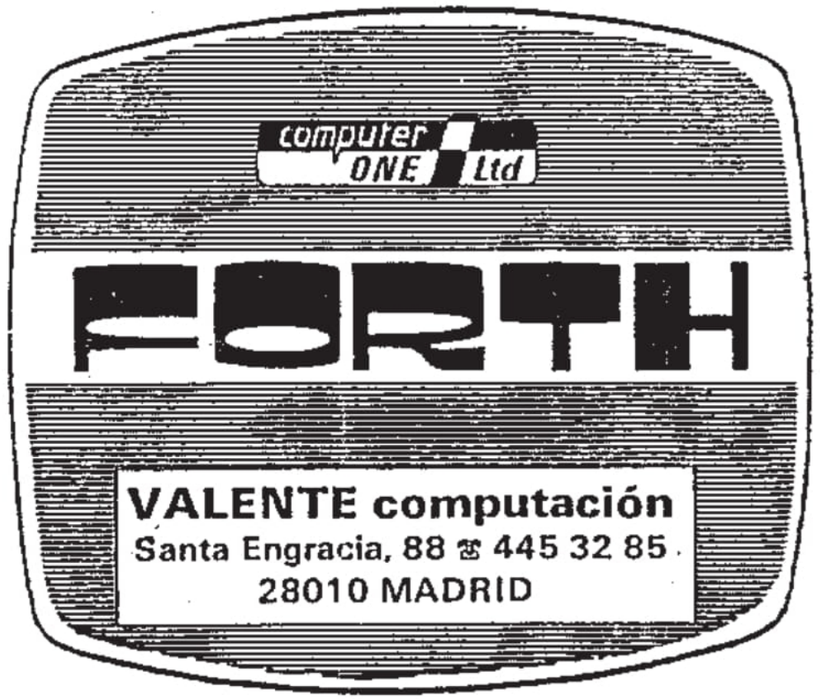


NUEVA DIRECCION
Caleruega, 8
28033 - MADRID
Teléfono 202 67 01

User Guide

For the Sinclair QL Computer

1.3: BACKUP

FORTH for the QL is supplied on a microdrive cartridge. This cartridge contains several files. Before you do anything else back up the supplied cartridge by using the CLONE program. It is recommended that you make two backups. The Master is then used as an emergency backup, and not to run the software.

Copies of the Master may be easily made using the 'clone' program (supplied on each Computer One software cartridge) as follows:

1. Place the Master copy in microdrive 2 (the right hand side drive).

2. Place a blank cartridge in microdrive 1.

3. Enter the following command:

```
LRUN mdv2_clone <ENTER>
```

4. The QL will respond with various instructions to name the new microdrive, and initiate the copying. Check you have inserted the cartridges in the correct drives, and invoke the copying.

5. The 'cloned' system may be used as soon as the microdrives have stopped running.

Repeat the procedure with another cartridge, and store the master and one of the copies in a safe place. Use the remaining copy as your working master - only use the others in dire emergency.

1.4: STARTUP

To startup the Computer One FORTH system, it is important to carry out the following :-

(1) Remove any cartridges from the two microdrives. It is inadvisable to reset the system with the cartridges in the drives, as they may be damaged.

(2) Press the reset button on the right hand side of the machine.

(3) Place the cloned FORTH cartridge in drive one - the left hand drive.

(4) Press F1 or F2 to select Monitor or TV mode.

When you have selected the appropriate mode, the FORTH system will automatically be loaded from the cartridge. The FORTH system image will be loaded and made resident in memory, and then started up ready for your input. 'Computer One FORTH' will appear on the screen, together with the word 'READY' to indicate all is well. The FORTH system image takes up approximately 20k, and

operates within 64k of memory leaving you about 44k as workspace.

NB: During this process you will see that drive 2 is accessed. FORTH is looking for a file on drive 2 called FORTH_SCR. This is the file that you use to contain screens of source code. If you insert a microdrive into drive 2 with your working copies of FORTH_SCR, then these will be used. Otherwise, by default, FORTH will use FORTH_SCR on microdrive one. The FORTH word 'BLOCK' will refer to blocks (screens) on this file. You can change the default screen file by typing USING <filename>.

1.5: CONTENTS OF THE SUPPLIED CARTRIDGE

The supplied cartridge will contain the following files:-

BOOT	initial loader
CLONE	program to back up the FORTH cartridge
IMAGE	FORTH itself
FORTH_SCR	useful goodies in source form
ELECTIVE_SCR	optional extensions to FORTH
EDITOR_BIN	full screen editor as a binary overlay
ASM68_BIN	68000 macro-assembler as a binary overlay
SCOPY_BIN	File copy utility as a binary overlay

1.6: GETTING STARTED WITH FORTH -- An example session

The only sure way of learning FORTH is to use it. This is because FORTH programmers tend to spend a greater proportion of their time at a terminal. This in turn is the result of being able to test each new definition as soon as it has been entered. The FORTH word : is used to start a new definition, and the word ; is used to finish it. The name of a word can contain any character you want - preferably printable ones.

The first thing you can do after you've loaded FORTH on the QL for the first time is to type WORDS <ENTER> this will display a list of the predefined words in FORTH. Note that if you have been using FIG-FORTH in the past, WORDS replaces VLIST.

Suppose we had a program in which we wanted to say hello to people. First we could define a word that says hello. We use a dot as the first character of the name because it is a FORTH convention that words that print things start with a dot.

```
: .HELLO    ." Hello "  ;
```

The word ." will print out every thing from then on up to the next " so that when you type .HELLO followed by a carriage return, FORTH will print out Hello followed by ok, which is FORTH's way of acknowledging that there were no errors in what it was last asked to do.

We could now define words to print people's names.

```
: .FRED     ." Fred "  ;

: .MARY     ." Mary "  ;

: .NEIL     ." Neil "  ;

: .LINDA    ." Linda " ;
```

We also need a word to link the names together.

```
: .AND      ." and "   ;
```

Now we can make a word to greet all these people. The word CR is used to print a carriage return followed by a line feed.

```
: .GREET    .HELLO .FRED .AND .MARY .AND .NEIL .AND .LINDA CR  ;
```

If we type GREET <ENTER> FORTH will respond:-

```
Hello Fred and Mary and Neil and Linda
ok
```

The secret of writing programs in FORTH is to keep everything simple. Simple things work, and all problems can be split into a sequence of very simple things. The programmer's job is to decide what those simple things should be and then write simple words to do them. If the names of the words you use reflect what the word has to do, then your code will be readable and easy to follow.

1.7: HOW FORTH IS DOCUMENTED

Abbreviations and nomenclature

FORTH words are described in the conventional FORTH style of showing what is required on the data stack before the word executes, and how the stack is left afterwards. The top of the stack is on the right and the execution point is denoted by two dashes --. e.g. *, the multiplication operator:-

```
*                n1 n2 -- n3
```

In the documentation all FORTH words are in upper case, but Computer One FORTH will accept FORTH words in upper or lower case. The following abbreviations are used for operands:

OPERAND	DESCRIPTION
n1,n2,...	16 bit signed numbers
d1,d2,...	32 bit signed numbers
u1,u2,...	16 bit unsigned numbers
ud1,ud2,.	32 bit unsigned numbers
addr1,addr2,...	16 bit addresses

b1,b2,...	8 bit byte right justified in 16 bit word
c1,c2,...	7 bit ASCII character right justified in 16 bit word
t/f,f1,f2,...	16 bit boolean flag zero for false non-zero for true
s1,s2,...	Character string
x1,x2,...	16 bit, 32 bit, or 64 bit number

1.8: WHERE TO FIND A WORD

If you find a word and want to know what it does, look it up in a glossary. A glossary is a FORTH term for an alphabetically sorted list of FORTH words with descriptions of what they do. We cannot call this a dictionary, because that is a term that refers to FORTH code itself. This manual contains eight glossaries.

There is a glossary for the FORTH nucleus itself. This includes the standard words which are included in the 'Required Word Set Glossary. The other glossaries describe general purpose extensions we have made to the nucleus.

The assembler has its own vocabulary called ASSEMBLER, and a glossary of its own in the assembler section of the manual. Only those words which are accessible are documented. Many other words used within the assembler are headerless and hence cannot be reached by the user. The Assembler words are described in Chapter 4 and the Assember Word Set Glossary.

Most of the words dealing with QDOS input and output are documented in Chapter 6. All the input and output words are accessible as we have tried to give the user full access to the operating system in a painless manner.

The floating point, graphics and sound words are all described in Glossary 8.

You may well find some words in the dictionary that are not documented at all. These have been left out deliberately and are words which are only used in passing as part of other words. They may not exist in later versions of COMPUTER ONE FORTH, and thus their existance should not be relied upon.

If you use the decompiler on some words you will find that they cannot be decompiled properly. This is because there are internal words which have been generated without dictionary headers, and so no name field exists for them.

1.9: ERROR MESSAGES

Error messages are held in screens 4 and 5 of FORTH_SCR, and may not be displayed if these screens are not open. (See 'Using').
A full list of error messages is provided in Appendix 1.

A Quick introduction to FORTH

2.1: WRITING PROGRAMS IN FORTH

FORTH is a different sort of computer language. It is interactive, in that you can enter commands from the keyboard, compiled in that the names of pre-defined functions that you type in are converted to addresses, and interpreted in that these addresses point to other pointers which eventually point to executable code which is then executed. FORTH is easy to use because it is interactive, fast because it is compiled, and not as fast as it might be because it is interpreted. FORTH is a language with a definite style.

In general FORTH takes a little longer to learn than other languages. One reason for this is that there are a lot of words to learn in FORTH before you can use it well - just like a language. In fact the predefined functions in FORTH are called words. These words are stored as a dictionary, and the group of words forming your area of interest - the context in which you work - is called a vocabulary. For example the words used to define the assembler are kept in a vocabulary called ASSEMBLER. There is an element of jargon in FORTH, as in all computer languages. Do persevere, FORTH is well worth the effort.

The FORTH run-time package is actually a remarkably compact interpreter, compiler, and memory management system. Any command or sequence of commands may be executed directly from the keyboard, or from the disc storage area. Programs in FORTH are compiled from combinations of existing commands (represented by words in the vocabulary), new commands as defined by the user, and control structures such as IF...ELSE...ENDIF or DO...LOOP. Usually, new commands are developed interactively, and at the terminal; the final version is then entered using the editor and saved on mircrodrive, where it can then be invoked from the keyboard, or used by another program.

The beauty and power of FORTH lies in its extensibility and flexibility. New vocabulary words, functions, and even data

types, can be added to the language at will either as high-level or assembler words. Programs are built up in the same way as people organise their thinking - by successively creating new functions in terms of old ones, progressively forming hierarchies of new levels of abstraction.

If your experience of programming has been restricted to such sequentially oriented languages as BASIC, you will initially find reading and writing programs in FORTH somewhat bizarre. Patience will bring rich rewards. FORTH can be practically self-documenting with good choice of word names; the language lends itself well to bottom-up design and coding; names of functions can be freely chosen to describe what they do, and you can use any character within a word name; embedded comments may be as long as you wish without a space or speed penalty in the compiled code. However, most FORTH programs keep most of their working variables on the stack, rather than as named variables, so that reading some sections of code can be a little mind-boggling even for the experienced. The secret is to keep definitions short and simple. Lazy programmers are good programmers because they make life easy for themselves - and part of making life easy is making sure that you can work out what the code is doing a year from now.

You may well find it profitable to study the source code of the demonstration programs supplied with COMPUTER ONE FORTH as a guide to style. Read the glossary documentation, and spend a while trying out the functions, and observing their action on the stack.

2.2: STACKS AND POSTFIX NOTATION

FORTH contains two stacks, one for storing return addresses (what was I doing last/where do I go back to?), and one for storing data. The first stack is called the return stack, and the second is called the data or parameter stack. The data stack is an efficient method of passing data between the words that make up a FORTH program. Any word that needs data takes it from the top of the stack, and puts any results back on top of the stack. Nearly all modern processors provide for the use of stacks, so these operations are very fast.

Because stacks are used for data handling, the use of post-fix, or reverse Polish notation, is very suitable. In this form of writing arithmetic expressions, operands (the data used) come before the operators (how you use the data), e.g

The normal (in-fix) notation expression :- `at**2 + bt + c`

is better expressed as :- `(at + b)t + c`

which is then expressed in post-fix as :- `a t * b + t * c +`

Notice that the use of brackets becomes unnecessary. This is because of the use of the stack to hold intermediate results.

Although the use of a stack is intimidating at first, after a while it becomes natural, and eventually you do not notice it except on rare occasions.

FORTH words are defined in this manual in terms of what they do with and to the stack. We use a convention quite popular with FORTH programmers. The top of the stack is to the right, and the point at which the word executes is marked by two dashes, --. So to define the word +, which adds two numbers from the stack, and leaves a third on the stack, we would comment it thus:-

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

In FORTH brackets are used to mark the start and end of comments.

2.3: INTERPRETING AND COMPILING

FORTH contains both an interpreter and a compiler.

Interpreting means taking the text fed in, converting it into a form the machine can execute, executing that form, and then discarding the executable form. Many forms of BASIC do just this, others convert the pre-defined keywords into tokens first, and then interpret the tokens. This is a very slow procedure, that can only be improved by using a very large BASIC.

Compiling means taking the input text, completely converting it into a machine executable form, keeping the executable form, and discarding the text. This can produce a program that runs very fast, but you cannot change anything without first editing the source text, then compiling it (using a separate program called a compiler), and then loading the executable code when you want to run it.

Any text fed to FORTH, either from the keyboard or from mass storage, is first compiled to a list of addresses, and this list of addresses is then interpreted if required. One reason for FORTH's speed is that this interpreter, called the inner interpreter, is actually very short, only two or three machine instructions on some processors. The slow and laborious job of compiling is performed as the text is entered - after you typed carriage return. The time taken to do this compilation is very short as far as you are concerned, but it allows subsequent execution of the code to be very fast.

Remember, all commands to FORTH are pre-defined 'words' in its 'vocabulary', consequently FORTH can look up the address of a given word for later execution. Some words in FORTH change the way the compiler section deals with text. For instance we could define a word that doubles the value given to it.

```
(n1 -- n1*2) : 2*  .        2 * ;
```


The address of the word : is found and : is executed; the action of : is tell the compiler section to define a new word whose name comes next (2*), and then compile into the new word the addresses of the words that follow. This would carry on for ever unless we had a way of stopping it, and this is provided by 'immediate' words such as ; which are always executed, regardless of what the compiler would otherwise be doing. The action of ; is to stop the compiler compiling word addresses, and return it to the mode of executing the addresses instead. There are other words (defining words) which are used to create words such as : - these are one of the keys to advanced use of FORTH. At all times remember, however, that the basis of FORTH is always very simple. FORTH is a language built from a number of very simple ideas, rather than one founded on a few complex systems.

2.4: DEFINING WORDS

Some words such as : in the previous section, are called defining words, because they are used to define new words (2* in the previous section); these words are one of the keys to the power of FORTH. The word : creates a new word in the dictionary, marks it as a high-level word (one written in FORTH rather than assembler) and switches FORTH from being an interpreter to being a compiler. Any word names met from now on will not be executed, but their execution addresses will be found and compiled into the dictionary. This process repeats until stopped, but it can only be stopped by the execution of a word, and all words are being compiled, not executed. This problem is dealt with by immediate words.

2.5: IMMEDIATE WORDS

The solution to the problem of the previous section is to have a class of words which are always executed, regardless of whether FORTH is supposed to be compiling. Such words are called immediate words. The word ; used to terminate a high level definition, is an example of such a word. When it executes, it firstly switches FORTH from being a compiler back to being an interpreter. Secondly, it compiles the address of the word ;S which performs the function of returning from a high level word.

2.6: CONTROL STRUCTURES

Control structures in FORTH allow conditional execution and looping based on the value on the stack. They are usually implemented by means of words that execute at compile time (immediate words). These words check for balanced structures (e.g that IF is followed by ENDIF) and compile the conditional branching structure itself. Additional error checking is also performed.

Control structures must be used inside a colon definition; they cannot be directly executed from the keyboard. Any one structure must be written entirely within one definition; you cannot put the IF in one word and the ENDIF in another.

IF .. ENDIF
Template:- flag IF words ENDIF

At IF the flag on the top of stack (TOS) is examined. If the flag is true (non-zero), the words between IF and ENDIF are executed, otherwise they are not.

If you wish to use the tested stack value inside the control structure you must duplicate it before the test. This can be done using DUP, or more conveniently in this case using -DUP, which only duplicates a number if it is non-zero.

IF...ENDIF structures may be nested, that is, one may contain another. BUT, nested structures must fit inside each other and may not overlap.

Example:-

```
: TEST   IF ." top of stack is non-zero" ENDIF  ;

1 TEST          top of stack is non-zero ok
0 TEST          ok
```

IF...ELSE...ENDIF
Template:- flag IF true words ELSE false words ENDIF

This structure behaves just like IF...ENDIF above except that an alternate set of words will execute when the flag is false (zero).

Example:-

```
: TEST   IF ." top of stack is non-zero"
         ELSE ." top of stack is zero"
         ENDIF  ;
```

DO...LOOP and DO...+LOOP
Template:- limit index DO words LOOP

limit index DO words increment +LOOP

This structure is very roughly the same as BASIC's FOR X=1 TO 10....NEXT. It allows looping or the repetitive execution of a set of words. The limits of the loop are defined by parameters on the stack at execution time.

The start and finishing indices must be on the stack before DO is reached. These are then transferred to the return stack by DO, and the top of stack represents the current loop index, and the

next on stack represents the limiting value. Execution carries on as far as LOOP or +LOOP, when the index is incremented by 1 in the case of LOOP, or by the value of the top of stack for +LOOP. If the new index is still less than the limit, execution resumes just after DO and the cycle repeats. If the index is greater than or equal to the limit, execution resumes after LOOP. You can force the program to leave the loop at the next test by using the word LEAVE, which sets the index to the limit so that when LOOP or +LOOP is next executed, the loop does not repeat. The current index may be inspected by the word I, which returns the index.

DO...LOOP structures may be nested to any level up to the capacity of the return stack. The index of the next outer loop may be inspected with the word J.

Warnings:-

** If you use the return stack for temporary storage within DO...LOOP, I and J will return incorrect values.

** Any data put on the return stack after DO must be removed before LOOP.

** Regardless of the value of the initial limit and index, the loop will always execute at least once.

** Because the test is performed after the index has been incremented, the limit value of the index is never used.

Example:-

```
: TEST   10 1  DO  I .  LOOP  ;

TEST     1 2 3 4 5 6 7 8 9  ok

: TEST2  10 1  DO  I .  3 +LOOP  ;

TEST2    1 4 7  ok
```

BEGIN...AGAIN
Template:- BEGIN words AGAIN

This structure forms a loop that never terminates unless an error condition occurs, or a word such as ABORT or QUIT is executed. The first example will read and echo characters from the keyboard forever, the second will exit when the carriage return key is pressed.

Example:

```
: TEST   BEGIN  KEY EMIT  AGAIN  ;

: TEST2  BEGIN  KEY DUP 13 = IF ABORT ENDIF  EMIT  AGAIN  ;
```

BEGIN...UNTIL
Template:- BEGIN words flag UNTIL

This structure forms a loop which is always executed at least once, and exits when the the word UNTIL is executed and the flag (on the data stack) is true (non-zero). If you need to use the terminating condition after the loop has finished use -DUP to duplicate the top item of the stack if it is non-zero. BEGIN...UNTIL loops may be nested to any level.

Example:-

```
: TEST   BEGIN  KEY DUP EMIT  13 = UNTIL  ;
```

BEGIN...WHILE...REPEAT
Template:- BEGIN words flag WHILE more words REPEAT

This is the most powerful and perhaps the most elegant (though certainly not some purists choice) of the FORTH control structures. The loop starts at BEGIN and all the words are executed as far as WHILE. If the flag on the data stack is non-zero the words between WHILE and REPEAT are executed, and the cycle repeats again with the words after BEGIN. This structure allows for extremely flexible loops, and perhaps because it is somewhat different to the structures of BASIC or PASCAL, this structure is often somewhat neglected. It does however, repay examination. In the example below, the console is polled until a key is pressed, and a counter is incremented while waiting.

Example:-

```
: TEST   BEGIN  ?TERMINAL 0=  WHILE  1 COUNTER +!  REPEAT  ;
```

CASE...OF...ENDOF...OF...ENDOF.....ENDCASE

```
Template:-      parameter CASE
                value1 OF words ENDOF
                value2 OF words ENDOF
                ........
                default words (otherwise clause)
                ENDCASE
```

CASE statements exist to replace a large chain of nested IFs, ELSEs, and ENDIFs. Such chains are unwieldy to write, prone to error, and lead to severe brain-strain. The intention of a CASE statement is to perform one action dependent on the value of the parameter passed into the CASE statement. If none of the conditions is met, a default action (the otherwise clause) should be available. Note that a select value must be available before each OF against which the entered parameter may be tested. The select value is top of stack, the parameter is next on stack (by requirement); OF then compares the two values, and if they are

equal, the words between OF and ENDOF are executed, and the program continues immediately after ENDCASE. If the test fails, the code between OF and ENDOF is skipped, so that the select value before the next OF may be tested. If all the tests fail the parameter is still on the data stack for the default action, and is then consumed by ENDCASE.

2.7: USING THE GRAPHICS EXTENSIONS

The QL graphics features have been added in Computer One FORTH. This section gives a commented example program on using the graphics words. Each word is described in the floating point and graphics glossary.

Computer One FORTH provides three channels which may be in use at any one time; these are called INPUT, OUTPUT and WORK. The input channel is the channel from which all input is taken; the output channel is the channel to which all text is output (including 'OK' prompt). The WORK channel is the channel to which all the graphic and windowing words apply. Eg: ELLIPSE, CLS, WINDOW etc.

Initially all three FORTH channels output to the same QDOS channel - channel 0. There are six FORTH words associated with the 3 channels - WORK, OUTPUT, INPUT, IS-WORK, IS-OUTPUT, IS-INPUT. The first 3 words return the current channel id for each of the channels. The other words take a channel id and set the work, output or input channels to that channel.

If we type:

```
WORK .
OUTPUT.
INPUT .
```

When the system has just been booted, all these operations will return 0, since this is the initial channel id for all three channels.

To define new files, for example a new window on the screen, it is necessary to give a name to a file control block (fcb), associate that name with a filename and finally to open the file.

Example:

```
fcb new_window  ( name of file control block )

new_window filename scr_512x256a0x0
                ( filename word associates the window dimensions
                  with the new window

new_window open-file     ( open the window )
```

To draw in the new window, we have to switch the work channel to the new channel. To find the channel id of a file, we use the word 'CHANNEL' which takes an fcb description, and returns the channel id.

```
Eg: new_window channel d.     (prints 32-bit channel id in decimal)
```

If we wish to draw a circle in the new window, we first have to make it the 'work' channel, then clear it and finally draw the circle in it.

```
Eg:  new_window channel is-work  (sets new window to be work channel)

     cls       ( cls operates on work channel  )
     f# 50 f# 30 f# 40 circle  ( draw circle in work window  )
```

Note 1: The word 'f#' is used to generate floating point values, which are used by many of the graphics commands (more in Glossary 8).
Note 2: To write text to this window we would have to switch the output channel to this window.

```
Eg:  new_window channel is-output
```

We now give an example program which resizes the default output channel, creates a new window, and draws a circle in it. Resizing the output channel, to, for example, the bottom third of the screen is a good idea if we are using the graphics interactively, since this prevents the whole screen scrolling whenever the prompt is output.

```
Variable param-ptr    ( Set up word whose address is put on stack
                        each time it is referred to              )

here param-ptr !      ( stores value 'here' at param ptr         )

8  Allot              ( allocate 8 bytes                         )

: par dup rot swap ! 2 + ;

: param-fill par par par ! ;

( These two definitions fill a 4-word parameter block for use in
resizing a window given the four values and address of the block ).

200 20 50 482 param-ptr @ param-fill

( window size  482x50a20x200 )

param-ptr @ window    ( resize work  )

fcb graph
graph filename scr_482x200a20x0
graph open-file       ( attach & open new window )
```



```
graph channel is-work                ( graphics is work window )

2 white border green paper cls

f# 50 f# 50 f# 30 circle      ( draw a circle  )
```

This allows you to use the newly created window interactively, without disturbing the window when the output channel scrolls.

An example program which draws a globe in different colours using the CIRCLE and OVER words may be found in screen 16 of the file 'elective_scr' on your Computer One FORTH microdrive.

THE FORTH–83 FULL SCREEN EDITOR

CHAPTER 3: THE FORTH-83 FULL SCREEN EDITOR

3.1: INTRODUCTION

Computer One FORTH includes a full screen editor which allows you to edit FORTH source text. Traditionally, FORTH has edited source files in terms of 1 k-byte blocks, presented as 16 lines of 64 characters. These are known as screens, and may be stored on the QL microdrives as described in the following section.

3.2: MASS STORAGE

The screens of source text are stored on microdrive in screen files, their size being limited by the available space on microdrive. The screens are numbered from zero upwards.

To open or change the default screen file being accessed, FORTH provides the word 'USING'. At startup, the file 'mdv1_forth_scr' is used as default. You can change the default screen file to 'mdv1_forth_scr' by typing:

```
using mdv1_forth_scr
```

The word 'BLOCK' is used to access blocks (screens) on the screen file, via FORTH's internal buffers. The word 'LOAD' is used to interpret input text from specified screens on the screen file. These words are described more elaborately in Glossary 1.

3.3: ENTERING THE EDITOR – command mode

To start the editor use the word EDIT, this will enter the first level of the editor, within which the following commands are available:-

3.3.1: Command Modes Control Codes

F1	Display menu.
C	Copy a single screen of text within the file. The user is prompted for the source and destination screen number.
E	Begin editing. The user is prompted for the number of the first screen to be edited.
I	Display index of current screen file.
M	Move a set of screens within the current file. The user is prompted for the number of the first source screen, and the first destination screen. Depending on the direction of the transfer, the routine begins at the appropriate end of the screen range so that when screen numbers overlap no data will be destroyed.
U	Change screen files. All updated buffers are written to the microdrive. The user is prompted for the drive assignment and the name of the new screen file. If the new file cannot be located, the previous screen file is reopened.
ESC	Write all updated buffers to drive, and return to FORTH.

When a screen number is requested in command mode, you must enter a decimal number (any number of digits) followed by a carriage return. You may use the usual keys to delete incorrect digits.

3.4: EDIT MODE

Edit mode is entered by using the E command in command mode, and is left by using the ESCAPE key.

All editing is done within the 16 row by 64 column box drawn on the screen. The four cursor keys will move the cursor within this box only. Typing any printable character will overwrite the existing character at the cursor position and move the cursor on one position.

Screens are stored and loaded by the words BLOCK and FLUSH.

3.4.1: Edit Mode Control Code

[cursor commands]

SHIFT Up	Home cursor.
SHIFT Down	Move cursor to end of screen.
Left	Move cursor left. If cursor is already at the

beginning of a line move it to the end of the previous line.

Right	Move cursor right. If cursor is already at the end of a line, move it to the beginning of the next.
Up	Move cursor up. If cursor is already on the top line move it to the left margin.
Down	Move cursor down. If cursor is already on the bottom line, move it to the right margin.
Enter	Move cursor to beginning of next line. If the cursor is already on the bottom line, move it to the right margin.
TAB	Tab right. Move cursor right to the next tab position. If it is at the end of the line move to the first position of the next line.
SHIFT TAB	Tab left. Move cursor left to the next tab position. If it is at the start of the line move to the last tab position of the next line.
ALT Right	Move cursor to end of text on the current line.

[word commands]

SHIFT Right	Move cursor forward one word.
SHIFT Left	Move cursor back one word.
SHIFT CTRL Right	Delete word to right of the cursor. Discard any trailing spaces, and bring the next word and the remainder of the line up to the cursor.

[line commands]

CTRL ALT Left	Erase the line containing the cursor, and replace it with a blank line.
CTRL Y	Delete the line containing the cursor, and move all the lines below it up one line, leaving a blank line at the bottom.
CTRL SHIFT J	Join lines. Bring as many words as possible from the next line up onto the current line, then left align the next line. If the next line is blank after this operation, delete it.
CTRL SHIFT S	Insert a blank line at the cursor, moving the lower lines down. The bottom line is lost.

CTRL ALT S	Split line at cursor. A new line is inserted. All the text to the right of the cursor is moved down to the new line. The bottom line is lost.
F2	Copy current line to the holding buffer, and delete the current line.
F3	Copy current line to the holding buffer. The line itself is unaffected, and the copy is displayed below the editing box. The number of lines in the buffer is also shown.
F4	Pop the top line of the holding buffer to the current line, which is overwritten. Note that the line is lost from the holding buffer.
F5	Pop the line stack and spread screen at the current line. Previous contents of line 15 is lost.

[character commands]

CTRL Left	Delete the character under the cursor moving the rest of the line to the left. Other lines are unchanged.
CTRL A	Insert a single space at the cursor. The rest of the line is moved to the right and the last character lost.
CTRL Q	Enter insert mode. All subsequent characters are inserted at the cursor and the rest of the line and the cursor are moved one space to the right. The rightmost characters drop off the end of the line and are lost. Insert mode terminates with the entry of another CTRL Q or ESC code.

[string commands]

CTRL Down	Find string. User is prompted for search argument. To use the same string argument as on previous search, just press <enter> in response to the prompt. The search may be interrupted by pressing any key.
CTRL Up	Find and replace string. User is prompted for search argument and replacement string. The replace string must be the same length or shorter than the search string. To use the arguments for subsequent search and replace operations, just push <enter> in response to the prompts. The search/replace may be interrupted by pressing any key.

CTRL L	Repeat previous search

[screen commands]

CTRL E	Clear the entire screen and leave the cursor at the top left corner.
CTRL SHIFT E	Erase screen from current cursor position to end.
CTRL N	Write current screen to store, and go to the next screen.
CTRL P	Write current screen to store, and go back to the previous screen.
CTRL SHIFT P	Write the current screen to store and go to screen 0 of the current file.
CTRL SHIFT N	Write the current screen to store and go to the last screen of the current file.
CTRL Z	Discards ALL the editing changes since the screen was last loaded from store.

[Miscellaneous commands]

F1	Display menu of control codes.
ESC	Leave editor and return to FORTH's command interpreter.

CHAPTER FOUR

MOTOROLA 68000 ASSEMBLER

The assembler may be used to create new machine language primitives (CODE definitions) in the FORTH dictionary. This capability is useful in applications, that are time dependent. FORTH programs can often be sped up a surprising degree by rewriting a few definitions in assembly language.

This assembler, like all FORTH assemblers of the usual FORTH design are intrinsically macro-assemblers. FORTH assemblers work by defining a set of words to correspond to the processor's registers, addressing modes etc. These words set flags and collect data which is processed and compiled into the dictionary by another set of words. These words are the assembler opcodes themselves. Like all FORTH operations the operands (registers, modes and addresses) must come before the operators (assembler opcode mnemonics e.g. ADD). Because of the full macro facilities, high level assembler structuring facilities can easily be added at a later date. Source code should be edited using FORTH's editor, or can be entered direct from the keyboard.

Load the assembler by typing 'ASM', and then switch to the assembler vocabulary by typing 'ASSEMBLER'.

4.1: EFFECTIVE ADDRESSES

The FORTH assembler accepts the following effective address (<ea>) formats. The abbreviations used are:

- Dx - a data register (x = 0...5) or OS W
- Ay - an address register (y = 0...3) or RP SP BP IP
- Rz - an address or data register

Effective Address	Motorola meaning	Example
Dx	Data Register Direct	D1
Ay	Address Register Direct	A2
(Ay)	Address Register Indirect	(A1)
d(Ay)	Address Register Indirect with displacement	6 d(A3)
(Ay)+	Address Register Indirect with Postincrement	(A2)+
-(Ay)	Address Register Indirect with Predecrement	-(A2)
d(Ax, Rz)	Address Register Indirect with Index	10 d(A1, D0)
#	Immediate	1 #
xxxx ,	Absolute short (source)	16 ,
, xxxx	Absolute short (destination)	, 1000
{ Rx..Ry..}	Register List	{ A0 D0 }

4.2: ASSEMBLER MNEMONICS

The 68k mnemonics are listed below in alphabetical order with their FORTH equivalents. Note that the presence or absence of commas and spaces is essential to avoid syntax errors. The error checking is not foolproof. Operand size is denoted by adding L. W. or B. before the opcode.

Eg: (SP)+ A0 L. MOVEA,

If no size is given, W. will be assumed unless the instruction requires otherwise.

Motorola Assembler		FORTH Assembler	
ABCD	Dy, Dx	Dy Dx	ABCD,
ADD	<ea>, Dn	<ea> Dn	ADD,
ADD	Dn, <ea>	Dn <ea>	ADD,
ADDA	<ea>, An	<ea> An	ADDA,
ADDI	#<data>, <ea>	<data> # <ea>	ADDI,
ADDQ	#<data>, <ea>	<data> # <ea>	ADDQ,
ADDX	Dy, Dx	Dy Dx	ADDX,
AND	<ea>, Dn	(Ay) -(Ax)	AND,
AND	Dn, <ea>	<ea> Dn	AND,
ANDI	#<data>, <ea>	<data> # <ea>	ANDI,
ANDI	#<data>, CCR	<data> # CCR B.	ANDI,
ANDI	#<data>, SR	<data> # SR	ANDI,
ASL	Dx, Dy	Dx Dy	ASL,
ASL	#<data>, Dy	<data> # Dy	ASL,
ASL	<ea>	<ea>	ASL,
ASR	Dx, Dy	Dx Dy	ASR,
ASR	#<data>, Dy	<data> # Dy	ASR,
ASR	<ea>	<ea>	ASR,
Bcc	<label>	<label>	Bcc,
BRA	<label>	<label>	BRA,
BSR	<label>	<label>	BSR,
BCHG	Dn, <ea>	Dn <ea>	BCHG,
BCHG	#<data>, <ea>	<data> # <ea>	BCHG,
BCLR	Dn, <ea>	Dn <ea>	BCLR,

BCLR	#<data>, <ea>	<data> # <ea>	BCLR,
BSET	Dn, <ea>	Dn <ea>	BSET,
BSET	#<data>, <ea>	<data> # <ea>	BSET,
BTST	Dn, <ea>	Dn <ea>	BTST,
BTST	#<data>, <ea>	<data> # <ea>	BTST,
CHK	<ea>, Dn	<ea> Dn	CHK,
CLR	<ea>	<ea>	CLR,
CMP	<ea>, Dn	<ea> Dn	CMP,
CMPA	<ea>, An	<ea> An	CMPA,
CMPI	#<data>, <ea>	<data> # <ea>	CMPI,
CMPM	(Ay)+, (Ax)+	(Ay)+ (Ax)+	CMPM,
DBcc	Dn, <label>	Dn <label> DBcc,	
DIVS	<ea>, Dn	<ea> Dn	DIVS,
DIVU	<ea>, Dn	<ea> Dn	DIVU,
EOR	Dn, <ea>	<ea> Dn	EOR,
EORI	#<data>, <ea>	<data> # <ea>	EORI,
EORI	#<data>, CCR	<data> # CCR B.	EORI,
EORI	#<data>, SR	<data> # SR	EORI,
EXG	Rx, Ry	Rx Ry	EXG,
EXT	Dn	Dn	EXT,
JMP	<ea>	<ea>	JMP,
JSR	<ea>	<ea>	JSR,
LEA	<ea>, An	<ea> An	LEA,
LINK	An, #<data>	An <data> #	LINK,
LSL	Dx, Dy	Dx Dy	LSL,
LSL	#<data>, Dy	<data> # Dy	LSL,
LSL	<ea>	<ea>	LSL,
LSR	Dx, Dy	Dx Dy	LSR,
LSR	#<data>, Dy	<data> # Dy	LSR,
LSR	<ea>	<ea>	LSR,
MOVE	<ea>, <ea>	<ea> <ea>	MOVE,
MOVE	<ea>, CCR	<ea>	MOVE,
MOVE	<ea>, SR	<ea>	MOVE,
MOVE	SR, <ea>	<ea>	MOVE,
MOVE	An, USP	An	MOVE,
MOVE	USP, An	An	MOVE,
MOVEA	<ea>, An	<ea> An	MOVEA,
MOVEM	<reg. list>, <ea>	<reg. list> <ea>	MOVEM,
MOVEM	<ea>, <reg. list>	<ea> <reg. list>	MOVEM,
MOVEP	Dx, d(Ay)	Dx d(Ay)	MOVEP,
MOVEP	d(Ay) Dx	d(Ay) Dx	MOVEP,
MOVEQ	#<data>, Dn	<data> # Dn	MOVEQ,
MULS	<ea>, Dn	<ea> Dn	MULS,
MULU	<ea>, Dn	<ea> Dn	MULU,
NBCD	<ea>	<ea>	NBCD,
NEG	<ea>	<ea>	NEG,
NEGX	<ea>	<ea>	NEGS,
NOP			NOP,
NOT	<ea>	<ea>	NOT,
OR	<ea>, Dn	<ea> Dn	OR,
OR	Dn, <ea>	Dn <ea>	OR,
ORI	#<data>, <ea>	<data> # <ea>	ORI,
ORI	#<data>, CCR	<data> # CCR B.	ORI,
ORI	#<data>, SR	<data> # SR	ORI,
PEA	<ea>	<ea>	PEA,

```
RESET
ROL     Dx, Dy              Dx Dy           RESET,
ROL     #<data>, Dy         <data> # Dy     ROL,
ROL     <ea>                <ea>            ROL,
ROR     Dx, Dy              Dx Dy           ROR,
ROR     #<data>, Dy         <data> # Dy     ROR,
ROR     <ea>                <ea>            ROR,
ROXL    Dx, Dy              Dx Dy           ROXL,
ROXL    #<data>, Dy         <data> # Dy     ROXL,
ROXL    <ea>                <ea>            ROXL,
ROXR    Dx, Dy              Dx Dy           ROXR,
ROXR    #<data>, Dy         <data> # Dy     ROXR,
ROXR    <ea>                <ea>            ROXR,
RTE                                         RTE,
RTR                                         RTR,
RTS                                         RTS,
SBCD    Dy, Dx              Dy Dx           SBCD,
SBCD    -(Ay), -(Ax)        -(Ay) -(Ax)     SBCD,
Scc     <ea>                <ea>            Scc,
STOP    #<data>             <data> #        STOP,
SUB     <ea>, Dn            <ea> Dn         SUB,
SUB     Dn, <ea>            Dn <ea>         SUB,
SUBA    <ea>, An            <ea> An         SUBA,
SUBI    #<data>, <ea>       <data> # <ea>   SUBI,
SUBQ    #<data>, <ea>       <data> # <ea>   SUBQ,
SUBX    Dy, Dx              Dy Dx           SUBX,
SUBX    -(Ay), -(Ax)        -(Ay) -(Ax)     SUBX,
SWAP    Dn                  Dn              SWAP,
TAS     <ea>                <ea>            TAS,
TST     <ea>                <ea>            TST,
TRAP    #<vector>           <vector> #      TRAP,
TRAPV                                       TRAPV,
UNLK    An                  An              UNLK,
```

4.3: LOCAL LABELS

The FORTH assembler has 5 predeclared labels. A local label is planted using the word:

```
n$:     (where n= 1...5)
```

It is referred to using the word:

```
n$
```

Note that forward jumps are possible

4.4: ASSEMBLER MACROS

Defined in the assembler are several "macros" that extend the usefulness of this assembler. The defined macros are:

NEXT,	Compiles a three-word routine that transfers control back to the "inner interpreter" of FORTH.
PUSHFORTH,	Pushes the FORTH registers (IP, W, RP, SP, BP, and OS,) onto the stack so that they can't be altered within a code definition.
POPFORTH,	Restores the registers pushed above.
L.JSR,	Assembles a 32-bit "jump to subroutine" for accessing system routines anywhere in the address space of the 68008.

4.5: USING THE ASSEMBLER

Normally the assembler will be used to create new FORTH words written in assembler. Such words use CODE and END-CODE in place of : and ; . To return to the FORTH interpreter the word NEXT, is used (ie: 'NEXT' followed by a comma and no space between them).

The word CODE creates a new dictionary header, and switches the vocabulary used for looking up words names in to the ASSEMBLER vocabulary. The vocabulary in which the word is built is unchanged. If you are confused by this ignore it, it is really only a technical detail. If you do not want a code header to be built, use the word ASSEMBLER to switch on the assembler words (try FORTH WORDS, and then ASSEMBLER WORDS).

As an example, study the definition of CMOVE in assembly language. CMOVE takes three arguments from the stack: The source address, the destination address, and the number of bytes (top). It then moves the specified number of bytes from the source to the destination address, incrementing both addresses with each iteration.

```
( source destination count -- )
        CODE CMOVE
             (SP)+ D0          MOVE,     ( pop the byte count )
             (SP)+ OS          MOVE,     ( pop dest. address, logical )
             0 d(BP, OS) A0    LEA,      ( convert dest. to real addr )
             (SP)+ OS          MOVE,     ( pop source address, logical )
             0 d(BP, OS) A1    LEA,      ( convert to real address )
        1$:  1 # D0            SUBQ,     ( decrement byte count )
             2$                BMI,      ( loop finished? )
             (A1)+ (A0)+ B.    MOVE      ( move one byte, post incr. )
             1$                BRA,      ( branch to start of loop )
        2$:                    NEXT,     ( end execution )
( Note NEXT, includes END-CODE .)
```

4.8: DEDICATED FORTH REGISTERS

FORTH	68000	preservation rules
RP	A7	Return stack pointer, preserved across FORTH words.
SP	A6	Parameter stack pointer, preserved across FORTH words.
BP	A5	Base pointer. Contains absolute starting address of FORTH. DO NOT TOUCH.
IP	A4	Interpretive pointer - FORTH's own internal program counter. Preserved across FORTH words except to cause a branch.
OS	D7	Offset register with high word set to 0. Used by FORTH with BP to form an actual 32-bit address from a 16-bit logical address. DO NOT MODIFY THE HIGH WORD.
W	D6	Sometimes output from NEXT. May be altered before jumping to NEXT.

All other registers may be freely used by assembler routines, but these registers may (will) be altered by QDOS system calls.

FORTH –83 Glossary Notation

Order

The glossary definitions are listed in ASCII alphabetical order.

Capitalization

Word names are capitalized throughout.

Stack Notation

The stack parameters input to output from a definition are described using the notation

```
before -- after

    before    stack parameters before execution
    after     stack parameters after execution
```

In this notation, the top of the stack is to the right. Words may also be shown in context when appropriate.

Unless otherwise noted, all stack notation describes execution time. If it applies at compile time, the line is followed by: (compiling) .

Attributes

Capitalized symbols indicate attributes of defined words:

C The word may only be used during compilation of a colon definition.
I Indicates that the word is IMMEDIATE and will be executed during compilation, unless special action is taken.
M This word has a potential multiprogramming impact.
U A user variable.

Serial Numbers

When a substantial alteration to a word's definition is made or when a new word is added, the serial number will be the last two digits of the year of the respective FORTH Standard in which such change was made (i.e., "83").

Stack Parameters

Unless otherwise stated all references to numbers apply to 16-bit signed integers. The implied range of values is shown as (from..to). The content of an address is shown by double braces, particulary for the contents of variables, i.e., BASE {2..72}.

The following are the stack parameter abbreviations and types of numbers used throughout the glossary. These abbreviations may be suffixed with a digit to differentiate multiple parameters of the same type.

Stack Abbrv.	Number Type	Range in Decimal	Minimum Field
flag	boolean	0=false, else=true	16
true	boolean	-1 (as a result)	16
false	boolean	0	16
b	bit	(0..1)	1
char	character	(0..127)	7
8b	8 arbitrary bits (byte)	not applicable	8
16b	16 arbitrary bits	not applicable	16
n	number (weighted bits)	(-32,768..32,767)	16
+n	positive number	(0..32,767)	16
u	unsigned number	(0..65,535)	16
w	unspecified weighted number (n or u)	(-32,768..65,535)	16
addr	address (same as u)	(0..65,535)	16
32b	32 arbitrary bits	not applicable	32
d	double number	(-2,147,483,683,648.. 2,147,483,647)	32
+d	positive double number	(0..2,147,483,647)	32
ud	unsigned double number	(0..4,294,967,295)	32
wd	unspecified weighted double number (d or ud)	(-2,147,483,648.. 4,294,967,295)	32
sys	0, 1, or more system dependent stack entries	not applicable	na

Any other symbol refers to an arbitrary signed 16-bit integer in the range (-32,768..32,767), unless otherwise noted.

Because of the use of two's complement arithmetic, the signed 16-bit number (n) -1 has the same bit representation as the unsigned number (u) 65,535. Both of these numbers are within the set of unspecified weighted numbers (w).

Input Text

<name>
An arbitrary FORTH word accepted from the input stream. This notation refers to text from the input stream, not to values on the data stack.

ccc
A sequence of arbitrary characters accepted from the input stream until the first occurrence of the specified delimited character. The delimiter is accepted from the input stream, but it is not one of the characters ccc and is therefore not otherwise processed. This notation refers to text from the input stream, not to values on the data stack. Unless noted otherwise, the number of characters accepted may be from 0 to 255.

Microdrive, Window, and Device Interface

As QDOS treats all devices in the same way, this section applies not just to microdrives and other file-oriented devices, but to any other devices, e.g. serial ports, that are supported by QDOS. In addition, the same techniques are used for windows (output only), and consoles (input and output).

Computer One FORTH uses the file handling interface standard established by Laboratory Microsystems. Files are handled through file control blocks. A file control block consists of a control area, and a 128 byte data buffer area. The layout of the control area varies between operating systems, but the data buffer is constant, and is used by the READ-SEQ, WRITE-SEQ, READ-RANDOM, and WRITE-RANDOM words. For this implementation the control area is 64 bytes long, and its organisation follows that of a microdrive header up to the end of the file name. A file control block is established and named by the word FCB, e.g.:

```
FCB FRED
```

As each file control block is defined, it is allocated space. When record operations are performed referencing a given file control block, the buffer address is returned if the operation is successful, otherwise 0 is returned. The number of file control blocks (and hence the number of open files) is limited only by dictionary space and QDOS.

The status code returned by all file operations is a 0 for successful operations, and a standard QDOS error code if the operation failed. The status code returned by record operations is either the disc buffer address for success, or 0 for failure. The error code of the last operation is contained in the variable DISK-ERROR.

6.1: FILE WORDS

BGET --

Used in the form:

BGET <filename>

Loads and relocates a binary image saved using BSAVE.

BSAVE --

Used in the form:

BSAVE <word-name> <filename>.

Before executing BSAVE, the source code must be compiled TWICE. <word-name> refers to the first word in the source code that has been compiled twice. A binary image of the compiled code will be saved as <filename>. This may be reloaded by BGET which will relocate it.

?BUFFER-ADDR fcb-addr -- disk-buffer-addr

Given the address of a file control block, the address of the relevant disk transfer buffer is returned.

CLOSE-FILE fcb-address -- status code

Close the file. Microdrive directories will be updated.

CONTROL-FCB -- (defining)
-- fcb address (execution)

Used in the form CONTROL-FCB <fcb-name>

DELETE-FILE fcb-address -- status-code

Delete the specified file, and update the directory.

DIR --

Used in the form:

DIR <medium name>

lists a directory of the selected files.

DISK-ERROR -- addr

Variable containing the value of the last file related error (or success) to have taken place.

FCB -- (defining)
-- fcb-address (execution)

Used in the form FCB <fcb-name>. Allocates and initialises a file control block and disk-buffer. Subsequent execution of the control block name will leave the address of the fcb control area on the stack.

PCBADDR --- IS-IO

Sets up the device whoose fcb addr is on the stack, as the current input/ output channel.

FILENAME fcb-address --

Used in the form:

(fcb-address --) FILENAME <file-spec>

FRED FILENAME MDV2_ELECTIVE_SCR

Format a file/device/window specification into the file control block.

INPUT-FILENAME fcb-address --

Queries the console for a file name, and passes the information to the control block.

MAKE-FILE fcb-address -- status-code

Creates and opens a new file with the name given in the file control block.

OPEN-FILE fcb-address -- status-code

Find and make available the file named in the file control block for further record operations.

PARSE-FILENM fcb-address string-address --

Transfer a file-name at the string address to the file control block, and initialise all reserved areas.

READ-HEADER fcb-address --- status-code

Read a microdrive file header into the control area.

READ-LEN length data-address channel-id
-- count status-code

Read length bytes from the channel specified by channel-id to the address given by data-address. The number of bytes actually transferred and the error status are returned. This is the general purpose read primitive.

READ-RANDOM fcb-address record-no. -- status-code

Read the designated record from storage into the disc buffer, returning either the buffer address or zero if the record cannot be read. A record size of 128 bytes is assumed.

READ-SEQ fcb-address --- status-code

Read the next sequential record (size=128 bytes) into the disc buffer, returning the buffer address for success, or 0 for failure.

SAVE --

Used in the form:

SAVE <filename>

Saves a copy of the complete system as a memory image to <filename>. This image will contain any code you have added. It may be reloaded using a copy of BOOT that has the filename of the binary file changed to <filename>.

SEEK-ABS dl fcb-address -- status-code

Modifies the file pointer so that the next i/o will start at the dl-th byte in the file.

SEEK-REL dl fcb-address -- status-code

Moves the file pointer by dl (signed) bytes.

WRITE-HEADER fcb-address -- status-code

Writes the file header from the file control block to QDOS.

WRITE-LEN length data-address channel-id
-- count status-code

Write length bytes from data-adress to the device. The number of bytes actually transferred and the error status are returned. This is the general purpose write primitive.

WRITE-RANDOM fcb-address record-number -- status-code

Write the record (size=128 bytes) from the buffer area to the device, returning the buffer address for success, or 0 for failure.

WRITE-SEQ fcb-address -- status-code

Write the buffer to the device, returning the buffer address for success, or 0 for failure.

A simple example using the interface functions. Open a file named USER_TMP, read record 0 into memory, then print word 0 of the record.

```
FCB FILE1                          ( establish fcb )
FILE1 FILENAME USER_TMP            ( set up file name )
: DEMO          FILE1 OPEN-FILE
                  ABORT" File not found"
                FILE1 0 READ-RANDOM DUP 0=
                  ABORT" Record does not exist"
                @ .
                FILE1 CLOSE-FILE DROP  ;
```

For more details of the error codes see the SuperBASIC manual.

FORTH Multi–tasking

CHAPTER 7: FORTH Multi-tasking

FORTH contains a simple multi-tasker which allows you to execute up to ten background tasks concurrently with the foreground task. The background tasks are given control in a round-robin manner whenever the foreground task is waiting for input from the keyboard.

By foreground task, we refer to any task which was invoked by simply entering its name followed by a carriage return. The foreground task has ultimate control of the keyboard, video display, and other i/o.

Background tasks designed by the user must obey certain rules in order for the system as a whole to perform properly:

1. Each background task must be self-contained. It must leave the parameter and return stacks balanced, i.e. no extra values may be consumed or left behind. If any information must be maintained from one invocation to another, it should be kept in a variable.

2. A background task must execute to completion in a reasonable length of time, otherwise the user will notice that keyboard response is delayed. If all the necessary work cannot be done quickly enough, it should be partitioned into several tasks which pass data through variables.

3. In general, background tasks should not attempt to access i/o devices.

The words used to control multi-tasking are:

KILL --

Used in the form: KILL <name> - removes a task from the background task list.

KILL-TASKS --

Kills all the background tasks.

START --

Used in the form: START <name> - adds a task to the background task list. Up to ten may run concurrently.

GLOSSARY ONE:

FORTH–83 Word Set Glossary

This glossary lists the minimal set of words required to meet the FORTH-83 standard.

! 16b addr -- 79 "store"
16b is stored at addr

+d1 -- +d2 79 "sharp"
The remainder of +d1 divided by the value of BASE is converted to an ASCII character and appended to the output string toward lower memory addresses. +d2 is the quotient and is maintained for further processing. Typically used between <# and #>.

#> 32b -- addr +n 79 "sharp-greater"
Pictured numeric output conversion is ended dropping 32b. addr is the address of the resulting output string. +n is the number of characters in the output string. addr and +n together are suitable for TYPE.

#S +d -- 0 0 79 "sharp-s"
+d is converted appending each resultant character into the pictured numeric output string until the quotient (see: #) is zero. A single zero is added to the output string if the number was initially zero. Typically used between <# and #>

#TIB -- addr U,83 "number-t-i-b"
The address of a variable containing the number of bytes in the text input buffer. #TIB is accessed by WORD when BLK is zero. {{0..capacity of TIB}}

' -- addr M,83 "tick"
Used in the form:
' <name>
addr is the compilation address of <name>. An error condition exists if <name> is not found in the currently active search order.

(-- I,M,83 "paren"
-- (compiling)
Used in the form:
(ccc)
The characters ccc, delimited by) (closing parenthesis), are considered comments. Comments are not otherwise processed. The

blank following (is not part of ccc. (may be freely used while interpreting or compiling. The number of characters in ccc may be from zero to the number of characters remaining in the input stream up to the closing parenthesis.

***** w1 w2 -- w3 79 "times"
w3 is the least-significant 16 bits of the arithmetic product of w1 times w2.

***/** n1 n2 n3 -- n4 83 "times-divide"
n1 is first multiplied by n2 producing an intermediate 32-bit result. n4 is the floor of the quotient of the intermediate 32-bit result divided by the divisor n3. The product of n1 times n2 is maintained as an intermediate 32-bit result for greater precision than the otherwise equivalent sequence: n1 n2 * n3 / . An error condition results if the divisor is zero or if the quotient falls outside of the range [-32,768..32,767].

***/MOD** n1 n2 n3 -- n4 83 "times-divide-mod"
n1 is first multiplied by n2 producing an intermediate 32-bit result. n4 is the remainder and n5 is the floor of the quotient of the intermediate 32-bit result divided by the divisor n3. A 32-bit intermediate product is used as for */ . n4 has the same sign as n3 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range [-32,768..32767].

+ w1 w2 -- w3 79 plus"
w3 is the arithmetic sum of w1 plus w2

+! w1 addr -- 79 "plus-store"
w1 is added to the w value at addr using the convention for +. This sum replaces the original value at addr.

+LOOP n1 -- C,I,83 "plus-loop"
sys -- (compiling)
n is added to the loop index. If the new index was incremented across the boundary between limit-1 and limit the the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues to just after the corresponding DO . sys is balanced with its corresponding DO . See: DO

, 16b -- F 79 "comma"
ALLOT space for 16b then store 16b at HERE 2- .

- w1 w2 -- w3 79 "minus"
w3 is the result of subtracting w2 from w1

-TRAILING addr +n1 -- addr +n2 79 "dash-trailing"
The character count +n1 of a text string beginning at addr is adjusted to exclude trailing spaces. If +n1 is zero, then +n2 is also zero. If the entire string consists of spaces, then +n2 is zero.

. n -- M,79 "dot"
The absolute value of n is displayed in a free field format with a leading minus sign if n is negative.

." -- C,I,83 "dot-quote"
-- (compiling)
Used in the form:
." ccc"
Later execution will display the characters ccc up to but not including the delimiting " (close-quote). The blank following ." is not part of ccc.

.(-- I,M,83 "dot-paren"
-- (compiling)
Used in the form:
.(ccc)
The characters ccc up to but not including the delimiting) (closing parenthesis) are displayed. The blank following .(is not part of ccc.

/ n1 n2 -- n3 83 "divide"
n3 is the floor of the quotient of n1 divided by the divisor n2. An error condition results if the divisor is zero or if the quotient falls outside the range [-32,768..32767].

/MOD n1 n2 -- n3 n4 83 "divide-mod"
n3 is the remainder and n4 the floor of the quotient of n1 divided by the divisor n2. n3 has the same sign as n2 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range [-32,768..32,767].

0< n -- flag 83 "zero-less"
flag is true if n is less than zero (negative).

0= w -- flag 83 "zero-equals"
flag is true if w is zero.

0> n -- flag 83 "zero-greater"
flag is true if n is greater then zero.

1+ w1 -- w2 79 "one-plus"
w2 is the result of adding one to w1 according to the operation of + .

1- w1 -- w2 79 "one-minus"
w2 is the result of subtracting one from w1 according to the operation of
- .

2+ w1 -- w2 79 "two-plus"
w2 is the result of adding two to w1 according to the operation of + .

ATTACH --

Used in the form ATTACH <fcb-name> <file-spec>. Allocates and initialises a file control block, associates the control block with the given file, and opens the file. For example

```
attach fred    scr_400x200x50x20
fred channel is-work cls
```

will open a window with the given parameters, make it the work channel and clear the window.

SET-FILE --

Used in the form SET-FILE <fcb-name> <file-spec>. This is the same as ATTACH, except that the file is not opened by the word.

PRINTER-IS --

Used in the form PRINTER-IS <file-spec>. Opens the given file to be used by the PRINTER word. For example, if a printer is attached to the serial port ser1 then to print screen 7 of the current screen file you would use the following :-

```
printer-is ser1
printer 7 list console ( print screen 7 and then reset
                         output to console )
```

Information on the exact specification of the serial port and other devices can be found in the QL User Guide under the keyword 'devices'.

D+ wd1 wd2 -- wd3

wd3 is the result of adding wd1 to wd2.

D* d u -- d

Multiplies a double word by an unsigned word to give a double word result.

D/ d u -- d

Divides a double word by an unsigned word to give a double word result.

DELETE --

Used in the form DELETE <file-spec>. Deletes the specified file. e.g DELETE mdv2_test_scr.

FORMAT --

Used in the form FORMAT <device-spec>. Formats the specified device. e.g. FORMAT mdv2_forthfiles.

Creating and using screen files

To create a screen file called, for example, mdv2_test_scr, use the following

```
FCB scr   scr FILENAME mdv2_test_scr
scr MAKE-FILE  scr CLOSE-FILE
```

Note that the file must be closed before you can use the word USING to make it the current screen file for use with the editor.

To ensure that the most recent edits on a screen file have been written to microdrive cartridge before resetting the system use the word USING with no parameters. This will close and reopen the current screen file, ensuring that the last updates have been written to the cartridge.

Multi-Tasking

When an application program is using multi-tasking, the foreground task, i.e. the one prompting for input, should use the word KEY and not EXPECT, since EXPECT will halt all background tasks while it is waiting for input.

Running stand alone programs

When the FORTH system is started up it uses a well-known FORTH location, Hex 18A, to find the code to start executing when it has finished initialising the system. You can cause your own application to be automatically run on startup by changing the value in this location so that it will execute a compiled FORTH word. For example, suppose we wish to run the NIBBLERS game as a stand alone program. First we have to load and compile the source, then setup the location 18A, save the current memory image and alter the boot file so that it will use the new image on startup.

Before setting up the stand alone program we could use FORGET to forget words that will not be required by our application.

```
using mdv1_forth_scr     ( this has the NIBBLERS source )
22 load                  ( load and compile the source )
' nibblers hex 18A !     ( Set up the start up location
                           with the NIBBLERS word )
save mdv2_nibblers       ( save the current image )
```

Now if we copy the BASIC boot file onto the micro drive with the NIBBLERS image and alter line 40 to give it the name of the NIBBLERS image, when this image is booted it will run the NIBBLERS game.

For experienced FORTH users all the guaranteed locations in the FORTH system are given below. All values are given in hex.

```
0126   initial value of &TIB
0128   initial value of WARNING
012A   initial value of FENCE
012C   initial value of DP
012E   initial value of VOC-LINK

0140   vector for          EMIT
0142   vector for          KEY
0144   vector for          ?TERMINAL
0146   vector for          TYPE
0148   vector for          GOTOXY
014A   vector for          CLEARSCREEN
014C   vector for          CLREOS
014E   vector for          CLREOL

0172   vector for          BLK-READ
0174   vector for          BLK-WRITE
0176   vector for          OPEN-SCR
0178   vector for          CLOSE-SCR
```

false. The words after UNTIL or REPEAT will be executed when either loop is finished. sys is balanced with its corresponding UNTIL or WHILE .

BLK -- addr U,79 "b-l-k"
The address of a variable containing the number of the mass storage block being interpreted as the input stream. If the value of BLK is zero the input stream is taken from the text input buffer. {{0..the number of blocks available -1}} See: TIB

BLOCK u -- addr M,83 "block"
addr is the address of the assigned buffer of the first byte of block u. If the block occupying that buffer is not block u and has been UPDATEd it is transferred to mass storage before assigning the buffer. If block u is not already in memory, it is transferred from mass storage into an assigned block buffer. A block many not be assigned to more than one buffer. If u is not an available block number, an error condition exists. Only data within the last buffer referenced by BLOCK or BUFFER is valid. The contents of a block buffer must not be changed unless the change may be transferred to mass storage.

BUFFER u -- addr M,83 "buffer"
Assign a block buffer to block u. addr is the address of the first byte of the block within its buffer. This function is fully specified by the definition of BLOCK except that if the block is not already in memory it might not be transferred from mass storage. The contents of the block buffer assigned to block u by BUFFER are unspecified.

C! 16b addr -- 79 "c-store"
The least-significant 8 bits of 16b are stored into the byte at addr.

C@ addr -- 8b 79 "c-fetch"
8b is the contents of the byte at addr.

CMOVE addr1 addr2 u -- 83 "c-move"
Move u bytes beginning at address addr1 to addr2. The byte at addr2 is moved first, proceeding towards high memory. If u is zero nothing is moved.

CMOVE> addr1 addr2 u -- 83 "c-move-up"
Move u bytes at address addr1 to addr2. The move begins by moving the byte at (addr1 plus u minus 1) to (addr2 plus u minus 1) and proceed to successively lower addresses for u bytes. If u is zero nothing is moved. (Useful for sliding a string towards higher addresses).

COMPILE -- C,83 "compile"
Typically used in the form:
: <name> ... COMPILE <namex> ... ;
When <name> is executed, the compilation address compiled for <namex> is compiled and not executed. <name> is typically immediate and <namex> is typically not immediate.

CONSTANT 16b -- M,83 "constant"
A defining word executed in the form:
16b CONSTANT <name>
Creates a dictionary entry for <name> so that when <name> is later executed, 16b will be left on the stack.

CONVERT +d1 addr1 -- +d2 addr2 79 "convert"
+d2 is the result of converting the characters within the text beginning at addr1+1 into digits, using the value of BASE , and accumulating each into +d1 after multiplying +d1 by the value of BASE . Conversion continues until an unconvertible character is encountered. addr2 is the location of the first unconvertible character.

COUNT addr1 -- addr2 +n 79 "count"
addr2 is addr1+1 and +n is the length of the counted string at addr1. The byte at addr1 contains the byte count +n. Range of +n is {{0..255}}.

CR -- M,79 "c-r"
Display a carriage-return and line-feed or equivalent operation.

CREATE -- M,79 "create"
A defining word executed in the form:
CREATE <name>
Creates a dictionary entry for <name>. After <name> is created, the next available dictionary location is the first byte of <name>'s parameter field. When <name> is subsequently executed, the address of the first byte of <name>'s parameter field is left on the stack. CREATE does not allocate space in <name>'s parameter field.

D+ wd1 wd2 -- wd3 79 "d-plus"
wd1 is the arithmetic sum of wd1 plus wd2.

D< d1 d2 -- flag 83 "d-less-then"
flag is true if d1 is less than d2 according to the operation of < except extended to 32 bits.

DECIMAL -- 79 F "decimal"
Set the input-output numeric conversion base to ten.

DEFINITIONS -- 79 "definitions"
The compilation vocabulary is changed to be the same as the first vocabulary in the search order.

DEPTH -- +n 79 "depth"
+n is the number of 16-bit values contained in the data stack before +n was placed on the stack.

DNEGATE d1 -- d2 79 "d-negate"
d2 is the two's complement of d1.

DO w1 w2 -- C,I,83 "do"
-- sys (compiling)

Used in the form:
DO ... LOOP
or
DO ... +LOOP Begins a loop which terminates based on control parameters. The loop index begin at w2, and terminates based on the limit w1. See LOOP and +LOOP for details on how the loop is terminated. The loop is always executed at least once. For example: w DUP DO ... LOOP executes 65,536 times. sys is balanced with its corresponding LOOP or +LOOP.

An error condition exists if insufficient space is available for at least three nesting levels.

DOES> -- addr C,I,83 "does"
-- (compiling)

Defines the execution-time action of a word created by high-level defining word. Used in the form:

: <namex> ... <create> ... DOES> ... ;
and then
<namex> <name>
where <create> is CREATE or any user defined word which executes CREATE .

Marks the termination of the defining part of the defining word <namex> and then begins the definition of the execution-time action for words that will later be defined by <namex>. When <name> is later executed, the address of <name>'s parameter field is placed on the stack and then the sequence of words between DOES> and ; are executed.

DROP 16b -- 79 "drop"
16b is removed from the stack.

DUP 16b -- 16b 16b 79 "dupe"
Duplicate 16b.

ELSE -- C,I,79 "else"
sys1 -- sys2

Used in the form:
flag IF ... ELSE ... THEN
ELSE executes after the true part following IF . ELSE forces execution to continue at just after THEN . sys1 is balanced with its corresponding IF . sys2 is balanced with its corresponding THEN . See: IF THEN

EMIT 16b -- M,83 "emit"
The least-significant 7-bit ASCII character is displayed.

EXECUTE addr -- 79 "execute"
The word definition indicated by addr is executed. An error condition exists if addr is not a compilation address

EXIT -- C,79 "exit"
Compiled within a colon definition such that when executed, that colon definition returns control to the definition that passed control to it by returning control to the return point on top of the return stack. An error condition exists if the top of the return stack does not contain a valid return point. May not be used within a Do-loop.

EXPECT addr +n -- M,83 "expect"
Receive characters and store each into memory. The transfer begins at addr proceeding towards higher addresses one byte per character until either a "return" is received or until +n characters have been transfered. No more than +n characters will be stored. The "return" is not stored in memory. No characters are received of transfered if +n is zero. All characters actually received and stored into memory will be displayed, with "return" displaying as space. See: SPAN

FILL addr u 8b -- 83 "fill"
u bytes of memory beginning at addr are set to 8b. No action is taken if u is zero.

FIND addr1 -- addr2 n 83 "find"
addr1 is the address of a counted string. The string contains a word name to be located in the currently active search order. If the word is not found, addr2 is the string addr1, and n is zero. If the word is found, addr2 is the compilation address and n is set to one of two non-zero values. If the word found has the immediate attribute, n is set to one. If the word is non-immediate, n is set to minus one (true).

FLUSH -- M,83 "flush"
Performs the function of SAVE-BUFFERS then deassigns all block buffers. (This may be useful for mounting or changing mass storage media).

FORGET -- M,83 "forget"
Used in the form:
FORGET <name>
If <name> is found in the compilation vocabulary, delete <name> from the dictionary and all words added to the dictionary after <name> regardless of their vocabulary. Failure to find <name> is an error condition. An error condition also exists if the compilation vocabulary is deleted.

FORTH -- 83 "forth"
the name of the primary vocabulary. Execution replaces the first vocabulary in the search order with FORTH . FORTH is initially the compilation vocabulary and the first vocabulary in the search order. New definitions become part of the FORTH vocabulary until a different compilation vocabulary is established. See: VOCABULARY

FORTH-83 -- 83 "forth-83"
Assures that a FORTH-83 Standard System is available, otherwise an error condition exists.

HERE -- addr 79 "here"
The address of the next available dictionary location.

HOLD char -- 79 "hold"
char is inserted into a pictured numeric output string. Typically used between <# and #>.

I -- w C,79 "i"
w is a copy of the loop index. May only be used in the form:
DO ... I ... LOOP
or
DO ... I ... +LOOP

IF flag -- C,I,79 "if"
-- sys (compiling)
Used in the form:
flag IF ... ELSE ... THEN
or
flag IF ... THEN
If flag is true, the words following IF are executed and the words following ELSE until just after the THEN are skipped. The ELSE part is optional.

if flag is false, words from IF through ELSE , or from IF through THEN (when no ELSE is used), are skipped. sys is balanced with its corresponding ELSE or THEN .

IMMEDIATE -- 79 "immediate"
Marks the most recently created dictionary entry as a word which will be executed when encountered during compilation rather than compiled.

J -- w C,79 "j"
w is a copy of the index of the next outer loop. May only be used within a nested DO-LOOP or DO-+LOOP in the form, for example:
DO ... DO ... J ... LOOP ... +LOOP

KEY -- 16b M,83 "key"
The least-significant 7 bits of 16b is the next ASCII character received. All valid ASCII characters can be received. Control characters are not processed by the system for any editing purpose. Characters received by KEY will not be displayed.

LEAVE -- C,I,83 "leave"
-- (compiling)
Transfers execution to just beyond the next LOOP or +LOOP . The next loop is terminated and loop control parameters are discarded. May only be used in the form:
DO ... LEAVE ... LOOP
or

DO ... LEAVE ... +LOOP
LEAVE may appear within other control structures which are nested within the do-loop structure. More than one LEAVE may appear within a do-loop.

LITERAL -- 16b C,I,79 P "literal"
16b -- (compiling)
Typically used in the form:
[16b] LITERAL
Compiles a system dependent operation so that when later executed, 16b will be left on the stack.

LOAD u -- M,79 "load"
The contents of >IN and BLK , which locate the current input stream, are saved. The input stream is then redirected to the beginning of screen u by setting >IN to zero and BLK to u . The screen is then interpreted. If interpretation from screen u is not terminated explicitly it will be terminated after the input stream is exhausted and then the contents of >IN and BLK will be restored. An error condition exists if u is zero. See: >IN BLK BLOCK

LOOP -- C,I,83 "loop"
sys -- (compiling)
Increments the DO-LOOP index by one. If the new index was incremented across the boundary between limit-1 and limit the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues to just after the corresponding DO . sys is balanced with its corresponding DO . See: DO

MAX n1 n2 -- n3 79 "max"
n3 is the greater of n1 and n2 according to the operation of >.

MIN n1 n2 -- n3 79 "min"
n3 is the lesser of n1 and n2 according to the operation of <.

MOD n1 n2 -- n3 83 "mod"
n3 is the remainder after dividing n1 by the divisor n2 . It has the same sign as n2 or is zero. An error condition results if the divisor is zero or if the quotient falls outside the range (-32,768..32,767).

NEGATE n1 -- n2 79 "negate"
n2 is the two's complement of n1, i.e., the difference of zero less n1.

NOT 16b1 -- 16b2 83 "not"
16b2 is the one's complement of 16b1.

OR 16b1 16b2 -- 16b3 79 "or"
16b3 is the bit-by-bit inclusive-or of 16b1 with 16b2.

OVER 16b1 16b2 --
16b1 16b2 16b3 79 "over"
16b3 is a copy of 16b1.

PAD -- addr 83 "pad"
The lower address of a scratch area used to hold data for intermediate processing. The address or contents of PAD may change and the data lost if the address of the next available dictionary location is changed. The minimum capacity of PAD is 84 characters.

PICK +n -- 16b 83 "pick"
16b is a copy of the +nth stack value, not counting +n itself. (0..the number of elements on the stack-1)
0 PICK is equivalent to DUP
1 PICK is equivalent to OVER

QUIT -- 79 "quit"
Clears the return stack, sets interpret state, accepts new input from the current input data device, and begins text interpretation. No message is displayed.

R> -- 16b C,79 "r-from"
16b is removed from the return stack and transferred to the data stack.

R@ -- 16b C,79 "r-fetch"
16b is a copy of the top of the return stack.

REPEAT -- C,I,79 "repeat"
sys -- (compiling)
Used in the form:
BEGIN ... flag WHILE ... REPEAT
At execution time, REPEAT continues execution to just after the corresponding BEGIN . sys is balanced with its corresponding WHILE . See: BEGIN

ROLL +n -- 83 "roll"
The +nth stack value, not counting +n itself is first removed and then transferred to the top of the stack, moving the remaining values into the vacated position. (0 .. the number of elements on the return stack-1)
2 ROLL is equivalent to ROT
0 ROLL is a null operation

ROT 16b1 16b2 16b3 -- 79 "rote"
16b2 16b3 16b1
The top three stack entries are rotated, bringing the deepest to the top.

SAVE-BUFFERS -- M,79 "save-buffers"
The contents of all block buffers marked as UPDATEd are written to their corresponding mass storage blocks. All buffers are marked as no longer being modified, but may remain assigned.

SIGN n -- 83 "sign"
If n is negative, and ASCII "-" (minus sign) is appended to the pictured numeric output string. Typically used between <# and #> .

SPACE -- M,79 "space"
Display an ASCII space.

SPACES +n -- M,79 "spaces"
Display +n ASCII spaces. Nothing is displayed if +n is zero.

SPAN -- addr U,83 "span"
The address of a variable containing the count of characters actually received and stored by the last execution of EXPECT . See: EXPECT

STATE -- addr U,79 "state"
The address of a variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself is system dependent. A standard program may not modify this variable.

SWAP 16b1 16b2 -- 16b2 16b1 79 "swap"
The top two stack entries are exchanged.

THEN -- C,I,79 "then"
sys -- (compiling)
Used in the form:
flag IF ... ELSE ... THEN
or
flag IF ... THEN
THEN is the point where execution continues after ELSE , or IF when no ELSE is present. sys is balanced with its corresponding IF or ELSE . See: IF ELSE

TIB -- addr 83 "t-i-b"
The address of the text input buffer. This buffer is used to hold characters when the input stream is coming from the current input device. The minimum capacity of TIB is 80 characters.

TYPE addr +n -- M,79 "type"
+n characters are displayed from memory beginning with the character at addr and continuing through consecutive addresses. Nothing is displayed if +n is zero.

U. u -- M,79 "u-dot"
u is displayed as an unsigned number in a free-field format.

U< u1 u2 -- flag 83 "u-less-then"
flag is true if u1 is less than u2.

UM* u1 u2 -- ud 83 "u-m-times"
ud is the unsigned product of u1 times u2. All values and arithmetic are unsigned.

UM/MOD ud u1 -- u2 u3 83 "u-m-divide-mod"
u2 is the remainder and u3 is the floor of the quotient after dividing ud by the divisor u1. All values and arithmetic are unsigned. An error condition results if the divisor is zero or if the quotient lies outside the range [0..65,535].

UNTIL flag -- C,I,79 "until"
sys -- (compiling)
Used in the form:
BEGIN ... flag UNTIL
Marks the end of a BEGIN-UNTIL loop which will terminate based on flag. If flag is true, the loop is terminated. If flag is false, execution continues to just after the corresponding BEGIN . sys is balanced with its corresponding BEGIN . See: BEGIN

UPDATE -- 79 "update"
The currently valid block buffer is marked as modified. Blocks marked as modified will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block or upon execution of FLUSH or SAVE-BUFFERS .

VARIABLE -- M,79 "variable"
A defining word executed in the form:
VARIABLE <name>
A dictionary entry for <name> is created and two bytes are ALLOTted in its parameter field. This parameter field is to be used for the contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When <name> is later executed, the address of its parameter field is placed on the stack.

VOCABULARY -- M,83 "vocabulary"
A defining word executed in the form:
VOCABULARY <name>
A dictionary entry for <name> is created which specifies a new ordered list of word definitions. Subsequent execution of <name> replaces the first vocabulary in the search order with <name>. When <name> becomes the compilation vocabulary new definitions will be appended to <name>'s list. See: DEFINITIONS

WHILE flag -- C,I,79 "while"
sys1 -- sys2 (compiling)
Used in the form:
BEGIN ... flag WHILE ... REPEAT
Selects conditional execution based on flag. When flag is true, execution continues to just after the WHILE through to the REPEAT which then continues execution back to just after the BEGIN . When flag is false, execution continues to just after the REPEAT , exiting the control structure. sys1 is balanced with its corresponding BEGIN . sys2 is balanced with its corresponding REPEAT . See: BEGIN REPEAT

WORD char -- addr M,83 "word"
Generates a counted string by non-destructively accepting characters from the input stream until the delimiting character char is encountered or the input stream exhausted. Leading delimiters are ignored. The entire character string is stored in memory beginning at addr as a sequence of bytes. The string is followed by a blank which is not included in the count. The first byte of the string is the number of characters [0..255]. If the string is longer than 255 characters, the count is unspecified. If the input stream is already exhausted as WORD is called, then a zero length character string will result.

If the delimiter is not found the value of >IN is the size of the input stream. If the delimiter is found >IN is adjusted to indicate the offset to the character following the delimiter. #TIB is unmodified.

The counted string returned by WORD may reside in the "free" dictionary area at HERE or above. Note that the text interpreter may also use this area.

XOR 16b1 16b2 -- 16b3 79 "x-or"
16b3 is the bit-by-bit exclusive-or of 16b1 with 16b2.

[-- I,79 "left-bracket"
-- (compiling)
Sets interpret state. The text from the input stream is subsequently interpreted. For typical use see LITERAL. See:]

['] -- addr C,I,M,83 "bracket-tick"
-- (compiling)
Used in the from:
['] <name>
Compiles the compilation address addr of <name> as a literal. When the colon definition is later executed addr is left on the stack. An error condition exists if <name> is not found in the currently active search order. See: LITERAL

[COMPILE] -- C,I,M "bracket-compile"
-- (compiling) ,83
Used in the form:
[COMPILE] <name>
Forces compilation of the following word <name>. This allows compilation of an immediate word when it would have otherwise have been executed.

] -- 79 "right-bracket"
Sets compilation state. The text from the input stream is subsequently compiled. For typical usage see LITERAL . See: [

GLOSSARY TWO:

FORTH—83 Double Number Extension Word Set

These are the double-number (32-bit) word set extensions to FORTH-83:

2! 32b addr -- 79 "two-store"
32b is stored at addr.

2@ addr -- 32b 79 "two-fetch"
32b is the value at addr.

2CONSTANT 32b -- M,83 "two-constant"
A defining word executed in the form:
32b 2CONSTANT <name>
Creates a dictionary entry for <name> so that when <name> is later executed, 32b will be left on the stack.

2DROP 32b -- 79 "two-drop"
32b is removed from the stack.

2DUP 32b -- 32b 32b 79 "two-dupe"
Duplicate 32b.

2OVER 32b1 32b2 --
32b1 32b2 32b3 79 "two-over"
32b3 is a copy of 32b1

2ROT 32b1 32b2 32b3 -- 79 "two-rote"
32b2 32b3 32b1
The top three double numbers on the stack are rotated, bringing the third double number to the top of the stack.

2SWAP 32b1 32b2 -- 32b2 32b1 79 "two-swap"
The top two double numbers are exchanged.

2VARIABLE -- M,79 "two-variable"
A defining word executed in the form:
2VARIABLE <name>
A dictionary entry for <name> is created and four bytes are ALLOTted in its parameter field. This parameter field is to be used for the contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When <name> is later executed, the address of its parameter field is placed on the stack. See: VARIABLE

D- wd1 wd2 -- wd3 79 "d-minus"
wd3 is the result of subtracting wd2 from wd1.

D. d -- M,79 "d-dot"
The absolute value of d is displayed in a free field format. A leading negative sign is displayed if d is negative.

D.R d +n -- M,83 "d-dot-r"
d is converted using the value of BASE and then displayed right aligned in a field +n characters wide. A leading minus sign is displayed if d is negative. If the number of characters required to display d is greater then +n, an error condition exists.

D0= wd -- flag 83 "d-zero-equals"
flag is true if wd is zero.

D2/ d1 -- d2 83 "d-two-divide"
d2 is the result of d1 arithmetically shifted one bit. The sign is included in the shift and remains unchanged.

D= wd1 wd2 -- flag 83 "d-equal"
flag is true if wd1 equals wd2

DABS d -- ud 79 "d-absolute"
ud is the absolute value of d. If d is -2,147,483,648 then ud is the saved value.

DMAX d1 d2 -- d3 79 "d-max"
d3 is the greater of d1 and d2

DMIN d1 d2 -- d3 79 "d-min"
d3 is the lesser of d1 and d2

DU< ud1 ud2 -- flag 83 "d-u-less"
flag is true if ud1 is less than ud2. Both numbers are unsigned.

GLOSSARY THREE:

FORTH—83 Assembler Extension Word Set

This glossary lists the relevant FORTH-83 words that control access to the assembler mnemonics:

;CODE -- C,I,/9 "semi-colon-code"
sys1 -- sys2
Used in the form:
: <namex> ... <create> ... ;CODE ... END-CODE
Stops compilation, terminates the defining word <namex> and executes ASSEMBLER. When <namex> is executed in the form:
<namex> <name>
to define the new <name>, the execution address of <name> will contain the address of the code sequence following the ;CODE in <namex>. Execution of any <name> will cause this machine code sequence to be executed. sys1 is balanced with its corresponding : . sys2 is balanced with its corresponding END-CODE . See: CODE DOES>

ASSEMBLER -- 83 "assembler"
Execution replaces the first vocabulary in the search order with the ASSEMBLER vocabulary. See: VOCABULARY

CODE -- sys M,83 "code"
A defining word executed in the form:
CODE <name> ... END-CODE
Creates a dictionary entry for <name> to be defined by a following sequence of assembly language words. Words thus defined are called code definitions. This newly created word definition of <name> cannot be found in the dictionary until the corresponding END-CODE is successfully processed (See: END-CODE). Executes ASSEMBLER . sys is balanced with its corresponding END-CODE.

END-CODE sys -- 79 "end-code"
Terminates a code definition and allows the <name> of the corresponding code definition to be found in the dictionary. sys is balanced with its corresponding CODE or ;CODE. See: CODE

GLOSSARY FOUR:

FORTH—83 System Extension Word Set

This glossary lists those words which define the standard functions needed during compilation. The average user will not need these.

<MARK -- addr C,83
Used at the destination of a backward branch. addr is typically only used by <RESOLVE to compile a branch address

<RESOLVE addr -- C,83
Used at the source of a backward branch after either BRANCH or ?BRANCH . Compiles a branch address using addr as the destination address.

>MARK -- addr C,83
Used at the source of a forward branch. Typically used after either BRANCH or ?BRANCH . Compiles space in the dictionary for a branch address which will later be resolved by >RESOLVE.

>RESOLVE addr -- C,83
Used at the destination of a forward branch. Calculates the branch address (to the current location in the dictionary) using addr and places this branch address into the space left by >MARK .

?BRANCH flag -- C,83
When used in the form: COMPILE ?BRANCH a conditional branch operation is compiled. See BRANCH for further details. When executed, if the flag is false the branch is performed as with BRANCH . When flag is true execution continues at the compilation address immediately following the branch address.

BRANCH -- C,83
When used in the form: COMPILE BRANCH an unconditional branch operation is compiled. A branch address must be compiled immediately following this compilation address. The branch address is typically generated by following BRANCH with <RESOLVE or >MARK .

CONTEXT -- addr U,79
The address of a variable which determines the dictionary search order

CURRENT -- addr U,79
The address of a variable specifying the vocabulary in which new word definitions are appended.

GLOSSARY FIVE:

FORTH—83 Controlled Reference Words

This glossary lists extra FORTH-83 words which are not neccessary for FORTH-83, yet are present and defined in Computer One FORTH. The use of these words is however controlled by the FORTH-83 standard.

;-> -- I,M,79 "next-block"
-- (compiling)
Continue interpretion on the next sequential block. May be used within a colon definition that crosses a block boundary.

.R n +n -- M,83 "dot-r"
n is converted using BASE and then displayed right aligned in a field +n characters wide. A leading minus sign is displayed if n is negative. If the number of characters required to display n is greater than +n, an error condition exists.

BL -- 32 79 "b-l"
Leave the ASCII character value for space (decimal 32).

BLANK addr u -- 83 "blank"
u bytes of memory beginning at addr are set to the ASCII character value for space. No action is taken if u is zero.

C, 16b -- 83 "c-comma"
ALLOT one byte then store the least-significant 8 bits of 16b at HERE 1-

DUMP addr u -- M,79 "dump"
List the contents of u addresses starting at addr. Each line of values may be preceded by the address of the first value.

EDITOR -- 83 "editor"
Execution replaces the first vocabulary in the search order with the EDITOR vocabulary. See: VOCABULARY

EMPTY-BUFFERS -- M,79 "empty-buffers"
Deassign all block buffers. UPDATEed blocks are not written to mass storage. See: BLOCK

END flag -- C,I,79 "end"
sys --
A synonym for UNTIL .

ERASE addr u -- (compiling) 79 "erase"
u bytes of memory beginning at addr are set to zero. No action is taken if u is zero.

HEX -- 79 "hex"
Sets the numeric input-output conversion base to sixteen.

INTERPRET -- M,83 "interpret"
Begin text interpretation at the character indexed by the contents of >IN relative to the block number contained in BLK, continuing until the input stream is exhausted. If BLK contains zero, interpret characters from the text input buffer.

LIST u -- M,79 "list"
The contents of screen u are displayed. SCR is set to u.
See: BLOCK

OCTAL -- 83 "octal"
Set the numeric input-output base to eight.

OFFSET -- addr U,83 "offset"
The address of a variable that contains the offset added to the block number on the stack by BLOCK or BUFFER to obtain the actual physical block number.

QUERY -- M,83 "query"
Characters are received and transferred into memory area addressed by TIB . The transfer terminates when either a "return" is received to the number of characters transferred reaches the size of the area addressed by TIB . The values of >IN and BLK are set to zero and the value of #TIB is set to the value of SPAN . WORD may be used to accept text from this buffer. See: EXPECT

SCR -- addr U,79 "s-c-r"
The address of a variable containing the number of the screen most recently LISTed.

SP@ -- addr 79 "s-p-fetch"
addr is the address of the top of the stack just before SP@ was executed.

U.R u +n -- M,83 "u-dot-r"
u is converted using the value of BASE and then displayed as an unsigned number right aligned in a field +n characters wide. If the number of characters required to display u is greater than +n, an error condition exists.

GLOSSARY SIX:

FORTH—83 Uncontrolled Reference Words

This glossary list the words included in Computer One FORTH, and whoose meanings are recommended (but not controlled) by the FORTH-83 standard.

+BLOCK w -- u "plus-block"
u is the sum of w plus the number of the block being interpreted.

;S -- "semi-s"
Stop interpretion of a block.

<> w1 w2 -- flag "not-equal"
flag is true if w1 is not equal to w2.

>< 16b1 -- 16b2 "bytes-swap"
Swap the high and low bytes within 16b1

AGAIN -- C,I "again"
sys -- (compiling)
Effect an unconditional jump back to the start of a BEGIN-AGAIN loop. sys is balanced with its corresponding BEGIN. See: BEGIN

ASCII -- char I,M "as-key"
-- (compiling)
Used in the form:
ASCII ccc
where the delimiter of ccc is a space. char is the ASCII character value of the first character in ccc. If interpreting, char is left on the stack. If compiling, compile char as a literal so that when the colon definition is later executed, char is left on the stack.

ASHIFT 16b1 n -- 16b2 "a-shift"
Shift the value 16b1 arithmetically n bits left if n is positive, shifting zeros into the least-significant bit positions. If n is negative, 16b1 is shifted right; the sign is included in the shift and remains unchanged.

B/BUF -- 1024 "bytes-per-buffer"
A constant leaving 1024, the number of bytes per block buff

BELL -- M "bell"
Activate a terminal bell or noise-maker as appropriate to the device in use.

FLD -- addr U "f-l-d"
A variable pointing to the field length reserved for a number during output conversion.

INDEX u1 u2 -- M "index"
Print the first line of the each screen over the range [u1..u2]. This displays the first line of each screen of source text, which conventionally contains a title.

LAST -- addr U "last"
A variable containing the address of the beginning of the last dictionary entry made, which may not yet be a complete or valid entry.

MOVE addr1 addr2 u -- "move"
The u bytes at address addr1 are moved to addr2. The data are moved in such that the u bytes remaining at address addr2 at the same data as was originally at addr1. If u is zero nothing is moved.

NUMBER addr -- d "number"
Convert the count and character string at addr, to a signed 32-bit integer, using the value of BASE . If numeric conversion is not possible, an error condiction exists. The string may contain a preceding minus sign.

S0 -- addr U "s-zero"
A variable containing the address of the bottom of the stack.

SHIFT 16b1 n -- 16b2 "shift"
Logical shift 16b1 left n bits if n is positive, right n bits if n is negative. Zeros are shifted into vacated bit positions.

USER +n M "user"
A defining word executed in the form:
+n USER <name>
which creates a user variable <name>. +n is the offset within the user area where the value for <name> is stored. Execution of <name> leaves its absolute user area storage address.

WORDS -- F M "words"
List the word names in the firstvocabulary of the currently active search order.

GLOSSARY SEVEN:

FORTH–83 QL Other Definitions

These are miscellaneous words present in Computer One FORTH.

!L 16b d -- "store-l"
16b is stored at d. See: !

-2 -1 0 1 2 3 -- n
These small numbers are defined as constants as they are used so often.

.CPU -- "dot-c-p-u"
Prints the processor name (i.e. 68008) from ORGIN+22H encoded as 32 bit, base 36 integer.

.FCB addr -- "dot-f-c-b"
addr is the address of the file-control-block. Displays fully qualified file name on the selected output device.

3DROP 16b1 16b2 16b3 -- "three-drop"
16b1 16b2 16b3 are removed from the stack.

4DROP 32b1 32b2 -- "four-drop"
32b1 32b1 are removed from the stack.

>BODY addr1 -- addr2 "to-body"
addr2 is the parameter field address corresponding to the compilation address addr1.

>LINK addr1 -- addr2 "to-link"
addr2 is the name field address corresponding to the compilation address addr1.

>NAME addr1 -- addr2 "to-name"
addr2 is the name field address corresponding to the compilation address addl.

?DO w1 w2 -- "query-do"
-- sys (compiling)
Used in the form:
?DO ... LOOP
or
?DO ... +LOOP
Begins a loop which terminates on control parameters. The loop begins at w2, and terminates based on the limit w1. If w1 equals w2 the loop is not executed. See: DO

?TERMINAL -- flag "query-terminal"
Performs a test of the keyboard for actuation of any key. A true flag indicates actuation, and will remain true on any number of calls until KEY in invoked to read the character.

@L d -- 16b "fetch-l"
16b is the value at d. See: @

ASM "load assembler"
Loads the FORTH assembler. Use ASSEMBLE to switch to assembler vocabulary.

BINARY -- "binary"
Set the numeric input-output conversion base to two.

BODY> addr1 -- addr2 "from-body"
addr2 is the compilation address corresponding to the parameter field address addr1.

BYE "bye"
Exit FORTH and renter QL BASIC.

C!L 16b d -- "c-store-l"
The least-significant 8 bits of 16b are stored into the byte at d. See: C!

C@L d -- 8b "c-fetch-l"
8b is the contents of the byte at d. See: C@

CONSOLE -- "console"
Make the system keyboard and VDU the principal ouput device. See: PRINTER

CMOVEL d1 d2 n -- "c-move-l"
Works like CMOVE , but allows string moves anywhere in the 68008 addressing space.

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

DLITERAL d -- d "d-literal"
d -- (compiling)
If compiling, compile a double number into a literal. When later executed the double number d will be left on the stack.

EDIT "edit"
Invokes the FORTH screen editor.

L>NAME addr1 -- addr2 "link-to-name"
addr2 is the name field address corresponding to thr link field address addr1.

LINK> addr1 -- addr2 "from-link"
addr2 is the compilation address corresponding to the link field address addr1.

N>LINK addr1 -- addr2 "name-to-link"
addr2 is the link field address corresponding to the name field address addr1.

NAME> addr1 -- addr2 "from-link"
addr2 is the compilation address corresponding to the link field address addr1.

NOOP -- "no-op"
A FORTH null word.

PRINTER -- "printer"
Make the line printer the principal output device. See: CONSOLE

RP! -- "r-p-store"
Initialize the return stack pointer from the user variable R0.

RP@ -- addr "r-p-fetch"
addr is the current value of the return stack pointer register on the parameter stack.

SCOPY "screen copy"
Copies a selected number of screens from the current screen file to another screen file. Scopy prompts for the destination screen file and for the screens to be copied. It gives the option of creating a new screen file if the specified one does not exist.

SP! -- "s-p-store"
Initialize the stack pointer register from S0 .

SP@ -- addr "s-p-fetch"
addr is the address of the top of the stack before SP@ was executed.

STRING, addr n -- "string-comma"
ALLOT n bytes and store the the string starting at addr at HERE -n.

TASK -- "task"
A non-operation word which can mark the boundary between applications. By forgeting TASK and re-compiling, an application can be discarded in its entirety.

TRAVERSE addr1 n -- addr2 "traverse"
Move across the name field of a FORTH-83 variable length name field. addr1 is the address of either the length byte or the first letter. If n=1, the motion is towards high memory, if n=-1, the motion is towards low memory. The addr2 resulting is the address of the other end of the name.

WARM "warm"
Clear the stacks and enter execution state. Return control to the keyboard.

GLOSSARY EIGHT:

FORTH—83 QL Graphics, Sound and Floating Point Word Set

All the graphics words may be assumed to work on the current WORK window. A window is created by QDOS as a device which has to be opened. First create a file control block using FCB - then give it a name using FILENAME - then open it using OPEN-FILE, e.g. FCB FRED FRED FILENAME CON_448X180A32X16_128 FRED OPEN-FILE .

OPEN-FILE returns an error code (0 for success). The error codes are those normally returned by QDOS. For more details of opening and closing files, and other device-related words, see the section on microdrives and devices. Remember, QDOS treats all input and output in the same way.

To switch between windows use the words WORK andIS-WORK. These words set and return the channel-id codes for the window used by the graphics commands. Use the words OUTPUT and IS-OUTPUT for switching the output window and use the words INPUT and IS-INPUT for switching the input channel. NOTE that default channel-id for all three channels is zero.

FLOATING POINT WORDS

Analaloques of the normal forth stack and maths operations are provided for use with floating point numbers. These words are:

F+ F- F* F/ FABS FNEG FDUP FROT FDROP FSWAP

The following other floating point functions are also available:

FSIN FCOS FTAN FCOT FASIN FACOS FATAN FACOT FSQRT FLN FLOG FEXP F**.

Conversion to and from is achieved using TRUNC ROUND and FLOAT.

TRUNC fn -- n
Truncates a floating point number to an integer.

ROUND fn -- n
Converts a floating point number to the nearest integer.

FLOAT n -- fn
Converts an integer to floating point.

F# (--- [compiling] ; ---fn [interpreting])
Used in the form
F# 3.456
This word generates a floating point number which is left on the stack or compiled depending on whether FORTH is interpreting or compiling.

F#IN (--- fn true / false)
Used to take a number from the input and attempt to convert it to a floating point number. If conversion fails, no number is returned.

F. (fn ---)
Prints fn in the same format as used by SuperBASIC.

GRAPHICS AND SOUND WORDS

NOTE: The FORTH graphics words CIRCLE, CURSOR, ELLIPSE, LINE and POINT all require floating point parameters. (See Section 2.7)

ADATE n --
Set the clock with the value n seconds.

ADJUST-CLOCK n --
Adjust the clock by n seconds.

BEEP-TABLE -- addr
returns the address of an eight word table containing the parameters for BEEP.

BEEP --
Uses the parameters in BEEP-TABLE to make a noise.

BEEPER duration pitch --
Sets the parameters into BEEP-TABLE and calls BEEP.

FILL-BLOCK width height x y colour --
Draw and fill a block of size width*height at position x,y relative to the origin of the window attached to the default channel.

BORDER width colour --
Redefine the border of a window.

CHANNEL fcb -- dl
Returns the channel-id of the channel associated with the given fcb.

CIRCLE fx fy fradius --
Draw a circle, centre at point x y .

CSIZE **width height --**
Set a new character print size for use in the default channel. See: QL manual keywords, CSIZE; or use the predefined words NARROW, WIDE, EXPANDED, SHORT and TALL e.g:-
WIDE EXPANDED TALL CSIZE

CURSOR **fx fy fxrel fyrel --**
Position screen cursor at position (xrel,yrel) relative to the graphics cursor position (x,y). Note that xrel and yrel are specified in the pixel co-ordingate system and x and y in the graphics coordinate system.

CURS-ON **--**
turn cursor on

CURS-OFF **--**
turn cursor off

DATE **-- date**
returns the setting of the QL clock as a 32-bit number of seconds.

ELLIPSE **fx fy fradius feccentricity fangle --**
Draws an ellipse centre x,y with the ratio between the major and minor axis being the eccentricity, angle is the rotational angle relative to the screen vertical.

FLASH **flag --**
Low resolution mode only. When flag is true the flash will be on.

GET-XY **-- x y**
Returns the current cursor character position

INK **colour --**
Sets the current ink colour. The standard colours are predefined words. Stipple colours may be generated using WITH and the stipple words 1-IN-4, H-STRIPES, V-STRIPES, and CHECKERS, e.g.;-
RED INK
RED BLUE WITH CHECKERS INK

INPUT **-- dl**
Returns the channel-id of the current input channel

IS-INPUT **dl --**
Sets the channel-id of the current input channel

IS-OUTPUT **dl --**
Sets the channel-id of the current output window.

IS-WORK **dl --**
Sets the channel-id of the current work window

LINE **fx1 fy1 fx2 fy2 --**
Draw a line from x1 y1 to x2 y2 .

OUTPUT **-- dl**
Returns the channel-id of the current output window.

OVERPRINT **n --**
Sets the type of overprinting required.
n = -1 print in ink over previous contents of screen
n = 0 print ink on strip
n = 1 print ink on transparent strip

PAN **n part --**
Move a window n pixels to the right or left.

PAPER **colour --**
Sets new paper colour, see INK.

POINT **fx fy --**
Plot a point at position x y .

RECOL **c0 c1 c2 c3 c4 c5 c6 c7 --**
Recolours individual pixels on the screen according to the parameters, for more details see the QL SuperBASIC manual.

SDATE **dl --**
SDATE allows the QL's clock to be reset to the number of seconds given.

SCROLL **n --**
Scroll the window attached to the default channel up or down. See: PAN

SET-MODE **dl --**
Sets the mode resolution to 4 colour or 8 colour. The double word put on the stack is 4 or 8.

SET-XY **x y --**
Sets the current character position.

STRIP **colour --**
Sets the current strip colour of the window attached to the default channel to colour. See INK.

UNDERSCORE **flag --**
Turns underline on for subsequent character outputs.

W-DATA **-- addr**
addr is the address of a four-word table containing the window's width, height, cursor x-position, cursor y-position. This table may be read using WSIZE-C or WSIZE-P

WINDOW **w-addr --**
Allows the user to change the position and size of a window. Any borders are removed if this window is redefined.

WORK **-- dl**
returns the channel-id of the current work window.

APPENDIX 1

Error Messages

FORTH error messages are stored in text form on screens 4 and 5 of systems that have discs, or other fast mass storage. On this version, only error numbers are given. Assembler error messages are detailed in the assembler section of the manual.

MESSAGE NUMBER	MESSAGE/ DESCRIPTION
1	**EMPTY STACK** The parameter stack is empty.
2	**DICTIONARY FULL** The dictionary space is exhausted.
3	**HAS INCORRECT ADDRESS MODE**
4	**IS NOT UNIQUE** The word defined already exists. This is a warning only.
5	**IS UNDEFINED** The word name being looked for cannot be found
6	**DISC RANGE ?** The disc block address is outside the range of the micro drive files.
7	**FULL STACK** The parameter stack space is exhausted.
8	**DISC ERROR** There is a microdrive error.
13	**BASE MUST BE DECIMAL** The current value of base is not decimal
14	**MISSING DECIMAL POINT** Floating point conversion failed to find a decimal point in the mantissa.
17	**COMPILATION ONLY, USE IN DEFINITION** A word which can be used only inside a colon definition has been used outsidea colon definition.
18	**EXECUTION ONLY** A colon has been used inside of a colon definition.
19	**CONDITIONALS NOT PAIRED** Conditional words are not paired or nested correctly.
20	**DEFINITION NOT FINISHED** A definition has been terminated by a semicolon before it has been completed.

21	**IN PROTECTED DICTIONARY** An attempt has been made to FORGET below FENCE.
22	**USE ONLY WHEN LOADING** The operation --> was executed from the keyboard, not from a screen being loaded from disc.
23	**OFF CURRENT EDITING SCREEN** An attempt has been made to edit a screen outside of the screen's bounds.
24	**DECLARE VOCABULARY** An attempt to FORGET has been made when the CONTEXT and CURRENT vocabularies are not the same.
27	**ILLEGAL FLOATING POINT FORMAT** Forth attempted to do a floating point operation and could not use the string supplied
28	**ILLEGAL DIMENSION IN ARRAY DEFINITION** Issued when an array is declared, this error message indicates an impossible (negative) dimension or that there is not enough memory for the array.
29	**NEGATIVE ARRAY INDEX** The indexing value of an array is negative.
30	**ARRAY INDEX TOO LARGE** A run-time access to an array attempted to index outside the declared size of the array.
33	**INCORRECT SOURCE ADDRESSING MODE** Assembler error message.
34	**INCORRECT DESTINATION ADDRESSING MODE** Assembler error message.
35	**OUT OF RANGE -128 <= N <= 127** Assembler error message.
36	**OUT OF RANGE 0 <= N <= 15** Assembler error message.
37	**OUT OF RANGE 1 <= N <= 8** Assembler error message.
40	**BUS ERROR** Privileged or non-existent memory.
41	**ADDRESS ERROR** Memory operation on odd address.
42	**UNRECOGNIZED OPCODE** Probable bad return stack.
43	**DIVIDE BY ZERO**

APPENDIX 2

Bibliography and References:

As yet, only FORTH Tools, Vol 1 is dedicated to FORTH-83. Despite this, the techniques described in these books are still of value as the major differences between the implementations are seen by the implementor and when using advanced techniques. The inexperienced FORTH programmer should still find these books useful.

FORTH 83 Standard.
FORTH Interest Group, PO Box 1105, San Carlos, CA 94070.

STARTING FORTH, Leo Brodie, Prentice Hall/Forth Inc.
This book is still the best introduction to FORTH, and is likely to be a classic.

FORTH PROGRAMMING, Scanlon
An introduction to FORTH with good chapters on string and disk handling, subjects often ignored by other books.

THE COMPLETE FORTH, Alan Winfield
A popular introductory book based on systems using FORTH-79

FORTH DIMENSIONS (Newsletter). FORTH Interest Group, PO Box 1105, San Carlos, CA 94070.
The American FORTH user group. The original FORTH magazine and still the best. The quality of some of the articles is quite outstanding. Leo Brodie presented a complete text editor and word processor in this magazine. They take credit cards and back issues are available.

SYSTEMS GUIDE TO FIG-FORTH, Ting, Offete Enterprises
This book is the implementor's bible. It describes the whys and hows of the internal workings of FORTH.

BYTE magazine.
The August 1980 issue of BYTE was devoted to FORTH and contains many useful articles.

Dr. DOBBS JOURNAL
Dr. Dobbs has become a serious and respected software magazine. It is usually a treasure chest of goodies for people interested in serious programming. The September issue each year is usually devoted to FORTH.

Computer ONE — Software Problem Report (FORTH)

Name .....
Address .....
.....
.....

Return to :

Computer One Ltd.,
Science Park,
Milton Road,
Cambridge CB4 4BH.

Telephone Number :

Nature of Problem (tick): Documentation error[ ] Software error[ ]

Operating System Version ..... (type 'print ver$')

Software Error : Please describe problem in as much detail as possible, giving the keystroke sequence which caused the error. (enclose listing if possible) -

Documentation Error : Please include page number in error description

Comments or Enquiries :

Detach and return sheet to Computer One at above address