1616: Forth Manual

Version 0.107
March, 1999

Applix 1616 microcomputer project
Applix pty ltd

1616 Forth Manual

Even though Applix has tested the software and reviewed the documentation, Applix makes no warranty or representation, either express or implied, with respect to software, its quality, performance, merchantability, or fitness for a particular purpose. As a result this software is sold "as is," and you the purchaser are assuming the entire risk as to its quality and performance.

In no event will Applix be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or its documentation.

The original version of this manual was written by Peter Fletcher
Additional introductory and tutorial material by Eric Lindsay

Comments about this manual or the software it describes should be sent to:

Applix Pty Limited
324 King Georges Road
Beverly Hills 2209
N.S.W. Australia
(02) 758 2688

ISBN 0 947341 ?? ?

1
Getting started with FORTH

To start up FORTH from the `/bin` directory, type (ignoring the prompt):

```
f0/bin>ff
```

This is a loader program which will allocate memory for FORTH, load it in and execute it.

FORTH will clear the screen and print a startup message. If it doesn't, check that you've got the disk in the correct drive and you're in the correct directory.

Now that FORTH has been loaded in, we can start to experiment.

Forth is typically somewhat uncommunicative in its responses. All you will see on the screen after most commands is the interactive Forth prompt, which is **>**. Type in a few numbers, leaving a space between each. Forth accepts a space as its input delimiter, so spaces are **very** important. Everything separated by spaces is either a number, or a Forth 'word' (a subroutine).

```
>4 5 6 7Enter
>
```

Not very exciting, is it? All that happened is that the numbers were put on the 'parameter stack' (which is an area of memory pointed to by register A6, for those of you into assembler). In fact, this is all Forth ever does with any number that you enter. This simple concept helps make Forth relatively easy to implement. By the way, in future, we won't show the Enter key at the end of each line, but don't you forget it.

To get rid of these four entries, type (don't forget, the > you see is merely the prompt, you don't type that).

```
>drop drop drop drop
```

Congratulations, you just entered your first Forth 'word'. As we mentioned, each Forth word is the equivalent of an assembler subroutine. In this case, it is simply `addq #4,a6`, which just moves the parameter stack pointer so that it ignores the top 32 bit number on the stack (not that you really need to know why it works yet).

The Stack.

All operations FORTH performs rely upon its 'parameter stack'. This is a simple last-in-first-out stack which stores 32 bit values. Let's give it a try. Try typing:

```
>1 2 3
```

When you press return, nothing happens. However, you have put three values on the parameter stack. You can print them one at a time, if you type '.' (spoken as 'dot') and press return:

```
>.
 3
>.
 2
>.
 1
```

(Why do they come out backwards ?)

‘.’ is a FORTH ‘word’. All FORTH programs (words) are built out of smaller words (programs). We’ll see how to define them later.

The stack is now empty - what happens if you try to print one more value?

```
>.
 [some number]
?Stack underflow.
>
```

You’ll probably see this message a lot when you’re trying things out. Unfortunately, FORTH doesn’t do too much of this sort of checking when it’s running, so if you’re careless you can crash the machine very easily. Don’t worry - you can always reboot everything after such crashes, and any substantial work you’ve done should be in a file anyway.

Arithmetic.

Now let’s do some real computing; let’s add two numbers:

```
>1 5 + .
 6
>
```

Now we’re really getting places. The strange way FORTH does arithmetic is called ‘reverse polish notation’. You may recognize it from Hewlett Packard calculators.

Now let us try a more complex expression: 5*(3+7)*(3*(2+3))

FORTH doesn’t do arithmetic this way, so we’ll have to translate it into RPN. It takes a while to get used to, but it’s not hard when you’ve written a few programs this way.

```
>5 3 7 + * 3 2 3 + * * .
 750
>
```

There! It’s easy, isn’t it?

Simple Programs.

Now that we’ve mastered arithmetic (you could try ‘mod’, ‘/’, ‘-’ too), let’s start writing programs.

There’s a special FORTH word for defining more words; it’s the colon: ‘:’. Let’s try it. Remember to type all the spaces:

```
>: first_word
>1000000 * .
>
```

(Remember to type all the spaces)

Hey! Why didn't FORTH print something when you entered the dot? Because the ': first_word' made FORTH enter 'compile mode', where any words that you enter are 'compiled' instead of being executed. So, the million, the times and the dot have been compiled away somewhere. How do we tell FORTH that our word does everything we want it to do and finish the definition? We use another FORTH word, semicolon: ';'.

```
>;
>0 .
 0
>
```

So, we're back to normal (by the way, if ';' is just another FORTH word, why doesn't FORTH just compile it, too???). Lets try executing our newly defined word:

```
>42 first_word
 42000000
>
```

Our very first word multiplies numbers by a million! Nifty, huh?

More stack words.

After all that excitement, it's time to get back to the parameter stack. Try this:

```
>1 2 3 4 5
>.s
 1 2 3 4 5
>
```

'.s' is a word which prints out the parameter stack, bottom element first. It does not affect the stack in any way, so it's useful to see what's on the stack. Now try these new words:

```
>abort
>.s
```

'abort' seems to clear the parameter stack (but it does a lot more- see later).

```
>42 dup .s
 42 42
>32 swap
 42 32 42
>over .s
 42 32 42 32
>+ .s
 42 32 74
>rot .s
 32 74 42
>drop drop .
 32
```

Try and work out what nip, tuck, rot-, 2dup, pick and roll do.

We’ve now learnt the main stack words. A complete list is:

```
dup swap over rot rot- drop
2dup 2swap 2drop
nip tuck
roll pick
```

Now you’ve got a start, look in the file `kern.txt`, where all the kernel words are explained. An incomplete list is given here.

Arithmetic Words

```
+ - * / mod ~ abs min max
*q /q
1+ 2+ 3+ 4+ 5+ 6+ 7+ 8+
1- 2- 3- 4- 5- 6- 7- 8-
2* 4* 8* 16*
2/ 4/ 8/ 16/
base hex decimal
```

Printing words.

```
.  .s  ."  .str type
```

Input/Output words.

```
expect dexpect pad
fopen fclose
fget fput fgetc fputc
```

Useful words.

```
load save words
trace+ trace-
debug+ debug-
echo+ echo-
altc+ altc-
edit delete dir ]
load? forget? vocab
```

Variables etc.

```
constant variable automatic struct ends string
array arraym
larray larraym uses
dconst dauto
@  !  +!  to
w@ c@ d@ w@x
w! c! d!
addr.of size.of
locals|
base0 base1
chk+ chk-
```

Structured Statements

```
do .. loop i j
if .. then
if .. else .. then
begin .. again
begin .. until
begin .. while .. repeat
```

```
case
  .. :- .. |
  .. :- .. |
  .. => ..
endcase
```

```
dropcase dropdo case@
return
label: goto
```

Dictionary.

```
find search dsearch vdel forget dict
allot literal , ,l next@ next! ’
compile code
```

Return stack.

```
>r <r r1 r2 r3 r4 dropr
mlink
```

Memory management.

```
link link@ unlink allocmem freemem allot
mem
```

Structured and unstructured statements.

FORTH allows all the structures of most common languages, yet it takes a little while to get used to the strange order in which words are specified.

do/loop

One of the most common operations in any language is looping. FORTH has several ways of looping, and the most often used is the ‘do/loop’ loop, which is very similar to the for/next loop in BASIC. To see how fast FORTH can do othing, try the following:

```
>: do1000000
>  1000000 1 do loop ;
>do1000000
```

It’s pretty fast! However, it’s not often we need to do nothing, so lets try something more useful: we’ll read in ten numbers and print their sum.

```
>: add10
>  0
>  10 1 do
>          i +
>  loop ." The sum of the first ten numbers is:" . ;
>add10
The sum of the first ten numbers is: 55
>
```

(Try entering this word with tracing enabled)

The loop indexes must be specified backwards (largest number first). If you want to loop down instead of up, or want a step other than one, use ‘+loop’ or ‘-loop’.

if/else/then

To make decisions with FORTH, there is an if/then statement. Remember that everything is backwards with FORTH: you must do everything in the right order:

{condition} if {condition true statements}
  else {condition false statements}
  then
{etc.}

The ‘else’ part may be left out.

```
>: doif
>  cget asc a = if
>          ." An ’a’ was pressed"
>  else
>          ." An ’a’ was not pressed"
>  then ;
```

begin/while/until/again/repeat

The word ‘begin’ is used in combination with while, until, repeat and again. However, these four words may not be mixed and matched indiscriminately; the allowable combinations are:

begin {body1} {condition} while {body2} repeat
begin {body} again
begin {body} until

To show what these words do, here are a few examples. Be careful with the first one; it goes for some time! The stuff between (brackets) is comments.

```
: star_forever ( print lots of stars )
  begin asc * emit again ;

: star_key ( print stars ’til key pressed )
  begin asc * emit c? until ;

: keys_prt ( print keypresses while altc not pressed )
  begin cget
          dup 0>= while ( altc? )
          emit
  repeat ;
```

case/:-/|/default/endcase.

The case statement is useful when many choices have to be made from one piece of data.

```
: case1
  cget
  case
          asc 1 :- ." one"        |
          asc 2 :- ." two"        |
          asc 3 :- ." three"      |
          asc 4 :- ." four"       |
          asc 5 :- ." five"       |
          asc 6 :- ." six"        |
          asc 7 :- ." seven"      |
          asc 8 :- ." eight"      |
          asc 9 :- ." nine"       |
          asc 0 :- ." zero"       |
      default :- ." Not a digit."

  endcase ;
```

Note that the last choice does not have a terminating ‘|’. If the default case is left out and the choice does not appear in the list, the ‘case’ statement does nothing.

If you need to get out of a case statement, you need to use ‘dropcase’ along with ‘return’ or ‘goto’.

The case statement is not smart; it simply checks the conditions in the order that they are entered. However, this makes it possible to simply check a flag in a case statement; see ‘=>’.

goto/label:/return

If you need to do something out of the ordinary, such as exiting upon an error condition, ‘goto’, ‘label’ or ‘return’ may be used. They need some care in their use, because many words temporarily store information on the return stack, which is where FORTH stores the return address to the word that called the word it is currently executing; if you write a ‘goto’ or ‘return’ that does not take care of this extra stuff on the return stack, the poor 68000 will attempt to return to an address given by a piece of data, which is not likely to do good things.

```
: spaghetti
  1 .
  goto fred

  ( this code is never executed )
  9999 .

label:    fred
  2 .
  goto count

label:    mary
  i .
  i 5 = if dropdo crlf ." Done." return then
  goto jill

label:  count
  5 3 do
          goto mary

label:    jill
  loop
( this code is never executed either )
  9999 . ;
```

As you can see, goto should be used rarely (if at all). If for some reason a label is never specified for a ‘goto’, the goto statement will do nothing. There is nothing to stop you ‘goto’ing between words, but this is asking for trouble.

Writing programs with the editor.

Let’s now try writing a program to print a times table. When writing any programs larger than a few lines, it’s better to edit them in a file so you can change them easily (it also means you can load them again later on). If you’re lazy, you could peek ahead and load the file ‘demo1.f’.

Try the following:

```
>edit forth1.f
```

You should now be in the editor. If you don’t want to write on your FORTH disk, you should put in a blank formatted disk at this stage. Now try entering the following program:

```
forget? forth1

vocab forth1
```

```
: timestab            ( --  | print a five times table )

( ----- print start message )
  crlf
  ." This is a times table for numbers from 1 to 5." crlf
  crlf

( ----- print a header )
  tab rvson
  5 1 do
          i . tab
  loop rvsoff crlf

( ----- print the table )
  5 1 do
  rvson i . tab rvsoff            ( ----- print first number )
          5 1 do
                      i j * . tab( ----- print table )
          loop
          crlf                    ( ----- start a new line )
  loop ;

interactive
```

What you’ve just entered needs a little explaining. The first line, ‘forget? forth1’, tells FORTH that a new vocabulary is being loaded, so if there is already anything by the name of ‘forth1’, it would be best forgotten. This allows the same file to be loaded lots of times in between edits, so there are no problems with words being redefined.

The next line, ‘vocab forth1’, tells FORTH that a new vocabulary called ‘forth1’ is being started in the dictionary.

The next lines form a colon definition for a word to print a times table.

The last line, ‘interactive’, tells forth that the current file it’s reading has come to an end, so it should close the file and return to the user (or the previous file it was reading).

In the actual definition of ‘timestab’, several things should be noted. On the line the definition opens, ‘(-- | etc’ appears. This is simply a comment, but it serves an essential purpose. Firstly, it explains what the word does. Just as importantly, however, it shows what the word expects on the stack before it executes, and shows what the word will leave. Every word you write should have such a comment.

The definition is spread liberally with comments, which for a word such as this are probably not necessary. However, FORTH code is reknowned for its impenetrability, so a few comments here and there cannot hurt. Unlike most languages, comment brackets may be nested successfully.

Now, exit the editor and you will be returned to the FORTH prompt. Load the file you’ve just entered with:

```
>load forth1
```

If any error messages occur, or things don’t seem to work any more, try the following:

```
>abort    ( just in case )
>;        ( in case you’ve left off the semicolon )
> ) ) )  ( in case you’ve left some comments open )
>system ( if things are really bad )
reset button! (FORTH is not extra forgiving)
```

You can see the new vocabulary in place by typing:

```
>words
forth1
sys
kern2
kern1
>words forth1
timestab {perhaps more words}
>
```

‘words’ allows you to list all vocabularies, or, if a vocabulary is specified, all words in a vocabulary. Try the following:

```
>words sys
>words kern2 i
>words *
```

Now you can try executing the new word:

```
>timestab
```

A beautiful times table should appear.

Saving programs to disk.

Rather than loading words from disk all the time, you may wish to save your current FORTH environment to disk. Just by typing

```
>save
```

you will have a copy of your current FORTH environment on disk, ready to be reloaded (Be careful: it will be given the name ‘forth’ unless you change the values if the string variables ‘name’ and ‘vname’.

If you want to create turnkeyed applications, look up ‘turnkey’ and ‘word1’.

Using Variables.

Constants.

FORTH has several types of variables. The simplest is not a variable at all. Try this:

```
>1000000 constant 1million
>1million .
 1000000
```

A constant can be used whenever you would enter a number. Constants are convenient, but they are slightly more expensive than entering straight numbers in terms of both speed and storage.

Variables.

The standard FORTH variable can be created with the FORTH word ‘variable’:

```
>variable a
>a .
24738
```

This number (which may be different when you try it) is the address of variable ‘a’. To access the contents of ‘a’, you need to use two words to access a’s contents.

The first of these, ‘!’ (’store’), stores a value in ‘a’. The second, ‘@’ (’get’ or ‘load’) retrieves a’s contents.

```
>56 a !
>a dup @ . .
 56 24738
```

‘@’ and ‘!’ are not used just for variables; they can be used for arrays, structures or anything else you put them to use on. These words always load four bytes (a longword) from the address you give to them, so if you accidentally pass them garbage you’ve got a good chance of causing an address error exception, which is very time consuming. Use them with a bit of care, and they’ll give you no trouble.

When using arrays and structures, allocating four bytes to each element is very expensive on memory usage. FORTH allows bytes and words to be loaded and stored just as easily, using the words ‘w@’, ‘c@’, ‘w!’ and ‘c!’.

Automatics.

Automatic variables, although not standard FORTH, are faster, smaller and easier to learn than standard variables. They also use the same syntax as local variables, which are explained later.

```
>automatic z
>zz .
9999
```

Automatic variables do not need the ‘@’ word; they leave their value on the stack as soon as they are executed. Some value had to be stored in z when it was initialized, and so 9999 was used because it was not zero. How do we store values in automatics ?

```
>-57 to z
>z .
 -57
```

Local variables & recursion.

Another sort of FORTH variable is the ‘local variable’. Local variables are a little trickier to use than the other sorts, but they have many advantages. They are faster again than automatic or standard variables, and allow FORTH to execute truly recursive algorithms. No storage space is allocated to local variables in either the dictionary or the code area, so they keep your vocabularies compact and neat. However, local variables are only ‘alive’ in the word in which they are defined, so their use is only local to the word in which they are defined.

Try this:

```
: fibonacci           ( n -- fib(n) | calculate fibonacci
number )
  locals| fibo |    ( get (n) from the stack )

( ----- zero or negative fibonacci number is a nono! )
fibo 0<= if
          " Cannot get fibonacci of negative or zero number!"
          error
then

( ----- fib(1) or fib(2) equals 1 )
1 fibo =
2 fibo =
or if
          1
else
          fibo 1- fibonacci ( calculate fib(n-1) )
          fibo 2- fibonacci ( calculate fib(n-2) )
          +
then ;
```

This word demonstrates the power of local variables. The same word could have been written using only the parameter stack, but it would have had lots of dup’s and drop’s, and would have been more difficult to fathom. Give it a try! However, this word is quite a heavy stack user (and very inefficient, the way it is written), so you may need to resize the parameter stack. The default parameter stack size is 256 entries (or 1024 bytes). To increase the stack size to 10000 entries, enter

```
>free .
 222482
>40000 resize
>free .
 183506
```

This example of recursion is about as complex as you’d normally need. However, if you’re into writing mutually recursive words, look up ‘::’.

Now let’s try a different application:

```
: thingo ( n p -- | print n to the p'th power )
  1 locals| a power n |
  power if
          power 1 do
                      a n * to a
          loop
  then
  n . ."  to the" power . " 'th power is: " a . ;
```

Note that local variables are listed in the opposite order to their initializing values on the stack, and a's initial value (1) was put into the word definition, and is not passed to it when calling 'thingo'. 'to' is used to give a local variable a new value, just as with automatics. If you need to find the address of a local variable (or an automatic), use 'addr.of'.

Structures and Arrays

FORTH allows structures to be defined with 'struct':

```
struct complex

          long:       .real
          long:       .imag

ends
```

Now the word 'complex' can be used just like 'variable', 'automatic' and 'constant':

```
complex z1
```

Let's see how to use them:

```
complex z1
complex z2
complex result

: c*      ( complex1 complex2 -- | multiply two complex
                      structures.  Return in 'result'. )

          locals| c2 c1 |
          c1 .real @
          c2 .real @ *
          c1 .imag @
          c2 .imag @ * ~
          + result .real !

          c1 .real @
          c2 .imag @ *
          c1 .imag @
          c2 .real @ *
          + result .imag !

          result ;
```

Now we'll try it out:

```
>1 z1 .real !
>1 z1 .imag !
>5 z2 .real !
>3 z2 .imag !
>z1 z2 c*
>result .real @ .
 2
>result .imag @ .
 8
```

The field words, '.real' and '.imag' simply add an offset to the base address of the structure, and 'z1' and 'z2', just like standard variables, return the address of their contents.

To create an array, we use the word 'array'. Firstly you must specify the size of each element, then the number of elements.

Arrays either start at element 0 or element 1, depending on which of 'base0' or 'base1' has been specified before declaring the array.

If 'chk+' is executed before the array is defined, code will be compiled into the array definition to check the bounds of the array.

```
>chk+
>base1
>size.of complex 10 array zfred
>0 fred

?Bad array index.
>0 1 fred .real ! ( store 0 in fred[1].real )
>0 1 fred .imag ! ( store 0 in fred[2].imag )
>11 fred

?Bad array index.
>
```

To save space in the object area, there is another type of array: the 'larray'. This array is defined exactly as with a normal array, but no space is allocated for it when it is declared. Space is allocated with the word 'uses', and the storage space for the array will remain around for as long as the word which 'uses' it is executing.

```
>chk+ base1
>1 80 larray characters
>: get80
>  uses characters
>  80 1 do
>         cget dup emit
>         dup 13 = if
>                     drop 0 i characters c! lastdo
>         else
>                     i characters c!
>         then
>  loop
>  80 1 do
>         i characters c@ if
>                     i characters .str crlf
>         else
>                     lastdo
>         then
>  loop ;
>get80
Type something!
ype something!
pe something!
e something!
 something!
something!
omething!
mething!
ething!
thing!
hing!
ing!
ng!
g!
!
```

The return stack.

When writing FORTH words, it is convenient to be able to stash away a value (or several values) somewhere else temporarily. One way to do this is with the ‘return stack.’ There are several FORTH words to push, pop and examine the top of the return stack.

‘>r’ and ‘<r’ push and pop values to the return stack
‘r@’ looks at the top element on the return stack
‘dropr’ drops the top element on the return stack
‘r1’, ‘r2’, ‘r3’ and ‘r4’ look at the first, second, third and fourth element on the return stack (‘r1’ is equivalent to ‘r4’).

Testing and debugging.

FORTH is notoriously unforgiving. However, to compensate partially for this fact, 1616 FORTH has a number of features to allow debugging a little easier.

Error messages.

As an aid to debugging, FORTH prints lots of warning messages when you do something wrong (if it doesn’t crash). The most often encountered is the ‘whazzat’ error:

```
>nosewash

nosewash?
>
```

This error message does not appear very useful, but all it is saying is that ‘nosewash’ is not a word in FORTH’s vocabulary.

The ‘whazzat’ error occurs at other times, but it usually appears in conjunction with other error messages.

The other messages that you may find are the following:

?Definition word only.
    Some words may only be used in colon definitions. If you attempt to use this word in direct mode, this error will be generated. If this word occurs while loading a file that looks OK, check that ‘interactive’ appears at its end.

?Stack mismatch in colon.
    A compound statement (such as if/then, begin/until) was incorrectly nested.

?Division by zero.
    Division by zero. Note that ‘/q’ will generate a genuine 68000 exception, as it uses the ‘DIVU’ instruction.

?Double local definition.
    Only one locals| definition is allowed per word.

?Out of memory.
    An attempt was made to ‘link’ or ‘allot’ more memory than was available. Use ‘free’ to find out how much space is available.

?Bad stack in load.
    When a file is loaded, it must leave the parameter stack in the same state as when the load began (file nesting information is stored on the parameter stack).

?EOF in load.
    ’interactive’ was not encountered before a forth file ended.

?No file.
    File not found when attempting to load. If multiple files are being loaded, turn on file echoing with ‘echo+’ to see where in the load the error occurs.

Break.

Either altc was pressed with altc checking turned on (altc+), or an ‘altc’ was executed while an altc was active, or a ‘stop’ was encountered. ‘cont’ can be used to continue from this point.

?Stack overflow.   ?Stack underflow.

The stack over/underflowed. This condition is usually not checked for when a word is executing, so if your program unexplainedly crashes, underflow or overflow might be the cause. (See the description of ‘trace mode’ below for help)

?Word not unique.

FORTH will not let you redefine a word that has already been defined. If you need to do such a thing, turn this checking off with ‘unique-’.

Trace mode.

When initially developing a program, it would be nice to have a facility to single-step code. This is possible by executing a ‘trace+’ before compiling your code; when the code is called, each word will be displayed before it is executed, and the 1616 will wait for a key. When a key is pressed, the contents of the stack after execution of the word will be displayed and the next word printed. Pressing ‘space’ will execute 24 words before waiting for the next key.

If trace mode is turned off with ‘trace-’, none of this information will be displayed, but altc and stack bounds will be checked after the execution of each word.

Altc & stack checking

Trace mode adds a large overhead to developing programs, so there is a mode which allows stack and altc to be checked regularly, without the overhead.

Executing ‘altc+’ will turn this checking on; any programs compiled with this flag turned on will have stack and altc checking compiled with them. The call to check these things is added after every ‘do’ and ‘begin’ in the code.

If ‘altc’ is pressed while such a program is executing, the message "Break." will be displayed and the prompt will appear as ‘~’. While the prompt looks like this, a stopped program may be restarted with ‘cont’. You must ensure that the parameter stack is the not changed when you continue a stopped word. If you do not wish to continue execution of the word, enter ‘abort’.

Coding conventions.

The ideas mentioned here are not rules; they are not even followed rigourously in the FORTH kernel. However, if you stick roughly to these conventions you should find it much easier to read your own code (and that of anybody else).

Nesting.

It is useful when writing code to indent the program text to give some idea of nesting level. One tab per nest is usually reasonable; if the code is being squashed too far to the right, either breaking the word up into simpler components or using a nesting level of four, say, would be reasonable.

Example:

```
: nest
  10 1 do
          10 1 do
                      10 1 do
                                  ." 1000 times."
                      loop
          loop
          crlf
  loop ;
```

It's also a good idea to break each line at logical breaks in the code, rather than when the line begins to fill up. This is sometimes a rather arbitrary choice to make, but as long as you're consistent it should make it easier to read.

Word names.

All FORTH objects are built out of the same stuff, so it is necessary to give some artificial guidelines to stick to to make different types of object recognizable.

%word	Automatic variables
$word	Constants
.word	Structure field names
(word)	Low-level words of minor importance (and some direct 1616 calls)
wordf	Floating point versions of standard words
word>	Codewriting words
word?	Words that leave flags or info on the stack
~@~	Words that load some object from memory
~!~	Words that store some object to memory
make_	Structure creating words
wword	Window words

Vocabulary names.

A vocabulary should be given the name of the file which contains it.

Using help.f

While you're learning FORTH, you may find it hard to remember what all the words in the vocabulary do. As an aid, you may want to load the file 'HELP.F' before you start work. To use this word, type:

>help {word}

Help will then scan through all the source files it knows about looking for a definition of {word}. If it finds it, it will print out the first line of the definition (which hopefully contains comments about the word's use).

Alternatively, if you want to look at the definition of a FORTH word, use the word 'list' instead of 'help'.

For either word, if you know the file the definition appears in, you may specify the file name on the line with 'help':

```
>help wopenq screen.f
```

This will only scan the file screen.f

Library files.

If you want to write assembler routines for FORTH, a library loading word is available in LIBLOAD.F.

Format of library file:

```
          org         $0000

          dc.l        fin-start

start     {code}
            .
            .
          {code}

fin       dc.b        len,'word1'
          dc.l        label
          dc.b        len,'word2'
          dc.l        label2
          dc.w        0

          end
```

Immediate words.

Many words defined in the FORTH kernel are not compiled when they are used in a colon definition, but execute while a word is being defined. Words such as 'if', 'do', 'to' and ';' do their work as the words which use them are being compiled.

Some other words, such as 'dup', '>r' and '1+' are immediate words, yet all they do is compile themselves into the word being defined. The reason this is done is because such words may be compiled in two bytes rather than the usual four, resulting in faster and smaller FORTH programs.

A word can be made immediate by executing ‘immediate’ immediately after it is defined.

This concept takes a little getting used to, and is not necessary for writing most FORTH programs. However, if you are interested in increasing FORTH’s pool of structured statements, look at the definitions (in FORTH) of many of the kernel words in the file WORDS.F, or even in the assembler source file FORTH.S.

Bugs.

FORTH is not perfect; however, none of the known ‘bugs’ which follow should present too many difficulties if you’re aware that they exist.

‘do/loop’ statements are always executed at least once; if this is not desirable, it is necessary to enclose the loop with an ‘if/then’.

Unlike standard FORTH, the highest loop index is counted as within the range of a do loop: for example, ‘10 1 do’ executes ten times with 1616 FORTH, nine times with standard FORTH.

Return stack overflow is not checked for with recursive calls, so a word which calls itself indefinitely will cause a crash. Similarly, parameter stack over/underflow is not always checked for unless one of trace or altc modes are turned on.

If FORTH is very low on memory, (less than 32K is unsafe), using the editor or transient programs may crash the machine.

When windows are open (using SCREEN.F) and an error occurs, an ‘abort’ should be executed immediately to ensure that the current window structure is reset to the default 1616 window.

When using the fast dot plotting routines (DOTFAST.F), the coordinates used are always relative to the whole screen, rather than the current window. If the screen may have scrolled, a ‘walign’ should be executed to ensure the CRTC scroll register is zeroed.

68000 hardware exceptions, such as bus error, address error, zero divide will crash the machine as expected.

Some operations of FORTH will move the return stack, which is a problem if any absolute addresses (such as window structures) reference or occur inside the return stack. These operations are:

i) A word added to the vocabulary, if the vocabulary space is full

ii) A ‘resize’ or ‘vocresize’

iii) Initializing the heap with the words in HEAP.F

However, all of these conditions can be avoided whilst a word is executing.

2
Kernel words.

!   (val addr --)
See also: @ c@ w@ w@x c! w! +!
Stores the longword {val} at {addr}.

"   (-- str)
Usage: " {string}"
See also: ." .str
Immediate:
Writes {string} into the dictionary and leaves its address on the stack.
Definition:
Writes {string} into dictionary. When code is executed, the string’s address is left on the stack.

%abase
%achk
%ccase
%struct   (-- val)

See also: array case struct chk+ chk- base0 base1 larray
Automatic variables used by case & structure declaring words.

’   (-- address)
Usage: ’ {token}
See also: dict search
Reads next token immediately and leaves its code address on the stack.

’c   (--)
Usage: ’c {token}
See also: ’ compile
Immediately read next token and compile it into the dictionary. Use this word to compile immediate words (such as if " dup etc.)

((--)
Usage: ({comments})
Start of a comment. Comment is terminated with ‘)’.
Comments may be nested and span multiple lines.

(.)   (n addr1 -- addr2)
See also: . .str pad expect dexpect
Converts the number {n} to a text string in the current base. {n} is right justified in a 35 byte field at {addr1}; {addr2} points to the first non-blank character of {n}. At least 36 bytes should be provided at {addr1}.

(do)   (--)
Word compiled by ‘do’.

*   (n1 n2 -- n1*n2)
See also: *q /q mod

*q   (n1 n2 -- n3)
16x16 bit multiply using 68000 MULU instruction. Faster than *.

+   (n1 n2 -- n1+n2)

+! (n address --)
See also: !
Adds longword n to value at address.

+loop (n --)
See also: do loop -loop
Adds n to loop counter, and if result is less than or equal to loop terminator, branches back to corresponding ‘do’.

,
,l (n --)
See also: code literal > >!l
Writes 16 or 32 bit value into object code area. Use to put 68000 machine instructions into word definitions.

- (n1 n2 -- n1-n2)

-loop (n --)
See also: do loop +loop
Subtracts n from loop counter, and if result is greater than or equal to the loop terminator, branches back to the corresponding ‘do’.

. (n --)
See also: number base (.) tab emit ."
Prints out value of n in current base.

." (--)
Usage: ." {string}"
See also: " .str crlf
Prints {string}. May be used only in colon definitions.

.card (n --)
See also: . (.)
Write out n with no leading sign

.s (--)
See also: trace+
Nondestructively prints out contents of stack, lowest element first. Note that this is opposite to some other Forths.

.str (string --)
See also: type
Prints null-terminated string.

/ (n1 n2 -- n1/n2)

/mod (n1 n2 -- (n1/n2) (n1 mod n2))
32 bit divide & modulus.

/modq (n1 n2 -- (n1/n2) (n1 mod n2))
32 by 16 bit divide & modulus.

/q (n1 n2 -- n1/n2)
32 by 16 bit divide using 68000 DIVU instruction. Faster than /.

0< 0<=
0<> 0=
0> 0>= (n -- f)

See also: if until while
Test n and leave a flag on the stack. -1=true, 0=false.

1+	(n -- n+1)	
2+	(n -- n+2)	
3+	(n -- n+3)	
4+	(n -- n+4)	
5+	(n -- n+5)	
6+	(n -- n+6)	
7+	(n -- n+7)	
8+	(n -- n+8)	
1-	(n -- n-1)	
2-	(n -- n-2)	
3-	(n -- n-3)	
4-	(n -- n-4)	
5-	(n -- n-5)	
6-	(n -- n-6)	
7-	(n -- n-7)	
8-	(n -- n-8)	
2*	(n -- n*2)	
4*	(n -- n*4)	
8*	(n -- n*8)	
16*	(n -- n*16)	
256*	(n -- n*256)	
2/	(n -- n/2)	
4/	(n -- n/4)	
8/	(n -- n/8)	
16/	(n -- n/16)	
256/	(n -- n/256)	
	See also:	+ - * / /mod /q /modq *q >> << >>x

2drop	(n1 n2 --)	
2dup	(n1 n2 -- n1 n2 n1 n2)	
2swap	(n1 n2 n3 n4 -- n3 n4 n1 n2)	
	See also:	dup swap rot over drop

: (--)
Usage: : {token} {definition} ;
See also: immediate ; :: forward
Defines {token} as an executable word, which will execute all the code in {definition} when it is invoked.

:- ({test} --)
See also: case endcase default dropcase
Selector in case statement.

:: (--)
Usage: :: {token} {definition} ;
See also: forward makes
Redefines {token} as an executable word by coding a BRA to {definition} into the previous definition. Use this word for writing mutually recursive words. Do not use this word for redefining kernel words unless you are sure that the redefinitions are permanent.

; (--)
See also: :
Completes a colon definition. Semicolon codes a RTS into the dictionary and checks that the parameter stack is the same as it was when ‘:’ was executed.

<	(n1 n2 -- f)	
	See also:	> < <= >= = 0= if while until
	True (-1) if n1 < n2.	
<<	(n1 count -- n1<<count)	
	See also:	2* 4* 8* 16* 256* >> >>x
	Left shift. Multiplies n1 by 2^count.	
<=		
<>	(n1 n2 -- flag)	
	See also:	<
<r	(-- value)	
	See also:	>r r@ r1 r2 r3 r4 dropr
	Pops top value from return stack.	
=		
>	(n1 n2 -- flag)	
	See also:	<
>!l		
>!w	(--)	
	See also:	literal ,
	Codes a move.l $xxxx,-(a6) or a move.l $xxxxxxxx,-(a6) into dictionary.	
>=	(n1 n2 -- flag)	
	See also:	<
>>	(n1 count n1>>count)	
	See also:	2/ 4/ 8/ 16/ 256/ >>x <<
	Shifts n1 right ‘count’ times, with no sign extension.	
>altc	(--)	
	See also:	altc+ altc- altc stop
	If altc checking has been enabled, compile an ‘altc’ into the current word definition. Otherwise, do nothing.	
>local	(n --)	
	See also:	locals| local? local> to
	Word to compile a store to a local variable.	
>r	(n --)	
	See also:	<r r1 r2 r3 r4 dropr
	Pushes n to return stack.	
>struct	(n -- n)	
	See also:	struct
	Word to create & compile a structure field.	
>uses	(address --)	
	See also:	larray uses
	Word compiled by ‘uses’	
?>	(--)	
	See also:	if
	Compiles TST.L (A6)+ into dictionary.	
@	(addr -- val)	
	See also:	c@ w@ w@x c! w!
	Loads a longword {val} from {addr}	

] (--)
Usage:] {1616 command}
See also: exec sexec shell system
In immediate mode, allows a 1616 command to be executed. FORTH copies itself to the stack when] is executed, so transients may be freely executed.

abort (anything . . . --)
See also: error cont stop
Closes all files currently being LOADED, resets parameter stack, return stack, deallocates space from links,turns off compile flag, turns off colon mode & (if screen.f loaded) resets default window.

abs (n -- +n)
See also: ~ - max min
Convert n to positive integer.

addr.of (-- address)
Usage: addr.of {auto. or local var}
See also: locals| to @ ! automatic
Leaves the address of automatic or local variable on stack.

again (--)
See also: begin repeat while until return
Used with a ‘begin’, causes code between ‘begin’ and ‘again’ to be executed forever.

align.w (n -- n aligned on even byte boundary)

allot (space --)
See also: link forget next!
Allot ‘space’ bytes in object area. To find the address of this space, execute a ‘next@’ before allotmentment. An attempt to allot too much space will result in an out of memory error.

aload (--)
See also: load args
If an argument was specified when FORTH was invoked, load it as a FORTH file.

altc (--)
See also: stop altc+ altc- cont
If altc has been pressed, ‘stop’. Otherwise, check the stack for over/underflow.

altc+ (--)
See also: >altc altc-
Tells FORTH to put checks into compiled code so that if altc is pressed, execution will halt immediately. If code has already been compiled, altc+ has no effect.

altc- (--)
See also: >altc altc+
Turns off run-time checking of altc. Code that has been compiled with the altc flag on will still check for altc.

altc? (-- flag)
See also: cget c? cget? altc
If altc has been pressed, flag is true (-1). This word also clears the altc status.

amul
See also: array larray
Word to compile multiplication part of array definition.

and
(n1 n2 -- n1 and n2)
See also: eor or not

args
(-- address)
See also: aload
Returns a pointer to the array containing information about the command line arguments passed to FORTH when it was invoked. See 1616 documentation for arguments format.

array
(elsize nels --)
Usage: {el. size} {no. els} {token}
See also: chk+ chk- base0 base1 larray size.of marray
Create an array with {nels} elements, each of size {elsize}.

```
Example:
base1
4 10 array values
: sum10        ( sum 10 numbers from keyboard )
  10 1 do
               32 pad expect ( get number )
               pad number ( convert it )
               if
                       i values !
               else
                       " Bad number" error
               then
                 loop
                 0 10 1 do
                       i values @ +
  loop . ;
```

This word reads 10 numbers from the keyboard, calculates the sum and prints the result.

asc
(-- value)
Usage: asc {chr}
See also: emit
Leave ascii value of {chr} on stack.

automatic
(--)
Usage: automatic {token}
{token} -> (-- value)
See also: to addr.of variable constant dauto
Defines {token} to be an automatic variable, whose value is placed on the stack when executed. The initial value is assigned to be 9999.

```
Example:
               >automatic a   ( create a )
               >a .           ( get a’s value )
               9999
               >42 to a       ( assigns 42 to a )
               >a .
               42
               >addr.of a @ .
               42
```

base
(-- value)
See also: hex decimal number .
System variable containing the current numeric base in use for numeric conversions and printing.

base0

base1
(--)
See also: array larray
If base0 is executed before defining arrays, array indices will start at zero.
If base1 is executed, array indices start at 1.

begin
(--)
See also: while repeat until again
Marks the start of a begin/again, begin/while/repeat or begin/until loop.

```
Examples:
: star_forever ( print lots of stars )
  begin asc * emit again ;

: star_key ( print stars ‘til key pressed )
  begin asc * emit c? until ;

: keys_prt ( print keypresses while altc not pressed )
  begin cget
                dup 0>= while (altc?)
                emit
  repeat ;
```

bell
(--)
Emits a bell character (7 emit)

beq>
bne>

bra>
(--)
Codes a branch instruction into dictionary.

bmove
(src dst count --)
See also: move
Does a block move. bmove always moves a multiple of four bytes, and {src} and {dst} must be alingned on a four-byte boundary. Use ‘move’ for moving character strings.

byte:
(addr offs -- addr offs+1)
Usage: byte: {token}
See also: long: word: string: struct ends bytes:
Declares {token} to be a byte length field in a structure.

bytes:
(addr offs nbytes -- addr offs+nbytes)
Usage: {n} bytes: {token}
See also: long: word: string: struct ends byte:
Declares {token} to be a field of length {nbytes} in a structure. ‘bytes:’ is equivalent to ‘string:’.

c!
(byte addr --)
See also: c@ @ ! w@ w!
Store {byte} at {address}.

c?
(-- f)
See also: altc? cget cget?
f is true (-1) if a key has been pressed. c? does not detect altc.

c@
(addr byte --)
See also: c! @ ! w@ w!
Loads {byte} from {addr}.

call (addr -- ??)

See also: compile

Call the assembler subroutine (or forth word) at {addr}. This routine should preserve a2-a5, a7 and d3-d7, and a6 should still point to a position in the FORTH stack. The routine should terminate with a ‘rts’. Note that this instruction is not simply a ‘jsr’; it checks to see whether or not (addr) contains 68000 code or a special FORTH construction.

case (value -- value)

See also: :- default | endcase dropcase

case is similar to if/then, but it allows many options to be checked for. Example:

```
: fred
  case
               1 :- ." one" |
               2 :- ." two" |
               3 :- ." three"
  endcase ;
```

Note that the last choice does not have a terminating ‘|’. The final choice may be ‘default’. Case stores the condition to be checked for on the return stack, so any exits from within the case statement must be handled carefully with ‘dropcase’. If a logical expression rather than a comparison with the condition needs to be checked for, use ‘=>’.

cget (-- chr)

See also: c? cget? altc?

Wait for a key from the keyboard and return its ascii value. If altc has been pressed, cget returns -1.

cget? (-- [chr] flag)

See also: c? cget altc?

As for cget, but does not wait for a key. Instead, cget? returns false (0) if no key has been pressed.

chk+

chk- (--)

See also: array base0 base1 larray

If chk+ has been previously executed, array declarations will be generated with code to check bounds.

clear (--)

Prints a clear screen character (12 emit).

code (--)

See also: , literal hex

Within a colon definition, all numbers will be compiled directly into the dictionary rather than compiled as literals. This word is used for compiling 68000 code into words. This word sets the base to 16; after a word with ‘code’ has been compiled, the base should be reset do decimal by executing ‘decimal’.

compile (address --)

See also: execute :

Compile a ‘JSR address’ into dictionary. If {address} is greater that $8000, a long JSR or relative BSR will be coded instead of the usual short JSR.

constant (value --)
See also: variable automatic
Usage: {expression} constant {token}
Generates a new word which, when executed, leaves a value on the stack. Constants are identical to automatic variables in some cases, but beware: small-valued constants are compiled as
MOVEQ #n,d0
MOVE d0,-(a6)
for efficiency.

cont (--)
See also: altc stop abort
If a ‘stop’ or an ‘altc’ has occurred, indicated by the ‘~’ prompt, cont will continue execution of a forth word at the point it left off. All registers, return stack, program counter and parameter stack pointer are restored.

copy (--)
See also: {see VLIST SYS}
Usage: copy {srcfile} {destfile}
As per the standard 1616 command. This command may only be used in direct mode, and must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

create (--)
Usage: create {token}
See also: variable allot header token
Reads the next token and make a header in the dictionary pointing to the current position in the object area.

crlf (--)
Prints a c/r linefeed. (13 emit 10 emit)

debug+

debug- (--)
See also: trace+ altc+
Executing ‘debug+’ tells forth not to ignore text entered after ‘\’ on a line. ‘debug-’ (the default) forces forth to ignore text after ‘\’. Use the ‘\’ in front of debugging code in your files.

decimal (--)
See also: hex base . number
Sets current base to ten. (10 base !)

default (-- n)
See also: case :- endcase dropcase
Default test for case statement.

delete (--)
Usage: delete {filename}
See also: edit dir
As per the standard 1616 command. This command may only be used in direct mode, and must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

delim? (chr -- flag)
flag is set true if chr is tab, space or null.

dexpect

(n addr --)
See also: expect pad (.) .str
Input a line of text from the keyboard, of maximum length {n} characters, to {addr}. If a string already appears at {addr}
(that is, if the first character is non-zero), display it on the line and allow it to be edited. If the string is longer than {n} characters, the string’s length will be the maximum.

dfind

(-- addr)
See also: dict dsearch find immediate ’
Find address of current token in the vocabulary table. This word returns the address of the dictionary entry, not the code, for a word. For the structure of each dictionary entry see dict.

dict

(-- address)
See also: ’ ’d dfind find? find search token
Returns a pointer to the dictionary pointer. dict @ points to the dictionary entry of the most- recently-compiled word.

Dictionary structure:
{count}
{characters . . .}
{code address}

{count} is a one byte character count, or’d with 128 to indicate immediate execution

{characters . . .} are {count} characters of the word, possibly with a zero fill byte

{code address} is a longword

dir

dirs

(--)
Usage: dir
or: dir {pattern}
or: dir {blk driver}
As per the standard 1616 command. This command may only be used in direct mode, and must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

do

(last first --)
See also: loop +loop -loop lastdo i j
Marks the start of a do/loop structure. The previous loop indices are placed upon the return stack.

```
Example:
: timestab     ( print 1-5 times table )
  1 5 do
               5 1 do
                       tab i j * .
               loop
               crlf
  1 -loop ;
```

docheck

(element top -- element)
See also: chk+ chk- array larray
Checks array bounds. This word is compiled into an array definition if chk+ has been executed.

drop (n --)
See also: dup swap rot over 2drop
Drops top element from stack.

dropcase (--)
See also: case :- endcase return dropdo
This word must be used when escaping from a case statement with a ‘return’ or ‘goto’. The case check is stored on the return stack, so this word is equivalent to ‘dropr’.

dropr (--)
See also: >r <r r1 .. r4
Drops top value from return stack.

dsearch (-- [address] flag)
See also: search token find dfind ’d
Searches for current token in dictionary and returns its dictionary table address if it is found.

dup (n -- n n)
See also: swap rot over drop 2dup nip tuck
Duplicates top entry on stack.

echo+

echo- (--)
See also: load
echo+ and echo- determine whether or not a file is echoed to the screen when it is loaded.

edit (--)
Usage: edit {filename}
See also:] exec sexec dir delete
Calls 1616 editor on filename. Be warned: FORTH copies itself to the return stack when edit is invoked, so there may not be enough memory to load all your file. However, this ensures that FORTH is not overwritten by large text files.

efcopy (--)
Usage: efcopy {files} {block driver}
As per the standard 1616 command. This command may only be used in direct mode, and must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

else (--)
See: if then

emit (chr --)
See also: type .str .
Outputs character to screen (or stdout)

endcase (n --)
See also: case default dropcase :- |
Terminates a case statement.

ends (addr offs --)
See also: struct byte: long: word:
Terminates a struct definition.

eor (n1 n2 -- n1 eor n2)
See also: and or not

error
(string --)
See also: abort
Prints ?{string} then performs an abort.

esc
(--)
Prints an escape character (27 emit)

exec
(string --)
See also: sexec] syscall shell system
Executes a 1616 command. Cannot be used for transients.

expect
(n addr --)
See also: dexpect pad .str
Input a line of text from the keyboard, of maximum length {n} characters, to {addr}.

f0/

f1/
(--)
Usage: f0/
As per the standard 1616 command. This command may only be used in direct mode, and must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

fclose
(fdesc -- status)
See also: fopen fget fput fgetc fputc
Close a file. Note that FORTH file descriptors always have an offset of 16 added to them, to allow file descriptors to be used directly in calls that may use devices such as cent:.

fcopy
(--)
Usage: fcopy {source file} {dest block driver}
As per the standard 1616 command. This command may only be used in direct mode, and must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

fget
(fdesc -- n)
See also: fput fgetc fopen fclose
Get a 32 bit (4 byte) integer from file (or device) {fdesc}.

fgetc
(fdesc -- c)
See also: fput fget fopen fclose
Get a byte from file (or device) {fdesc}.

find
(-- address)
See also: whazzat dfind ’
Searches for current token, giving error message if not found, or the code address.

fkey
(--)
Usage: fkey .n "{string}"
As per the standard 1616 command. This command may only be used in direct mode, and must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

flip
(w1w2 -- w2w1)
Swap the two words of the top element on the stack.

float:
See ‘struct’

fopen (name mode -- fdesc)
Open a file on the current block device, with mode being 1 for read, 2 for append/write, 3 for read/write. See ’file.f’ for more usefule file manipulation words/constants. If an error occurs, fdesc will be -1.

forget (--)
Usage: forget {token}
See also: vdel vocab
Removes {token} and all after it from the dictionary, and resets next (a5) to the code address of {token}.
Beware: if the dictionary is out of order, (as with a libload) forget may have unpredictable results, as definitions may be removed from the code area while their dictionary entries are not. It is safest to forget vocabularies.

forget? (--)
Usage: forget? {token}
See also: oad? forget
If token exists, forget it. This word should appear at the head of every forth file to ensure that if a file is loaded twice, its previous incarnation is removed.

forward (--)
Usage: : {token} forward ;
See also: ::
This word does not do a thing, but is useful for indicating that the real definition for {token} lies ahead. Essential for writing mutually recursive words.

free (-- value)
See also: mem vocfree allot link used vocused
Returns the amount of space it is possible to allot or link. The value returned by free allows 4k between the bottom of the return stack and the top of FORTH for interrupts etc.

goto (--)
Usage: goto {label}
See also: label:
Jump to {label}. See ‘label:’ for warnings about the use of ‘goto’.

header (--)
See also: create dict next@ unique+ unique-
Creates a dictionary entry for the current token. If the token already exists, an error message is generated unless the unique flag has been turned off.

hex (--)
Sets the current base to 16.
See also: base number . decimal

i (-- n)
See also: i+ j do loop +loop -loop
Returns the current value of the do/loop counter (d3).

i+ (n -- n+i)
See also: i j do loop +loop -loop
Adds the current value of the loop counter to n.

if (flag --)
Usage: if {code1} then
or: if {code1} else {code2} then
See also: then case else if0
If flag is non-zero, execute {code1} [else execute {code2}]

```
Example:
: iftst
  cget
  asc a = if
                .” Was an ""a"""
  else
                .” Was not an ""a"""
  then
  crlf ;
```

if0 (flag --)
As for if, but code is executed if flag is zero.

imm? (-- flag)
{flag} is false if executed while a word is being defined.

immediate (--)
See also: immword dict
Changes dictionary entry for the most recently compiled word so that it will execute inside colon definitions rather than be compiled.

immword (--)
Check imm? flag, and if word is not being used inside a colon definition, error.

incase? (--)
See also: case
Checks that :-, default, dropcase or endcase are executed while a case statement is active.

instruct? (--)
See also: struct
Checks that we’re inside a struct definition.

interactive (fdesc marker --)
See also: load
Indicates to FORTH that the current file has come to an end. It must be used at the end of every word definition file.

j (-- n)
See also: do i loop
Returns the value of an outer loop counter.

join (addr --)
See also: split if
Fills in a previously compiled forward branch so that it jumps to the current dictionary position.

label: (--)
See also: goto dropdo dropcase dropr return
Usage: label: {labelname}
Specifies a label for a goto inside a word definition. A label may be declared before or after the corresponding goto, and may be in a separate word. However, any do loops, case statements or return stack pushes left hanging are likely to have nasty effects, so it is best to jump to labels within the same word, being very careful about cleaning up.

larray (elsize #els --)
Usage: {expr1} {expr2} larray {token}
See also: uses array chk+ base0
Creates a locally useable array whose element size is {expr1} and has

{expr2} elements. Using larray to define an array does not allocate any space for the array: the word ‘uses’ must be used inside a definition to allocate space for the array on the return stack.

```
Example:
base0 4 10 larray fred
: fred_is_used ( fill fred with zeroes )
               uses fred
               9 0 do
                      0 i fred !
               loop
  loop ;       ( fred is now deallocated )
```

lastdo

(--)
See also: do loop
Sets the loop counter to its terminal value, so that when ’loop’ is next encountered the loop will terminate.

line!

(addr --)
Set the address of the interpreter’s line buffer. Not very useful.

line?

(-- addr)
Get the address of the first character of the interpreter’s line buffer.

line@

(-- address)
See also: token
Returns the current pointer within FORTH’s command line (a4). This word is useful for examining command lines directly.

link

(n --)
See also: link@ unlink locals|
Allocates n bytes in memory. The space is freed when the word containing the link completes, unless the top element of the return stack is temporarily popped when the link is executed, in which case the space will be freed when the calling word terminates. Executing an abort or causing an error will cause all linked blocks to be freed. Only 256 link’ed blocks may be in use at any one time.

link@

(-- addr)
See also: link unlink
Returns the address of the last link.

```
Example:
: slinky ( fdesc -- | gets line from file & print )
               128 link        ( allocate space )
               link@ swap fgets ( get line )
               link@ .str      ( print line )
               crlf
;                      ( return & deallocate space )
```

If it is necessary to keep the allocated space on the stack for the calling word (see wopen@ in screen.f), use the following method:

```
: slinky2 ( fdesc -- addr | gets line from file )
               <r 128 link >r  ( allocate space )
               link@ swap fgets ( get line )
;              ( return )
```

The space on the stack will be deallocated when the word beneath completes.

literal

(n --)
See also: constant number
Compiles code in the object area to put n on the stack.

load (-- fdesc marker)
Usage: load {filename}
See also: interactive aload
Causes FORTH to open {filename} and read lines from it instead of the keyboard until ‘interactive’ is encountered. Don’t put more commands on the line with the load, as FORTH will attempt to execute them before the file is loaded.

Loads may be nested to a depth of 16 (limited by number of open files allowed).

If the file specified is not found, FORTH tries adding a ’.f’ extension (save does not do this).

’load’ stores nesting information on the parameter stack, so the stack should be left in the same state when ‘interactive’ is executed as when a file began loading.

load? (-- fdesc marker) or (--)
Usage: load? {token} {filename}
See also: forget?
If {token} doesn’t exist, load {filename}. This word should be put at the head of word files to ensure that any other necessary word files have been loaded, and the token will usually be a vocabulary name.

local> (local# --)
See also: local? >local
Compiles a local variable.

local? (-- [local#] f)
If current token is a local variable, f is true and local# is its index.

locals| (a1 .. an --)
Usage: locals| v1 v2 v3 v4 |
See also: mlink to addr.of
Declares local variables and loads their values from the stack. Only one locals definition is allowed per word.

The pointer to local variables is (a2).

```
Example:
: quadratic ( x c0 c1 c2 -- n | n=x*x*c2+x*c1+c0 )
               0 locals| xx c2 c1 c0 x |
               x x * to xx
               c0
               c1 x *
               c2 xx *
               + + ;
```

A ‘.’ appearing before a local variable will declare it to be a double (or floating). For example (using FLOAT.F):

```
: quadratic ( . x . c0 . c1 . c2 -- . n )
               0 0 locals| . xx . c2 . c1 . c0 . x |
               x x *f etc ... ;
```

logoff (--)
See also: save turnkey
Copies the vocabulary above the object area and sets internal pointers appropriately for a save.

long: (addr offs -- addr offs+4)
See also: struct
Longword field for structure.

loop (--)
See also: do +loop -loop
Adds one to the loop index (i), checks it, and if <= loop terminator, go back to do.

makes (--)
Usage: ’ {token1} makes {token2}
See also: vdel
Allows more than one token to be linked to the same piece of code. This allows several synonyms to be defined for convenience. By the use of ‘makes’ with ‘vdel’, words may be redefined.

Example: redefine emit to print control characters as
[val]

```
’ emit makes oldemit
unique-        ( allow emit to be redefined )

: emit
  dup 32 < if
               asc [ oldemit
               .
               asc ] oldemit
  else
               oldemit
  then ;

vdel oldemit
unique+
```

Note that words defined with the older version of emit will execute unchanged.

mark (-- addr)
See also: split join
Leaves the current object pointer on the stack to allow calculation of branches in if’s, again’s etc.

mark++ (-- addr)

mark-- (-- offset)
’mark++’ and ‘mark--’ as a pair calculate the offset for a backward branch.

max (n1 n2 -- max{n1,n2})

mem (--)
See also: vocused free used HEAP.F
Prints info about current memory usage.

min (n1 n2 -- min{n1,n2})

mlink (a1 .. an n --)
See also: locals| link mlink
Allocates n*4 bytes on the return stack with link, then reads n values from the stack into this space. locals| uses this word. Note that (a2) is set to point to local variables, so that future links will not supersede the local definitions. The link pointer (a3) is also set to point to the local variables, but any future link will supercede this.

mod (n1 n2 -- n1 mod n2)
See also: modq /q /modq /mod
32 by 32 bit integer modulus. Should only be used with positive numbers.

modq (n1 n2 -- n1 mod n2)
See also: mod /q /modq
32 by 16 bit integer mod. This uses the 68000 DIVU instruction, so it is quick.

move (src dst #bytes --)
See also: bmove
Move {#bytes} bytes from {src} to {dst}. This is a byte move, so will be slower (but more flexible) than bmove.

msg1 (-- string)
See also: word1
Message first printed by FORTH when it is invoked.

newline (--)
Put a null into FORTH's input buffer so that no more words will be executed from the current input line.

next! (addr --)

next@ (-- addr)
See also: allot header compile
Set/read the code pointer (a5).

next_vocab

next_word (addr -- addr+offs)
See also: vlist type_word
Used by vlist to find the next vocabulary/word in the FORTH dictionary.

nip (n1 n2 -- n2)
See also: dup tuck rot swap over drop

nop (--)
Execute nothing.

not (addr -- addr eor $ffffffff)
See also: 0= ~
One's complement. This is not a logical not: use '0=' for this purpose.

number (string -- [number] flag)
See also: line . (.)
Convert {string} to a number. If the conversion was successful, flag is true (-1).

one.of

or.of See 'struct'.

or (n1 n2 -- n1 or n2)
See also: not and eor

org (-- addr)
Returns the address FORTH was loaded at ($4000).

over (n1 n2 -- n1 n2 n1)
See also: rot dup swap drop

pad (-- addr)
See also: expect dexpect (.) number
Returns the address of a 512-byte buffer, suitable for inputting lines from the keyboard, building strings or anything else. The space at 'pad' is totally unused by the FORTH kernel.

pick

(n -- val)
See also: roll
Pick the n'th value from the stack; '1 pick' is equivalent to 'dup', '2 pick' is equivalent to 'over'.

prompt

(--)
Type a prompt: '>' or '~', printing a carriage return if necessary.

r1

r@

r2

r3

r4

(-- n)
See also: >r <r dropr do case
Get a value from the return stack. r1 is (a7), r2 is 4(a7), r3 is 8(a7), r4 is 12(a7) and r@ is equivalent to r1.

rd/

rename

(--)
Usage: rd/
rename {file1} {file2}
See also: edit dir
As per the standard 1616 commands. These commands may only be used in direct mode, and each must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary 'sys'.

repeat

(--)
See also: begin while again until
Marks the end of a begin/while/repeat loop.

resize

(space --)
See also: vocresize bmove
Moves the return stack to give the parameter stack more/less room. Any absolute addresses pointing within the return stack will become meaningless, but local variables will survive. By using local variables and links, the 256 entries provided by default by FORTH should be plenty for any program.

return

(--)
See also: dropdo dropcase
Gets out of a word early. Be careful that all case, return stack pushes and loops have been correctly killed.

Example:

```
: rtntst
  10 1 do
             cget case
                    asc q :- dropcase dropdo
                           return |

             13    :- 10 13 emit emit |

             default :- case@ emit
             endcase
  loop ;
```

roll (a1 a2 a3 .. an n -- an a1 a2 a3 .. an-1)
See also: pick
Roll {n} elements in the stack. This word is not very fast; it should only be used if really necessary.

rot (n1 n2 n3 -- n2 n3 n1)

rot- (n1 n2 n3 -- n3 n1 n2)
See also: swap drop over dup

rp@ (-- addr)
See also: sp@ rtnhi
Returns the return stack pointer.

rtnhi (-- address)
See also: rp@
Returns the base of the return stack.

rts> (--)
See also: return
Codes a RTS into the object area.

rvsoff

rvson (--)
Turns off & on reverse field printing.

save (--)
Usage: save {filename}
See also: turnkey word1
Saves current FORTH environment to disk. The ‘.f’ extension is not automatically supplied.

sbod Save routine used by save and turnkey.

search (-- [address] flag)
See also: dict find
Searches for current token. If it is found, its code address and true (-1) is left on the stack. If it is not found, false (0) is left on the stack.

shell (--)
See also: system] exec sexec
Temporarily exit FORTH to the 1616os shell. Transients can be executed, as FORTH saves itself on the return stack.

size.of (-- size)
Use: size.of {structure}
See also: addr.of struct
Returns the size of a structure to allow arrays, links etc. to be declared of appropriate size.

sp@ (-- addr)
See also: r@ stkhi
Returns the pointer to the parameter stack: that is, {addr} points to the entry below itself.

split (-- addr)
See also: join mark split0
Code a beq into the dictionary and leave an address to allow a later join. Used by if, while.

split0 (-- addr)
As for split, but uses bne.

ssasm (--)
Usage: ssasm {filename}

ssddutil (--)
Usage: ssddutil
As per the standard 1616 commands. These commands may only be used in direct mode, and each must be the first and only FORTH word on the command line. All the system words are in FORTH vocabulary ‘sys’

stkhi (-- addr)
See also: sp@ rtnhi
Returns variable containing base of parameter stack.

stklen (-- addr)
See also: resize
Returns variable containing size of parameter stack (maximum number of entries is size/4)

stop (--)
See also: cont abort error
Saves registers, stack pointers and sets cont flag before returning to FORTH prompt.

string (size --)
Usage: string {token}
See also: .str " ."
Creates a string variable which, when executed, returns a pointer to a buffer of {size} bytes.

string: (addr offs size -- addr offs+size+1)
Usage: {expr} string: {token}
See also: struct
Creates a string field for a structure.

struct (--)
See also: size.of
Begins a structure definition.

Example:
struct wind

```
word: .xstart
               word: .ystart
               word: .xend
               word: .yend
               word: .bg_col
               word: .fg_col
               word: .curs_x
               word: .curs_y
               long: .wsave
               long: .oldwin
ends (wind can now be used to create variables: )
wind wtest     ( create wtest )
               0 wtest .xstart !word
               80 wtest .xend !word
               0 wtest .ystart !word
               25 wtest .yend !word
size.of wind 10 array 10windows ( create a ten-element array
)
```

swap (n1 n2 -- n2 n1)
See also: drop dup rot over 2swap nip tuck

syscall (an .. a1 #args call# -- return)
See also:] exec sexec
1616 os system call.

Example:
y x 2 55 syscall . (read pixel colour at (x,y))

system (--)
See also: shell
Quit forth permanently.

tab (--)
Print a tab (9 emit).

then (--)
See also: if

to (--)
Usage: {expr} to {token}
See also: locals| automatic addr.of
Sets the value of a local or automatic variable to {expr}. If variable is double-size (8 bytes), two entries are taken from the stack.

token (--)
See also: search find ’
Read the next token from the input line.

trace+ (--)
Turn on trace mode - words compiled when trace mode is on will be compiled with trace information. Words executed with trace mode on which contain this information may be single stepped through each word, with the parameter stack being shown after each step. If the trace mode is of while the trace’d word is executed, no single-step information will be printed out, but altc and stack over/underflow will be checked for with each word executed. trace’d code is about twice the size of un-trace’d code.

trace++ (--)
Turn on trace mode from within a colon definition to trace tiny sections. This word will not turn trace mode on when the word is executed; the trace mode must be on for the trace’d code to be single-stepped.

trace- (--)
Turn off trace mode.

trace-- (--)
Turn off trace mode from within a colon definition.

tuck (n1 n2 -- n2 n1 n2)
See also: nip dup over rot drop swap

turnkey (--)
Usage: turnkey {filename}
See also: save
Save FORTH without the vocabulary to conserve space. For this to work, word1 must be redefined. Please save FORTH applications for public distribution with ‘turnkey’, not ’save’.

Example: create a turnkey application to clear the screen.

```
>:: word1 clear system ; ( define first word exec’d )
               >turnkey clear.exec ( save forth environment )
               >system                 ( quit forth )
               f0>clear.exe            ( clear screen )
```

type (buffer count --)
See also: .str emit " ." crlf
Emit {count} bytes from {buffer}.

type_word (addr --)
See also: next_word next_vocab vlist
Used by vlist to type a word from the dictionary.

unique+

unique- (--)
Turns on and off a check to ensure all newly defined words are unique.

unlink (--)
See also: locals| link mlink
Deallocate space from last link, mlink, locals| etc. Whatever data was there will be clobbered within the next twenty milliseconds by the retrace interrupt.

’unlink’ must be executed at the same nesting level as the initial ‘link’ or ‘locals|’ was.

until (flag --)
See also: begin
If flag is true (-1), fall through to the next statement. Otherwise, go back to the last begin.

until0 (flag --)
As for until, except a flag of zero indicates termination.

used (-- space)
See also: mem free
Returns code space used.

uses (--)
Usage: uses {larray}
See also: link larray
Allocates space within a word for an array defined with larray. Other words may use the array after ‘uses’ has been executed, but ‘uses’ may not be nested with one array.

variable (--)
Usage: variable {token}
See also: automatic constant
Creates a variable which, when executed, returns the address of its four byte value. Although non-standard, ‘automatic’ and local variables are more efficient and easier to use than ‘variable’.

```
Example:
>variable xxx
>57 xxx !       ( sets value of xxx )
>xxx @ .        ( get value )
57
```

vdel (--)
Usage: vdel {token}
See also: forget makes
Removes {token} from the dictionary, but leaves object code intact. vdel is a useful word for non-destructively redefining kernel words.

vlist (--)
Usage: vlist
vlist {vocabname}
vlist {vocabname} i
vlist *
See also: immediate vocab
’vlist’ on its own will list vocabularies present in the dictionary. ‘vlist {vocabname}’ will list all words contained in vocabulary {vocabname}, while appending ‘i’ to the command will list all immediate words in {vocabname} in reverse field. ‘vlist *’ will list all words in all vocabularies.

vocab (--)
Usage: vocab {vocabname}
See also: load? forget? forget vlist
Marks the start of a new vocabulary. Use this word at the beginning of forth files, as explained in the section at the end of this document.

vocfree (-- space)
See also: vocused free used
Returns amount of free space for vocabulary. Free space is automatically allocated in 1k chunks when this runs out.

vochi (-- address)
Returns a variable pointing to the top of the vocabulary space.

voclen (-- address)
Returns a variable pointing to the total amount of space allocated to the vocabulary.

vocmove (--)
See also: word1
Moves vocabulary from just above the object code area into the vocabulary area. This word is only called once when FORTH is initially invoked. ’vocmove’ should appear in the definition of ‘word1’ before a ‘save’ for FORTH to function correctly, but should not appear in ‘turnkey’ed ‘word1’s.

vocpnt (-- addr)
When FORTH is initially invoked, vocpnt points to the end of the vocabulary sitting above the object area.

vocresize (space --)
Moves the return stack and parameter stack down to allow more space for the vocabulary. vocresize is automatically called when vocab space runs out. Calling ‘vocresize’ with {space} less than that returned by ‘vocused’ will almost certainly crash the FORTH system.

vocused (-- space)
Calculates how much vocabulary space has been used.

w! (word addr --)
See also: ! c@ c! w@
Store the 16-bit value {word} at {addr}.

w@ (addr -- {word})
See also: @ ! c@ c! w!
Get the 16-bit value from {addr}.

whazzat (--)
See also: error
Prints a currently offending token followed by a question mark. Aborts any load currently in progress.

while (flag --)
See also: begin repeat until
If flag is false (0), exit begin/while/repeat loop.

while0 (flag --)
As for while, except that loop is exitted if flag is non-zero.

word1 (--)
See also: save :: turnkey vocmove
Word called when FORTH is invoked. It can be modified to print a different sign-on message or start a turnkey application. Look in ‘WORDS.F’ for its default definition.

Initially word1 is defined as follows:

```
: word1
            msg1     ( sign-on message )
            .str     ( print it )
            vocmove ( move vocabulary into place )
            aload    ( load a FORTH file passed as arg )
;
```

word1 can be redefined with ::

:: word1 {turnkey application} system ;

word: (addr offs -- offs+2)
See also: struct
Specifies a word length field in a structure definition.

words (--)
See also: vlist
A synonym for ‘vlist’.

| (--)
See also: case endcase default dropcase :-
Separates choices in a case statement.

~ (n -- -n)

FORTH memory map:

```
HIGH MEMORY  -------------------------------------
                     1616os original return stack
             -------------------------------------
                     FORTH system buffers
                     line input buffer      <- line@ (a4)
                     512 byte user scratch<- pad
                     miscellaneous
             -------------------------------------<- vochi @
                     Vocabulary moving down
                        .
                        .
```

```
                                  forget?
     voclen @ {       trace++
                                  trace--
                                    .
                                    .
                                  newestword             <- dict @
          vocfree {               *** free vocab space
          hfree       {           Heap space if HEAP.F loaded
                      -------------------------------------<- stkhi @
                                  Parameter stack moving down
     stklen @         {                                  <- sp@ (a6)
                                  *** free stack space
                      -------------------------------------<- rtnhi @
                                  Return stack moving down
                                  Local arrays,
                                  Local vars,            <- (a2)
                                  Linked data,
                                  Last linked data       <- link@ (a3)
                      -------------------------------------   <- r@ (a7)
                                  4096 byte safe area
                      -------------------------------------

          free        {           *** free space

                                  Original vocab loaded with<- vocpnt
                                    FORTH
                                  Newly read tokens
                      -------------------------------------<- next@ (a5)
          used        {           Object code
                      -------------------------------------<- org ($4000)
```

Suggested format for .f files.

For a file containing vocabulary ‘fred’, using the supplementary files screen.f’, ‘file.f’ and ‘float.f’, the fred.f file might look like this:

```
forget? fred          ( if fred is already loaded, forget it )

load? file file.f     ( if the vocabulary ‘file’ is not loaded, load it )
load? floats float    ( if ‘floats’ is not loaded, load float.f )
load? screen screen

vocab fred

{word definitions...}

interactive                       ( that’s the end! )
```

Quick reference:

```
!
"
%abase
%achk
%ccase
%struct
’
’c
(
(.)
(do)
*
*q
+
+!
+loop
,
,l
-
-loop
.
."
.s
.str
/
/mod
/modq
/q
0<
0<=
0<>
0=
0>
0>=
1+
1-
16*
16/
2*
2+
2-
2/
256*
256/
2drop
2dup
2swap
3+
3-
4*
4+
4-
4/
5+
5-
6+
6-
7+
7-
8*
8+
8-
8/
:
:-
::
;
<
<<
<=
<>
<r
=
>
>!l
>!w
>=
>>
>>x
>altc
>local
>r
>struct
>uses
?>
@
]
abort
abs
addr.of
again
align.w
allot
aload
altc
altc+
altc-
altc?
amul
and
args
array
asc
automatic
base
base0
base1
begin
bell
beq>
bmove
bne>
bra>
byte:
bytes:
c!
c?
c@
call
case
case@
cget
cget?
chk+
chk-
clear
code
compile
constant
cont
copy
create
crlf
decimal
default
delete
delim?
dexpect
dfind
dict
dir
dirs
do
docheck
drop
dropcase
dropdo
dropr
dsearch
dup
echo+
echo-
edit
efcopy
else
emit
endcase
ends
eor
error
esc
exec
expect
f0/
f1/
fclose
fcopy
fget
fgetc
find
fkey
flip
float:
fopen
forget
forget?
forward
fput
fputc
free
goto
header
help
hex
i
i+
if
if0
imm?
immediate
immword
incase?
instruct?
interactive
j
join
label:
larray
lastdo
line!
line?
line@
link
link@
literal
load
load?
local>
local?
locals|
logoff
long:
loop
makes
mark
mark++
mark--
max
mem
min
mlink
mod
modq
move
msg1
newline
next!
next@
next_vocab
next_word
nip
nop
not
number
or
org
over
pad
pick
prompt
r1
r2
r3
r4
r@
rd/
rename
repeat
resize
return
roll
rot
rot-
rp@
rtnhi
rts>
rvsoff
rvson
save
sbod
search
sexec
shell
size.of
sp@
split
split0
ssasm
ssddutil
stkhi
stklen
stop
string
string:
struct
swap
syscall
system
tab
then
to
token
trace+
trace++
trace-
trace--
tuck
turnkey
type
type_word
unique+
unique-
unlink
until
until0
used
uses
variable
vdel
vlist
vocab
vocfree
vochi
voclen
vocmove
vocpnt
vocresize
vocused
w!
w@
w@x
whazzat
while
while0
word1
word:
words
|
~
```

```
!
"
%abase
%achk
%ccase
%chksize
%maxoffs
%oneoffs
%struct
'
'c
(
(.)
(array)
(dims)
(do)
(rseed)
(struct)
(unlink)
(uses)
*
*q
+
+!
+loop
,
,l
-
-loop
.
."
.card
.s
.str
.token
/
/mod
/modq
/q
0<
0<=
0<>
0=
0>
0>=
1+
1-
16*
16/
2*
2+
2-
2/
256*
256/
2drop
2dup
2swap
3+
3-
4*
4+
4-
4/
5+
5-
6+
8+
8-
8/
:
:-
::
;
<
<<
<=
<>
<r
=
=>
>
>=
>>
>>x
>altc
>local
>r
?>
@
]
abort
abs
addr.of
again
align.w
allocfmem
allocmem
allot
aload
altc
altc+
altc-
altc?
amul
and
args
array
arraym
asc
automatic
base
base0
base1
begin
bell
beq>
bmove
bne>
bra>
byte:
bytes:
c!
c?
c@
call
case
case@
cd
cget
cget?
chk+
chk-
clear
copy
create
crlf
d!
d@
dauto
debug+
debug-
decimal
default
delete
delim?
dexpect
dfind
dict
dir
dirs
do
docheck
drop
dropcase
dropdo
dropr
dsearch
dup
dvar
echo
echo+
echo-
edit
efcopy
else
emit
end.of
endcase
ends
entries
eor
error
esc
exec
execute
expect
fclose
fcopy
fget
fgetc
find
fkey
flip
float:
fopen
forget
forget?
forward
fput
fputc
free
freemem
goto
header
help
hex
i
i+
if
if0
interactive
j
join
label:
larray
larraym
lastdo
libload
line?
link
link@
literal
literal!
literal@
litlea
llce
load
load?
loc@
local>
local?
locals|
logon
long:
loop
makes
mark
mark++
mark--
max
mem
min
mkdir
mlink
mod
modq
move
msg1
name
needed
newline
next
next_vocab
next_word
nextend
nip
nop
not
number
one.of
option
or
or.of
org
over
pad
pick
prompt
r1
r2
r3
r4
r@
random
rename
repeat
resize
rp@
rtnhi
rts>
rvsoff
rvson
save
search
shell
size.of
sp@
split
split0
ssasm
ssdutil
stkhi
stklen
stop
string
string:
struct
swap
syscall
sysfree
system
tab
then
ticks?
tline
to
tokbuf
token
trace+
trace++
trace-
trace--
tuck
turnkey
type
type_word
unique+
unique-
unlink
until
until0
used
uses
variable
vdel
vlist
vname
vocab
vocfree
vochi@
voclen
voclo
vocused
w!
w@
w@x
whazzat
while
while0
word1
word:
words
|
~
```

6-
7+
7-
8*
code
compile
constant
cont
imm?
immediate
immword
instruct?
return
roll
rot
rot-

3
File manipulation words, in file.f

%ferror (-- f)
See also: ferror+ ferror-
Automatic variable set to true if automatic reporting is turned on; set false if errors ignored.

.fbakup (buf -- buff+offs)
.fblock#
.fdate
.flen
.fload
.fname
.ftype
.funused
See also: fstat fstatus
Field names for file status structure. See ‘fstat’ for structure description.

closeall (--)
See also: fclose fopen eof?
Close all currently open files.

d$cent: (-- 3)
d$con: (-- 0)
d$sa: (-- 1)
d$sb: (-- 2)
Constants representing file descriptors for devices.

eof? (fdesc -- f)
See also: ferror? ferror+ ferror-
If {fdesc} represents a file at EOF or {fdesc} is invalid, {f} is true. Otherwise, {f} is false.

f$abs (-- 0)
f$eof (-- 2)
f$skip (-- 1)
See also: fseek
Constants for different types of seek mode.

f$char (-- n)
f$comp (-- n)
f$exec (-- n)
f$pict (-- n)
f$temp (-- n)
f$text (-- n)
f$xtxt (-- n)
See also: fcreate
Each of these is an obsolete file type, used in 1616OS versions 1.x. In the fcreate system call, a file type of 0 is now used.

f$r/w (-- 2)
f$read (-- 0)
f$write (-- 1)
See also: fopen fcreate
File opening modes.

f$r/w indicates a random-access file that may be both written to and read from.
f$read indicates a file which may be read from but not written to.
f$write indicates a file to be appended to.

fclose (fdesc -- status)
See also: ferror+

Close a file. Status will be zero if the file was successfully closed, otherwise will be negative.

fcreate (ld.addr (type) name -- fdesc)
See also: ferror+

Create a file. If the file is non-executable, the load address should be zero. For systems with 1616os 2.x, {type} should be zero. {name} should be a null-terminated string of fewer than 32 characters. If an error occurred while creating the file, {fdesc} will be negative.

fdout (n -- n+16)
See also:

Convert a 1616 file descriptor to a FORTH file descriptor. FORTH disk files are always offset by 16 to ensure that they are compatible with all system calls. Only devices will have file descriptors smaller than 16.

ferrmes (-- addr)
See also: ferror+

A fifty byte string for error messages.

ferror (error -- addr)
See also: ferrmes

Convert a negative error status to a string.

ferror+ (--)
See also: ferror- ferror? ferror ferrmes

Turn on automatic error checking. After this word is executed, any errors occurring while using FORTH file words will cause an abort with appropriate error message.

ferror- (--)
See also: ferror+

Turn off error checking. After this word is executed, the user (or a FORTH program) must check for errors after every file operation.

ferror? (status -- status)
See also: ferror+

If {status} is negative and error checking has been turned on with ‘ferror+’, abort with an appropriate error message.

fget (fdesc -- n)
See also: fput fget fputc fgetc putchar getchar

Get a 32 bit (four byte) number from {fdesc}. Numbers will be input most significant byte first.

fgetc (fdesc -- ch)
See also: fget fput fputc putchar getchar ferror+

Get a byte from {fdesc}. If an error occurs, {ch} will be negative.

fgets (buf fdesc --)
See also: fget eof?
Read a crlf-terminated string from {fdesc} into {buf}.
’fgets’ will not detect errors, but a null string will
be stored at {buf}. If error detection is required,
use the word ‘eof?’.

fkill (name -- status)
See also: fcreate fopen
Delete a file.

fopen (mode name -- fdesc)
See also: f$read f$write f$r/w fclose eof? ferror+
Open a file.

fpos (fdesc -- pos/error)
See also: fseek eof?
Find the position (in bytes read) of the current file.

fput (date fdesc -- status)
See also: fget
Write a 32 bit number to {fdesc}.

fputc (char fdesc -- status)
See also: fgetc getchar
Write a character to {fdesc}.

fputs (buf fdesc --)
See also: fgets putcrlf
Write a null-terminated string to {fdesc} from {buf}.
’fputs’ does not write a crlf after the string; use
’putcrlf’ to do so.

fread (#bytes buf fdesc -- status)
See also: fwrite
Read {#bytes} to {buf} from {fdesc}.

frename (oldname newname -- status)
See also: fkill fcreate
Rename a file.

fseek (seekmode offset fdesc -- newpos/status)
See also: fpos f$abs f$eof f$skip
Seek to a position in a file. If newpos/status is negative,
an error has occurred.

fstat (64bytebuf name -- code)
See also: fstatus
Get the status of a file.

fstatus (--)
See also: fstat .fbakup
Structure definition used for accessing a file’s status
record. Its definition is as follows:

```
struct fstatus

    32      bytes: .fname
    8       bytes: .fdate
             word: .ftype
             long: .fload
             long: .flen
```

```
                    word: .fbakup
                    word: .fblock#
          10        bytes: .funused
       ends
```

fwrite (#bytes buf fdesc -- status)
See also: fread
Write {#bytes} to {fdesc} from {buf}.

getchar (-- char/error)
See also: putchar stdin set_sip
Get a character from standard input. This word is similar to ‘cget’, but will not return an ‘altc’.

istatus (-- #chars)
See also: ostatus stdin set_sip
Return the number of characters ready at stdin. If standard input is a file, this call will usually return 1.

ostatus (-- #chars)
See also: istatus stdout set_sip
Return the number of characters needed to fill the output buffer of stdout.

putchar (chr -- status)
See also: fputc
Write a character to stdout.

putcrlf (fdesc --)
See also: fputs putc
Write a crlf to {fdesc}.

set_sip (fdesc -- old_stdin)
See also: set_sop istatus stdin
Set standard input.

set_sop (--)
See also: set_sip ostatus stdout
Set standard output.

sgetc (fdesc -- #chars)
See also: istatus
Return number of characters ready at {fdesc} for reading.

sputc (fdesc -- #chars)
See also: ostatus
Returns number of characters needed to fill the output buffer of {fdesc}.

stdin (-- fdesc)
See also: stdout set_sip istatus sgetc
Returns file descriptor of standard input.

stdout (-- fdesc)
See also: stdin set_sop ostatus sputc
Returns file descriptor of standard input.

ty (--)
Usage: ty {filename}
See also:
Prints a file to the screen. ‘ty’ may only be used in direct mode.

ungetc (char --)
See also: fgetc getchar
Put a character back onto standard input. Only one character at a time may be put back into the input.

Summary

%ferror
.fbakup
.fblock#
.fdate
.flen
.fload
.fname
.ftype
.funused
closeall
d$cent:
d$con:
d$sa:
d$sb:
eof?
f$abs
f$char
f$comp
f$eof
f$exec
f$pict
f$r/w
f$read
f$skip
f$temp
f$text
f$write
f$xtxt
fclose
fcreate
fdout
ferrmes
ferror
ferror+
ferror-
ferror?
fget
fgetc
fgets
fkill
fopen
fpos
fput
fputc
fputs
fread
frename
fseek
fstat
fstatus
fwrite
getchar
istatus
ostatus
putchar
putcrlf
set_sip
set_sop
sgetc
sputc
stdin
stdout
ty
ungetc

4
Floating point words, in float.f

!f (ne nm addr --)
See also: @f varf constf
Store 6 bytes at {addr}

$1/e
$1/log10
$e
$log10
$pi
$pi*2
$pi/2

*f (ne1 nm1 ne2 nm2 -- xe xm)
See also: /f +f -f >=<f
Multiply two floating point numbers.

+!f (ne nm addr --)
See also: @f !f +f varf
Add f.p. to that at {addr}

+c? (n1 n2 -- n1+n2 f)
See also: <<?
Add n1 and n2. f is true if a carry was generated.

+f (ne1 nm1 ne2 nm2 -- xe xm)
See also: -f *f /f
Add two f.p. numbers.

+real (ne1 nm1 ne2 nm2 -- xe xm)
See also: +f
Add two (unsigned) f.p. numbers (should not be used).

-f (ne1 nm1 ne2 nm2 -- xe xm)
See also: +f *f /f
Subtract fp#2 from fp#1.

-real (ne1 nm1 ne2 nm2 -- xe xm)
See also: -f
Subtract two (unsigned) f.p. numbers (should not be used).

.f (ne nm --)
See also: .fe .fu
Print out a floating point number. If the number is within range, it is printed out in decimal format. Otherwise, it is printed in exponential notation.

.fe (ne nm --)
See also: .f .fu
Print a floating point number in exponential notation, with leading sign, nine significant digits and signed exponent.

.fu (ne nm #dp #sigfigs --)
See also: .f .fe
Print a floating point number with dp decimal places and sigfigs significant digits.

.int (n --)
See also: .fe
Print a 2-digit integer with leading sign. This word prints the exponent of ‘.fe’.

/f (ne1 nm1 ne2 nm2 -- xe xm)
See also: *f +f -f
Divide fp#1 by fp#2.

/real (ne1 nm1 ne2 nm2 -- xe xm)
See also:
Divide two (unsigned) f.p. numbers (should not be used).

0maxf (ne nm -- ne nm or 0 0)
See also: maxf minf
If f.p. number is positive, leave it. Otherwise,
return f.p. 0.

1/10f Constant

10*f (ne nm -- xe xm)
See also: 10f 2*f 2/f 1/10f
Multiply f.p. number by 10.

10f (-- ne nm)
See also: 10*f $e
Constant: 10

1f (-- ne nm)
See also: 10f $e
Constant: 1

2*f (ne nm -- xe xm)
See also: 2/f 10*f
Multiply f.p. number by 2.

2/f (ne nm -- xe xm)
See also: 2*f 10*f
Divide f.p. number by 2.

2dupf (ne1 nm1 ne2 nm2 -- ne1 nm1 ne2 nm2 ne1 nm1 ne2 nm2)
See also: dupf swapf dropf
Duplicate two f.p. numbers.

<<? (n -- n<< cnt)
See also: +c?
Shift n left until the sign bit is set; leave the count of the number of shifts on the top of the stack.

<<f (ne nm cnt -- ne’ nm’)
See also: >>f 2*f 2/f
Multiply by 2^cnt

<f <=f <>f (ne1 nm1 ne2 nm2 -- f)
See also: >f =f >=<f
f is true if fp#1>fp#2.

=f (ne1 nm1 ne2 nm2 -- f)
See also: <f >f >=<f
f is true is fp#1>fp#2.

>=<f (ne1 nm1 ne2 nm2 -- f)
See also: <f >f =f
f is +ve if fp#1 > fp#2
f is 0 if fp#1 = fp#2
f is -ve if fp#1 < fp#2.

>=<real (ne1 nm1 ne2 nm2 -- f)
See also: >=<f
Unsigned version of >=<f

>f >=f (ne1 nm1 ne2 nm2 -- f)
See also: >=<f <f =f
f is true if fp#1>fp#2

@f (addr -- ne nm)
See also: !f +!f varf
Load a six byte f.p. number from {addr}.

_/f (n1 n2 -- x1)
See also: /f
No longer in use.

absf (ne nm -- +ne +nm)
See also: ~f
Returns the absolute value of f.p. number.

acosf (ne nm -- arctan(ne nm))
See also: tanf sinf cosf

asinf (ne nm -- arctan(ne nm))
See also: tanf sinf cosf

atnf (ne nm -- arctan(ne nm))
See also: tanf sinf cosf

autof (--)
Usage: autof {token}
See also: constf varf @f !f +!f
Create a floating point variable. When {token} is executed, it will leave a f.p. number on the stack. Use ‘to’ to change the variable’s value.

chkexp (ne nm -- ne nm)
See also: *f /f
’?Overflow error’ if ne>2047. Underflow is not checked for.

clip (ne nm -- n)
See also: fix
Truncate the decimal portion of a number (that is, all numbers are rounded towards zero).

constf (ne nm --)
See also: varf f"
Usage: constf {token}
Define a f.p. constant.

cosf (ne nm -- cos(ne nm))
See also: sinf tanf acosf
Returns the cosine of a number.

digit? (chr -- f)
See also: .f f"
f is true if {chr} is a decimal digit.

dropf (ne nm --)
See also: dupf 2dupf swapf
Drop a f.p. number from the stack.

dupf (ne nm -- ne nm ne nm)
See also: 2dupf dropf swapf
Duplicate a f.p. number.

expf (xe xm -- ne nm)
See also: sinf $e $1/e
Return e^x.

f" (-- ne nm)
See also: .f scanf float
Treat the next token in the FORTH input stream as a f.p. number; leave its value on the stack (can be used in either immediate mode or definitions).

fcoeff (n -- addr)
See also: expf sinf
Table of coefficients for calculation of sin and exp.

fix (ne nm -- n)
See also: clip float
Round a f.p. number down (that is, positive numbers towards 0, negative numbers towards negative infinity).

float (n -- ne nm)
See also: fix f"
Convert an integer to f.p. format.

log10f (ne nm -- ne’ nm’)
See also: expf $e $1/e log10f
Calculate log base 10

logf (ne nm -- ne’ nm’)
See also: expf $e $1/e log10f
Calculate log base e

lsines (n -- addr)
See also: sinfq sinfq_init
Local array used for the quick sine routines. ‘lsines’ must be initialized by sinfq_init before use by sinfq.

maxf (ne1 nm1 ne2 nm2 -- ne nm)
See also: minf 0maxf
Returns the maximum of fp#1 and fp#2.

maxint (-- $80000000)
See also: minint
Returns minimum integer (corresponds to sign bit in the mantissa of a f.p. number)

minf (ne1 nm1 ne2 nm2 -- ne nm)
See also: maxf 0maxf
Returns the minimum of fp#1 and fp#2.

minint (-- $7fffffff)
See also: maxint
Returns maximum integer (corresponds to the unsigned portion of the mantissa)

normalize (ne1 nm1 -- ne1+offs nm1<<)
See also: float +f -f
Changes an unsigned f.p. number to have a 1 in the most-significant bit, changing the exponent accordingly.

odd? (n -- f)
Returns true if n is odd.

oflow (--)
See also: chkexp
Force an overflow error. This word is a link for the machine language library.

scanf (addr -- ne nm)
See also: f"
Transform the character string at {addr} to a f.p. number.

sinf (ne nm -- xe xm)
See also: expf sinfq sinfq_init
Calculate the sine of f.p. number in radians.

sinfq (n -- x)
See also: sinfq_init lsines
Return the integer sine (-32767 to +32767) of n. The length of one cycle of this sine wave is 1024. sinfq_init must have been called before this word will work.

sinfq_init (--)
See also: sinfq
Initialize the sine table used by sinfq. The table stored in ‘lsines’ will be deallocated when the word containing ’sinfq_init’ terminates.

sqrt (ne nm -- xe xm)
See also: sinf expf
Calculate the square root of a f.p. number.

swapf (ne1 nm1 ne2 nm2 -- ne2 nm2 ne1 nm1)
See also: dupf 2dupf dropf
Swap two f.p. numbers.

tanf (ne nm -- tan(ne nm))
See also: cosf sinf atanf
Returns the tangent of a number.

varf (--)
Usage: varf {token}
See also: autof constf @f !f +!f
Create a floating point variable. When {token} is executed, it will leave the address of a six byte area ready for storing a f.p. number.

y^x (ne nm me me -- ne’ nm’)
Evaluate n^m.

~f (ne nm -- xe xm)
See also: -f +f absf
Negate a f.p. number.

Summary

!f
$1/e
$1/log10
$e

-f
-real
.f
.fe

<<f
<=f
<>f
<f

constf
cosf
digit?
dropf

minf
minint
normalize
odd?

$log10
$pi
$pi*2
$pi/2
(.fu)
(atnf)
(atnf1)
(atnf2)
(fo)
(scanf)
*f
+!f
+c?
+f
+real
.fu
.int
/f
/real
0maxf
1/10f
10*f
10/f
10^x
10f
1f
2*f
2/f
2dupf
<<?
=f
>=<f
>=<real
>=f
>>f
>f
@f
_/f
absf
acosf
asinf
atnf
autof
chkexp
clip
dupf
expf
f"
fcoeff
fix
float
floats
get_exponent
get_frac_part
get_int_part
log10f
logf
lsines
maxf
maxint
oflow
scanf
sinf
sinfq
sinfq_init
sqrt
swapf
varf
y^x
~f

Floating point format:

Mantissa: MSB (bit 31) is sign - 1=-ve, 0=+ve
for operations, MSB is assumed to be 1 (unless number is zero) other bits in mantissa form a binary weighted fraction

Exponent: Exponent is zero iff number is zero.
Otherwise, value is .mantissa*2^(exponent-1024)
Example:
Mantissa: $50000000
(01010000 00000000 00000000 00000000)
Exponent: 1025

Sign bit is 0, so number is positive.
Implicit 1 makes mantissa = .1101
Exponent multiplies mantissa by 2^(1025-1024)=2
Resulting number = 1.101 = 1.625

5
Turn off VIA interrupts, irqkill (--)

See also: install

imsg (-- address)

A variable whose value is left on the stack for the currently executing interrupt routine. The value left on the stack by the interrupt routine is in turn left in imsg.

See also: install irqkill istack

install (rate address --)

Installs an interrupt service routine, using the VIA timer interrupts.

Before using this word, the following initializations are necessary:

the variable istack must point to enough space for a parameter stack for the routine

imsg can be given a value to pass to the routine

rate is inversely proportional to the calling rate: as a rough guide, 3748 for rate corresponds to about 50 hz.

See also: istack imsg irqkill

istack (-- address)

Points to a pointer to the parameter stack used by the interrupt routine.

See also: install imsg irqkill

6
Screen/Graphics words.

(640mode) (0/1 --)
See also: open640 open320 close320 close640

1616 syscall to clear/Set 640 pixel mode.

(cursor) (mask enable rate --)
See also: cursor_on cursor_off cursor?

1616 syscall to set the cursor mode.

(dotmode) (dotmode --)
See also: dot_xor dot_write dot_or dot_and

1616 syscall to set the point plotting mode.

(move_wind) (mode buff --)
See also: wopenq wcloseq (wget) wopen wclose (wsize)

1616 syscall to move current window contents to/from {buff}

(open640) See also: open640 close640 open320 close320

Utility word to open a new screen mode.

(pall) (--)
See also: .pal0 .pal1 .pal2 .pal3 make_pallette

Structure used for pallete definitions.

(pallette) (colour pallettepos --)
See also: make_pallette

1616 syscall to set a pallette entry.

(rseed) (-- addr)
See also: random

Variable containing current 32-bit random seed value.

(w320) (-- w.addr)
See also: open320 open640 close320 close640 make_window

Window structure for default 320 pixel window.

(w640) (-- w.addr)
See also: open320 open640 close320 close640 make_window

Window structure for default 640 pixel window.

(wclose) (--)
See also: wclose wcloseq close320 close640 (wopen)

Close a window without restoring original screen contents.

(wget) (w.addr --)
See also: (move_wind)

Put the contents of the window {w.addr} is defined for
into a buffer on the return stack, setting the appropriate
fields in {w.addr}

(wind) (--)
See also: make_window

Structure definition of a window.

(window) (w.addr/0/1 -- w.addr)
See also: make_window w_default w_default?
w_current? w_reset

1616 syscall to set the current window.

(wopen) (w.addr --)
See also: (wclose) wopen wclose

Open a screen window without saving its contents.

(wput) (--)
See also: (wget) wclose

Restore a window's original contents.

(wset) (w.addr --)
See also: (window)

Set the current window.

(wsize) (w.addr -- size.in.bytes)
See also: (wget) (wput)

Returns the number of bytes needed to store the
contents of {w.addr}.

(x) (-- x-value)
See also: line_to (y) dot_pos

Last x-value used in a line_to or dot_pos.

(y) (--)
See also: line_to (x) dot_pos

Last y-value used in a line_to or dot_pos.

(w.addr -- w.addr+offs)
See also: make_window (wind) wopen wclose

Words to access the fields of a window structure.

.bg_col	word:	background colour mask
.curs_x	word:	cursor pos x rel. to current window
.curs_y	word:	cursor pos y rel. to current window
.fg_col	word:	foreground colour mask
.oldwin	longword:	link to previously open window
.wsave	longword:	link to buffer containing previous screen contents
.xend	word:	x right boundary+1
.xstart	word:	x left boundary
.yend	word:	y bottom boundary+1
.ystart	word:	y top boundary

(p.addr -- p.addr+offs)
See also: make_pallette pallette? show_pallette

Words to access the four colours in a pallette.
Each is a byte in length.

.pal0
.pal1
.pal2
.pal3

640? (-- f)
See also: open640 close640

f is true (1) if the current screen mode is 640 pixels.

abort (--)
See also: edit shell

Same as kernel ‘abort’, except that ‘abort’ now closes down any open windows, turns the cursor on and sets the screen mode to 640.

bd_col (colour --)
See also: fg_col bg_col

Set the border colour.

bg_col (colourmask --)
See also: bd_col fg_col .bg_col bg_col?

Set the screen backgroud colour (will not affect text already on the screen).

bg_col? (-- colourmask)
See also: bg_col

Returns the current background colour.

close320 (--)
See also: open320

Closes the currently open 320 pixel screen and returns to the previous window, restoring the previous graphics mode.

close640 (--)
See also: open640

Closes the currently open 640 pixel screen and returns to the previous window.

colour (colour --)
See also: dot line_to colour? colourdot

Sets the current graphics drawing colour.

colour? (-- colour)
See also: colour dot line colourdot

Returns the current graphics colour.

colourdot (colour y x --)
See also: dot line

Plots a dot in colour {colour}.

cursor? (-- f)
See also: cursor_on cursor_off cursor_pos?

f is true (-1) if the cursor is currently on.

cursor_off (--)
See also: cursor_on cursor? cursor_pos

Turn cursor off.

cursor_on (--)
See also: cursor_off cursor? cursor_pos

Turns cursor on.

cursor_pos (y x --)
See also: cursor_pos? cursor?

Set the cursor position on the screen.

cursor_pos? (-- y x)
See also: cursor_pos cursor?

Returns the cursor position on the screen.

dot (y x --)
See also: dot? colour colourdot line line_to

Set a point on the screen in the current graphics colour.

dot? (y x -- colour)
See also: dot colour colourdot line line_to

Returns the colour of the dot at (x,y).

dot_and
dot_or
dot_write
dot_xor (--)
See also: dot line (dotmode)

Set cursor plot mode.

dot_pos (y x --)
See also: line_to

Set position for subsequent ‘line_to’

downarrow (-- ch)
See also: uparrow leftarrow rightarrow

Returns the ascii value of the downarrow character.

edit (--)
See also: abort shell

The original kernel ‘edit’, except that the current screen is saved & 640 pixel mode entered when edit is invoked.

fg_col (colourmask --)
See also: bg_col .fg_col

Set text foreground colour.

fg_col? (-- colourmask)
See also: bg_col? .bg_col

Get text foreground colour.

flushc (--)
See also:

Flush any characters from the keyboard buffer.

home (--)
See also:

Home the cursor (put it at (0,0) relative to current window)

line (y1 x1 y2 x2 --)
See also: line_to colour dot_pos

Draw a line in the current graphics colour from (x1,y1) to (x2,y2).

line_to (x y --)
See also: line dot_pos

Draws a line from (x,y) to the previous dot_pos. (x,y) becomes the next dot_pos.

make_pallette (p3 p2 p1 p0 --)
See also: make_window .pal0 show_pallette pallette?
Usage: n3 n2 n1 n0 make_pallette {token}

Makes a pallette file for setting the pallette in 640 pixel mode.

make_window (yend xend ystart xstart --)
See also: wopen wclose
Usage: {ye} {xe} {ys} {xs} make_window {token}

Makes a window structure called {token}.

open320 (--)
See also: wopenq close320 wopen open640

Opens the whole screen as a window and sets the current graphics mode to 320 pixels. ‘open320’s may be nested.

open640 (--)
See also: wopenq close640 wopen open320

Opens the whole screen as a window and sets the current graphics mode to 640 pixels. ‘open640’s may be nested.

pallette? (-- pall.addr)
See also: make_pallette .pal0

Returns the address of the default pallette structure.

random (n -- rand#)
See also: (rseed)

Returns a random number between 0 and (n-1), inclusive. ’random’ may only be used to generate random numbers between 0 and 65535, although the cycle length of the generator is 2^32.

rightarrow (-- chr)
See also: leftarrow downarrow uparrow

Returns the ascii value of the rightarrow character.

shell (--)
See also: abort edit

The same as the kernel ‘shell’, except when screen is loaded ’shell’ sets the screen mode to 640 pixels upon entry and restores the FORTH window after ‘quit’.

show_pallette (pall.addr --)
See also: make_pallette pallette? .pal0

Put the pallette structure at pall.addr into the 1616’s hardware pallette register.

ticks? (-- n)
See also: wait

Returns the number of 50Hz ticks counted since 1616 turn-on. This word is useful for timing things.

Example:

```
: time1000000
  ticks
  1000000 1 do loop ( loop 1000000 times )
  ticks - ~ ( calculate ticks taken )
  . ."  ticks taken to loop 1000000 times." crlf ;
```

uparrow (-- chr)
See also: downarrow leftarrow rightarrow

Returns the ascii character for uparrow.

w_copy (src.w.addr dst.w.addr --)
See also: wopenq make_window

Copies the contents of {src.w.addr} to {dst.w.addr}. This word is usually used to copy template windows into the return stack, so that the same window may nested several times.

w_current? (-- w.addr)
See also: wopen make_window w_reset w_default? w_default

This word returns the window structure currently in use. As well as returning its address, this word also causes the current cursor position etc. to be copied into the window structure.

w_default (--)
See also: w_reset w_default? wopen abort

This word resets the current window to be the default 1616 window, so if the current window has become corrupted, w_default will restore the 1616 window to be the original.

w_default? (-- w.addr)
See also: w_default w_reset wopen

Returns the address of the default 1616 window.

w_reset (--)
See also: w_default

Resets the current window to be the default, and sets the cursor position to (0,0), resets the background colour and sets the margins to their usual state.

wait (n --)
See also: ticks

Waits for {n} ticks, or n/50 seconds.

wclose (--)
See also: wopen open320 wopenq (wclose)

Closes the currently open window and restores the previous window contents. If no previous window was opened, or an attempt is made to perform this in direct mode, a crash is possible. ‘wclose’ does not free the space allocated on the return stack for the screen buffer; this space will only be freed when either an ‘unlink’ is executed or the word containing ‘wopen’ terminates.

wcloseq (--)
See also: wclose close320 (wclose)

Closes the window currently opened with ‘wopenq’ and restores the previous window contents. See ‘wclose’ for more info. Note that a window opened with ‘wopenq’ requires two ‘unlink’s to free the space allocated to it.

wopen (w.addr --)
See also: wopenq open320 wclose

Opens the window structure pointed to by w.addr, saving the screen contents. wopen’s with the same w.addr may not be nested.

wopenq (--)
See also: wopen open320 wcloseq

Open a new window with the same characteristics as the current window, saving the screen contents. wopenq’s may be nested.

Summary

(640mode)
(cursor)
(dotmode)
(move_wind)
(open640)
(pall)
(pallette)
(rseed)
(w320)
(w640)
(wclose)
(wget)
(wind)
(window)
(wopen)
(wput)
(wset)
(wsize)

(x)
(y)
.bg_col
.curs_x
.curs_y
.fg_col
.oldwin
.pal0
.pal1
.pal2
.pal3
.wsave
.xend
.xstart
.yend
.ystart
640?
abort

bd_col
bg_col
bg_col?
close320
close640
colour
colour?
colourdot
cursor?
cursor_off
cursor_on
cursor_pos
cursor_pos?
dot
dot?
dot_and
dot_or
dot_pos

dot_write
dot_xor
downarrow
edit
fg_col
fg_col?
flushc
home
line
line_to
make_pallette
make_window
open320
open640
pallette?
random
rightarrow
shell

show_pallette
ticks?
uparrow
w_copy
w_current?
w_default
w_default?
w_reset
wait
wclose
wcloseq
wopen
wopenq

Window examples.

1. Opening a simple full-screen window

```
            : wtest1
              wopenq                    ( opens window and saves contents
)
              clear
              1000 1 do i . loop        ( fill the screen with garbage )
              cget drop                 ( wait for a key & drop it )
              wcloseq                   ( closes window & restore screen
contents )
            ;
```

2. Opening a small message window

```
            15 60 10 20 make_window messagewindow( make a window )
            hex 5555 constant col1 decimal( constant for background col )
            col1 messagewindow .bg_col w!( set its bg colour )

            : wtest2
              messagewindow wopen       ( opens 5x40 window & saves con-
tents )
              clear
              crlf crlf ."  This is a message."
              cget drop
              wclose
            ;
```

3. Plotting points in 320 pixel mode

```
            : plot3
              open320                   ( set screen mode to 320 pixels )
              wopenq                    ( save screen contents )
              clear                     ( clear screen )
              cursor_off                ( turn off annoying cursor )
              15 colour                 ( set default colour )
              begin
                          200 random    ( get random y )
                          320 random    ( get random x )
                          dot           ( plot point )
                          cget?         ( key pressed ? )
              until                     ( if no, back to begin )
              drop                      ( drop keypress )
              cursor_on                 ( on again! )
              wcloseq                   ( must close windows in right
order )
              close320
            ;
```

For more examples, see the numerous demo programs.

7
The files on the Forth disk are:

FORTH.S	- Source code for FORTH
FORTH	- Assembled code
WORDS.F	- Kernel word definitions
SCREEN.F	- Screen words
FLOAT.F	- Floating point words
FILE.F	- File manipulation words
START.TXT	- Little tutorial
KERN.TXT	- Kernel documentation
FLOAT.TXT	- Float documentation
FILE.TXT	- File documentation
SCREEN.TXT	- Screen documentation
MAND3.F	- Demo program - start with ‘mshell’
HEAP.F	- Heap management words
DEMO.F	- Demo program - start with ‘demostart’
IRQ.F	- IRQ demos
KERN.F	- Help helper
FSORT.F	- File sorter
SPHERE.F	- Demo program - start with ‘spheres’
HELP.F	- Help program - ‘help {word} [{file}]’
STRING.F	- String comparisons/copies
SIEVE.F	- Prime number calculator - start with ‘sieve’
GRAV.F	- Gravity demo - start with ‘grav’
README	- What you read now
MEM	- Print free memory (transient)
TY	- Type a file (transient)
SESAME	- SSASM a file & leave listing file in errors.
SSASM	- SSASM
MSCREEN	- Screen for mand3.f

To get FORTH started, type:

```
*FORTH words.f
```

To get an idea of what it does,

```
>load demo.f
>demostart
```

Enjoy!

8
The Last Read Me file

Here is a working version of my FORTH for the new version of the operating system. The documentation has not yet been fully upgraded; however, ninety percent of it is correct.

To start it up, you must ‘cd /f0/bin’ before executing ‘ff’, which allocates 64k of space for FORTH, and the loads and executes it. To load most FORTH stuff, you must be in the /f0/forth directory. It would be nice to add a search path for FORTH later on.

I’ve written a bit more since the last version you would have seen. The floating point routines have been debugged and speeded up somewhat, and a tiny Pascal->FORTH compiler thingy and a small LISP interpreter have been hacked together.

I’ve found only three actual bugs in the OS which you probably already know about: firstly, cd doesn’t always work, and secondly, there are some crashes I’ve been in which produce another error each time the reset button is pushed; turning the machine off was the only solution. The third bug is that the system fails to close files after a program finishes.

Other nice features worth having:

For some screen applications (that is, games and graphics), the clock is a pain: however, turning it off by freeing its interrupt slot is not the right solution, as it cannot be turned back on. Does the MRD stuff allow you to turn things like this off and on ?

Is it possible to guarantee the status of registers not involved in syscalls? I haven’t checked since the first version of the roms I had, but one or two syscalls trashed registers: having to save all registers in use before every syscall is wasteful.

The new graphics routines in 80 column mode are lovely and fast: however, the 40 column routines don’t seem as fast. (Perhaps this is the fault of my FORTH, not your routines).

The memory manager gets very slow when lots of little blocks have been allocated, especially when freeing them all after completing a program.

It would be nice to be able to open a file given a search path, or even just open a file on the execution path.

’Bye,

Peter.

PS: I’ll be contactable at home from about the first of July until the
fourteenth on (062) 86 2964.

Table of Contents