SPECTRUM FORTH

Addendum

Saving: to save your own words; enter a lower case s from command mode and when requested enter a file name, max 9 letters with no space, then press the enter key.

SPECTRUM FORTH

16K & 48K VERSIONS

USER MANUAL

© CP SOFTWARE 1983

CONTENTS

USER MANUAL (C) Copyright CP SOFTWARE 1983.
Spectrum FORTH and Game (C) Copyright M.Hampson 1983

This User Guide and the material on the Spectrum FORTH tape, or any part thereof, shall not be copied or reproduced for any use by any other person or organisation, neither shall it be loaned nor hired, without prior permission in writing.
While every effort has been made in the production of this program the publisher undertakes no responsibility for errors nor liability for damage arising from its use.

CP SOFTWARE, 17 Orchard Lane, Prestwood, Great Missendin, BUCKS. HP16 ONN, ENGLAND.

GENERAL INSTRUCTIONS FOR SPECTRUM FORTH

Spectrum FORTH is a cassette based implementation of FORTH. The cassettes are recorded as shown:

Side 1 has:- FORTH V48, GAME, FORTH V16

Side 2 has:- FORTH V16, GAME, FORTH V48

To load, enter LOAD "" press the ENTER key and start the tape recorder. When correctly loaded the prompt ***> appears at the bottom left of the screen indicating that you are in command mode.

V16 (16K version) supports fewer words than V48, for details consult the Word Lists.

Most of the resident Spectrum FORTH words must be entered using capital letters. A space must be put between all FORTH words in a command or definition. For this purpose pressing SPACE or the ENTER key are interchangeable.

VLIST lists the names of all the words, both resident and new, in the dictionary; use the Y key to scroll the list when the flashing cursor appears at bottom right of screen.

The Spectrum key and extended address words are not used and any that are resident in the dictionary must be typed in full. The undefined symbols and extended address words/ functions can be used to name new words.

The usual editing facilities ← → cursor control and delete keys can be used on the input line (newline) before pressing ENTER. On V16 once ENTER has been pressed no further editing is possible. On V48 the f (forget) word allows limited editing. When typing a long definition or program it is therefore advisable to periodically test the input by

entering it in small blocks. If it is OK you get the prompt 'continue definition': Otherwise the message is 'typing error' and you should then retype the input from the point that it stopped compiling, ie, the last word of the definition shown at the top of the screen. With some mistakes you may get a BASIC report code in which case enter GO TO 100 to return to command mode or enter LET N $ = "": GO TO 5ØØØ to continue definition. If you continually get BASIC error report codes this could mean that your current input is invalid or that you have corrupted a flag or register etc., in which case it may prove necessary to reload. Spectrum FORTH does not expect elementary mistakes to be made, such as using a command inappropriately, as these may cause the system to crash.

SPECTRUM FORTH GAME

The sample game program, for 48K machines only, is recorded after the Spectrum FORTH. To load it enter LOAD "GAME" press the enter key and start the tape. Once correctly loaded the command mode prompt ***> appears. To play the game enter GAME ; (don't forget the space between GAME and ;) and press the ENTER key.

The game Hampson's Plane by M. Hampson was orginally published in BASIC in SYNC magazine, USA.

The object of the game is to turn over all the squares to yellow side up, as they were before the start. First enter skill level, from 1 to 9, I suggest you start at 1. When you have done this, a number of squares in a random pattern will change from yellow to blue, or light to dark if you are using a black and white TV.

To change a square back to yellow all you have to do is enter the co-ordinates of that square, letter first then number. There is just one problem, not only does that square change colour but also the eight that surround it. Try, A Ø1 a few times to get the idea.

When the game is finished you are returned to FORTH, if you want another game enter GAME ; again.

A listing of the game is given in Appendix A, and it can be typed into a 16K machine.

AN INTRODUCTION TO FORTH

This introduction is intended to provide those new to FORTH, very briefly, with some of the main ideas behind the language. No attempt is made to go into detail, which is adequately covered in other works, although most of the points covered are illustrated with examples. For the newcomer two excellent books can be suggested, Starting FORTH - by L. Brodie, published by FORTH INC/ Prentice Hall, ISBN 0-13-842922-7 and The Complete FORTH - by Alan Winfield, published by Sigma Technical Press, ISBN 0-905-104-22-6.

Before continuing read through the General Instructions if you have not already done so. It is assumed that you are familiar with BASIC, and comparisons between FORTH and BASIC will be made when appropriate.

FORTH is based on the concept of the WORD and it uses the Reverse Polish Notation which operates on a Stack. This is unlike BASIC, which is based on lines and line numbers and with algebraic notation for entering arithmetic instructions.

All FORTH commands or instructions are called WORDS. Even symbols like . + ? in FORTH are WORDS. Each Word has its own special task which is clearly defined. A list of all the Words resident in Spectrum FORTH together with their definitions and examples are given in the word list and dictionary which follow. One of the advantages of FORTH is that you can teach it new Words, based on those already in the dictionary, to do just about anything you like. The new Words that you create can be added to the dictionary and then used, to create more words if you choose, whenever you like. This, infact is the basis of FORTH programming.

The Stack is a pile of numbers and, as in many piles, the last item on is the first one off. The FORTH Word that puts a number onto the Stack is

anything which looks like a whole number. To be specific any integer in the range -32768 to 32767. The Word that removes a number from the Stack and prints it, on the screen, is a . (called a dot). There are, of course, other ways of putting items onto and taking them off the Stack and there are various things that can be done with the numbers while they are on the Stack. For example the Word + takes the first and second items off the stack adds them and puts the result on the stack. This example brings us back to the Reverse Polish Notation (usually abbreviated to RPN) which uses the stack for its operations. Put simply, the RPN is nothing more than a method of writing down and carrying out arithmetic operations, i.e. addition, subtraction, multiplication and division. It is also the way in which all FORTH commands are input, with the operand (data) first, followed by the operator (FORTH Word).

The examples which follow will give you practice in using RPN, which is really quite easy. They are also used to illustrate the difference between RPN and the algebraic notation, which is normally used for writing down arithmetic operations and is the method used in Basic. If RPN is new to you, or if you are out of practice I suggest you load Spectrum FORTH now (using LOAD"" - ENTER) and work through the examples as you come to them.

First just a few words on using Spectrum FORTH. The prompt ***> appears, at the bottom left of the screen, to indicate Command Mode. This means that it is ready for your input, just like the L and K cursors in BASIC. All commands or instructions and definitions must be terminated by the Word ; (semicolon) this indicates that the command is complete. If the prompt - 'continue definition' : - appears after pressing the ENTER key it usually means that you have forgotten the ; . To correct, just enter a ; (semicolon) eg 'continue definition' : ; - ENTER.
It is important to put a Space between each Word.

(Note, in Spectrum FORTH the ENTER key can be used instead of a Space, see also the General Instructions.) Failure to do so will usually result in the message - 'typing error'. If this happens it is necessary to re-enter the instruction or definition. Sometimes it will result in an incorrect input, for example if you want to put, say, a 5 and a 10 on the stack, then the correct way is ***>5 10 ; ENTER with a space between the 5 and 10 and 10 and ; . However, if you missed the space between the 5 and 10 , ***>510 ; ENTER then 510 would be put on the stack. This may seem trivial but it is worth taking care with because, unlike BASIC, the numbers are not seperated by punctuation marks or arithmetic operators.

Now for the first example which we will work through in detail. Put a number on the stack, for example 6.
***>6 ; Don't forget the space between the 6 and ; The ; indicates the end of this command. Now press the ENTER key and you should get the report stack:1 which is telling you that there is one item on the stack. Now put another number on the stack, this time use 2.
***>2 ; ENTER The report should now say stack:2 indicating two items on the stack.
Next, add the two numbers together, this is done using the + Word.
***>+ ; ENTER Did you remember the space between + and ; ? The report now indicates stack:1, i.e. only one item on the stack, and that is the answer of the sum. Now print out the answer using the . Word, ***>. ; The screen report should be 8, ie, the answer to 6+2, and stack:0, indicating that the stack is empty. Before continuing you should note that the . (dot) Word does two things, it prints out the top of the stack (TOS) and also deletes that item from the stack. There are other words which let you see what is on TOS without deleting it, these will be covered later.

To continue, normally putting two items onto the

stack, adding them together and printing out the answer, is carried out with a single instruction:-
***>6 2 + . ; ENTER (don't forget the spaces)

The report should be 8, the answer, and stack:0. That is how Reverse Polish Notation works, data (numbers) first then the instruction, in this case +. For comparison, this sum, in algebraic notation would be written 6 + 2 = , while in RPN it is written 6 2 + , and in the Spectrum's BASIC it is PRINT 6 + 2 . The same applies to other arithmetic operations and all FORTH commands as you will see.

Try the following examples. For comparison the FORTH instructions are shown on the left with the BASIC statements, in brackets, on the right.

FORTH (RPN)	BASIC (Algebraic)
***> 45 100 + . ;	[PRINT 45 + 100]
***> 100 45 + . ;	[PRINT 100 + 45]
***> 100 45 - . ;	[PRINT 100 - 45]
***> 45 100 - . ;	[PRINT 45 - 100]
***> 1024 2 / . ;	[PRINT 1024/2]
***> 111 7 * . ;	[PRINT 111*7]
***> -50 3 * . ;	[PRINT -50*3]
***> 12 16 14 6 + + + . ;	[PRINT 12+16+14+6]

If you have any difficulty with these examples try them again, and if you still have problems read one of the books suggested above.

The following need a little more care, again FORTH on the left and BASIC on the right.

```
***> 3 20 * 6 / . ;              [PRINT (3*20)/6]
***> 12 3 + 6 * 5 / . ;          [PRINT ((12+3)*6)/5]
***> 11 3 - 4 3 * * . ;          [PRINT (11-3)*(4*3)]
***> 3 3 3 * * . ;               [PRINT 3*3*3]
***> 15 35 + 5 5                 [PRINT
     * / 3 + . ;                    ((15+35)/(5*5))+3]
```

You will notice that we have only tried examples of integer arithmetic, ie, whole number operands that

have yielded only whole number answers. In fact this is one of the constraints of FORTH in that it only operates with integers. Divisions can be carried out which yield a non-integer quotient and this operation is explained later. The result of using non integer numbers is discussed below. It is up to you to ensure that the result of any operation yields an integer or make due allowance for a non-integer result. Decimal fractions (ie non-integer numbers) whether input data or the result of some operations are, in Spectrum FORTH, rounded up or down.

If the number n < 0.5 then it rounds down; if n $\geq$ 0.5 then it rounds up, to the next integer. Try the following:

```
***> 1.49 . ;
***> 1.5 . ;
***> -0.51 . ;
***> -1.5 . ;
***> -1.49 . ;
```

Do not use numbers in the range $-0.5 \leq n < 0$

If you have any questions or ideas don't be afraid to try them; the worst that can happen is that you will 'lock up' or crash the program and then have to reload. Should at any time you get a BASIC error message then enter GO TO 100 - ENTER to return to the Command mode.

When carrying out a division where the quotient (result) is a non-integer and you want to know what the remainder is, use the Word MOD or /MOD. MOD carries out the division and gives, as a result, the remainder.

```
***> 17 6 MOD . ;
***> 17 6 / . ;
```

While /MOD carries out the division and gives both the quotient and remainder.

```
***> 17 6 /MOD . ;
```


So far we have been using . to print out TOS but as you know this also deletes the TOS. If you want to look at the TOS but not delete it from the stack then the Word DUP allows you to so this. DUP duplicates or copies the TOS onto the stack so that when the . is used the TOS is unchanged.

```
***> 6 DUP . ;   prints one item and leaves 1 item
on stack.  Print it out:
***> . ;   the stack should now be empty.
```

You will find DUP a very useful Word.

If you print out more than one item at a time using the . you will find that the items appear all on the same line with no spaces between them. Two commands, Field and CR, allow you to format the screen, they are similar to the , and ; in BASIC. The Word FIELD moves the print position across into the next quarter of the screen and the Word CR starts a new line.

```
***> 3 4 6 . . . ;
***> 3 4 6 . FIELD . FIELD . ;
***> 3 4 6 . CR . CR . ;
```

Finally, a few comments about defining new words. Defining a new Word is just like entering a command except that a colon followed by a space and the name of the new word is put in front of the command. The command then becomes the definition of the new word. This is known as the colon definition. When correctly entered, the name of the new word will appear at the top left of the screen and it's position in memory at the top right. The definition will be printed below this. If you have a printer connected, this information will be printed out. For short definitions this can use a lot of paper, so to economise, once the definition has been printed, BREAK (shifted Space key) then GO TO 100 to get back to command mode.
As an example, a Word can be defined that will square a number (raise it to the power 2). A

method of doing this is to put the number to be squared on the stack and copy it to 2OS using DUP, then multiply TOS and 2OS using *. The new Word can be called SQR and its definition will be DUP *. Remember the spaces, enter:-

```
***> : SQR DUP * ;
```

When correctly entered the new word SQR is added to the dictionary and can be used either as a command or in the definition of a new word as required. Try the following:-

```
***> 3 SQR . ;
***> 12 SQR . ;
```

are the results consistent with 3 squared and 12 squared?

Lower-case, capital letters and combinations of both can be used to name new words but when subsequently used as a command exactly the same format must be used. To list the dictionary, including your own words, enter VLIST and when the flashing cursor [] appears at the bottom right of the screen use the Y key to scroll the list.
Programming in FORTH is just an extension of defining a new word. Words are built on words until the program is complete and comprises just one word.

ABOUT SPECTRUM FORTH

The two versions of Spectrum FORTH, V16 for the 16K Spectrum and V48 for the 48K Spectrum, together with a sample GAME (48K only) are all on one tape. V16 will, of course, also run on 48K machines although V48's UDG's and the extra memory will not be usable.

The main differences between the two versions, as may be expected, relate to the amount of free memory available for new words and user definable graphics, UDG's.

V16 will allow you to define about 114 new words, have 60 stack items and 21 UDG's.

V48 will allow you to define about 1000 new words, have 800 stack items and 256 UDG's

To exceed these limits may cause problems.

There is a separate memory map and word list for both V16 and V48 following this section. A dictionary of the resident words supported by Spectrum FORTH follows the Word lists. Not all the words in the dictionary are supported by V16, consult the Word list for details. Certain parts of this section refer specifically to V48, if in doubt, consult the Word lists.

Some standard FORTH words have been omitted and several non-standard words added. The purpose of most of these non-standard words is to enable the use of the Spectrum's hires graphics, colours and the printer.

A feature of the input routine concerning the use of brackets () and quotes "" should be noted. Quotes within brackets or quotes within quotes should not be used. If brackets within quotes are used, there must be no spaces after the open bracket (and brackets must always be in pairs

within the quotes, ie an open bracket (must always have a close bracket) within the quotes.
Now a look at the user definable graphics UDG's. As mentioned earlier, on the 16K version you can have 21 UDG's with letter names. The same as in the Spectrum's BASIC (See Spectrum manual, chapter 14).

On the 48K version you can have 256 UDG's with numerical labels from Ø to 255. Numbers 32 to 127 inclusive are the characters used by BASIC but the others are only available in Spectrum FORTH. In fact on the 48K version two characters ! and @ have already been redefined. As will be seen by putting them on the screen.

To define or redefine a character, use command mode, and enter a lower case d then, when prompted, give the UDG a label. The label should be a letter for V16 and a number for V48. Finally the required character is created by entering 8 lines of 8 bit binary words i.e. eight ones or zeros or any combination. Using a 1 bit to turn a pixel ON and a Ø bit to turn a pixel OFF. The lines are entered one at a time as prompted and an enlarged version of the character is created in the middle of the screen as the lines are entered. It is the same method as used in BASIC and is covered in chapter 14 of the Spectrum manual.

You may find it useful to draw the required character on 8 x 8 squared paper and put a 1 on each square that the character occupies and a Ø for each background square. The other way round gives an inverse video effect. The required 8 bit words for the input are then read off horizontally from the top of the paper.

If you are using a printer the new character will be printed out when the definition is complete.

After defining the character, or after the printer stops, your are returned to command mode. To see

what the character looks like, actual size, enter its label. To print out the entire character set enter:

```
***> 256 Ø DO I EMIT LOOP ; .
```

To define a word the : (colon) definition is used. From command mode enter a : followed by a space and the name of the new word and then its definition. The definition is terminated in the usual way with a ; . Note, names should not start with a minus sign or digit. Only valid words already in the dictionary can be used in the definition of a new word, that is, both resident Spectrum FORTH words and any new words already added may be used. A space must be included between the : and word name, the name and definition words and between each word in the definition, in the usual way. For example a word to raise a number to the power of 2 (ie.square it) could be defined thus:-

```
***> : SQR DUP * ;
```

After pressing the ENTER key the name of the new word, in this case SQR, will appear at the top left of the screen, its position in memory at the top right and below will be its definition, in this case DUP *. If you are using a printer the name and definition will be printed out. Note, if it is only a short definition you can save printer paper by Breaking (shifted Space key) and then entering GO TO 100 to return to command mode. The new word is added to the dictionary and can be saved along with the rest of the dictionary. If you save the dictionary containing a word(s) for future use you are advised to maintain a record of the name(s) and corresponding definition(s). The section "An Introduction to FORTH" gives more detail on defining new words. Each word used in a definition either resident or one of your own, occupies about 3 bytes of memory. Limits of routines memory area are approximately, 1K for V16 and 20K for V48, see memory maps.

Now to discuss the number of bits operated on by

some of the commands. Logical operators AND, OR and XOR are full 16 bit operations.

NOT only compliments bit zero of TOS. If a 16 bit 1's compliment is required use -1 XOR.

IF, WHILE and UNTIL test bit zero of TOS only.

DO LOOP, during a DO loop three loop control items exist on the return stack.

Variables. In the V48 version it is not necessary to declare variables as there are 25 already available. They are labelled A, B,.....Y, Z but I is excluded for use in loops. These labels are in fact words and the variable can, if required, be renamed. To store TOS in a variable, use the letter (ie label) and ! (store). Example to set D to 50 enter:-

```
***> 50 D ! ;
```

To read a variable onto the stack use the label and @ (fetch). Example, to read D enter:-

```
***> D @ . ;
```

To directly read a variable use ? Example to read D enter:-

```
***> D ? ;
```

Variables can be incremented or decremented using +! Examples, to increment D by 50 and read the result enter:-

```
***> 50 D +! D ? ;
```

and to decrement D by 50 enter:-

```
***> -50 D +! D ;
```

The variable labels are words which put an address onto the stack. The above functions can be used like PEEK and POKE in BASIC by using an address instead of the label. The labels are not included in the following word list but can be listed from the program by using the command VLIST.

Using machine code. All the words in Spectrum FORTH, either resident or new, are stored as

compiled machine code routines which are called from each other. Machine code can be entered into any point in a definition as follows:-

From command mode or the prompt 'continue definition' enter the lower case letters mc. After pressing the ENTER key the prompt 'mc byte:' appears. Respond by entering one byte of machine code in decimal and pressing ENTER. The 'mc byte:' prompt will appear again when you can enter another single byte. Continue in this fashion until you have completed the mc and then enter -1 (minus one)

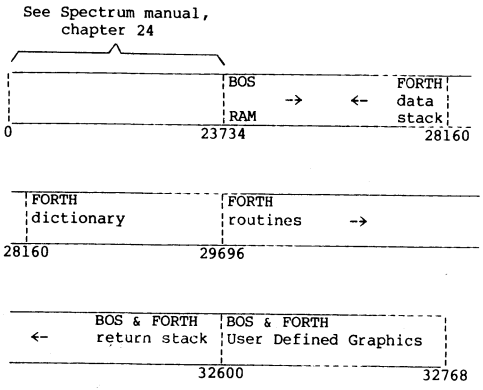
The screen can be copied to the printer from command mode by entering the lower case cp.

System variables used are:-

DC - CC for print positions
23681 for WORD displacement with the PAD;
23728 to store the stack pointer not currently in SP.

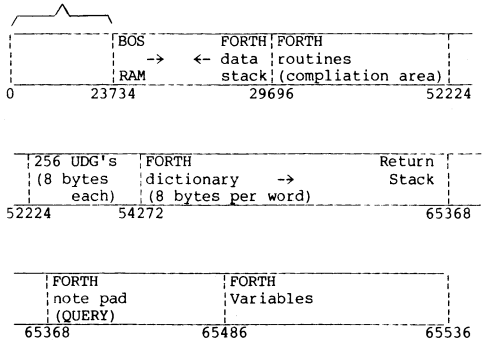
MEMORY MAP - 16K Version (V16)

Addresses Ø to 23734 as defined in the Spectrum manual, chapter 24.



MEMORY MAPS - 48K Version (V48)

Addresses Ø to 23734 as defined in the Spectrum manual, chapter 24.



SPECTRUM FORTH V16 WORD LIST - for 16K Version

The numbers in [] before each word refers to its location in the following dictionary.

[63]	!	[77]	'	[57]	()	[7]	*
[5]	+	[2]	.	[6]	-	[56]	."
[4]	:	[3]	;	[8]	/	[12]	<
[13]	>	[11]	=	[75]	?	[61]	@
[62]	C@	[64]	C!	[76]	C?	[65]	P@
[66]	P!	[44]	R@	[43]	R>	[42]	>R
[18]	ABS	[14]	AND	[22]	AT	[82]	ATTR
[48a]	BEGIN	[87]	BORDER	[84]	BRIGHT	[73]	CDUMP
[26]	CLS	[70]	CMOVE	[97]	cp	[25]	CR
[69]	DELETE	[46a]	DO	[29]	DROP	[74]	DUMP
[30]	DUP	[47b]	ELSE	[23]	EMIT	[68]	ERASE
[51]	EXIT	[52]	EXITLP	[24]	FIELD	[67]	FILL
[83]	FLASH	[88]	HIRES	[45]	I	[47a]	IF
[85]	INK	[58]	INKEY	[59]	KEY	[46d]	LEAVE
[46b]	LOOP	[46c]	+LOOP	[54]	MAX	[98]	mc
[55]	MIN	[9]	MOD	[71]	MOVE	[19]	NEGATE
[17]	NOT	[1]	number	[15]	OR	[32]	OVER
[91]	OVRPLOT	[86]	PAPER	[33]	PICK	[89]	PLOT
[92]	POINT	[49a]	REPEAT	[60]	RND	[36]	ROLL
[35]	ROT	[20]	SPACE	[21]	SPACES	[34]	SWAP
[47c]	THEN	[72]	TYPE	[90]	UNPLOT	[48b]	UNTIL
[95]	VLIST	[53]	WAIT	[49b]	WHILE	[16]	XOR

SPECTRUM FORTH V48 WORD LIST - for 48K Version

The numbers in [] before each word refers to its location in the following dictionary.

[63]	!	[77]	'	[57]	()	[7]	*
[5]	+	[2]	.	[6]	-	[56]	."
[4]	:	[3]	;	[8]	/	[12]	<
[13]	>	[11]	=	[75]	?	[61]	@
[62]	C@	[64]	C!	[76]	C?	[65]	P@
[66]	P!	[44]	R@	[43]	R>	[42]	>R
[28]	ABORT	[41]	>IN	[18]	ABS	[14]	AND
[22]	AT	[82]	ATTR	[94]	BEEP	[48a]	BEGIN
[87]	BORDER	[84]	BRIGHT	[73]	CDUMP	[26]	CLS
[70]	CMOVE	[81]	COUNT	[97]	cp	[25]	CR
[69]	DELETE	[37]	DEPTH	[46a]	DO	[29]	DROP
[74]	DUMP	[30]	DUP	[31]	?DUP	[47b]	ELSE
[23]	EMIT	[68]	ERASE	[79]	EXECUTE	[51]	EXIT
[52]	EXITLP	[80]	EXPECT	[96]	f	[24]	FIELD
[67]	FILL	[83]	FLASH	[50]	flgtst	[88]	HIRES
[45]	I	[47a]	IF	[85]	INK	[58]	INKEY
[93]	INVERSE	[59]	KEY	[46d]	LEAVE	[46b]	LOOP
[46c]	+LOOP	[54]	MAX	[98]	mc	[55]	MIN
[9]	MOD	[10]	/MOD	[71]	MOVE	[19]	NEGATE
[17]	NOT	[1]	number	[15]	OR	[32]	OVER
[91]	OVRPLOT	[40]	PAD	[86]	PAPER	[33]	PICK
[89]	PLOT	[92]	POINT	[34]	QUERY	[27]	QUIT
[49a]	REPEAT	[60]	RND	[36]	ROLL	[35]	ROT
[20]	SPACE	[21]	SPACES	[99]	STKSWP	[34]	SWAP
[47c]	THEN	[72]	TYPE	[90]	UNPLOT	[48b]	UNTIL
[95]	VLIST	[53]	WAIT	[49b]	WHILE	[39]	WORD
[78]	wrdsch	[16]	XOR				

Spectrum FORTH Dictionary

Abbreviations TOS - top of Stack, 2OS - second on stack, 3OS third on stack etc. are used thoughout and refer to the data stack.

NOTE Not all the Words in this dictionary are supported by V16, see the Word list for V16.

Examples These should be entered immediately following the ***> command mode prompt. Ensure that the stack is empty before entering each example. Spaces must be used between words, and numbers where required. The ENTER key must be pressed to input the example.

No. Word	Definition	Example and Comments Enter example after ***> prompt
(1) number	not a word in itself, any word starting with a digit or minus sign is evaluated and put on the top of the stack (TOS). In practice any denary integer number with or without a preceding minus sign. Remember any non-integer numbers are rounded either up or down as appropriate.	-3 . ; prints out -3, stack:0 4.5 . ; rounds up prints out 5 , stack:0 4.4 . ; rounds down prints out 4 , stack:0
(2) .	dot, the dot words prints out the TOS at the same time deleting it. A minus sign and comma after the thousands are included as appropriate.	10000 . ; prints out 10,000, stack : 0
(3) ;	semicolon, used to terminate all instructions, new word definitions etc.	
(4) :	colon, is used to create a dictionary entry.	: SQR DUP * ; defines a new word called SQR that will find the square of a number.

The next 5 words are arithmetic operators. The priority of the arithmetic operators is given by the order in which they are entered.

(5) +	sum, adds the 2OS to the TOS and puts the result onto the stack.	7 9 + . ; prints 16, stack:0
(6) -	difference, subtracts the TOS from the 2OS and puts the result on the stack.	7 9 - . ; prints -2, stack:0
(7) *	product, multiplies the TOS by 2OS and puts the result on the stack.	6 3 * . ; prints 18, stack:0
(8) /	division, takes 2OS and divides it by TOS, puts the result on the stack.	6 3 / . ; prints 2, stack:0
(9) MOD	MOD performs a division in exactly the same way as / but instead of giving the quotient as the result it gives the remainder. Very useful in FORTH'S integer arithmetic.	17 7 / . ; prints 2, stack:0 17 7 MOD . ; prints 3, stack:0
(10) /MOD	same as MOD except it gives both the quotient and remainder as the result. The quotient is TOS and the remainder is 2OS.	17 7 /MOD . CR . ; prints 2 3 stack:0

The next 3 words are comparisons.

(11) =	equality, compares and tests for equality 2OS with TOS. The result, 1 for true, 0 for false, is put on the stack.	1 0 = . ; prints 0, stack:0 1 1 = . ; prints 1, stack:0
(12) <	less than, compares 2OS with TOS. If 2OS is less than TOS, result is 1, if 2OS is greater than or equal to TOS result is 0	4 5 < . ; prints 1, stack:0 5 4 < . ; prints 0, stack:0
(13) >	greater than, compares 2OS with TOS. If 2OS is greater than TOS, result is 1. If 2OS is less than or equal to TOS result is 0.	13 11 > . ; prints 1, stack:0 11 13 > . ; prints 0, stack:0

The next three words are logical operators and are used for combining the results of comaprisons as in BASIC. They are full 16 bit operations.

(14) AND	does an AND operation on TOS and 2OS and leaves the result on the stack. Result is 1 if both TOS and 2OS is 1, otherwise result is 0.	1 1 AND . ; prints 1, stack:0 0 0 AND . ; prints 0, stack:0 1 0 AND . ; prints 0, stack:0
(15) OR	does an OR operation on TOS and 2OS and leaves the result on the stack. Result is 1 if either TOS or 2OS is 1 otherwise result is 0.	1 0 OR . ; prints 1, stack:0 0 0 OR . ; prints 0, stack:0 1 1 OR . ; prints 1, stack:0
(16) XOR	does an XOR operation on TOS and 2OS and leaves the result on the stack. Result is 1 if, and only if, TOS and 2OS are different otherwise result is 0.	1 0 XOR . ; prints 1, stack:0 1 1 XOR . ; prints 0, stack:0 0 0 XOR . ; prints 0, stack:0
(17) NOT	changes TOS from a conditional true result to a conditional false result by complementing bit zero (LSB). NOT is designed for use with the Words: IF, WHILE and UNTIL.	0 NOT . ; prints 1, stack:0
(18) ABS	gives the absolute value of TOS. ie. if TOS is negative the minus sign will be removed.	-20 ABS . ; 20 ABS . ;
(19) NEGATE	changes the sign of the TOS. If it is a positive number it becomes negative and vice versa.	3 NEGATE . ; -3 NEGATE . ;
(20) SPACE	moves the print position on by one space.	1 3 5 7 ;

(21) SPACES	takes TOS and performs SPACE that many times. Can be used for tabulating the screen a printout. Warning - Ø SPACES performs it 65,536 times!	. 2 SPACES . 4 SPACES . 6 SPACES . ;
(22) AT	used for formating the screen, takes 2OS as the line number and TOS as column number. Similar to AT in BASIC.	CLS 12 5 AT . "OOOOO" ;
(23) EMIT	takes TOS and prints out the associated (Spectrum) character that it represents. Or a user defined character. Similar to CHR$ in BASIC.	127 EMIT ;
(24) FIELD	moves the print position along into the next quarter of the screen.	1 10 100 1000 1000 . FIELD . FIELD . FIELD . FIELD . ;
(25) CR	starts, the print position, onto a new line.	substitute CR for FIELD in above example
(26) CLS	clear screen as in BASIC, clears display and attributes	CLS 127 EMIT WAIT ; press the Y key to return to command mode
(27) QUIT	returns to the FORTH command mode.	
(28) ABORT	clears the stack and performs QUIT	1 2 3 ABORT ; Prints 0 stack:0
(29) DROP	discards TOS, making the stack one item shorter.	3 2 1 . . . CR 3 2 1 DROP . . ;
(30) DUP	copies (duplicates) TOS so that it is on the stack twice.	3 2 DUP . . . ;
(31) ?DUP	performs DUP only if TOS is non-zero. Otherwise ignored	compare the following result 1 ?DUP . cr . ; Prints 1 1 stack:0 0 ?DUP . CR . ; Prints 0 0 stack:0
(32) OVER	copies 2OS onto TOS, pushes the stack down, ie TOS becomes 2OS, 2OS becomes 3OS etc.	3 2 OVER . . . ;
(33) PICK	like DUP and OVER, but it copies any stack item selected.	40 30 20 10 ; 3 PICK . ; or 4 PICK . ;
(34) SWAP	swaps round TOS and 2OS	9 8 7 SWAP . . . ;
(35) ROT	removes 3OS and puts it on top of the stack pushing down the old TOS and 2OS into the space left.	9 8 7 6 ROT . . . . ;

(36) ROLL	like ROT but for any stack item specified, 6 ROLL will put the 6OS on top and push the old TOS etc down.	19 18 17 16 12 14 5 ROLL . . . . . ;
(37) DEPTH	gives the number of items on the stack, before DEPTH was put on.	DEPTH . ; Prints 0 stack:0
(38) QUERY	inputs characters from the keyboard into the note-pad area of memory (see memory map) until ENTER is pressed. The enter character (13) is stored to mark the end of the data. Do not use more than 100 characters in response to a QUERY.	
(39) WORD	reads one character off the pad onto the stack, working through the pad each time it is called.	
(40) PAD	gives the address of the pad.	
(41) >IN	gives the address containing an indicator of how far WORD has read through the pad. >IN c@ gives the number of characters WORD has read so far.	
(42) >R	(To R), transfers one item from the data stack to the return stack.	For use with Return Stack
(43) R>	(From R), transfers one item from the return stack to the data stack.	For use with Return Stack
(44) R@	(R fetch), copies the top of the return stack onto the data stack.	For use with Return stack
(45) I	the looping variable in a DO-LOOP. I is the same as R@, the top of the return stack is the loop index variable.	See DO-LOOP
(46) DO-LOOP (LEAVE)	this type of loop can be nested to any depth. I always refers to the innermost loop at that point. Note a loop of length Ø will execute 65536 times. DO cannot exist in a word without LOOP or +LOOP and vice versa.	
(46a) DO	(similar to FOR in BASIC), takes the TOS as the first value and 2OS as the loop limit (first 'illegal' value). It places three loop control items on the Return stack and must therefore always be used in the same definition/command as LOOP	1001 0 DO 23 0 AT I . LOOP ;
(46b) LOOP	(similar to NEXT in BASIC), tests loop count against looping variable which is always I.	128 0 DO I EMIT FIELD LOOP ;
(46c) +LOOP	used instead of LOOP in a DO-LOOP. +LOOP takes TOS as the loop step value (as STEP in BASIC). Only positive increments can be used.	128 0 DO I EMIT FIELD 3 +LOOP ;

(46d) LEAVE	forces an early exit from a DO style loop by making the looping variable I equal to the top limit. LEAVE must occur in the same word/command as the DO and LOOP (or +LOOP) it refers to.	32 300 0 DO 1 + DUP EMIT DUP 127 = IF LEAVE ELSE THEN LOOP DROP ;
(47) IF ¦ ELSE ¦ THEN	a conditional loop, all three words must appear in a word/command. It does not matter if there are no words between, for example, ELSE and THEN. IF cannot be nested but new words can be defined for inner IFs should nesting be required.	5 5 = IF 444 . ELSE 333 . THEN ;
(47a) IF	takes TOS and tests it, if the result is true (1) the words between IF and ELSE are executed	
(47b) ELSE	if the result is false (0) the words between ELSE and THEN are executed.	5 6 > IF 444 . ELSE 333 . THEN ;
(47c) THEN	finally after having executed one set of words (47a) or (47b), the words following THEN are executed.	5 6 > IF 444 . ELSE 333 . THEN 128 32 DO I EMIT FIELD LOOP ;
(48) BEGIN ¦ UNTIL	another conditional loop and like IFs they cannot be nested.	
(48a) BEGIN	marks the beginning of the loop	8192 BEGIN DUP . CR 2 / DUP 0 = UNTIL ;
(48b) UNTIL	marks the end of the loop and it must be preceded by a condition. The words after BEGIN are executed UNTIL the condition is true.	
(49) BEGIN ¦ WHILE ¦ REPEAT	similar to a BEGIN-UNTIL loop. BEGIN marks the start as in (48a) above.	
(49a) REPEAT	marks the end of the load and is <u>not</u> conditional.	
(49b) WHILE	is used to test a condition and execution, of the loop, continues WHILE the condition is true. It jumps out of the loop when the condition is found to be false.	8192 BEGIN DUP 1 > WHILE DUP . 61 50 47 EMIT EMIT EMIT 2 DUP . CR REPEAT DROP ;
(50) flgtst	reads TOS bit 0 and sets the Z flag appropriately. Used by IF, WHILE and UNTIL. Therefore these read even numbers as false and odd numbers as true.	
(51) EXIT	when used in a command of definition causes the words which follow EXIT to be abandoned. It cannot be used from inside a DO style loop	5 , 4 , EXIT 3 . 2 . 1 . ;
(52) EXITLP	the same as EXIT but designed for use in a DO style loop.	100 0 DO I . FIELD I 10 = IF EXITLP ELSE THEN LOOP ;
(53) WAIT	halts execution until the Y key is pressed, a flashing block at the bottom right of the screen acts as a reminder that the computer is waiting.	
(54) MAX	takes TOS and 2OS and replaces only the larger of the two back onto the stack.	5 6 MAX . FIELD 6 5 MAX . FIELD ;
(55) MIN	takes TOS and 2OS and replaces only the smaller of the two back onto the stack.	5 6 MIN FIELD 6 5 MIN . FIELD ;
(56) ."	print string, the string to be printed must have quotes both before and after and be preceded by a . (dot).	."dot-quotes." CR ." works for spaces and strings."
(57) ()	everything with brackets in FORTH is ignored. Like a REM statement in BASIC. See rules for nesting brackets.	
(58) INKEY	puts the ASCII value of the currently pressed key onto the stack, or 255 if no key is pressed.	
(59) KEY	waits for a key to be pressed and then puts its value onto the stack	100 0 DO KEY EMIT LOOP ;
(60) RND	places a random number between Ø and 255 onto the stack	60 0 DO RND . FIELD LOOP ;
(61) @	fetch, fetches the two byte number from the address pointed to by TOS and puts that number onto the stack.	
(62) C@	byte-fetch, fetches the single byte from the address pointed to by TOS and puts it on the stack.	
(63) !	store, stores the two byte number 2OS in the address pointed to by TOS	
(64) C!	byte-store, stores the least significant byte of the number 2OS in the address pointed to by TOS	
(65) P@	port fetch, port IN control instruction fetches the number from the port, address indicated by TOS and puts the result onto the stack	248 254 P! 254 P@ . WAIT ;
(66) P!	port out, outputs the value 2OS to the port, address, indicated by TOS	

(67) FILL	fills 2OS bytes with the value of TOS starting at address indicated by 3OS.	Care should be taken when using this word that required data is not destroyed.
(68) ERASE	fills TOS bytes with the value zero starting at address indicated by 2OS	Care should be taken when using this word that required data is not destroyed.
(69) DELETE	fills TOS bytes with the value 32 (ASCII space) starting at address indicated by 2OS.	Care should be taken when using this word that required data is not destroyed.
(70) CMOVE	moves single byte values in memory to another area of memory. TOS indicates the number of bytes to be moved, 3OS indicates the address they are to be moved from and 2OS indicates their new location.	If data is being moved up memory the source and destination blocks must not overlap.
(71) MOVE	is the same as CMOVE except it transfers two byte values, not bytes.	If data is being moved up memory the source and destination blocks must not overlap.
(72) TYPE	prints, on the screen as characters a number of bytes, indicated by TOS, starting at address indicated by 2OS.	5000 500 TYPE ;
(73) CDUMP	prints, on the screen a list, length indicated by TOS, of addresses and corresponding single byte contents, starting from the address in 2OS. The Word WAIT is incorporated in this command requiring the Y key to be pressed to print the list.	23552 8 CDUMP ;
(74) DUMP	same as CDUMP except the two byte contents of the addresses are printed out.	23552 4 DUMP ;
(75) ?	read, prints on the screen, the two byte value stored at the address indicated by TOS.	
(76) C?	byte read, same as ? except it prints the single byte value.	
(77) ' or FIND	puts, as a result, onto the stack the address of the next Word (routine) in the current input command/definition, that Word itself not being executed. Use only to find new words. ie. do not use for resident words.	
(78) wrdsch	searches for the word whose name is stored in six bytes at 23264, in the dictionary. It returns the address of the dictionary entry (or zero if not found) in the BC register pair. Each entry consists of 6 bytes of word name followed by 2 bytes of compilation address.	
(79) EXECUTE	causes a jump to address TOS	
(80) EXPECT	inputs a number, up to the value of TOS, characters from the keyboard and stores them at address 2OS. Input can be terminated early by pressing ENTER, in which case the value 13 is stored.	
(81) COUNT	for use with a table of bytes in which the first byte contains the length of the table. The tables address is TOS. TOS is incremented and the length of the table is placed above it on the stack.	
(82) ATTR	converts a column number, given by TOS, and a line number, given by 2OS, into an address within the attribute file.	ATTR C@ is equivalent to the BASIC ATTR.
(83) FLASH (84) BRIGHT (85) INK (86) PAPER	these four words ((83) to (86)) are similar to the BASIC statements of the same name but only change the status of a single character cell. Consequently they are usually used in loops. The character cell changed is at line given by 2OS and column given by TOS and the operand is at 3OS and should be the same value as would be used in the Spectrum's BASIC. Note, ATTR is used in the definition of these Words.	0 12 10 0 12 10 1 12 10 1 12 10 FLASH BRIGHT WAIT FLASH BRIGHT ; 127 EMIT 6 23 0 INK 2 23 0 PAPER WAIT ;
(87) BORDER	same as in BASIC, the BORDER colour, ie. operand data, is held in TOS and should be in the range 0 to 7, ie. Spectrum colour range.	5 BORDER WAIT ; 3000 0 DO 8 0 DO I BORDER LOOP LOOP ;
(88) HIRES	used in the definition of the next four ((89) to (92)) words. Takes the hires coordinates x at 2OS, y at TOS and sets DF-CC to point at the byte containing that pixel, leaving 8 times the bit number on the data stack. x is in the range 0 to 255 and y is in the range 0 to 191.	
(89) PLOT	plot, usually used in loops. The pixel coordinates and ranges are given in (88) above.	120 -199 CLS DO I 128 + I I * 75 / PLOT LOOP WAIT ;
(90) UNPLOT	unplot, changes the pixel, coordinates and ranges as given in (88) above, colour to that of the paper.	32 24 * 1 DO 0 EMIT LOOP 120 -119 DO I 128 + I I * 75 / UNPLOT LOOP WAIT ;
(91) OVRPLOT	changes the pixel, coordinates and ranges as given in (88) above, from ON to OFF or vice versa.	
(92) POINT	this puts, as a result, onto the stack -1 for ON and 0 for OFF.	

(93) INVERSE — inverses the entire character set

```
INVERSE 255 0 DO I
EMIT LOOP ;
INVERSE ;
```

(94) BEEP — to control the Spectrum's BEEPer. Takes 2OS as the duration and TOS as the pitch. The same pitch value always gives the same pitch but durations depends on both 2OS and pitch.

```
1000 1000 BEEP ;
200 0 DO 10 I BEEP
LOOP ;
```

(95) VLIST — lists the names of all the words both resident and new, in the dictionary. The flashing cursor at the bottom right of the screen reminds that the Y key should be pressed to scroll the list and return to command mode.

(96) f — forget, used for deleting words from the dictionary. Enter a lower-case f and then the name of the WORD to be forgotten. Warning forget deletes the named word and all those subsequently defined.

(97) cp — lower-case cp copies the screen to the printer.

Not a FORTH Word

(98) mc — lower-case mc for entering machine code. Can be used during command or definition. n1, n2, ...n, is a machine code byte, in decimal.

Not a FORTH Word

(99) STKSWP — stack swap, used a first time, makes SP point at the data stack. When next used, points SP back at return stack.

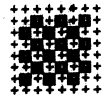
APPENDIX A

SPECTRUM FORTH GAME

```
: SCREEN                                                    31361

CLS.."£££abcdefghijklmnopqrstuvwxyz£££££$$$$$$$$$$
$$$$$$$$$$$$$$$$$$£££"19 1 DO I 10 < IF ."0" ELSE
THEN I . ."$$$$$$$$$$$$$$$$$$$$$$$$$$$$$" I 10 < IF
."0" ELSE THEN I . LOOP ."£££$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$ £££££abcdefghijklmnopqrstuvwxyz£££" 22528 768
49 FILL 6 BORDER ;

: TURN9                                                     31633

48 - SWAP 48 - 10 * = 32 * SWAP 65 - + 22530 + 3 0
DO DUP DUP @ 16191 XOR SWAP ! DUP 2 + DUP C@ 63 XOR
SWAP C! 32 + LOOP DROP ;

: INP3                                                      31775

BEGIN 23 0 AT ."Give letter   " KEY DUP 64 > OVER 9
1 < AND NOT WHILE DROP REPEAT BEGIN 23 0 AT . "Give
1st digit" KEY DUP 48 = OVER 49 = OR NOT WHILE DROP
REPEAT BEGIN 23 0 AT ."Give 2nd digit" KEY DUP 47 >
OVER 58 < AND NOT WHILE DROP REPEAT ;

: RAND                                                      31992

23672 @ BEGIN 23672 @ OVER = NOT UNTIL 16383 AND C@
;

: SKILL                                                     32034

SCREEN 23 0 AT ."Skill level" KEY 48 - 100 * 0 DO
BEGIN RAND 31 AND DUP 25 > WHILE DROP REPEAT 65 +
RAND 1 AND BEGIN RAND 15 AND OVER 10 * OVER + 17 >
WHILE DROP REPEAT 49 + SWAP 48 + SWAP TURN9 LOOP ;
```

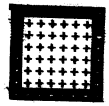
```
: FINI?                                                    32218

1 23296 22528 DO I C@ 49 = AND LOOP ;

: GAME                                                     32257

SKILL BEGIN INP3 TURN9 FINI? UNTIL WAIT ;
```

Redefinition of character 35



Character 35

Redefinition of character 36



Character 36

APPENDIX B

SPECTRUM FORTH - COMPARATIVE TIMINGS

These timings, in seconds, are for 1000 operations. Those for the Jupiter Ace and Spectrum BASIC are taken from Jupiter Cantabb's own advert while those for Spectrum FORTH are from the author. This comparison shows Spectrum FORTH to be faster than the Jupiter Ace except for screen handling which is not relevant due to the Spectrum's HIRes graphics used by this FORTH.

OPERATION	Time - seconds		
	JUPITER ACE	SPECTRUM BASIC	SPECTRUM FORTH
Empty loop	0.12	4.2	0.05
Print number	7.5	19	1.9
Print character	0.62	7.5	0.7
Addition	0.45	7.5	0.23
Multiplication	0.9	7.5	0.5