ABERSOFT Fig-FORTH

SOFT 188 (Cassette)

SOFT 1188 (Disc)

For the Amstrad CPC464/664

Published by **AMSOFT**, a division of

Amstrad Consumer Electronics plc
Brentwood House
169 Kings Road
Brentwood
Essex

First edition 1985

Contents

Chapter 1. Introduction.

Starting Out.

This manual is not intended to be a tutorial of FORTH, there are many books available on the market that do this more than adequately, however, Chapter 2 gives a short but useful guide to the basics of FORTH, along with the extensions presented for the Amstrad CPC464. For the beginner to the language, a good book is 'Starting FORTH' by Leo Brodie, even though this describes FORTH-79 rather than fig-FORTH. For the more advanced user, a useful reference is 'The Systems Guide to fig-FORTH' by C.H.Ting.

FORTH is a language which combines the features of high level languages with the speed of machine code. The FORTH interpreter interprets each line of commands by searching through an internal dictionary of words which it understands. It then acts on these words. Some of the words allow creation of new words, which in turn form the basis for even higher level words. Eventually a whole program comes together, and is called by a single word. This gradual development allows for better checking of sections of a program than BASIC does.

Setting Up.

1) Tape System.

To install the basic system on your Amstrad CPC464 system, type

```
RUN "FORTH"
```

and press play on the DATACORDER.

When the compiler has loaded, you will be greeted by the message:-

```
Amstrad CPC464 fig-FORTH 1.1B
(c)Abersoft:1984
```

and the standard cursor.

2) Disc System.

To install the basic system on your Amstrad CPC464 CP/M system, type

```
A>FORTH <filename>
```

where filename is the name of a file that will be used to store editor source screens on. If no filename is given, `FORTH.FIG` is defaulted. Whatever name is given, the extension `.FIG` is added. (This file is the first file that is used during a session with the compiler, if at any time during programming another `.FIG` file is required, `FILE` ‹filename›, e.g. `FILE EDITOR`, is used. This closes the present file and opens the new one for I/O. Note that no extension is allowed.

When the compiler has loaded, you will be greeted by the message:-

```
Amstrad CPC464 CP/M fig-Forth 1.1B
(c)Abersoft:1984
```

and the standard cursor.

To test that the system is running, just hit the **[ENTER]** key and the system should respond

```
ok
```

You are now ready to begin programming in FORTH.

Chapter 2. Beginners Guide to Forth

The Stack.

One of the major differences between FORTH and most other high-level languages is that FORTH uses Reverse Polish Notation (RPN). Normally, the operator + comes between the numbers you wish to add (e.g. 4 + 7). In RPN, the operator comes after the numbers (i.e. 4 7 +) instead. (Note that you need to insert spaces between the 4, 7 and +). This is because a stack is used to store numbers when evaluating expressions. (In case you don't know what a stack is, imagine a pile of plates. The last plate put on the pile, being on top, is the first to come off again). When FORTH finds a number, it puts it onto the stack. When it finds an operator, it takes the required numbers off the stack, then puts the result back onto the stack. The word dot . takes a number off the top of the stack and prints it.

The BASIC program line

```
PRINT (3+7)*(8+2)
```

is, in FORTH,

```
3 7 + 8 2 + * .
```

(note the spaces are important).

After executing: the stack becomes:

3	(3) top of stack on this side
7	(3,7)
+	(10)
8	(10,8)
2	(10,8,2)
+	(10,10)
*	(100)
.	() 100 is printed.

Practice is the only way to get familiar with RPN. Probably the best thing to do now, if you are a beginner to programming, is to sit at your keyboard and try a few examples. After a short while, RPN will become as easy as normal arithmetic and you will find no problem in your application programming. You may find sometimes that you get the message ? MSG # 1 when you try to do something, this merely means that you tried to use more information than you have placed on the stack. Similar words explained in the reference guide are:

```
+ - / */ */MOD /MOD MOD MAX MIN AND OR XOR MINUS +- 1+ 2+.
```

There are some words which shuffle the numbers on the stack
DUP duplicates the top number
DROP discards the top number
SWAP swaps the two top numbers
ROT rotates the top three numbers, bringing the third to the top.
OVER copies the second value over the top as the new first value.

So, for example:

```
1 DUP . . prints 1 1 ok
1 2 DROP . prints 1 ok
1 2 SWAP . . prints 1 2 ok
```

See also: SØ SP@

Variables and Constants.

A variable in FORTH must be explicitly created before it is used, using the word VARIABLE.

```
3 VARIABLE A1
```

will create a variable A1 with initial value 3. Any sequence of alpha-numeric characters will work as a name.

```
6 A1 !
```

stores 6 in A1 once A1 has been defined as a variable. (This is equivalent to A1=6 in BASIC)

A1 @ . (can also be written as A1 ?)

will fetch the contents of A1 and print it. The word A1 actually puts the address of A1 on the stack. ! takes an address off the stack then takes a number off the stack and stores it at that address. The word @ takes an address off the stack and replaces it with the contents of memory at that address. The word ? is the equivalent of @ . for those who are lazy. See also the word +!.

A constant in FORTH is a fixed number which is given a name, using the word, CONSTANT.

```
1Ø CONSTANT TEN
```

creates a constant, TEN, with the value 10. From now on, each time the word TEN is found, 10 will be pushed onto the stack, so

```
TEN .
```

will print:

```
1Ø ok
```

Note that ! and @ are not needed with constants, and in fact ! cannot be used to change a CONSTANT, because CONSTANTs are constant!

Among the predefined system variables there is BASE. BASE contains the current number base used for input and output. HEX stores 16 (base 10) in BASE, setting hexadecimal mode. DECIMAL stores 10 (base 10) in BASE, setting decimal mode.

Types of numbers.

FORTH is a typeless language. There is no distinction between a number and a character for example. Things on the stack are taken by a word and interpreted as that word sees fit. Some of the more common interpretations, other than the normal 16 bit signed numbers (range -32768 to 32767) which have been used, are:

Unsigned:

range 0 to 65535. The word U. acts like . but it assumes that the number is unsigned rather than signed.

Double precision:

The top two numbers on the stack are interpreted as a 32 bit number, either signed or unsigned. The top number is taken as the high 16 bits. The second number is taken as the low 16 bits. To put a 32 bit number onto the stack, type the number with a decimal point somewhere in the number. The system variable DPL contains the number of digits following the decimal point, in case it is important.

```
70.000 . . DPL @ .
```

will print 1 4464 3 ok. The 32 bit number 70000 (not 70) is put onto the stack as two numbers. The first . prints the top half as a signed number. The bottom half is then printed (note 70000 = 1 x 2 ↑ 16 + 4464). The contents of DPL are 3 since there are 3 digits after the decimal point.

The words U* U/MOD M* M/ and M/MOD are mixed mode versions of * / and /MOD where the operands and results are of different types.

Byte:

Sometimes only 8 bit numbers are required, for instance to represent a character. They are represented on the stack by 16 bit numbers with the 8 highest bits zero. C@ and C! fetch and store only a single byte at a time.

Flags:

To make decisions, the values true and false are needed. For instance:

```
2 3 = .
```

prints 0 (false), whereas:

```
3 3 = .
```

prints 1 (true). In fact any non-zero number is treated as true. (So, - can be used for <>, since x-y=0 (false) only if x=y.) Other comparisons are:

< U< 0< (equivalent to 0 <) and 0= (equivalent to 0 =).

NOT changes a true flag to false and vice versa. (This is actually equivalent to 0=.)

FORTH modes.

FORTH has two different ways of working. One is the interpretive mode and the other is the defining (or compiling) mode.

In the interpretive mode, FORTH takes whatever is input and tries to execute it immediately. If you type

```
4 7 + .
```

the computer will give an immediate answer of

```
11 ok
```

However in compiling mode, FORTH takes what is input and stores it away in its dictionary and uses it later. As an example, if you wanted to input the sum above, but only see the result later (a rather strange thing to want to do) you could define a new word, thus:

```
: STRANGE 4 7 + ;
```

(: is the word used by FORTH to mean begin compiling, ; is the word for finish compiling). If you then type

```
STRANGE
```

and **[ENTER]**, the computer will then give you

```
11 ok
```

Once a word is defined in this way, it can be used in other definitions:

```
: TOOSTRANGE STRANGE STRANGE ; TOOSTRANGE
```

compiles a word TOOSTRANGE which will perform STRANGE twice, then executes the word, printing

```
11 11 ok
```

Note that as far as FORTH is concerned, once a word has been defined using : ... ; it then becomes part of the language and can be used in exactly the same way as any other word, like SWAP and DROP. When defining new words, try to name them so that they mean what they do. In the middle of a large piece of work, a definition named CENT-FAHR is pretty obvious, whereas CF will not be. (As an example of a more useful FORTH definition than STRANGE, here is CENT-FAHR

```
: CENT-FAHR     ( expects a single number on the stack)
9 *             ( multiply by 9)
5 /             ( divide by 5)
32 +            ( Add 32)
.               ( Print the answer)
;               ( Definition finished)
```

now typing 10 CENT-FAHR (return) will give a screen display of

```
10 CENT-FAHR 50 ok ).
```

Control Structures.

The FORTH structure IF.....ENDIF (or IF....ELSE....ENDIF) allows conditional execution (only) within a definition. The equivalent of BASIC's IF A>2 THEN PRINT "TOO BIG" is

```
: TEST A @ 2 > IF ." TOO BIG" ENDIF ;
```

A @ 2 > put a flag on the stack stating whether A is greater than 2. IF removes the flag. If the flag is true (non-zero) then the following code is executed. The word ." prints everything up to the next double quote ". (Note that there has to be a space after ." for it to be recognised properly). If the flag is false (zero) then the code up to ENDIF is skipped. In the case of IF ... ELSE ... ENDIF, if the flag is true the code between IF and ELSE is executed and the code between ELSE and ENDIF is skipped. If the flag is false, only the ELSE ... ENDIF code (and that which follows ENDIF as usual) is executed. Inside any IF ... ENDIF clause you can have another one.

BASIC's FOR A=1 TO 10 : PRINT A: NEXT A

is FORTH's

```
: TEST 11 1 DO I . LOOP ;
```

When TEST is executed, DO takes two numbers off the stack. The top number (1 here) is the starting value. The second number (11) is the limit. I copies the loop index (A in the BASIC program) onto the stack, where it is printed by . . LOOP increments the loop index by 1 . If the limit is reached or exceeded, execution continues as normal, otherwise it loops back to the DO. Hence the limit being 11 in order to count to 10. Note that the loop is done at least once no matter what the limit is. n +LOOP instead of LOOP is FORTH's equivalent of STEP n in BASIC. LEAVE changes the limit so that the loop will be left as soon as LOOP is reached. J returns the value of the outer loop counter if you have a a nested loop structure. For example:-

```
: DEMO CR
         4 1 DO
           9 7 DO
             I J . . CR
           LOOP
         LOOP
;
```

will print:-

```
1 7
1 8
2 7
2 8
3 7
3 8
```

Another type of loop is BEGIN ... flag UNTIL.

```
: TEST BEGIN ... A @ 2 > UNTIL ;
```

is the equivalent of

```
10 REM BEGIN
20 . . .
30 IF NOT(A>2) THEN GOTO 10
```

i.e. ... will be executed over and over until A>2.

This is a very useful instruction when we need to repeat a loop over and over until the **[ESC]** key is pressed. This can be done like so:-

```
: TEST BEGIN ." LOOPING" CR ?TERMINAL UNTIL ;
```

the ?TERMINAL leaves a flag on the stack of true if **[ESC]** is pressed and false if it is not.

If it is required for a loop to continue forever, BEGIN .... AGAIN can be used. This has no conditions and would be very useful when writing a game program. e.g.

```
: GAME
        TITLES
        BEGIN
                PLAY-GAME
                END-MESSAGE
        AGAIN
;
```

This also shows how programs should be put together, with the final word used to call a program, consisting of only a few, already tested and built up words.

The problem with the above conditional structures is that they are all executed at least once, as the test is cariied out at the bottom of the loop. Certain programs will require the test to be carried out at the top of the loop, this is done with the BEGIN ..... flag WHILE .... REPEAT structure. For example:-

```
: DEMO
        BEGIN DUP 4 > WHILE
        DUP . 1 - REPEAT
        DROP
;
```

If 7 DEMO is entered, the output will be 7 6 5 , if 3 DEMO is entered, there will be no output.

One of the most useful commands in a language such as Pascal, is the CASE statement. This allows for the testing of a number for many different values and executing different procedures on each value. This is available in standard FORTH only by using many nested IF ... IF ... IF ... ENDIF ENDIF ENDIF and can be very tedious. A new structure in Abersoft FORTH is naturally called the CASE structure. Its use is :-

..... (instructions leaving a single value on the stack)

```
CASE
        8 OF ." Thisis8" CR ENDOF
        12 OF ." Thisis12" CR ENDOF
        99 OF ." Thisis99" CR ENDOF
ENDCASE
```

This structure provides a much more readable and less error prone way of making multiple decisions. In the above example, if 99 was left on the stack This is 99 would be printed. Any value other than 8, 12 or 99 would produce no output at all.

Keyboard and Screen I/O.

You have met the word ." which prints a fixed message on the screen. EMIT expects the ASCII code for a character on the stack. It then prints that character.

```
65 EMIT
```

will print the letter A (ASCII code 65).

TYPE is used for printing strings. The variable TIB contains the address of the Terminal Input Buffer where what you type is stored. To type out the first 10 characters in the buffer, you need to supply TYPE with the starting address, and the number of characters to be printed:

```
TIB @ 10 TYPE
```

will print

```
TIB @ 10 T
```

SPACES takes a number n from the stack and prints out n spaces.

KEY waits for a key to be pressed, then puts the ASCII code ofthat key onto the stack.

EXPECT requires an address and a count, just lik TYPE. However, EXPECT takes the specified number of characters from the keyboard (or all the characters up to **[ENTER]**) and tacks one or two nulls (ASCII 0) on the end. The word QUERY is defined as 'TIB @ 80 EXPECT.

GET is similar to KEY but whereas **[KEY]** only allows ASCII codes 13 and 32 to 127 to be input and switches on the cursor, GET returns all key values, without the cursor appearing.

The Amstrad Extensions

These extensions are presented in alphabetic order in the same way as the Basic User Guide.

```
n1 n2 BORDER
```

Sets the border to colours n1 and n2. if the border is required to be one steady colour, n1 and n2 should be the same.

```
CLG
```

Clears the graphic window.

n CHAN

Sets the text stream to n.

n1 n2 DRAW

Draws a line from the current position to n1,n2.

n1 n2 DRAWR

Draws a line n1,n2 relative to the current position.

(nnc nnb nna...n1c n1b n1a) p n ENT

Sets tone envelope n to be set to the parameters n1a etc, the number of groups of parameters being p, and being the same as explained in the User Guide.

(nnc nnb nna...n1c n1b n1a) p n ENV

Sets amplitude envelope n to be set to the parameters n1a etc, the number of groups of parameters being p, and being the same as explained in the User Guide.

n GPAPER

Sets the graphics paper to ink n.

n GPEN

Sets the graphics pen to ink n.

n1 n2 n3 n4 GWINDOW

Sets the graphics window to n1 (left), n2 (right), n3 (top), n4 (bottom).

n1 n2 n3 INK

Sets ink n3 to colours n1 and n2.

n1 INKEY n2

Tests key n1, returns -1 if key is not pressed, Ø if key alone is pressed, +32 if **[SHIFT]** is pressed, +128 if **[CTRL]** is pressed.

INKEY$ n

Returns the value of the current key pressed.

n1 JOY n2

Returns the value n2 of joystick n1.

n1 n2 LOCATE

Sets the text cursor to n1,n2 on the current text stream.

n MODE

Sets the screen mode to n.

n1 n2 MOVE

Moves the graphics cursor to n1, n2.

n1 n2 MOVER

Moves the graphics cursor to n1,n2, relative to the current plot position.

n1 n2 ORIGIN

Sets the starting point for the graphics cursor to n1,n2.

n PAPER

Sets the paper on the current text stream to ink n.

n PEN

Sets the pen on the current text stream to ink n.

n1 n2 PLOT

Plots the point n1,n2.

n1 n2 PLOTR

Plots the point n1,n2 relative to the current plotting position.

POS n

Returns the horizontal position of the text cursor on the current text stream.

n RELEASE

Releases the sound on channel n.

SCREEN n

Returns the character at the current position. If no recognisable character, returns 0.

n1 n2 n3 n4 n5 n6 n7 SOUND

Refer to Chapter 6 of the USER GUIDE. (6.10) Where n1=M, n2=L, n3=K, n4=J, n5=I, n6=H and n7=G.

n1 n2 SPEEDINK

Sets the flash rate of inks to n1,n2.

n1 n2 SPEEDKEY

Sets the start delay and repeat period to n1 and n2 respectively.

n SPEEDWRITE

If n is zero sets the slow cassette rate, else sets the high rate.

n1 SQ n2

Returns n2, the status of sound queue n1.

n1 n2 n3 n4 n5 n6 n7 n8 n9 SYMBOL

Redefines character n9 to the values n1-n8 as long as that character has been allowed to be redefined in SYMBOLAFTER. n8 is the top line of the character and n1 the bottom line.

n SYMBOLAFTER

Allows all characters after n to be user-defined by symbol.

TAG

Sets the current text stream cursor to the current graphics cursor.

TAGOFF

Reverses the action of TAG.

n1 n2 TEST n3

Returns the ink value of the graphics point n1,n2.

`n1 n2 TESTR n3`

Returns the ink value of the graphics point `n1,n2` relative to the current graphics cursor.

`VPOS n`

Returns the vertical position of the text cursor on the currenttext stream.

`n1 n2 n3 n4 WINDOW`

Sets the text window to `n1`(left), `n2`(right), `n3`(top), `n4`(bottom) on the current text stream.

`n1 n2 WINDOWSWAP`

Exchanges the text windows `n1` and `n2`.

`XPOS n1`

Returns the x position of the graphics cursor.

`YPOS n1`

Returns the y position of the graphics cursor.

Vocabulary.

`VLIST` prints out all known words in the current vocabulary. (A vocabulary is a subsection of the whole dictionary. The normal vocabulary is called FORTH and is selected using the word `FORTH`.)

`FORGET <word>`

will forget all words defined from <word> onwards. A handy thing to do is to compile the word `TASK` as the first new word

```
: TASK ;
```

Then, if after compiling more words, you decide to get rid of them all,

```
FORGET TASK
```

will do the job. The system variable `FENCE` contains an address, below which you cannot `FORGET` a word. To protect the words you have just compiled type `HERE FENCE !`

To create a new vocabulary MYWORDS

type

```
VOCABULARY MYWORDS IMMEDIATE
```

The word MYWORDS will now cause this new vocabulary to be searched when interpreting words (the system variable CONTEXT is set to MYWORDS). New definitions will, however, be added to the old vocabulary (the system variable CURRENT is still pointing to FORTH). To select the new vocabulary as the one to add new definitions to, type:

```
MYWORDS DEFINITIONS
```

This sets the current vocabulary to the context vocabulary (made MYWORDS by MYWORDS). To go back to adding definitions to the FORTH vocabulary, type

```
FORTH DEFINITIONS
```

Some words, such as FORTH, are immediate. This means that they will be executed, even in compile mode.

The Ram Disc.

One of the problems with using FORTH on a non-disc system is that once a word has been defined, the original source for that definition is lost. For a large definition, this is an obvious nuisance if it is found that it does not subsequently do what was expected of it. In Abersoft FORTH (cassette version) this has been circumvented by using the top of store as a small, pretend disc of 11k bytes in total size. This may seem like a small amount, but given FORTH's compactness, a surprisingly large application can be written. As an example, the editor described later took up only 8 pages of this disc (a page is 1024 bytes long arranged as 16 lines of 64 characters) and that included copious comments.

The pages on the disc are numbered from 0 to 10, page 0 being reserved for comments, text is then input to the disc by means of the editor described later. To compile a program from RAM-disc, use

n LOAD (where n is the page number of the first definition)

if the definition or definitions, spread over more than one page, the final word on the page should be

--> (pronounced next-screen)

To prepare the RAM disc for writing, the command

```
INIT-DISC
```

should be used. This simply clears the area with blanks. To list the current contents of the disc pages, use

```
n LIST
```
(where n is the page to be listed)

When a disc has been filled, it can be saved to tape with the command

```
SAVET
```

To reload a disc area created previously, use

```
LOADT
```

but remember, this overwrites whatever is already there.

CPM

When using the CPM version, most of the above, about RAM-DISC applies, but whereas there are only 11k of disc in that version, the only restriction on screens in CPM is the size of the disc. Note that to be sure that PIP can copy files, you must make sure that page 0 is clear, use `Ø CLEAR FLUSH` when creating a new file, either by using a new name at load time, or by using `FILE` <filename>. The redundant parts of the above are, `SAVET, LOADT` and `INIT-DISC.`

The Outside World.

Two commands are available for communication through the standard I/O ports of the CPC464. These are:-

```
n1 INP n2
n1 n2 OUTP
```

`INP` returns a value to the stack from port `n1`.
`OUTP` puts the value `n1` out onto port `n2`.

Miscellaneous Additions.

The final changes to the standard are :-

`FREE`

this returns a value on the stack of the amount of store remaining in bytes. For example:-

```
: BYTES FREE . ." bytes remaining" CR ;
```

when `BYTES` is typed, the message `18919 bytes remaining` or similar will be printed.

`SIZE`

this returns a value on the stack of the current size of the dictionary.

`f LINK`

this command (where `f` is a flag) links the printer and the screen together so that anything output to the screen will also be printed on the printer.

On the CPM version there are a number of commands to manipulate BDOS built into the system, most of these are for internal FORTH use, the only user available ones are:-

`FILE <filename>`

Closes the current .FIG file and opens a new file filename.

`n1 n2 BDOS n3`

Sets C to `n2` and DE to `n1`, returns `n3`=A to the system.

Chapter 3. Advanced Features of Forth

Saving an extended dictionary.

As it can be seen from any work on FORTH, a completed FORTH application is simply an extension of the dictionary. In Abersoft FORTH, if you have a completed program and do not wish to compile from RAM-disc each time you want to use it, or you have a set of standard routines that you wish to use every time you enter FORTH, these may be saved by the following method:-

Tape version

1) Find out the new total length of your program by typing:-

```
SIZE .
```

2) Change the COLD START parameters by typing the following:-

```
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN !
HERE 28 +ORIGIN !
HERE 30 +ORIGIN !
HERE FENCE !
' FORTH 8 + 32 +ORIGIN !
```

(If using a different vocabulary than FORTH use ' vocab 6 + . . .)

3) Type:-

```
HERE ENVIR
```

to save your new extended version of FORTH to tape. (Use your own tape to do this on, NOT the master FORTH tape.) This new tape is of course for your own use only and not for re-sale, hire or lending purposes.

4) When you have a stand alone program, that will never need to be changed, or if you want to create a program for re-sale, you need to make sure that the application will never return to the command level, and then type:-

```
ZAP <word>
```

where <word> is the final word of the program, like the word `GAME` in the example of Chapter 2. This will then save an image of the system to tape. This new program can be run with `RUN "` (accessed by pressing **[CTRL]** and small **[ENTER]** keys simultaneously.)

CPM Version.

1) Find out the new total length of your program in save pages by typing:-

```
HERE 256 / .
```

2) Change the COLD START parameters by typing the following:-

```
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN !
HERE 28 +ORIGIN !
HERE 30 +ORIGIN !
HERE FENCE !
' FORTH 8 + 32 +ORIGIN !
```

(If using a different vocabulary than FORTH use ' vocab 6 + ...)

3) Type:-

```
MON
```

to leave forth and then

`SAVE n FORTHEX.COM` (where `n` is the number found in 1)

to save your new extended version of FORTH to disc. (Use your own disc to do this on, NOT the master FORTH disc.) This new disc is of course for your own use only and not for re-sale, hire or lending purposes.

4) When you have a stand alone program, that will never need to be changed, or if you want to create a program for re-sale, you need to make sure that the application will never return to the command level, and then type:-

`ZAP <word>`

where <word> is the final word of the program, like the word GAME in the example of Chapter 2. This will then save an image of the system to disc. This new program can be run with

<Filename>

the same as any CPM transient program.

Register Usage.

The programmer who wishes to use machine code routines (using `CREATE` or `;CODE`) or Assembler in 1.1B will require the register usage of Abersoft FORTH.

These are as follows:-

REGISTERS

FORTH	Z80	
IP	BC	Must be preserved across words.
W	DE	Input only when PUSHDE is used.
SP	SP	Data Stack.
UP	IX	User area pointer. Must be preserved across words.
	HL	Input only when PUSHHL used.

To understand this area fully will require Ting's book as mentioned earlier.

Chapter 4. Forth Assembler

Assembler.

This allows easy construction of either full words using:-

```
CODE. . .(assembly mnemonics). . .C;
```

or new defining words using:-

```
:. . . . .;CODE. . .(assembly mnemonics). . .C;
```

e.g.

```
CODE DOUBLE          ( take top word of stack and double it)
HL POP               ( pop top word)
HL DAD               ( ADD HL,HL)
HL PUSH              ( push word back on top of stack)
NEXT                 ( compile jump back to next)
C;
```

Subroutines that may be required in a FORTH low-level word may be defined thus:-

```
LABEL SUB1
HL DCX  (decrement HL)
RET
C;

CODE DECREMENT
HL POP
SUB1 CALL
PUSHHL  (alternate return to next which pushes HL first)
C;
```

As can be seen from the above example, FORTH Assemblers are structured in Reverse Polish as is the rest of the language. Below is the complete FORTH Asssembler, with standard Z80 mnemonics as reference:-

Where:-
r = A,B,C,D,E,H,L or (HL)
cc = condition (Z,NZ etc)
rp = register pair
ir = index register IY or IX

Use of the index registers is thus:-

n (IX) B LDX = LD B,(IX+n)

FORTH	Z80	FORTH	Z80	FORTH	Z80
CCF	CCF	DI	DI	XRA	XOR A
EI	EI	CPL	CPL	HALT	HALT
DAA	DAA	NOP	NOP	SCF	SCF
EXAF	EX AF,AF'	RET	RET	EXX	EXX
EXDEHL	EX DE,HL	LDADE	LD A,(DE)	LDABC	LD A,(BC)
STADE	LD (DE),A	STABC	LD (BC),A	IMn	IMn
NEG	NEG	LDAI	LD A,I	LDAR	LD A,R
RETI	RETI	RETN	RETN	LDRA	LD R,A
LDIA	LD I,A	r1 r2 MOV	LD r2,r1	n r MVI	LD r,n
n rp LXI	LD rp,nn	rp n SXR	LD (n),rp	n rp LXR	LD rp,(N)
r INR	INC r	n RST	RST n	rp POP	POP rp
rp PUSH	PUSH rp	rp INX	INC rp	r DCR	DEC r
rp DCX	DEC RP	rp DAD	ADD HL,rp	r ADD	ADD A,r
n ADI	ADD A,n	r ADC	ADC A,r	n ACI	ADC A,n
r SUB	SUB r	n SUI	SUB n	r SBC	SBC r
n SCI	SBC n	r AND	AND r	n ANI	AND n
r XOR	XOR r	n XRI	XOR n	r OR	OR r
n ORI	ON r	r CMP	CMP r	n CMPI	CMP n
n IN	IN A,(n)	n OUT	OUT A,(n)	DADX	ADD IX,IX
DADY	ADD IY,IY	rp ir DAD	ADD ir,rp	rp DADC	ADC HL,rp
rp DSBC	SBC HL,rp	n JMP	JMP n	n CALL	CALL n
n STA	LD (n),A	n LDA	LD A,(n)	EX(SP)	EX (SP),HL
ir | HL JPI	JP (HL) | (ir)	SPHL	LD SP,HL	r RLC	RLC r
r RL	RL r	r RRC	RRC r	r RR	RR r
r SLA	SLA r	r SRL	SRL r	r SRA	SRA r
RRCA	RRCA	RLA	RLA	RRA	RRA
r n BIT	BIT n,r	r n RES	RES n,r	r n SET	SET n,r
LDI	LDI	CPI	CPI	INI	INI
OUTI	OUTI	LDD	LDD	CPD	CPD
IND	IND	OUTD	OUTD	LDIR	LDIR
CPIR	CPIR	INIR	INIR	OTIR	OTIR
LDDR	LDDR	CPDR	CPDR	INDR	INDR
OTDR	OTDR	r IN(C)	IN r,(c)	r OUT(C)	OUT r,(C)
n cc CALLC	CALL cc,n	n cc RETC	RET cc,n	n cc JPC	JP cc,n
n cc JRC	JR cc,n	n JR	JR n	n DJNZ	DJNZ n

Jumps and loops can be used as in normal assembly language, but this can be difficult. FORTH Assembler allows similar looping structures to normal FORTH. These are:-

BEGIN. . .AGAIN (Infinite loop)

BEGIN. . .cc UNTIL

this is the equivalent of

```
L1: mnemonics

    JR cc,L1 (or if branch > 128 JP cc,L1)

BEGIN ... cc WHILE ... REPEAT

cc IF ... ENDIF

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

There is also a DO ... LOOP, which uses the DJNZ instruction.i.e.

```
5 DO A INR LOOP
```

would assemble

```
      LD B,5
L1:   INC A
      DJNZ L1
```

Because the register names A,B,C etc. are easily confused with the HEX numbers, and also that most people prefer to write assembler in HEX, remember to specify hex numbers with a leading Ø. i.e ØA, ØB etc.

The FORTH assembler does not have many checks on the way you write your code, and can easily allow illegal instructions to be used. Checks are made at the end of assembly that the stack is clear, but the onus is on YOU for most error checking.

To pass values back to the stack, and to return to the next FORTH word, the registers stated must be preserved. To return without any values, NEXT assembles a jump to the FORTH NEXT, i.e. use NEXT not NEXT JMP at the end of assembly. If you have one value, pass it in the HL register and use PUSHHL. If you have two values pass them in the HL and DE registers and use PUSHDE. This will leave the stack as:-

```
DE HL (tos).
```

Chapter 5. The Disc Editor

Arrangement of the RAM-disc.

FORTH organises all mass storage as screens of 1024 characters. The RAM-disc has 11k and its screens are numbered 0 to 11.

Each screen is organised by the system into 16 lines of 64 characters per line. The FORTH screens are an arrangement of virtual memory and do not correspond to the CPC464 screen format.

Inputting to a Screen.

To start an editing session, type `EDITOR`. This invokes the EDITOR vocabulary and allows text to be input.

The screen to be edited is selected, using either:-

`n LIST` (list screen n and select for editing) OR
`n CLEAR` (clear screen n and select for editing)

To input text after `LIST` or `CLEAR`, the `P` (put) command is used.

Example:

```
0 P This is how
1 P to input text
2 P onto lines 0, 1, and 2 of the selected screen.
```

Line Editing.

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

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

Line Editor Commands.

`n H` Hold line n in PAD (used by system more often than by user).
`n D` Delete line n but hold it in PAD. Line 15 becomes blank as lines n+1 to 15 move up 1 line.
`n T` Type line n and save it in PAD.
`n I` Insert the text from pad at line n, moving the old line n and following lines down. Line 15 is lost.

n E Erase line n with blanks.
n S Spread at line n. n and subsequent lines move down 1 line. Line n becomes blank. Line 15 is lost.

String Editing.

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

Commands to position the cursor.

TOP	Position the cursor at the start of the screen.
n M	Move the cursor by a signed amount n and print the cursor under score.
n LIST	List screen n and select it for editing.
n CLEAR	Clear screen n with blanks and select it for editing.
n1 n2 COPY	Copy screen n1 to screen n2.
L	List the current screen. The cursor line is relisted after the screen listing, to show the cursor position.
F <text>	Search forward from the current cursor position until string <text> is found. The cursor is left at the end of the text string, and the cursor line is printed. If the string is not found, an error message is given and the cursor is repositioned at the top of the screen.
B	Used after F to back up the cursor by the length of the most recent text.
N	Find the next occurence of the string found by an F command.
X <text>	Find and delete the string <text>.
C <text>	Copy in text to the cursor line at the cursor position.
TILL <text>	delete on the cursor line from the cursor till the end of the text string <text>.

NOTE: Typing C with no text will copy a null into the text at the cursor position. this will abruptly stop later compiling! To delete this error type TOP X **[ENTER]**.

The Screen Editor

Inputting to a Screen.

To start an editing session, type SCREDIT. This invokes the SCREEN EDITOR vocabulary and allows text to be input.

The screen to be edited is selected, using:-

n SCRED (list screen n and select for editing)

The editor starts in OVERWRITE mode. The cursor can be moved by using the arrow keys, and anything typed will be entered on the screen. **[ENTER]** just moves the cursor to the beginning of the next line. **[DEL]** deletes the previous character as would be expected. Care should be taken at the end of the screen lines, as FORTH treats the screen as one contiguous block, so if a word finishes in column 64, it will take column 1 of the next line as part of the previous line. The other mode is INSERT mode, toggling between the modes is accomplished with **[COPY]**. In this mode, all characters entered move characters along the screen line you are on, losing excess off the right-hand side. **[DEL]** closes up a line, and **[ENTER]** performs the same function as in OVERWRITE mode. Two other commands, **[CTRL] S** and **[CTRL] D**, either open up a gap in the screen at the current cursor position losing the 15th line, or delete the current line, filling the 15th line with spaces. To complete the edit, **[ESC]** is pressed, The option is then given to either complete the edit, or quit the edit leaving the screen as it was.

Chapter 6. Error Messages

If the compiler finds an error at any point, it clears both the data and return stack, and gives an error message with a numeric value.The meaning of these error messages is:-

MSG·	Meaning
0	Word not found.
1	Stack empty.
2	Dictionary full.
3	Has incorrect address mode.
4	Is not unique.
6	RAM Disc Range?(not pages 0 to 10)
7	Full Stack.
9	Trying to load from page 0.
17	Compilation only use in a definition.
18	Execution only.
19	Conditionals not paired.
20	Definition not finished.
21	In protected dictionary.
22	Use only when loading.
23	Off current editing screen.
24	Declare vocabulary.

Chapter 7. Forth Reference Guide.

The reference guide contains all of the word definitions in this release of Fig-FORTH (the extensions for the Amstrad CPC64 have been presented in earlier chapters) in the main vocabulary. The definitions are presented in the order of their ASCII sort.

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

The symbols include:

addr	memory address
b	8 bit byte(i.e. high 8 bits zero)
c	7 bit ASCII character.
d	32 bit signed double integer.
f	boolean flag. 0=false, non-zero=true.
ff	boolean false flag=0.
n	16 bit signed integer number.
u	16 bit unsigned integer number.
tf	boolean true flag = non-zero.

Unless otherwise noted, all references to numbers are for 16 bit signed integers. The high byte is on top of the tack, with the sign in the leftmost bit. For 32 bit numbers the most significant part is on top.

All arithmetic is implicitly 16 bit signed integer, with error and under-flow indication unspecified.

Acknowledgements are duly made to the FORTH INTEREST GROUP for parts of this compiler and manual,and it is recommended that membership of this august body is taken out. The address of Fig(UK) is:-

The Membership Secretary
FIG(UK)
24 Western Avenue
Woodley
Reading
RG5 3BH

! n addr ---

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

!CSP

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

d1 --- d2

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

#> d --- addr count

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

#BUF --- n

A constant returning the number of disc buffers allocated.

#S d1 --- d2

Generates ascii text in the text output buffer, by the use of #, until a zero double number results. Used between <# and #>.

' --- addr

Used in the form:

' nnnn

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

(

Used in the form:

(cccc)

Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required.

(."")

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

(;CODE)

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

(+LOOP) n ---

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

(ABORT)

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

(DO)

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

(FIND) addr1 addr2 --- pfa b tf (ok)
addr1 addr2 --- ff (bad)

Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left.

(LINE) n1 n2 --- addr count

Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 40 indicates the full line text length.

(LOOP)

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

(NUMBER) d1 addr1 --- d2 addr2

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

* n1 n2 --- prod

Leave the signed product of two signed numbers.

`*/` n1 n2 n3 --- n4

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

`*/MOD` n1 n2 n3 --- n4 n5

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

`+` n1 n2 --- sum

Leave the sum of n1+n2.

`+!` n addr ---

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

`+-` n1 n2 --- n3

Apply the sign of n2 to n1, which is left as n3.

`+BUF` addr1 --- addr2 f

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

`+LOOP` n1 --- (run)
addr n2 --- (compile)

Used in a colon-definition in the form:

```
DO ... n1 +LOOP
```

At run-time. +LOOP selectively controls branching back to the corresponding DO based on n1, the loop index and the loop limit. The signed increment n1 is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or greater than the limit (n1>0), or until the new index is equal to or less than the limit (n1<0). Upon exiting the loop, the parameters are discarded and execution continues ahead.

At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. n2 is used for compile time error checking.

`+ORIGIN` n --- addr

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

.CPU

Prints the message at signon of your computer.

, n ---

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

- n1 n2 --- diff

Leave the difference of n1-n2.

-->

Continue interpretation with the next disc screen. (pronounced next-screen).

-DUP n1 -- n1 (if zero)
n1 -- n1 n1 (if non-zero)

reproduce n1 only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

-FIND --- pfa b tf (found)
--- ff (not found)

Accepts the next text word (delimited by blanks) in the input stream to HERE. and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left.

-TRAILING addr n1 --- addr n2

Adjusts the character count n1 of a text string beginning address to suppress the output of trailing blanks.

. n ---

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

."

Used in the form:

."cccc"

Compiles an in-line string cccc (delimited by the trailing ") with an execution procedure to transmit the text to the selected output device. If executed outside a definition, ." will immediately print the text until the final ". See (.").

.LINE line scr ---

Print on the screen, a line of text from the RAM disc by its line and screen number. Trailing blanks are suppressed.

.R n1 n2 ---

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

/ n1 n2 --- quot

Leave the signed quotient of n1/n2.

/MOD n1 n2 --- rem quot

Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 --- n

These small numbers are used so often that it is attractive to define them by name in the dictionary as constants.

0< n --- f

Leave a true flag if the number is less than zero (negative),otherwise leave a false flag.

0= n --- f

Leave a true flag if the number is equal to zero, otherwise leave a false flag.

0BRANCH f ---

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

1+ n1 --- n2

Increment n1 by 1.

2+ n1 --- n2

Increment n1 by 2.

2! nlow nhigh addr ---

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

2@ addr --- nlow nhigh

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

2CONSTANT d ---

A defining word used in the form:

d 2CONSTANT cccc

to create word cccc, with its para-meter field containing d. When cccc is later executed, it will push the double value of d to the stack.

2DROP d ---

Drop the double number from the stack.

2DUP n2 n1 --- n2 n1 n2 n1

Duplicates the top two values on the stack. Equivalent to OVER OVER.

2OVER d1 d2 --- d1 d2 d1

Copy the first double value d1 to the top of the stack.

2SWAP d1 d2 --- d2 d1

Exchange the top two double numbers on the stack.

2VARIABLE

A defining word used in the form:

d VARIABLE cccc

When VARIABLE is executed, it creates the definition cccc with its parameter field field initialized to d. When cccc is later executed, the address of its parameter field (containing d) is left on the stack, so that a double fetch or store may access this location.

:

Used in the form called a colon definition:

: cccc ... ;

Creates a dictionary entry defining cccc as equivalent to the following sequence of Forth word definitions '...' until the next ; or ;CODE. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled.

;

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

;CODE

Used in the form:

: cccc .... ;CODE
assembly mnemonics

Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following mnemonics. If the ASSEMBLER is not loaded, code values may be compiled using , and C, .

When cccc later executes in the form:

cccc nnnn

the word nnnn will be created with its execution procedure given by the machine code following cccc. That is when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S

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

< n1 n2 --- f

Leave a true flag if n1 is less than n2; otherwise leave a false flag.

<#

Setup for pictured numeric output formatting using the words:

<# # #S SIGN #>

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

<BUILDS

Used within a colon-definition:

: cccc <BUILDS ...
DOES> ... ;

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

cccc nnnn

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

= n1 n2 --- f

Leave a true flag if n1=n2; otherwise leave a false flag.

> n1 n2 --- f

Leave a true flag if n1 is greater than n2; otherwise a false flag.

>R n ---

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

? addr ---

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

?COMP

Issue error message if not compiling.

?CSP

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

?ERROR f n ---

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

?EXEC

Issue an error message if not execuuting.

?LOADING

Issue an error message if not loading

?PAIRS n1 n2 ---

Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.

?STACK

Issue an error message if the stack is out of bounds.

?TERMINAL --- f

Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation.

@ addr --- n

Leave the 16 bit contents of address.

ABORT

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

ABS n --- u

Leave the absolute value of n as u.

AGAIN addr n --- (compiling)

Used in a colon-definition in the form:

BEGIN ... AGAIN

At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below).

At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking.

ALLOT n ---

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

AND n1 n2 --- n2

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

ASSEMBLER

The vocabulary that holds the FORTH assembler.

B/BUF --- n

This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK.

B/SCR --- n

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

BACK addr ---

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

BASE --- addr

A user variable containing the current number base used for input and output conversion.

BEGIN --- addr n (compiling)

Occurs in a colon-definition in form:

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

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

BL --- c

A constant that leaves the ascii values for "blank".

BLANKS addr count ---

Fill an area of memory begining at addr with blanks.

BLK --- addr

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

BLOCK n --- addr

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

BRANCH

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

BUFFER n --- addr

Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the disc The block is not read from the disc. The address left is the first cell within the buffer for data storage.

C! b addr ---

Store 8 bits at address.

C/L --- n

Constant leaving the number of characters per line; used by the editor.

C, b ---

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

C;

Terminate an assembler definition.

C@ addr --- b

Leave the 8 bit contents of memory address.

CASE --- n (compiling)

Occurs in a colon definition in the form:

```
CASE
n OF ..... ENDOF
.....
ENDCASE
```

At run-time, CASE marks the start of a sequence of OF...ENDOF statements.

At compile-time CASE leaves n for compiler error checking.

CFA pfa --- cfa

Convert the parameter field address of a definition to its code field address.

CLS

Performs the clear screen-home cursor function.

CMOVE from to count ---

Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceding toward high memory.

CODE

Creates a new word, and sets the current vocabulary to ASSEMBLER, ready for code definitions.

COLD

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

COMPILE

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

CONSTANT n ---

A defining word used in the form:

```
n CONSTANT cccc
```

to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.

CONTEXT --- addr

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

COUNT addr1 --- addr2 n

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

CR

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

CREATE

A defining word used in the form:

CREATE cccc

by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the words parameter field. The new word is created in the CURRENT vocabulary.

CSP ---- addr

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

D+ d1 d2 --- dsum

Leave the double number sum of two double numbers.

D+ -d1 n --- d2

Apply the sign of n to the double number d1, leaving it as d2.

D. d ---

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

D.R d n ---

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

DABS d --- ud

Leave the absolute value ud of a double number.

DECIMAL

Set the numeric conversion BASE for decimal input-output.

DEFINITIONS

Used in the form:

cccc DEFINITIONS

Set the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.

DIGIT c n1 --- n1 tf (ok)
c n1 --- ff (bad)

Converts the ASCII character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.

DLITERAL d --- d (executing)
d --- (compiling)

If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it to the stack. If executing, the number will remain on the stack.

DMINUS d1 --- d2

Convert d1 to its double number two's complement.

DO n1 n2 --- (execute)
addr n --- (compile)

Occurs in a colon-definition in form:

DO ... LOOP DO ... +LOOP At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial values n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop I will copy the current value of the index to the stack. See I, LOOP, +LOOP, LEAVE.

When compiling within the colon definition, DO compiles (DO), leaves the following address addr and n for later error checking.

DOES>

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

DP ---- addr

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

DPL ---- addr

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

DROP n ---

Drop the number from the stack.

DUP n --- n n

Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2 (compiling)

Occurs within a colon-definition in the form:

IF ... ELSE ... ENDIF

At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect.

At compile-time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.

EMIT c ---

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

EMPTY-BUFFERS

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

ENCLOSE addr1 c -- addr1 n1 n2 n3

The text scanning primitive used by WORD. From the text address addr1 and an ASCII delimiting character c, is determined the byte offset to the first non-delimiter character n1, the offset to the first delimiter after the text n2, and the offset to the first character not included. This procedure will not process past an ASCII 'null', treating it as an unconditional delimiter.

END

This is an 'alias' or duplicate definition for UNTIL.

ENDCASE addr n --- (compile)

Occurs in a colon definition in a form:

```
CASE n OF ..... ENDOF ..... ENDCASE
```

At run-time ENDCASE marks the conclusion of a CASE statement.

At compile-time, ENDCASE computes forward branch offsets.

ENDIF addr n --- (compile)

Occurs in a colon-definition in form:

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

At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. Both names are supported in FIG-FORTH. See also IF and ELSE.

At compile-time, ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests.

ENDOF addr n --- (compile)

Used as ENDIF but in CASE statements.

ERASE addr n ---

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

ERROR line --- in blk

Execute error notification and restart of systems. WARNING is first examined. If WARNING=positive, n is just printed as a message number (RAM disc installation). If WARNING is -1, the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). FIG-FORTH saves the contents of IN and BLK to assist in determining the location of the error. Final action is execution of QUIT.

EXECUTE addr --

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

EXPECT addr count ---

Transfer characters from the terminal to address, until a **[ENTER]** is typed or the count of characters have been received. One or more nulls are added at the end of the text.

FENCE --- addr

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

FILL addr quan b ---

Fill memory at the address with the specified quantity of bytes b.

FIRST --- n

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

FLD --- addr

A user variable for control of number output field width. Presently unused in FIG-FORTH.

FLUSH

Write all updated disc buffers to RAM disc.

FORGET

Executed in the form:

FORGET cccc

Deletes definition named cccc from the dictionary with all entries physically following it. In FIG-FORTH, an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same.

FORTH

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

HERE --- addr

Leave the address of the next available dictionary location.

HEX

Set the numeric conversion base to sixteen (hexadecimal).

HLD --- addr

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

HOLD c ---

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

I --- n

Used within a DO .... LOOP to copy the loop index to the stack. See R.

I/ --- n

Copies the last but one value from the return stack.

ID. addr ---

Print a definition's name from its name field address.

IF f --- (run-time)
--- addr n (compile)

occurs is a colon-definition in form:

IF (tp) ... ENDIF
IF (tp) ... ELSE (fp) ... ENDIF

At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f is just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional. If missing, false execution skips to just after ENDIF.

At compile-time IF compiles OBRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.

IMMEDIATE

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

IN --- addr

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

INDEX from to ---

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

INIT-DISC

Wipes all information off the RAM disc prior to first use.

INTERPRET

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

J --- n

Used within nested DO loops. Returns the index value of the outer loop.

KEY --- v

Leave the ASCII value of the next terminal key struck.

LATEST --- addr

Leave the name field address of the topmost word in the CURRENT vocabulary.

LEAVE

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

LFA pfa --- lfa

Convert the parameter field address of a dictionary definition to its link field address.

LIMIT ---- n

A constant leaving the address just above the highest memory available for a disc buffer. Usually this is the highest system memory.

LINE n --- addr

Leave address of line n of current screen. This address will be in the disc buffer area.

LIST n ---

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

LIT --- n

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

LITERAL n --- (compiling)

If compiling, the compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon definition. The intended use is: : xxx [calculate] LITERAL ; Compilation is suspended for the compile time calculation of a value. Compilation is resumed and LITERAL compiles this value.

LOAD n ---

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

LOOP addr n --- (compiling)

Occurs in a colon-definition in form:

DO ... LOOP

At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.

At compile-time, LOOP compiles (LOOP) and used addr to calculate an offset to DO. n is used for error testing.

M* n1 n2 --- d

A mixed magnitude math operation which leaves the double number signed product of two signed number.

M/ d n1 --- n2 n3

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

M/MOD ud1 u2 --- u3 ud4

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

MAX n1 n2 --- max

Leave the greater of two numbers.

MESSAGE n ---

Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headerts. If WARNING is zero, the message will simply be printed as a number (RAM disc system).

MIN n1 n2 --- min

Leave the smaller of two numbers.

MINUS n1 --- n2

Leave the two's complement of a number.

MOD n1 n2 --- mod

Leave the remainder of n1 / n2, with the same sign as n1.

MON

Exit to BASIC or operating system.

NEXT

This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directlty executed but is the return point for all code procedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by the address pointed to by W. W points to the code field of a definition which contains the address of the code which excecutes for that definition. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific. (See earlier note on FORTH convention).

NFA pfa --- nfa

Convert the parameter field address of a definition to its name field.

NOOP

A FORTH no-operation.

NOT f --- f

Leaves a false flag if a true flag is on the stack, else leaves a true flag. (Actually executes 0=)

NUMBER addr --- d

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

OFFSET --- addr

A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK. Messages by MESSAGE are independent of OFFSET. See BLOCK, MESSAGE. This is REDUNDANT in FORTH 1.1b

OR n1 n2 -- or

Leave the bit-value logical or of two 16 bit values.

OUT --- addr

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

OVER n1 n2 --- n1 n2 n1

Copy the second stack value, placing it as the new top.

PAD --- addr

Leaves the address of the text output buffer, wich is a fixed offset above HERE.

PFA nfa --- pfa

Convert the name field address of a compiled definition to its parameter field address.

PREV ---- addr

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

QUERY

Input 80 characters of text (or until a **[ENTER]**) from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.

QUIT

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

R --- n

Copy the top of the return stack to the computation stack.

R# --- addr

A user variable which may contain the location of an editing cursor, or other file related function.

R/W addr blk f ---

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

R> --- n

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

RØ --- addr

A user variable containing the initial location of the return stack. Pronounced 'R-'zero. See RP!

REPEAT addr n --- (compiling)

Used within a colon-definition in the form:

BEGIN ... WHILE ... REPEAT

At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN.

At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.

ROT n1 n2 n3 --- n2 n3 n1

Rotate the top three values on the stack, bringing the third to the top.

RP@ --- addr

Leaves the current value in the return stack pointer register.

RP!

A computer dependent procedure to initialize the return stack pointer from user variable RØ.

S->D n -- d

Sign extend a single number to form a double number.

SO --- addr

A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SP!

SCR --- addr

A user variable containing the screen number most recently reference by LIST.

SIGN n d --- d

Stores an ASCII - sign just before a converted numeric output string in the next output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>.

SMUDGE

Used during word definition to toggle the 'smudge bit' in a definitions' name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error.

SP!

A computer dependent procedure to initialize the stack pointer from SØ.

SP@ --- addr

A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ . . . would type 2 2 1)

SPACE

Transmit an ASCII blank to the output device.

SPACES n ---

Transmit n ASCII blanks to the output device.

STATE --- addr

A user variable containing the compilation state. A non-zero value indicates compilation.

SWAP n1 n2 --- n2 n1

Exchange the top two values on the stack.

TASK

A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.

TEXT c ---

Accept following text to PAD. c is the text delimeter.

THEN

An alias for ENDIF.

TIB --- addr

A user variable containing the address of the terminal input buffer.

TOGGLE addr b ---

Complement the contents of addr by the bit pattern b.

TRAVERSE addr1 n --- addr2

Move across the name field of a fig-FORTH variable length name field. add1 is the address of either the length byte or the last letter. If n=1, the motion is toward high memory; if n=-1, the motion is toward low memory. The addr resulting is address of the other end of the name.

TRIAD scr ---

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

TYPE addr count ---

Transmit count characters from addr to the selected output device.

U< u1 u2 --- f

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

U* u1 u2 --- ud

Leave the unsigned double number product of two unsigned numbers.

U. u ---

Prints an unsigned 16-bit number converted according to BASE. A trailing blank follows.

U.R u n ---

Print the unsigned number u right aligned in a field whose width is n. No following blank is printed.

U/MOD ud u1 --- u2 u3

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

UNTIL f --- (run-time)
addr n --- (compile)

Occurs within a colon-definition in the form:

BEGIN ... UNTIL

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

At compile-time, UNTIL compiles ØBRANCH and an offset from HERE to addr. n is used for error tests.

UPDATE

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

USE --- addr

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

USER n ---

A defining word used in the form:

n USER cccc

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

VARIABLE

A defining word used in the form:

n VARIABLE cccc

When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location.

VLIST

List the names of the definitions in the context vocabulary. **[ESC]** will terminate the listing.

VOC-LINK --- addr

A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control by FORGETting thru multiple vocabularys.

VOCABULARY

A defining word used in the form:

VOCABULARY cccc

to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence cccc DEFINITIONS will also make cccc the CURRENT vocabulary into which new definitions are placed.

In FIG-FORTH, cccc will be so chained so to include all definitions of the vocabulary in which cccc is itself defined. All vocabulary ultimately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.

WARNING --- addr

A user variable containing a value controlling messages. If = 1 disc is present, and screen 4 of drive Ø is the base location for messages. If = Ø, no disc is present and messages will be presented by number. If = −1, execute (ABORT) for user defined procedure. See MESSAGE, ERROR.

WHILE f --- (run-time)
ad1 n1 --- ad1 n1 ad2 n2

Occurs in a colon-definition in the form:

BEGIN ... WHILE (tp) ... REPEAT

At run-time, WHILE selects conditional execution based on boolean flag f. If f is true (non-zero), WHILE continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution skips to just after REPEAT, exiting the structure.

At compile time, WHILE emplaces (ØBRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr

In fig-FORTH, a user variable containing the maximum number of letters saved in the complication of a definitions' name. It must be 1 thru 31, with a default value of 31. the name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits.

WORD c ---

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

X

This is psuedonym for the 'null' or dictionary entry for a name of one character of ASCII null. It is the execution procedure to terminate interpretation of a line of text from the terminal or within a disc buffer, as both buffers always have a null at the end.

XOR n1 n2 --- xor

Leave the bitwise logical exclusive- or of two values.

[

Used in a colon-definition in form:

```
: xxx [ words ] more ;
```

Suspend compilations. The words after [are executed, not compiled. This allows calculations or compilation exceptions before resuming compilation with]. See LITERAL,].

[COMPILE]

Used in a colon-definition in form:

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

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

]

Resume compilation, to the completion of a colon-definition. See [.