	X,Y,Z	X,Y
PICK	Replaces TOS by the TOS-th item on the stack. Say if the 4 is the 7th item on the stack: Note that 2PICK is hence identical to OVER	3,4,5,7	3,4,5,4
PLOT	Plots the pixel (2OS, TOS). Works with integer operands too	X,Y,Z	X
POINT	Reads the pixel (2OS, TOS) and gives the result, O (paper) or 1 (ink) in pure integers. Works only with pure integer operands	X,Y,Z	X
PROVER	Sets the OVER control to TOS (should be O or 1). Use before print commands	X,Y,Z	X,Y
prstrg	Run-time routine — see 8.2		
QUERY	This puts characters from the keyboard into the notepad area of memory (see Appendix C and PAD) until you press enter, when the ENTER code (13) is stored to mark the end of data. The maximum string length for QUERY is 141 characters, after which it automatically ends. QUERY also resets the WORD displacement (stored in System Variable 23681 — see >IN). The note pad can be read using WORD	X,Y,Z	X,Y,Z
QUIT	Makes the computer return to command mode (***> for FP5∅, ###> for ED5∅)	X,Y,Z	X,Y,Z
R@	Refer to 8.2		
>R	Refer to 8.2		
R>	Refer to 8.2		
RND	Puts a random number (0 <= RND <1) onto the stack	X,Y,Z	X,Y,Z,O.7077179
ROLL	The TOS-th item is removed from its position and put on top of the stack. So, if the 4 is the 7th item on the stack:-	3,4,5,7	3,5,4
ROT	Rotates →3OS is removed from its position and put on top of the stack	X,Y,Z	Y,Z,X
SIND	Replaces TOS degrees by its sine	X,Y,30	X,Y,0.5
SINR	Replaces TOS radians by its sine	X,Y,2	X,Y,0.90929743
SGN	Replaces TOS by 1 if it was positive, -1 if it was negative or 0 if it was 0	X,Y,-3	X,Y,-1
SPACE	Moves the print at position one to the right, or if impossible to a new line	X,Y,Z	X,Y,Z
SPACES	Moves the print at position along TOS spaces. The operand should be a non-zero pure integer, less than 256	X,Y,Z	X,Y
SPtoCS	Refer to 8.2		
SQR	Replaces TOS by its square root	X,Y,2	X,Y,1.4142136
STKSWP	Refer to 8.2		

SWAP	Swaps around 2OS and TOS	X,Y,Z	X,Y,Y
TAB	Moves the print at position to column TOS	X,Y,Z	X,Y
TAND	Replaces TOS degrees by its tangent	X,Y,45	X,Y,1
TANR	Replaces TOS radians by its tangent	X,Y,1	X,Y,1.5574077
-TRAIL	Refer to 8.2		
TYPE	Refer to 8.2		
VLIST	Lists the words (and run time routines) in the FORTH dictionary. Use the Y key to scroll when the flashing block appears	X,Y,Z	X,Y,Z
WAIT	If you have the Y key pressed, this command has no effect. If you do not have the Y key pressed, it waits until you press it, showing a flashing block in the bottom right hand corner of the screen. It can be used for scroll control and to prevent immediate return to command mode after producing a screen display/graph	X,Y,Z	X,Y,Z
WORD	This reads one character off the QUERY notepad onto the stack, working upwards through the pad. Its displacement is stored in the system variable 23681. See QUERY, PAD, >IN. If the next unread character in the pad is "B"	X,Y,Z	X,Y,Z,66
wrdsch	Run time routine — see 8.2		
A,B,C..Y,Z (excl. I,J,K)	The 23 variables, initially set to zero		

Note all Integer FORTH words are listed separately in Appendix B.

The commands below work like FORTH words, but are not really words (and hence do not appear in the dictionary).

:	Prefix 9 used for word definition
;	Instruction end indicator
(space key)	Word separator
e	To load ED5Ø from within FP5Ø
f	FORGET — see 5.5
proff	Switches off the printer
pron	Switches on the printer
ret	To load FP5Ø from within ED5Ø
S	SAVE onto tape — see 5.6
Z	COPYs screen onto printer

APPENDIX B — EXISTING INTEGER FORTH WORDS

Note: Where no description of operation is given, refer to the corresponding word (without the % sign) in Appendix A.

INTEGER FORTH WORDS	*WORD DESCRIPTION (where necessary)*
%.	
%+	
%—	
%*	
%/	
%>	
%<	
%=	
%>=	
%<=	
%<>	
%!	
%@	
%+!	Except that 65536 % will give a decrement, and so on.
%∅=	
%1+	
%1—	
%2+	
%2—	
%AND	
%DO	
%DUMP	
%FIELD	Moves the print position into the next character field. Does not work with printer.
%LEAVE	
%+LOOP	
%LOOP	
%MAX	
%MIN	
%MOD	This yields the remainder after a division. For example 13 4 %MOD puts 1 on TOS (because 13 — 4 leaves remainder 1)
%/MOD	This puts the integer part of the quotient on TOS and the remainder becomes 20S. For example, 13 4 %/MOD leaves TOS = 3 and 20 S = 1.
%MOVE	
%NOT	
%OR	
%XOR	If one of TOS and 20S was > ∅ and the other was not (ie. was = ∅ since we assume pure

integers), the two are taken off the stack and replaceD by the value of the one greater than ∅. If not (ie. both = ∅ or both > ∅) then they Are taken off the stack and replaced by TOS set to ∅.

Hence 4 7 %XOR . ; prints ∅
4 ∅ %XOR . ; prints 4
∅ ∅ %XOR . ; prints ∅

%I This is the loop control variable — refer to 6.6

EXIT%L This is the only Integer Forth word not prefixed by a %. Its FP equivalent is EXITLP.

APPENDIX C — THE MEMORY MAP — FP5∅

Addresses	*Function*
0 — 16383	ROM
16384 — 23295	Screen
23296 — 23546	Buffer
23547 — 23733	System Variables (including some extra FORTH ones)
23734 — 32767	BASIC program and data stack, both growing inwards
32768 — 32882	Variables (23 x 5 bytes)
32883 — 33023	Query Notepad
33024 — 33791	Characters (96 x 8 bytes)
33792 — 44031	Dictionary
44032 — 65367	Routines Compilation Area and Return Stack, both growing inwards
65368 — 65535	UDG's