XFORTH TECHNICAL MANUAL

Table of Contents

Preface

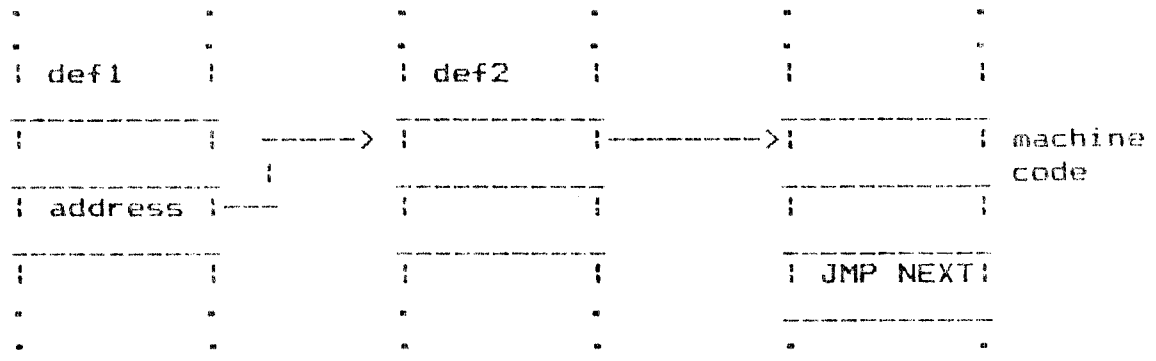
This manual is not intended for the novice. It is not really suitable for someone who does not have a good grasp of the fundamentals of FORTH programming, at the very least to the extent that he understands the xForth introductory manual and all of its appendices very well. The main part of the manual is the glossary, but the first part contains useful information such as memory maps.

If you have difficulty with this manual, try reading one of the excellent books which are now becoming available, such as 'Threaded Interpretive Languages' by Ronald Loeliger, published by Byte Books (a division of McGraw-Hill).

Chapter 1

The inner interpreter

Forth uses 'indirect threaded code' to achieve its fast and compact form. All this means is that the definitions of most Forth words - those defined via the colon - consist of queues of addresses. The addresses are so-called compilation addresses (sometimes called code-field addresses) for which the corresponding memory location contains another address, this time pointing to actual machine code.



This sounds unnecessarily complicated at first, and it has often been suggested that the middle link shown in the diagram should be removed, giving 'direct threaded code', but the present consensus seems to be that the standard Forth method's gains in flexibility for advanced use outweigh the slight losses in speed and simplicity.

The diagram shows a colon definition (def1) being executed. At present we are at a point where def2, which appeared in def1's definition, is to be used. The compilation address of def2 contains a pointer to machine code which is executed and which terminates in a jump to a system label NEXT. Control will then return to the 'inner interpreter', which arranges for the next compilation address in def1's queue to be used, and so on. (If def2 is a colon word too, the machine code executed will save def1's queue position on a special stack, then tell the inner interpreter to start looking up the compilation addresses in def2's queue. This is discussed in Chapter 2.)

For the moment we are concerned only with how to run through a queue of compilation addresses such as those in def1. The exact method differs for different CPU's but the general idea is to keep the pointer which marks the place in def1's queue in a register, and to perform the indirection by a short routine starting at label NEXT. This routine has the job of looking up the contents of the address pointed to by the register, then looking up the address pointed to by what it has just looked up, and finally jumping to this last address. All machine code segments have the duty to save the relevant register; in fact, the run-time code for colon actually saves it on the return stack (to be restored by semicolon) then puts a different value in the register so the inner interpreter will start executing from a new queue. Other words such as BRANCH (the code compiled by ELSE) alter the register value without putting it on a stack, so giving the effect of a jump.

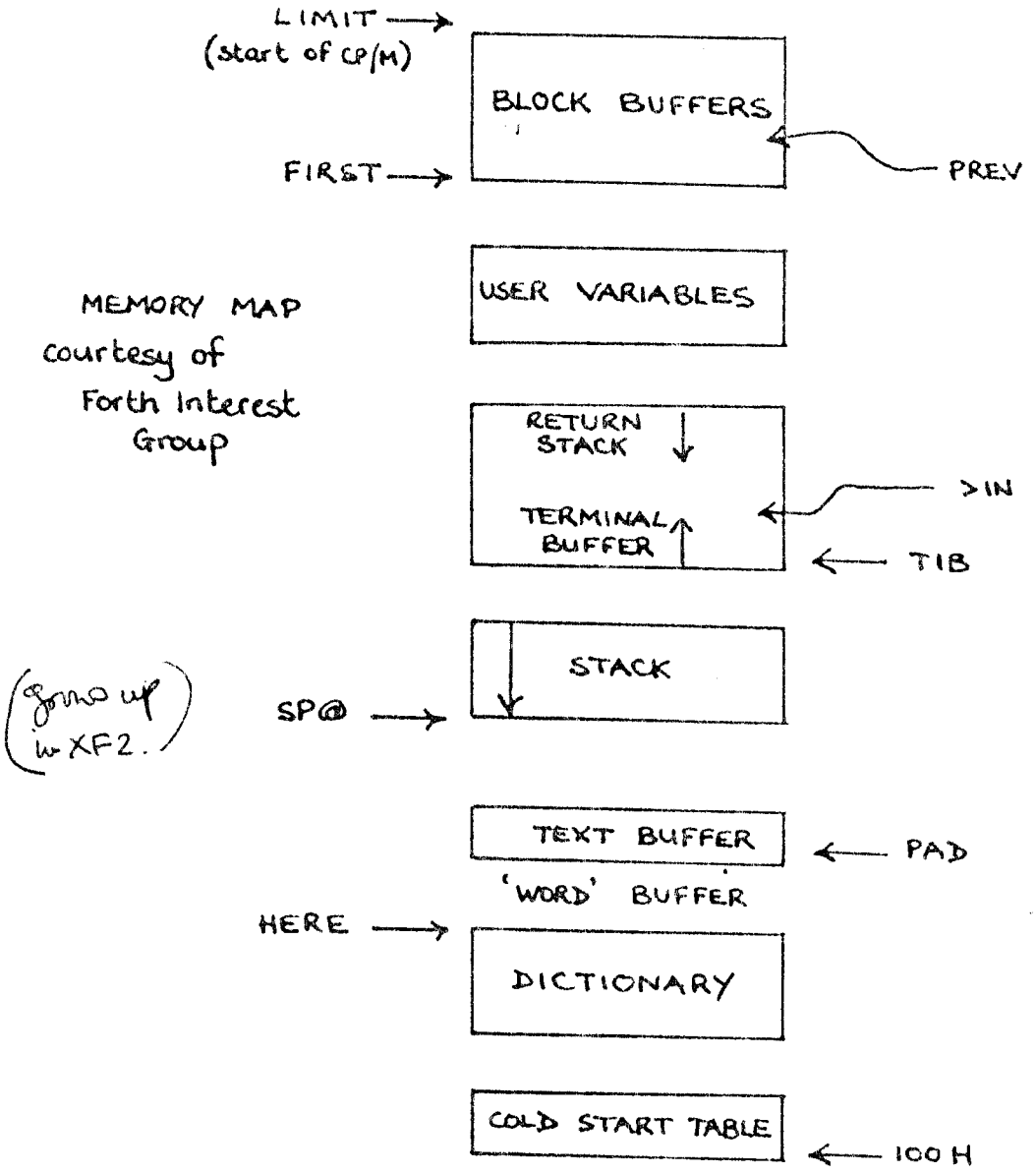
In xForth, the inner interpreter lives in low memory, just after the table of initialization values. Its address is placed on the stack by the constant NEXT, so assembler words exit using JMP NEXT (or its equivalent in some other assembly language than Intel's 8080). The rest of this discussion refers to the xForth implementation in 8080 code, version 1.20 for CP/M2.2. Other versions and implementations for other languages and operating systems will differ in detail, but not in overall concept.

The queue pointer referred to above is kept in the BC register, which must be saved and restored if necessary by any code you write. The value is pre-incremented: that is, by the time there's some code being executed it points to the queue position to be used next rather than the one currently being used. When the inner interpreter jumps to machine code, the DE register contains 1 more than the compilation address (of def2 in the diagram). This value need not be preserved but is often useful to know. The return stack, used by colon and semicolon to push and pop BC register values, is an area in high memory at the top end of the terminal input buffer; the stacking mechanism is simulated, using a memory location as a stack pointer.

Most of xForth is written at high level (i.e. as colon definitions): all of the disc handling and virtual memory, for example, uses high level words and communicates with CP/M via the word CPM-CALL, which is a simple register-twiddle-and-jump assembler word. The dictionary, which is a collection of all definitions, grows up from low memory, starting just after the inner interpreter. The stack (sometimes called the parameter stack, to distinguish it from the return stack) uses the normal 8080 stack pointer and grows down from high memory. Above the stack base are the storage locations used by system variables

such as XCURSOR and BASE, and above them are the disc input-output buffers, whose addresses are returned by the word BLOCK in virtual memory operations. The word SYSADAPT allows the number of blocks and the position of the top of the last block to be altered, by resetting various pointers and calling for a cold start. (Any words which try this should be written with great care, particularly if a cold start is not called, since the order of operations is significant; for example, stacks can end up overwriting one another if one is not careful.) All of this is summed up in the following memory map.

XFORTH 1
MEMORY MAP.



MEMORY MAP
courtesy of
Forth Interest
Group

Chapter 2

The outer interpreter

When xForth is first loaded, some initialization is done (see Chapter 7) and then QUIT is executed. QUIT is a colon-defined word whose function is to pass control to the terminal: it does this by clearing the return stack (which may have had return addresses or data on it, if QUIT was executed from within some word) and then going into a loop.

Clearing the return stack does not affect QUIT because the inner interpreter has its own pointer; if semicolon were to be executed there would be trouble, because semicolon tries to pop the return stack to find a return address, but QUIT's loop is one from which there is no escape except via another QUIT.

The important point about the loop is that it contains the word INTERPRET. Initially, INTERPRET is set by QUIT to read words from the terminal by setting variable BLK to zero. However, words such as LOAD reset the pointer BLK so that INTERPRET reads from elsewhere - in the case of LOAD, from virtual memory. At first INTERPRET merely calls on the inner interpreter to execute words successively from the input, until the input is exhausted. As explained in the introductory manual, some words such as colon switch INTERPRET over to storing the compilation addresses of words it meets, rather than executing them at once. This is done by changing the value of the variable STATE. If it is zero, words are executed as they are read while if it is nonzero, words are compiled unless they have been marked (by IMMEDIATE) for immediate execution.

At this stage it is necessary to look a little more closely at the action of colon. When colon is executed by INTERPRET, control passes to CREATE which reads the next word from the input and begins a dictionary entry for it, as described in Chapter 4. The dictionary entry has, in its compilation address field, a pointer to some special code which will adjust the return stack and the execution queue pointer when the new word is used later. The variable STATE is then set to a nonzero value (implementation-dependent) and control returns to the interpreter. From then on, until STATE is reset (by semicolon,

usually) non-immediate words are compiled: that is, their compilation addresses are entered into the queue in the dictionary entry for the new word. A typical immediate word is IF, which compiles a word OBRANCH. At execution time OBRANCH tests the stack top and, if it is zero, adjusts the queue pointer to point past the ELSE or ENDIF matching IF. (Obviously, the amount of adjustment is calculated at compile time, when ENDIF is executed.) Semicolon is an immediate word that resets STATE as already described, and compiles the word ;S which pops the return stack into the queue pointer.

The net effect of all this is to give the familiar FORTH behaviour: words are read from the terminal or virtual memory as required, and are executed or compiled as required.

Sometimes it is necessary to alter the usual compiling process, though. This is done by the words [COMPILE] and [and]. If an IMMEDIATE word has to be compiled (such as a vocabulary name, if we want the vocabulary change to take place at execution time) it can be preceded by [COMPILE]. The effect of [COMPILE] (which is itself IMMEDIATE) is to force the word following it to be compiled whether or not it is immediate. For example,

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

More or less the opposite effect is obtained with [which is immediate and whose action is to set STATE to zero. STATE can be reset again by] so that

```
: yyy ... [ HEX ] 10 [ DECIMAL ] 10 ... ;
```

will compile code that when executed will put decimal 16 on the stack and then decimal 10 on the stack. (Had we not put in the word DECIMAL, there would be two 16s instead of a 16 and a 10, and at compile time the number base would still be set to 16 after the semicolon.) The words [and] are useful for, say, calculating quantities known at compile time. They may be used in conjunction with LITERAL which is an immediate word that compiles the value the stack top as a literal to be put back on the stack at execution time. So

```
: yyy ... [ HEX 10 ] LITERAL
          [ DECIMAL 10 ] LITERAL
      ... ;
```

achieves the same effect as the previous definition of yyy.

Non-trivial examples of these and other wonders are to be found in the system code on blocks 30 to 42 (in most systems).

Chapter 3

Dictionary headers

A dictionary entry consists of two parts: a header and a body. The header part consists of the name of the word and information linking the word to previous words to allow the dictionary to be searched. In versions of xForth up to 1.21, the header consists of a **name field** followed by a link field. In earlier versions (before 1.20) the name field is a character count and then a text string, while in later versions the text string comes before the character count and is reversed; this is done to speed up dictionary searches. In all versions the link field is merely a pointer to the name field of the previous word in the dictionary. The character count is only valid in its low order 5 bits, since the remaining bits are used by SMUDGE and by IMMEDIATE. The precise details of all this may change in future systems.

The body consists of a **code field** followed by a **parameter field**. The code field address is often also called the **compilation address**. The code field contains the threading pointer and in the case of a colon definition it points to the run-time code for colon. This code pushes the BC register on the return stack and then sets BC to the first address of the parameter field, which will be the first member of the queue of words to be executed by that colon word. For a variable, the code field points to some code that returns the address of the parameter field on the stack. For other words, the code field will point to whatever special code they correspond to.

The word CREATE is used by all words that make named dictionary entries. CREATE reads the next word from the input stream and makes a dictionary header for it, then sets the code field to point to the code for VARIABLE, so that you can build tables by doing

```
CREATE name  1 , 2 , 3 , 4 , 5 ,
```

which makes a dictionary entry called name, followed by a code field address which returns the address of a 10 byte region containing the numbers 1 to 5, in order. (The word , (comma) takes the stack top and inserts it as a 2 byte value in the

dictionary, then increments the dictionary pointer.) Usually, CREATE is used in conjunction with a word DOES> that alters this tentative code field pointer to whatever is needed for the entry being created.

The code field of a word can always be located using FIND and the parameter field can always be located using ' (tick). For example, FIND DROP will return the code field address of DROP (which contains a pointer to the very next 16 bit memory location, as it happens) while ' DROP will return the parameter field address of DROP, which is 2 more than the code field address in versions up to 1.21. It is possible to locate all of the fields in present versions of xForth, though for compatibility with FORTH-79 and with future versions of xForth this fact should not normally be used. The relevant words are CFA, LFA, NFA and PFA, which are described in the glossary.

Chapter 4

;CODE and DOES>

Defining words such as : or STRING have two jobs to do when they execute: they must create a dictionary entry and they must create a definition corresponding to that entry. The definition, as we have seen, is a code field - a location containing the address of code to be executed - followed possibly by a parameter field, whose contents depend on the particular word. For example, the parameter field of a word defined by STRING is a pair of 16 bit locations containing the maximum length and the current length, followed by enough space to contain the maximum length string. The code field points to code that returns the address of the text space and the current length. For a word defined by : on the other hand, the code field points to code that manipulates the return stack, while the parameter field is the queue of addresses mentioned in Chapter 1, namely a sequence of pointers to other definitions' code fields (compilation addresses), interspersed with in-line data such as numeric values of literals to be placed on the stack at execution time.

We have already seen how CREATE makes a dictionary entry, but how are the code field and the parameter field of a new word produced? As we saw, CREATE does build the beginning of a definition after it makes the dictionary entry, since it makes the code field point to code that leaves the parameter field address on the stack. Often this will have to be changed, and nearly always a parameter field will have to be created.

The easiest way to do this is to use the word DOES> after CREATE. Let us look at a simplified version of the word STRING. We want 10 STRING s to create a dictionary entry called s with a parameter field containing 10, then the present length of s, then 10 bytes. When s executes we want it to leave the address of the start of those 10 bytes followed by the present length of s, so the code field of s will have to point to code that accomplishes this. Here is the definition.

```
: STRING  CREATE          ( Make a dictionary header )
          DUP , 0 ,       ( Put in max & present lengths )
          ALLOT           ( Leave enough space )
```

```
 DOES>          ( Point to address at exec time )
2+ DUP @        ( Present length on stack )
 SWAP 2+ SWAP ; ( Exit with addr & length )
```

When STRING executes, CREATE reads the next word from the input stream (e.g. the word following STRING) and makes a dictionary entry for it. So 10 STRING s makes an entry called s. CREATE sets up a code field like that of VARIABLE but allots nothing to the parameter field. Then DUP , stores the stack top (10 in the above example) in the parameter field and allots space for it so that the next assignment to the parameter field will use another memory location. Next 0 , puts zero in the next memory location (so the string has initial length zero) and ALLOT takes the 10 that was DUPed to decide how much space to allocate to the text string. The dictionary pointer is left pointing to the end of the text space, ready for another word to be created.

Next, the special word DOES> comes into action. Its job is to make the code field point to the correct code, which is the code between DOES> and the semicolon in the definition of STRING. At execution time, the address of the parameter will be pushed on the stack and the code following DOES> will be executed. Here, we add 2 to the start of the parameter field to get the location containing the max length and then two again to get the address of the text space.

As another example, here is a simplified version of []VARIABLE.

```
: []VARIABLE  CREATE
              2* ALLOT
              DOES>  2* + ;
```

In this case, 10 []VARIABLE vec makes a dictionary entry called vec and then allots 20 bytes (space for 10 integers). Executing 3 vec uses the address of vec's parameter field, plus 6, to be left on the stack. In our simple example, no range checking is done either when vec is defined or when vec is executed, so -100 []VARIABLE xx will cause disaster and 0 20 vec ! may well do so.

It is also possible to define execution time code in assembly language, by replacing DOES> with ;CODE. In that case, everything between ;CODE and END-CODE will be assembled, and at execution time the interpreter will jump to just after ;CODE. To use this successfully, you have to know the details of how your particular implementation simulates the Forth pseudo-machine, as in Chapter 1. For example, the 8080/Z80 system has the BC register as the threading pointer for the

inner interpreter, and the DE register contains 1 less than the parameter field address. The BC register must be restored before jumping to NEXT but the DE and HL registers need not be. Versions of xForth for different processors are supplied with the corresponding information about where the pointers are kept.

Thus we could define []VARIABLE for the 8080/Z80 system by

```
: []VARIABLE  CREATE  2* ALLOT   ( Make space at )
                                 ( compile time. )
              ;CODE              ( And do this   )
                                 ( at exec time: )
                      D INX,     ( D now points  )
                                 ( to parm field.)
                      H POP,     ( Get stack top.)
                      H DAD,     ( Double it.    )
                      D DAD,     ( Add the parm  )
                                 ( field address )
                                 ( to get addr   )
                                 ( of required   )
                                 ( element in HL.)
                      NEXT 1- JMP, ( Push HL &   )
                                 ( return.       )
              END-CODE
```

where we have used the fact that location NEXT-1 contains H PUSH, Note that ;CODE does exactly what is there: there is no hidden pushing of the parameter field address onto the stack as in DOES>, so we have to get it from the DE register.

Chapter 5

Virtual memory and the filing system.

All virtual memory operations are done through the word BLOCK. Executing BLOCK results in the removal of a block number from the stack and its replacement by an address in true memory.

BLOCK determines whether the required block is in memory and if so, returns the buffer address at once. If the block is not in memory, BLOCK uses an approximation to a 'least recently used' algorithm for buffer allocation, to select a new buffer. Each buffer has a 16 bit tag whose high bit is set if UPDATE has been called. This method of marking blocks as updated may change in systems beyond 1.20. If the required buffer has this bit set it is written to disc, the relevant file being created if necessary. The final stage is to read the appropriate part of a disc file into the buffer and leave the buffer address on the stack. If the block has never before been written to, the first byte will be a CP/M end of file mark (control/Z) and the next 127 bytes will be zero. The subsidiary words BUFFER and R/W are called by BLOCK to perform disc operations.

The connexion between disc files and block numbers, as described in the Appendix of the users' manual, is established by the word GET-FILE, which maintains a table of files. The 3rd file in the table corresponds to the 3rd virtual memory segment and so on; if no file is associated with a segment the stored address is 0. Executing get-file causes a block number to be taken off the stack and divided by seg-size to give an index in the file table. If the corresponding table entry is zero an error message is given, and otherwise the file and the appropriate CP/M record number are left on the stack.

For example, if table entry 6 points to the xForth file structure file6 (so that 6 'th-FILE places the same thing in the stack as does file6) then

```
6004 get-file
```

leaves the same stack result as

24 file6

would, since the 4th 1k block of file6 starts with CP/M sector 24 of the file.

Sequential input-output is actually simulated using standard xForth virtual memory operations. This allows spooling to be done in virtual memory so that by allocating a large buffer space using SYSADAPT the user can minimize disc activity. Note that if the sequential input-output is needed, QUIT is redefined to type out the contents of the output spool, if any, and SAVE-BUFFERS is redefined to delete the input and output spools.

Byte-oriented input-output via getc and putc is done by maintaining block and character pointers. If it is too slow the simplest solution is to write new words that do their own buffering and only call BLOCK occasionally rather than for every byte. (In particular, put$ could have been defined as a loop containing putc, but you can see how it does a block memory move instead.)

The greatest advantage of using virtual memory for sequential input-output is in the simplicity of the pipe and redirection words, which (like COPY) just renumber virtual memory blocks using (copies). The word (copies) in (file-voc) is like COPIES except that COPIES flushes virtual memory then optimizes disc access, and always copies the requested number of blocks, while (copies) does not flush virtual memory, stealing blocks from its input even if they should have been written to disc. Also, (copies) stops at any block beginning with an end-of-file. Note that this means the input to (copies) may not be correctly represented on disc (which doesn't matter for a spool file) and that, depending on the exact details of your CP/M implementation, it will usually stop at the first 1k block following the end of file.

Chapter 6

ABORT, COLD and the cold start tables

When xForth is loaded and run, control is transferred to a short assembler segment that sets up the stack and return stack correctly, and then COLD is called. The job of COLD is to initialize such things as XEMIT, the variable containing the code executed by the word EMIT before the output pointer OUT is incremented. All the information needed at initialization is contained in a table at the base of memory, which also contains information such as the version number. This table is read by COLD and written by PROTECT; the main part of the operation is a block memory transfer between the cold-start table (whose address is given by CS-TABLE) and the 'user variable' area, a region in high memory used for the current values of things like XEMIT.

The last thing COLD does is call EMPTY-BUFFERS and then ABORT. ABORT resets the stack and return stack pointers (unnecessarily in this case, but often needed) and executes the code pointed to by XSIGNON. Then it calls QUIT which gives control to the terminal.

Because of subtleties concerning the way multiple vocabularies are linked together, COLD is redefined by the xForth high level system to call EMPTY first if it is executed by the user or by a user-defined word. If this were not done, it would be possible to have mysterious delayed crashes caused by corruption of the dictionary structure in a way that only becomes evident later.

Appendix A

Additional notes.

The following are a few hints that may be helpful for users who want to change their systems.

1. A minimal system can be constructed by altering block 1 of FORTH.BLK and the blocks and files that block 1 loads when SYSGEN is typed from the kernel system. You only need the structuring words (except CASE and its friends) and the most elementary output words, plus a few assorted others, to have a FORTH-79 system.

2. If you want to reserve an area in memory for use by other code (say, code from another compiler) and you are nervous about putting it directly into the dictionary, you can leave a gap between the top of xForth and the CP/M system. Do this by changing 6 @ in the word SYSADAPT to, say, 6 @ 1024 - and then regenerating the system from the kernel. Then when you type, say, 4 SYSADAPT the system will be set to observe the new upper limit and will put its buffers below this.

3. The metacompiler can make words with no headers. This makes for a much smaller system.

4. The two locations before the start of the inner interpreter contain D PUSH, and H PUSH, so to push both D and H from a CODE definition, end up by jumping to NEXT 2- while to push just H jump to NEXT 1-

Appendix B

Error handling.

The FORTH-79 Standard defines the following types of error condition. In xForth, the word ERROR is executed whenever an error is detected, and leads to the effect described below as long as the system variable WARNING has the value 1. For the effects of other values of WARNING, see the entry for WARNING.

1. Input stream exhausted before a required <name>. The xForth system gives the error message 'Unexpected end of input'.

2. Empty stack and full stack for the text interpreter. The xForth system gives the error message 'Empty stack' or 'Full stack'.

3. An unknown word which is not a valid number for the text interpreter. The xForth system gives an error message consisting of the unknown word, followed by a question mark.

4. Compilation of incorrectly nested conditionals. The xForth system gives the error message 'Conditionals incorrectly nested'.

5. Interpretation of words restricted to compilation. If the word is IMMEDIATE, the xForth system gives the error message 'Compilation only; tried to use outside definition'. If the word is not IMMEDIATE, the error is ignored and execution continues as usual. See the notes on the glossary for more details.

6. FORGETting within the system to a point that removes a word required for correct execution. The xForth system gives the error message 'in protected dictionary'.

7. Insufficient space remaining in the dictionary. The xForth system gives the error message 'Dictionary full'.

Appendix C

Notes on the glossary.

xForth is a FORTH-79 Standard System with Assembler Standard Extensions (if the assembler has been purchased). It contains nearly all of the words in the Double Number Standard Extension.

The glossary that follows describes the action of nearly all xForth words in the vocabulary FORTH. Words in other vocabularies are not described, as they are typically specialised words not intended for use outside that vocabulary, e.g. (file-voc) and (EDITOR). In general, words enclosed in parentheses are specialised and not intended for normal use; if the parenthesised word is a vocabulary name none of the words in the vocabulary are intended for normal use.

The glossary entries consist of a header with the word and its effect on the stack, followed in some cases by some information in square brackets. The stack effect is described in the form

(stack-before --- stack-after)

or, if the word reads from the input stream when it executes (e.g. VARIABLE), in the form

(stack-before +++ stack-after)

Here each of stack-before and stack-after contains an ordered list of stack items, with the top of the stack (most accessible) on the right. The stack items are any of the following:

n	a signed 16 bit integer;
u	an unsigned 16 bit integer;
addr	a 16 bit address;
byte	a 16 bit stack item with only the low order 8 bits of interest;

char	a 16 bit stack item with only the low order 7 bits of interest;
a	a string identifier consisting of an address and a count of the number of characters;
file	the address of the base of a file control structure;
d	a 32 bit signed integer;
ud	a 32 bit unsigned integer;
flag	a boolean flag: TRUE is defined as 1, FALSE as zero. All xForth supplied words treat anything nonzero as TRUE.

The stack items are sometimes followed by digits 1, 2 etc where necessary to avoid ambiguity.

The information in square brackets is one or more of the following:

- A number corresponding to the number of the FORTH-79 definition.

- The letter C, meaning that the word should only be used during compilation (i.e. within a colon definition. All such words that are also immediate execute ERROR if called outside a colon definition. Words such as >R do not give an error message in such cases, unless the DEBUG vocabulary is being used. The general action for words that give no error message is to continue execution, ignoring the error. However, the side effects of the incorrect use may be such as to crash the system. For this reason, we recommend that you use the DEBUG vocabulary for all testing.

- The letter I, meaning that the word is marked for immediate execution and so will execute even if encountered during a colon definition (e.g. IF). To compile an immediate word, precede it by [COMPILE]. See the entries for [COMPILE] and IMMEDIATE.

- The letter U, meaning the word is the name of a user variable (a system variable). See the glossary entry for USER.

! (n addr ---) [112]

Store n at address. "store"

!CSP (---)

Save the stack position in variable CSP. Used for error checks during compiling.

" (+++ s) [I]

Interpreted or used in a colon definition in the form:

" ccc"

Read the following text from the input stream, terminated by a double-quote. If executing, leave the address and length of a text string which will remain equal to the text read at least until the input stream is exhausted. If compiling, compile so that later execution will leave the address and length of such a text string, valid at all times. At least 127 characters are allowed in the text. If the input stream is exhausted before the terminating double-quote, ERROR is executed. "double-quote"

(ud1 --- ud2) [158]

Generate from an unsigned double number ud1 the next ASCII character, which is placed in an output string. Result ud2 is the quotient after division by BASE and is maintained for further processing. Used between <# and #>. "sharp"

#->$ (d --- s)

Convert double number d to a text string, according to the current base for output conversion. The string contains only digits, except that the first character is a minus sign if d is negative. If d is zero, the string consists of one zero.

Otherwise, the leading digit is always nonzero.

#> (d --- addr n) [190]

End pictured numeric output conversion. Drop d, leaving the text address, followed by the character count, suitable for TYPE. "sharp-greater"

#BUFF (--- n)

A constant returning the number of buffers at present allocated for virtual memory operations.

#S (ud --- 0 0) [209]

Convert all digits of an unsigned 32 bit number ud, adding each to the pictured numeric output text, until remainder is zero. A single zero is added to the output string if the number was originally zero. Use only between <# and #>. "sharp-s"

#files (--- n)

A constant returning 8, the number of virtual memory segments normally accessible to the user.

$! (s1 s2 ---)

Assign the string literal s1 to the string variable s2, truncating s1 at the right if necessary.

$+ (s1 s2 --- addr3 n)

Concatenate strings. The string s2 is joined to the right of the string s1, the result being left at the PAD. The PAD address addr3 and the length of the combined string is left on the stack.

$-># (**s --- d flag**)

Attempt to convert string s to a signed double number, according to the present BASE. The string may optionally be preceded by a minus sign, but otherwise may contain only digits in the present base. If the conversion succeeds, d contains the result and the flag is set to TRUE.

$->U# (**s --- ud flag**)

Attempt to convert string s to an unsigned double number. The behaviour is identical with that of $-># (which calls $->U#) except that no leading minus is allowed.

$< (**s1 s2 --- flag**)

Return TRUE if s1 is lexically prior to s2, using ASCII ordering.

$= (**s1 s2 --- flag**)

Return TRUE if the strings are identical.

$FIND (**s --- addr**)

Attempt to locate the string s in the vocabulary, according to the rules for FIND. Identical to FIND except that string s is used instead of a string read from the input being interpreted.

' (**+++ addr**) [I,171]

Used in the form ' <name>

If executing, leave the parameter field address of the next word accepted from the input stream. If compiling, compile this address as a literal; later execution will place this value on the stack. ERROR is executed if <name> is not found after a search of CONTEXT and FORTH vocabularies (and any vocabularies

contained in CONTEXT). Within a colon-definition, ' <name> is identical to [' <name>] LITERAL. "tick"

's-FCB (addr1 --- addr2)

Given the address addr1 of a file control structure, return the address addr2 of a part of memory used to communicate file control information with the operating system.

's-STATUS-BYTE (file --- addr)

Return the address of a byte in the control structure of file that is used for information on the present status of the file (e.g. whether it is open).

's-name (addr --- s)

Given the address of a file control structure, return the address and length of a string containing the ASCII name of the corresponding CP/M file.

'th-FILE (n --- addr)

Return the address of the file control structure owning the n'th virtual memory segment, if this segment has been allocated to a file. Otherwise return 0. If n is negative or too large ERROR is executed. The meaning of 'too large' depends on the system.

((+++) [I,122]

Used in the form (ccc)

Read and ignore characters from the input stream, until the next right parenthesis. Being a FORTH-79 word, the left parenthesis must be followed by one blank. It may be used freely while executing or compiling. ERROR is executed if the input stream ends before a right parenthesis has been found.

(EDITOR) (---) [I]

The name of the vocabulary of internal screen editor words. Execution makes (EDITOR) the CONTEXT vocabulary.

(FLUSH) (---)

Request the operating system to write all blocks to mass-storage that have been flagged as UPDATEd, but do not necessarily force them or their directory information to be written physically. (The operating system may keep them in its own buffers.) Not intended for normal use.

(ID) (addr --- s)

Given a name field address addr, leave a string s containing the name.

(LINE) (n1 n2 --- s)

Return a 64 character string containing the text from line n1 of block n2.

(file-voc) (---) [I]

The name of the vocabulary of words used internally by the filing system and not normally intended to be accessed by other words. Contains the file $$$.

(skip-until) (addr1 char --- addr2)

Increase addr1 by 0 or more until the byte address pointed to contains char or 0. Used by skip-until but may also be used independently.

(skip-while) (addr1 char --- addr2)

Increase addr1 by 0 or more until the byte address pointed to does not contain char or does contain 0. Used by skip-while but may also be used independently.

* (n1 n2 --- n3) [138]

Leave the arithmetic product of n1 and n2. "times"

*/ (n1 n2 n3 --- n4) [220]

Multiply n1 by n2, divide the result by n3, and leave the quotient n4, rounded towards zero. The intermediate product is maintained as a 32 bit value for greater precision than the otherwise equivalent n1 n2 n3 * / "times-divide"

*/MOD (n1 n2 n3 --- n4 n5) [192]

Multiply n1 by n2, divide the result by n3, and leave the remainder n4 and the quotient n5. The intermediate product is maintained as a 32 bit value. The remainder has the same sign as n1. "times-divide-mod"

+ (n1 n2 --- n3) [121]

Leave the arithmetic sum of n1 and n2. "plus"

+! (n addr ---) [157]

Add n to the 16 bit signed value at the address. "plus-store"

+- (n1 n2 --- n3)

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

+LOOP (n ---) [I,C,141]

Add the signed increment n to the loop index and compare the total with the limit. Return execution to the corresponding DO unless the new index is equal to or greater than the limit (n>0), or unless the new index is less than the limit (n<0). Upon exit from the loop, discard the loop control parameters, continuing exection beyond +LOOP. Index and limit are signed integers in the range (-32768 ... 32767). "plus-loop" (Comment: It is an unfortunate Standards Committee decision that the limit for n<0 is irregular. Further consideration of the characteristic is likely.)

, (n ---) [143]

Allot two bytes in the dictionary and store n there. "comma"

- (n1 n2 --- n3) [134]

Subtract n2 from n1 and leave the difference n3. "minus"

--> (---) [I,131]

Continue interpretation with the next block. "next block"

-TRAILING (addr n1 --- addr n2) [148]

Adjust the character count n1 of a text string beginning at addr to exclude trailing blanks, i.e. the characters at addr+n2 to addr+n1-1 are blanks. ERROR is executed if n1 is negative. "dash-trailing"

. (n ---) [193]

Display n converted according to BASE in a free-field format with one trailing blank. If n<0 display a negative sign. "dot"

." (+++) [I,133]

Interpreted or used in a colon definition in the form:

." ccc"

Read the following text from the input stream, terminated by a double-quote. If executing, transmit this text to the selected output device. If compiling, compile so that later execution will transmit the text to the output device selected at execution time. At least 127 characters are allowed in the text. If the input stream is exhausted before the terminating double-quote, ERROR is executed. "dot-quote"

.BASE (---)

Type the present base for numeric input/output, in decimal.

.COVOC (---)

Type the name of the present context vocabulary.

.CPU (---)

Type the name of the CPU the present xForth version was designed for.

.CUVOC (---)

Type the name of the present current vocabulary.

.R (n1 n2 ---)

Type a signed number in the current base, right aligned in a field of width n2.

.SIZE (---)

Type the number of CP/M 256 byte pages that need to be SAVEd to make an executable copy of xForth.

.STACK (---)

Type a copy of the stack on the currently selected output device, without disturbing the stack.

.STORE (---)

Type the number of bytes presently available for dictionary and stack expansion.

.VERSION (---)

Type the version number of xForth being used.

/ (n1 n2 --- n3) [178]

Divide n1 by n2 and leave the quotient n3, rounded towards zero. Overflow or divide by zero may not be detected on some processors, for efficiency reasons. If it is not faulted, division by zero gives the result -1. "divide"

/MOD (n1 n2 --- n3 n4) [198]

Divide n1 by n2 and leave the remainder n3 and the quotient n4. n3 has the same sign as n1. "divide-mod"

0< (n --- flag) [144]

True if n is negative. "zero-less"

0= (n --- flag) [180]

True if n is zero. "zero-equals"

0> (n --- flag) [118]

True if n is positive. "zero-greater"

OBRANCH (flag ---)

The execution time procedure to branch conditionally. If flag is 0 the following in-line number is added to the interpretive pointer (in BC in 8080/Z80 systems) to branch ahead or back. Compiled by IF, UNTIL and WHILE.

1+ (n --- n+1) [107]

Increment the stack top. "one-plus"

1+! (addr ---)

Add 1 to the 16 bit number at address. "one-plus-store"

1- (n --- n-1) [105]

Decrement the stack top. "one-minus"

1-! (addr ---)

Subtract 1 from the 16 bit number at address. "one-minus-store"

2! (d addr ---)

Store the double number d in 4 bytes starting at addr. "two-store"

2* (n --- 2*n)

Double the number on the stack top. "two-times"

2+ (n --- n+2) [135]

Add 2 to the stack top. "two-plus"

2- (n --- n-2) [129]

Subtract 2 from the stack top. "two-minus"

2@ (addr --- d)

Fetch a double number from 4 bytes starting at addr. "two-fetch"

2DROP (d ---)

Discard the double number on top of the stack.

2DUP (d --- d d)

Duplicate the double number on top of the stack.

2OVER (d1 d2 --- d1 d2 d1)

Copy the second top stack double number.

2SWAP (d1 d2 --- d2 d1)

Exchange the top two stack double numbers.

79-STANDARD (---) [119]

Execute ensuring that a FORTH-79 Standard system is available. Note: On xForth systems with version numbers 1.1x, this merely sets the FORTH vocabulary, which is guaranteed to contain all the FORTH-79 words. However, it is the user's responsibility to ensure that none of these words has been redefined. It may be that the standard expects redefinition to be faulted: this is under consideration.

: (+++) [116]

A defining word used in the form:

: <name> ... ;

Select the CONTEXT vocabulary to be identical to CURRENT. Create a dictionary entry for <name> in CURRENT, and set compilation mode. Words thus defined are called "colon-definitions". The dictionary entry created is temporarily made invisible to FIND by setting a "smudge" bit. The compilation addresses of subsequent non-immediate words from the input stream are stored into the dictionary to be executed when <name> is executed later. Immediate words are executed as encountered.

Words in the input stream are looked up in the dictionary according to the convention for FIND. This means that the CONTEXT vocabulary is searched first, followed by any vocabularies contained in the CONTEXT vocabulary, in the reverse order they were defined. Note that all vocabularies chain to FORTH eventually, so FORTH is always searched. The chaining is such that an entire vocabulary is always searched before another one is tried; this includes definitions made after the vocabularies were originally chained together.

If a word is not found, conversion and compilation of a literal single or double number or (with the floating point option installed) a floating point number is attempted, with regard to the current BASE. That failing, ERROR is executed.

Note that colon is not immediate in FORTH-79, though it is made so in FIG-Forth and in the xForth DEBUG vocabulary in order to catch the common error of a missing semicolon.

"colon"

; (---) [I,C,196]

Terminate a colon definition and stop compilation. Make the latest word created visible to FIND by unsetting its smudge bit. If compiling from mass storage and the input stream is exhausted before encountering ; ERROR is executed. "semi-colon"

;CODE (---) [I,C,206]

Define the run-time action of a word created by a mixed high and low level defining word. Used in the form:

: <name> ... CREATE ... ;CODE ... END-CODE

and later <name> <namex>

Marks the termination of the defining part of the defining word <name>, sets the CONTEXT vocabulary to ASSEMBLER, and begins the definition of the run time action for words such as <namex> that will later be defined by <name>. On execution of <namex> the assembler language sequence between ;CODE and END-CODE will be entered. "semicolon-code"

;S (---)

The execution code compiled by semicolon. Pops the return stack into the execution pointer (BC in 8080/Z80 systems).

< (n1 n2 --- flag) [139]

True if n1 is less than n2. "less-than"

<# (---) [169]

Initialize pictured numeric output. The words:

<# # #S HOLD SIGN #>

can be used to specify the conversion of a double precision number into an ASCII character string stored in right-to-left order. "less-sharp"

<< (+++)

Read the next word in the input stream as a CP/M file name, and make that file the spooled input for reading with getc. "redirect-in"

<= (n1 n2 --- flag)

True if n1 is less than or equal to n2.

<> (n1 n2 --- flag)

True if n1 is not equal to n2. "not-equal"

<CMOVE> (addr1 addr2 u ---)

Move u bytes from addr1 to addr2, if u is nonzero. Overlapping moves are handled correctly. "bidirectional-cmove"

= (n1 n2 --- flag) [173]

True if n1 is equal to n2. "equals"

== (---)

Make the previous spooled output for putc the new spooled input for getc. "pipe"

021

> (n1 n2 --- flag) [102]

True if n1 is greater than n2. "greater-than"

>= (n1 n2 --- flag)

True if n1 is greater than or equal to n2.

>> (+++)

Read the next word in the input stream as a CP/M file name, and copy to that file the spooled output written previously by putc. "redirect-out"

>IN (--- addr) [U,201]

Leave the address of a variable that contains the present character offset within the input stream. (Range 0 ... 1023) "to-in"

>LINE (--- addr)

A variable that is incremented by CR. May be used for paging control etc.

>R (n ---) [C,200]

Transfer n to the return stack. Every >R must be balanced by a R> in the same control structure nesting level of a colon-definition. "to-r"

? (addr ---) [194]

Display the signed 16 bit number at the address. "question-mark"

?BRANCH See ØBRANCH

?COMP (---)

Execute ERROR if not compiling.

?CSP (---)

Execute ERROR if the present stack position differs from that saved in variable CSP.

?DEPTH (n ---)

Execute ERROR if the stack has less than n entries below n itself.

?DUP (n --- n) or (n --- n n) [184]

Duplicate n if it is non-zero. "query-dup"

?ERROR (flag n ---)

If flag is TRUE, execute n ERROR. If flag is false, remove it and n and continue normally.

?EXEC (---)

Execute ERROR if not executing.

?LOADING (---)

Execute ERROR if not loading from virtual memory.

?PAIRS (n1 n2 ---)

Execute ERROR if n1 is not equal to n2. Used in checking syntax of conditionals.

?PAUSE (---)

If the variable XOFF-CHAR contains -1, do nothing. Otherwise, execute ?TERMINAL and if no key has been struck, do nothing; if control/C has been struck, execute 6 ERROR; if the key has the ASCII code in XOFF-CHAR then pause until another key is struck and then continue unless the key is control/C (which aborts as before). Called by CR after every new line is output.

?STACK (---)

Execute ERROR if the stack is out of bounds.

?TERMINAL (--- flag)

Return TRUE if any key has been struck, leaving the actual key to be read if desired from LAST-KEY

@ (addr --- n) [199]

Leave on the stack the number contained at the address. "fetch"

ABORT (---) [101]

Clear the data and return stacks, setting execution mode. Return control to the terminal and execute the code pointed to by XSIGNON.

ABS (n --- |n|) [108]

Leave the absolute value of n. "absolute"

ALLOT (n ---) [154]

Add n bytes to the parameter field of the most recently defined word. The bytes are not initialized

or changed by ALLOT.

AND (n1 n2 --- n3) [183]

Leave the bitwise logical 'and' of n1 and n2.

ASCII (+++ char)

Read the next word from the input and leave the ASCII code of its first character.

ASSEMBLER (---) [I,166]

The name of the vocabulary of assembler words. Execution makes ASSEMBLER the CONTEXT vocabulary.

B/BUF (--- 1024)

A constant returning 1024, the number of bytes in a virtual memory buffer.

BASE (--- addr) [U,115]

Leave the address of a variable containing the current input-output numeric conversion base, which must lie in the range 2...70.

BEGIN (---) [I,C,147]

Used in a colon definition in the forms:

BEGIN ... flag UNTIL or
BEGIN ... flag WHILE ... REPEAT

BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until t flag is true; a BEGIN-WHILE-REPEAT loop will be repeated until the flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. flag is always dropped

after being tested.

The effect of BEGIN and the other loop words is achieved by the fact that they are immediate and so can calculate branches, and can compile appropriate code, while a colon definition is being formed.

BELL (---)

Send ASCII code 7 to the terminal. This normally sounds a noise-maker.

BINARY (---)

Set the base for numeric input/output to 2.

BL (--- 32)

A constant returning the ASCII code of a blank.

BLANKS (addr n ---)

Fill memory with n blanks starting at addr. If n<=0 do nothing.

BLK (--- addr) [U,132]

Leave the address of a variable containing the number of the mass storage block being interpreted as the input stream. If the content is zero, the input stream is taken from the terminal. "b-l-k"

BLOCK (n --- addr) [191]

Leave the address of the first byte in block n. If the block is not already in memory, it is transferred from mass storage into whichever memory buffer has been least recently accessed. If the block occupying that buffer has been UPDATEd (i.e. marked as modified) it is rewritten onto mass storage

before block n is read into the buffer. n is an unsigned number. If correct mass storage read or write is not possible, ERROR is executed. Only data within the latest block referenced by BLOCK is valid by byte address, due to sharing of the block buffers.

BRANCH (---)

The execution time procedure to branch unconditionally. The following in-line number is added to the interpretive pointer (in BC in 8080/Z80 systems) to branch ahead or back. Compiled by ELSE and REPEAT.

BUFFER (n --- addr) [130]

This is used by BLOCK but not normally by other words. Obtain the next block buffer, assigning it to block n. The block is NOT read from mass storage. If the previous contents of the buffer has been marked as UPDATEd it is written to mass storage. If correct writing to mass storage is not possible, ERROR is executed. The address left is the first byte within the buffer for data storage. n is an unsigned number.

BYE (---)

Call SAVE-BUFFERS and then return control to the operating system.

C! (n addr ---) [219]

Store the least significant 8 bits of n at addr. "c-store"

C, (byte1 ---)

Allot a byte in the dictionary, storing byte1 there.

C/L (--- n)

A constant returning the number of characters in a line of output. (Set to 80 on delivery.)

C@ (addr --- byte) [156]

Leave on the stack the contents of the byte at addr (with higher bits zero, in a 16 bit field). "c-fetch"

CAN-KEY (--- addr) [U]

A system variable containing the ASCII code of a key used during input by EXPECT and QUERY to remove all characters so far typed.

CASE (n --- n) [I,C]

Used in a colon definition in the form

```
CASE     ... OF ... ENDOF
         ... OF ... ENDOF
         ......
         DEFAULT ....
ENDCASE
```

Execute the first part between the OF ... ENDOF for which the stack top on entry to CASE matches the stack top on entry to OF. Note that the stack top is dropped by OF if the OF ... ENDOF part is performed. If any OF ... ENDOF part is performed, control passes from ENDOF to beyond ENDCASE. If there is no match execute any code between the final ENDOF and DEFAULT or ENDCASE, then execute the DEFAULT part if there is one. Note that DEFAULT does not remove the stack top, and ENDCASE only removes the stack top if there was no DEFAULT. In all cases continue beyond ENDCASE.

CFA (addr1 --- addr2)

Convert the parameter field address addr1 of a definition to its code field address addr2. Not

guaranteed to be available in future versions of xForth.

CLOSE (addr --- flag)

Attempt to close the file whose file control structure is at addr, leaving a true flag if successful and a false flag otherwise.

CMOVE (addr1 addr2 n ---) [153]

Move n bytes beginning at addr1 to addr2. The contents of addr1 are moved first, then those of addr1+1 and so on. If n<=0 nothing is moved. See also <CMOVE>. "c-move"

CODE (+++) [111]

A defining word used in the form:

CODE <name> ... END-CODE

to create a dictionary entry for <name>, to be defined by a following sequence of assembly language words. ASSEMBLER becomes the context vocabulary.

COLD (---)

Reset the system to the state it had when PROTECT was last called, or to the initial start-up state. This includes removing all new definitions, resetting the filing system without saving buffers or closing files, and resetting all system and execution variables. Finally call ABORT.

COMPILE (---) [C,146]

When a word containing COMPILE executes, the 16 bit value following the compilation address of COMPILE is copied (compiled) into the dictionary.

Thus COMPILE DUP will copy the compilation address of DUP and COMPILE [0 ,] will copy zero.

CONFIG (---)

Load block 2. This is usually set up to conduct a question and answer session with the user to set the values of things like DEL-KEY.

CONSTANT (n +++) [185]

A defining word used in the form:

n CONSTANT <name>

to create a dictionary entry for <name>, leaving n in its parameter field. When <name> is later executed, n will be left on the stack.

CONTEXT (--- addr) [U,151]

Leave the address of a variable specifying the vocabulary in which dictionary searches are to be made (e.g. by FIND and during compilation).

CONVERT (d1 addr1 --- d2 addr2) [195]

Convert the text beginning at addr1+1 to a double number, with regard to BASE, and add it to d1 to give d2. addr2 is the address of the first non-convertible character.

COPIES (n1 n2 n3 ---)

Copy block n1 to n2, n1+1 to n2+1 etc until n3 blocks have been copied. Overlapping shifts are done correctly.

COPY (n1 n2 ---)

Copy block n1 to n2.

COUNT (addr --- addr+1 n.) [159]

Used to convert text from packed to string form: in packed form the number of characters is stored in the first byte, so COUNT merely returns the contents n of that byte and increments the address. Range of n is 0,...,255.

Renamed DOS-CALL

~~CPM-CALL~~ (n1 byte --- n2)

Make a call to the CP/M system for function byte with n1 in the DE register. The contents of the HL register on return are left as n2. In CP/M1.4 the contents of BA rather than the contents of HL are left.

Renamed DOS-CALLB

~~CPM-CALLb~~ (n byte1 --- byte2)

Execute CPM-CALL and then mask off the high order byte of the result.

CR (---) [160]

Make the current output device take a new line. (Send ASCII return and line feed.) Then increment the variable >LINE and execute ?PAUSE. Any word that produces output thus automatically checks for control/S (or other pause character set in XOFF-CHAR) and for control/C.

CREATE (+++) [239]

A defining word used in the form:

CREATE <name> to create a dictionary entry for <name>, without allocating any parameter field memory. When <name> is subsequently executed, the address of the first byte of its parameter field is left on the stack unless DOES> has modified the

dictionary entry.

CR8 (n ---)

Perform CR n times if n>0. Do nothing if n<=0.

C8-TABLE (--- addr)

Return the address of the base of a table used to initialize system variables on initial start or after COLD is typed.

C8P (---- addr)

A variable used for temporarily storing the stack pointer position. Used for error checks during compiling.

CTRL (+++ char)

Read the next word from the input and leave the low order 5 bits of the ASCII code of its first character.

CURRENT (--- addr) [U,137]

Leave the address of a variable specifying the vocabulary into which new word definitions are to be entered.

CURSOR (n1 n2 ---)

Execute the code pointed to by XCURSOR. The code should move the terminal's cursor to row n1 and column n2, relative to 0 0 as the top left corner.

D+ (d1 d2 --- d3) [241]

Add signed double numbers d1 and d2 and leave the result as d3. "d-plus"

D+- (d1 n --- d2)

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

D. (d ---) [129]

Type a signed double number in the current base, followed by a blank. Display the sign only if negative. "d-dot"

D.R (d n ---)

Type a signed double number in the current base, right aligned in a field of width n. Display the sign only if negative. "d-dot-r"

D< (d1 d2 --- flag) [244]

True if d1 is less than d2. "d-less-than"

DABS (d1 --- d2)

Leave as a positive double number d2 the absolute value of a double number d1. The result lies in the range 0,...,2147483647.

DEBUG (---) [I]

The vocabulary used for debugging and tracing.

DECIMAL (---) [197]

Set the input-output numeric conversion to 10.

DEFAULT (n --- n) [I,C]

At compile time, arrange for ENDCASE not to compile code to drop the stack top. At execution time, do nothing.

DEFINITIONS (---) [155]

Set CURRENT to the CONTEXT vocabulary so that subsequent definitions will be created in the vocabulary previously selected as CONTEXT.

DEL-KEY (--- addr) [U]

A system variable containing the ASCII code of a key used during input by EXPECT or QUERY to remove the last character typed.

DEPTH (--- n) [238]

Leave the number of 16 bit values contained in the data stack, before n was added.

DIR (n ---)

Display the directory of the disc in CP/M drive n, where n=1 gives drive A and so on.

DLITERAL (d ---)

If compiling, compile the stack double value d as a 32 bit literal, to be left on the stack at later execution.

DNEGATE (d --- -d) [245]

Leave the twos complement of a double number.

DO (n1 n2 ---) [I,C,142]

Used in a colon definition in the form:

DO ... LOOP or DO ... +LOOP

Start a counting loop. The loop index begins at n2 and terminates according to a test on the present value of the index and the limit n1. See LOOP or +LOOP for details of termination. Note that the loop is always performed at least once, because the test is made at the end. The index I only returns a valid quantity between DO and LOOP or +LOOP.

DO ... LOOP may be nested to a great depth: the precise value depends on the system but will usually be at least 20.

DOES> (---) [I,C,168]

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

: <name> ... CREATE ... DOES> ... ;

;and later <name> <namex>

Marks the termination of the defining part of the defining word <name> and begins the definition of the run time action for words that will later be defined by <name>. On execution of <namex> the sequence of words between DOES> and ; will be executed, with the address of <namex>'s parameter field on the stack. "does"

DP (--- addr) [U]

A system variable, the dictionary pointer, which contains the address of the next free memory location above the dictionary. Do not alter directly: read by HERE and change by ALLOT.

DPL (--- addr)

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

DROP (n ---) [233]

Discard the number on top of the stack.

DUP (n --- n n) [205]

Duplicate the number on top of the stack.

ELSE (---) [I,C,167]

Used in a colon definition in the form:

IF ... ELSE ... ENDIF or
IF ... ELSE ... THEN

(The THEN form is the FORTH-79 standard.) ELSE executes after the true part following IF and passes execution to just beyond ENDIF or THEN. It also acts as a marker for the part to be executed if there is a false flag on the stack when IF executes. ELSE has no effect on the stack. See IF.

EMIT (char ---) [207]

Transmit char to the currently selected output device and increment OUT.

EMITP (char ---)

Transmit char to the CP/M list device (the 'printer'). Do not increment OUT or look to see what devices are selected.

EMITT (char ---)

Transmit char to the CP/M console device (the 'terminal'. Do not increment OUT or look to see what devices are selected.

EMPTY (---)

Remove all unprotected definitions without altering the stacks. See PROTECT.

EMPTY-BUFFERS (---) [145]

Mark all blocks as empty, whether or not they were actually marked as updated.

END-CODE (---)

Terminate a code definition, resetting the CONTEXT vocabulary to the CURRENT vocabulary. If no errors have occurred, the code definition is made available for use.

ENDCASE (---) or (n ---) [I,C]

At compile time, complete the action of a CASE statement as follows. Fill in all the branch addresses from instances of ENDOF. Then if DEFAULT has not been executed, compile DROP so that at execution time, the stack top will be dropped if the point where ENDCASE was is reached rather than being jumped past from an ENDOF.

ENDIF (---) [I,C]

A synonym for THEN

ENDOF (---) [I,C]

Pass control to beyond the next ENDCASE.

ENSURE-LINE (n ---)

If there are less than n character positions remaining on the present output line, execute CR .

EOF (--- n)

A constant returning the value (26) of the CP/M end-of-file marker.

ERASE (addr n ---)

Fill memory with n zeroes starting at addr. If n<=0 do nothing.

ERROR (n ---)

Execute the code pointed to by XERROR. By default, the code is STD-ERROR which issues error message n and executes QUIT.

EXECUTE (addr ---) [163]

Execute the dictionary entry whose compilation address is on the stack.

EXIT (---) [C,117]

When compiled within a colon-definition, terminate execution of that definition, at that point. May not be used within DO ... LOOP or if anything has been left on the return stack.

EXPECT (addr n ---) [189]

Transfer characters from the terminal beginning at addr, upward, until a "return" has been received or the count of n has been reached. Take no action for n<=0. One or two nulls are added at the end of

the text.

EXPECT$ (addr n1 --- addr n2)

Execute EXPECT and return the address of the string read from the terminal, with its actual length n2. (Note n2<=n1.)

FALSE (--- 0)

A constant returning the value of a FALSE flag, namely 0.

FCREATE (addr ---)

Given the address of a file control structure, attempt to create a new disc file with the name given in the structure. If unsuccessful, execute ERROR.

FENCE (--- addr) [U]

A system variable containing an address below which FORGET will refuse to work. Set by PROTECT.

FILE (+++)

A defining word used in the form:

FILE <name>

to create a file control structure. The initial operating system name of the corresponding file is set to be the same as the xForth name. Since names are checked for correct operating system syntax, ERROR is executed if <name> (after conversion to upper case if necessary) is not a legal operating system name. Execution of <name> leaves the address of the file control structure on the stack. See also fname! and 's-FCB.

FILE-INIT (---)

Reset the filing system, without writing UPDATEd buffeers to disc. All block/file allocations are removed except that blocks 1 to seg-size -1 are allocated to SYSFILE. The name of SYSFILE is set to its standard value, which is FORTH.BLK on the current default drive for CP/M systems.

FILL (addr n byte ---) [234]

Fill memory beginning at addr with a sequence of n copies of byte. If n<=0 take no action.

FIND (+++ addr) [203]

Leave the compilation address of the next word, which is read from the input stream. If that word cannot be found in the dictionary after a search of the CONTEXT vocabulary and all vocabularies contained in it, leave 0. Note that all vocabularies chain to FORTH eventually, so FORTH is always searched. Chaining is done in such a way that all entries in a vocabulary are searched before the next vocabulary is tried.

FIRST (--- addr)

A constant returning the address of the first virtual memory buffer.

FORGET (+++) [186]

Used in the form:

FORGET <name>

Set the CONTEXT vocabulary to CURRENT and attempt to find <name> according to the conventions for FIND. If <name> cannot be found, ERROR is executed. If <name> is found, check whether it is protected and if so, cause an error; if not, remove <name> and all words added to the dictionary after <name>, regardless of their vocabulary.

FORTH (---) [I,187]

The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary.

New definitions become a part of FORTH until a different CURRENT vocabulary is established.

User vocabularies conclude by 'chaining' to FORTH, so it should be considered that FORTH is 'contained' within each user's vocabulary.

GET-FILE (n1 --- addr n2)

For block n1, find the corresponding file if there is one, and return its address addr and the operating-system dependent sector number n2 of its base. If no file owns block n1, execute ERROR.

HERE (--- addr) [188]

Return the address of the next available dictionary location.

HEX (---)

Set the base for numeric input/output to 16.

HLD (--- addr)

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

HOLD (char ---) [175]

Insert char into a pictured numeric output string. May only be used between <# and #>.

I (--- n) [C,136]

Copy the loop index onto the data stack. May be only used in the form:

DO . . . I . . . LOOP or DO . . . I . . . +LOOP

ID. (addr ---)

Given a name field address addr, type the name.

IF (flag ---) [I,C,210]

Used in a colon-definition in the forms:

flag IF . . . ELSE . . . THEN or
flag IF . . . THEN

If flag is true, the words following IF are executed and the words following ELSE are skipped. The ElSE part is optional.

If flag is false, words between IF and ELSE, or between IF and THEN (when no ELSE is used), are skipped. IF-ELSE-THEN conditionals may be nested.

In xForth, ENDIF is a synonym for THEN.

IMMEDIATE (---) [103]

Mark the most recently made dictionary entry as a word which will be executed when encountered during compilation rather than compiled.

INDEX (n1 n2 ---)

Execute PAGE then type the first 64 characters each of blocks n1 to n2, executing CR after each 64 characters. May be stopped by hitting control/C.

INSTALL-$$$ (+++)

Read the next word from the input stream as a CP/M file and make it the CP/M name of a file control structure owning the segment of virtual memory beyond the last user-accessible segment. The file control structure has xForth name **$$$** in vocabulary (file-voc). It is used for temporary allocation by words such as LOAD-FILE.

INTERPRET (---)

Execute (or compile, if STATE is nonzero) the text from the present input. The input is from the keyboard if the value of BLK is nonzero and from the virtual memory block numbered by BLK otherwise.

J (--- n) [C,255]

Return the index of the next outer loop. May only be used within a nested DO-LOOP in the form:

DO ... DO ... J ... LOOP ... LOOP

KEY (--- char) [100]

Leave the ASCII value of the next available character from the current input device.

L/S (--- n)

A constant returning the number of lines in a VDU screen. (Set to 24 on delivery.)

LABEL (+++)

A defining word used in the form

LABEL <name>

to set the CONTEXT and CURRENT vocabularies to ASSEMBLER and create a dictionary entry for <name>. Note that <name> will therefore be in the ASSEMBLER vocabulary.

LAST-KEY (--- char)

Return the ASCII code of the last key read by KEY or ?TERMINAL.

LATEST (--- addr)

Leave the name field address of the most recently created word in the CURRENT vocabulary.

LEAVE (---) [C,213]

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

LFA (addr1 --- addr2)

Convert the parameter field address addr1 of a definition to its link field address addr2. Not guaranteed to be available in future versions of xForth.

LIMIT (--- addr)

A constant returning the address immediately beyond the last address used by xForth. By careful alteration of the value put into LIMIT by SYSADAPT the user can create areas of memory which can be reserved for other programs.

LIST (n ---) [109]

List the ASCII symbolic contents of screen n on the current output device, setting SCR to contain n. n is unsigned.

LIST-FILE (+++)

Read the next word from the input stream as a CP/M file name and list the file in LIST form if the file extension is .BLK Otherwise assume the file is an ASCII text file and list it as such.

LISTS (n1 n2 ---)

LIST blocks n1 to n2, inclusive, calling CR between each.

LIT (--- n)

The execution time code compiled by LITERAL (and by INTERPRET for in-line constants). Leaves on the stack the 16 bit quantity following it.

LITERAL (n ---) [I,215]

If compiling, then compile the stack value n as a 16-bit literal, which when later executed, will leave n on the stack.

LOAD (n ---) [202]

Begin interpretation of screen n by making it the input stream; preserve the locators of the present input stream (from >IN and BLK). If interpretation is not terminated explicitly it will be terminated when the input stream is exhausted. Control then returns to the input stream containing LOAD, determined by the input stream locators >IN and BLK.

LOAD-FILE (+++)

Read the next word from the input stream as a CP/M file name and LOAD the first block of the file if the file extension is .BLK Otherwise ERROR is executed.

LOOP (---) [I,C,124]

Increment the DO-LOOP index by one, terminating the loop if the new index is equal to or greater than the limit. The limit and index are signed numbers in the range -32,768,...,32,767.

M* (n1 n2 --- d)

Multiply n1 by n2, leaving the result as a 32 bit signed number d.

M/ (d n1 --- n2 n3)

A mixed magnitude operator that leaves the signed remainder n2 and signed quotient n3 from a 32 bit signed dividend d and a 16 bit signed divisor n1. The remainder takes its sign from the dividend.

M/MOD (ud1 u1 --- u2 ud2)

An unsigned mixed precision version of /MOD, leaving a double quotient ud2 and a single remainder u2 obtained by dividing the single number u1 into the double number ud1.

MAX (n1 n2 --- n3) [218]

Leave the greater of two numbers. "max"

MESSAGE (n ---)

Type error message n but continue normally.

MIN (n1 n2 --- n3) [127]

Leave the lesser of two numbers. "min"

MOD (n1 n2 --- n3) [104]

Divide n1 by n2, leaving the remainder n3, with the same sign as n1. "mod"

MOVE (addr1 addr2 n ---) [113]

Move the specified quantity n of 16-bit memory cells beginning at addr1 into memory at addr2. The contents of addr1 is moved first. If n is negative or zero, nothing is moved.

MYSELF (---)

Compile a reference to the latest definition (usually the definition MYSELF is contained in). Allows recursion.

NEGATE (n --- -n) [177]

Leave the two's complement of a number, i.e., the difference of 0 less n.

NEXT (--- addr)

A constant returning the address of the xForth inner interpreter, which must be jumped to by assembler words if normal threading is to resume.

NFA (addr1 --- addr2)

Convert the parameter field address addr1 of a definition to its name field address addr2. Not guaranteed to be available in future versions of xForth.

NOOP (---)

Do nothing.

NOT (flag1 --- flag2) [165]

Reverse the boolean value of flag1. This is identical to 0=.

NUMBER (addr --- d)

Convert a character string left at addr, with a count at addr-1, to a signed double number d according to the present value contained in BASE. If a decimal point is encountered in the text, its position will be given in DPL but no other effect occurs. If numeric conversion is impossible, ERROR will be executed.

OF (n1 n2 --- n1) or (n1 n2 ---) [I,C]

If n1 is not equal to n2, pass control to beyond the next ENDOF, leaving n1 on the stack for further testing. If n1=n2, continue execution beyond OF having removed n1 and n2 from the stack.

OPEN (addr --- flag)

Attempt to open the file whose file control structure is at addr, leaving a true flag if successful and a false flag otherwise.

OR (n1 n2 --- n3) [223]

Leave the bitwise inclusive-or of two numbers.

OUT (--- addr)

A variable holding the present position of the output pointer. Incremented by EMIT, reset to 0 by

CR and adjusted by all other xForth output words.

OVER (n1 n2 --- n1 n2 n1) [170]

Leave a copy of the second number on the stack.

P! (byte1 byte2 ---)

Send byte1 to the 8080 port address given by byte2. "p-store"

P@ (byte1 --- byte2)

Return the byte obtained by inputting from the 8080 port address given by byte1. "p-fetch"

PAD (--- addr) [226]

The address of a scratch area used to hold character strings for intermediate processing. The minimum capacity of PAD is 64 characters (addr through addr+63).

PAGE (---)

Execute the definition pointed to by XPAGE. Usually causes a paper throw or blanks a screen.

PFA (addr1 --- addr2)

Convert the name field address addr1 of a definition to its parameter field address addr2. Not guaranteed to be available in future versions of xForth.

PICK (n1 --- n2) [240]

Return the contents of the n1-th stack value, not counting n1 itself. ERROR is executed if n is less than one.

2 PICK is equivalent to OVER.

PREV (--- addr) [U]

A system variable containing the number of the block most recently referenced via BLOCK. The high bit of the block number is set if the block is updated.

PRINTER-ON? (--- addr)

A variable that if set to 1 causes output to be copied to the printer. Toggled by control/P during EXPECT or may be set and reset under program control.

PROTECT (---)

Set the initialisation sequence so that the present state will be restored by COLD. Set the variable FENCE so that all definitions presently existing are protected against removal by FORGET or EMPTY.

QUERY (---) [235]

Accept input of up to 80 characters (or until a 'return') from the operator's terminal, into the terminal input buffer. WORD may be used to accept text from this buffer as the input stream, by setting >IN and BLK to zero.

QUIT (---) [211]

Clear the return stack, setting execution mode, and return control to the terminal. No message is given.

R# (--- addr) [U]

A system variable that is used by the editors (for example) to keep track of the cursor position.

R/W (addr n flag ---)

Read or write block n to a 1K buffer starting at addr. If flag is 1 then read; if flag is 0 then write. This word does all file allocation and, if necessary, creation or opening of disc files. Normally for system use only; not guaranteed to be present in future releases of xForth.

R> (--- n) [C,110]

Transfer n from the return stack to the data stack. "r-from"

R@ (--- n) [C,228]

Copy the number on the top of the return stack to the data stack. "r-fetch"

REPEAT (---) [I,C,120]

Used in a colon definition in the form:

BEGIN ... WHILE ... REPEAT

At run-time, REPEAT returns to just after the corresponding BEGIN.

REPLACED-BY (addr +++)

Find the compilation address of the next word in the input stream, and store it in addr. If the next word has not previously been compiled, execute ERROR.

RESTORE-$$$ (---)

If used after INSTALL-$$$ and before control has been returned to the console, restores the name of the temporary file $$$ to what it was before INSTALL-$$$ was executed.

REVERSE (s ---)

Reverse the order of the characters of s. Note that this is done in place so s should always be a 'safe' copy of an original. Used mainly by the system, but can be used with care by users.

ROLL (n ---) [236]

Extract the n'th stack value to the top of the stack, not counting n itself, moving the remaining values into the vacated position. ERROR is executed if n is less than one.

```
3 ROLL = ROT
     1 ROLL = null operation
```

ROT (n1 n2 n3 --- n2 n3 n1) [212]

Rotate the top three values, bringing the deepest to the top. "rote"

RP! (---)

Resets the return stack pointer. Note that this means execution of ;S (as compiled by ;) may crash the system: RP! should not normally be called by the user.

S->D (n --- d)

Extend the signed single number n to a signed double number d.

SAVE-BUFFERS (---) [221]

Write all blocks to mass-storage that have been flagged as UPDATEd. ERROR is executed if mass-storage writing is not completed.

SCR (--- addr) [U,217]

Leave the address of a variable containing the number of the screen most recently listed. "s-c-r"

SEC/BLK (--- n)

An operating-system dependent constant equal to the number of operating-system basic mass-storage units in a 1K block. Not guaranteed to be present in future versions of xForth.

SEE (n ---)

Invoke the screen editor with the cursor pointing to the first character of block n.

SEE-FILE (+++)

Read the next word from the input stream as a CP/M file name and invoke the screen editor with the cursor pointing to the first character of the file, if the file extension is .BLK

SIGN (n ---) [C,140]

Insert the ASCII "-" (minus sign) into the pictured numeric output string, if n is negative.

SMUDGE (---)

Used during word definition to toggle the 'smudge bit' of the name field of the latest definition,

i,e. the one whose name field address is contained in the variable LATEST. This prevents the word from being found by FIND and so makes an uncompleted definition invisible until it is completed without error.

SP! (any ---)

Discard all items on the stack. "stack pointer store"

SP@ (--- addr)

A system dependent procedure to return the address of the stack top as it was before SP@ was executed. Not normally for user execution.

SPACE (---) [232]

Transmit an ASCII blank to the current output device.

SPACES (n ---) [231]

Transmit n spaces to the current output device. Take no action if n<=0.

STATE (--- addr) [U,164]

Leave the address of the variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself may depend on the version of xForth.

STD-ERROR (n ---)

The default code executed by ERROR. Executes n MESSAGE and then QUIT.

STRING (n +++)

A defining word used in the form:

n STRING <name>

to create a dictionary entry which is a string variable to hold a text string of length up to n characters. If n exceeds 255, the maximum length will be 255. At execution time, <name> returns the address of the first character and the current length of the string in a form suitable for TYPE.

SWAP (n1 n2 --- n2 n1) [230]

Exchange the top two stack numbers.

SYSADAPT (n ---)

Call PROTECT then set the initialisation sequence so that n buffers will be used for virtual memory, and set LIMIT so that when COLD is executed, all possible memory will be used. Then call COLD. Note that the code of SYSADAPT may be changed to set LIMIT to some different value so that not all possible memory will be used.

SYSFILE (--- addr)

Return the address of a file control structure owning virtual memory segment 0. (Blocks 1 to 1000 in the standard system).

SYSGEN (---)

Generate the standard system from the kernel system. If the present system is not the kernel system, execute ERROR.

TAB (n ---)

Type as many spaces as needed to bring the output pointer to position n, if possible. Otherwise do nothing.

THEN (---) [I,C,161]

Used in a colon-definition, in the form:

IF ... ELSE ... THEN or
IF ... THEN

THEN is the point where the execution resumes after ELSE or IF (when no ELSE is present).

TIB (--- addr) [U]

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

TOGGLE (addr byte1 ---)

Complement the byte at addr by the bit pattern contained in byte1.

TRIAD (n ---)

Type 3 screens, including n, in a form suitable for hard copy. May be stopped by hitting control/C.

TRUE (--- 1)

A constant returning the value of a TRUE flag, namely 1.

TYPE (addr n ---) [222]

Transmit n characters beginning at address to the current output device. No action takes place for n less than or equal to zero.

U$ (s --- s)

Apply UCHAR to each character of s, so changing it to upper case if it was a letter and stripping the high bit in all cases. Note that the operation is done in place, i.e. the actual s is changed.

U* (un1 un2 --- ud3) [242]

Perform an unsigned multiplication of un1 by un2, leaving the double number product ud3. All values are unsigned. "u-times"

U. (un ---) [106]

Display un converted according to BASE as an unsigned number, in a free-field format, with one trailing blank. "u-dot"

U.R (u n ---)

Print the unsigned 16 bit value u right aligned in a field of width n. If n is too small, overflow to the right as required.

U/MOD (ud1 un2 --- un3 un4) [243]

Perform the unsigned division of double number ud1 by un2, leaving the remainder un3, and quotient un4. All values are unsigned. "u-divide-mod"

U< (un1 un2 --- flag) [150]

Leave the flag representing the magnitude comparison of un1 < un2 where un1 and un2 are treated as 16-bit unsigned integers. "u-less-than"

UCHAR (byte --- char)

Remove the high bit of byte; if the corresponding character is lower case, convert it to upper case.

UNTIL (flag ---) [I,C,237]

Within a colon definition, mark the end of a BEGIN-UNTIL loop, which will terminate based on a flag. If flag is true, the loop is terminated. If flag is false, execution returns to the first word after BEGIN. BEGIN-UNTIL structures may be nested.

UPDATE (---) [229]

Mark the most recently referenced block as modified. The block will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block, or upon execution of SAVE-BUFFERS.

USER (n +++)

A defining word used in the form:

n USER <name>

to create a dictionary entry for a system (or 'user') variable. The value of n is the offset relative to the user register pointer. When <name> is executed, the address left will be the sum of n and the user register pointer. Used to put variables in RAM when the system itself is in ROM, and in multi-user systems.

VARIABLE (+++) [227]

A defining word used in the form:

VARIABLE <name>

to create a dictionary entry for <name> and allot two bytes for storage in the parameter field. The application must initialize the stored value, i.e. xForth does not set it to any particular value. When <name> is later executed, it will place the storage

address on the stack.

VLIST (---)

List all the words in the present context vocabulary, and all vocabularies contained in it. May be stopped by hitting control/C.

VOC-LINK (--- addr) [U]

A system variable used for vocabulary linking. Not for user alteration!

VOCABULARY (+++) [208]

A defining word used in the form:

VOCABULARY <name>

to create (in the CURRENT vocabulary) a dictionary entry for <name>, specifying a new ordered list of word definitions. Subsequent execution of <name> will make it the CONTEXT vocabulary. When <name> becomes the CURRENT vocabulary (see DEFINITIONS), new definitions will be created in that list.

In xForth, new vocabularies chain to the vocabulary they were created in, so that when a dictionary search through a vocabulary is exhausted, the vocabulary they chain to is searched. The search starts with the most recently created word, whether or not that word was created after the outer vocabulary.

WARM (---)

Call EMPTY-BUFFERS and then ABORT.

WARNING (--- addr) [U]

A system variable used for error handling. If its low order bit is set to 1, error messages are read from blocks 4 onwards of FILE-A. If the low order bit is set to 0, numeric error messages are given. If the high order (sign) bit is set to zero, a warning message (message 4) is given when a word is redefined. If the high order bit is set to 1, message 4 is suppressed.

WHERE (n1 n2 ---)

Invoke the screen editor with the cursor pointing after character n1 of block n2. If used after ERROR caused by a compilation error while reading from disc, will leave the cursor just after the word where the error was detected.

WHILE (flag ---) [I,C,149]

Used in a colon definition in the form:

BEGIN ... WHILE ... REPEAT

Select conditional execution based on the flag. On a true flag, continue execution through to REPEAT, which then returns back to just after BEGIN. On a false flag, transfer execution to just after REPEAT.

WIDTH (--- addr) [U]

A system variable containing the maximum number of letters saved in the compilation of a definition's name. The default value is 31 and the allowed range is 1,...,31. If the value of WIDTH is less than the natural character count for a word, only WIDTH characters will be saved, together with the actual character count.

WORD (char +++ addr) [181]

Receive characters from the input stream until the non-zero delimiting character is encountered or the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed

string at addr, with the character count at addr and the string following. The actual delimiter encountered (char or null) is stored at the end of the text but is not included in the count.

Note: The facts that (i) leading delimiters are skipped, and (ii) WORD needs to move its text to insert the count, often lead to difficulties. The xForth words skip-while and wrap achieve the same effects as WORD with greater flexibility.

XCANCEL (--- addr) [U]

A system variable containing the compilation address of the code used in EXPECT to cancel all input.

XCURSOR (--- addr) [U]

A system variable containing the compilation address of the code used by CURSOR.

XEMIT (--- addr) [U]

A system variable containing the compilation address of the code used by EMIT.

XERROR (--- addr) [U]

A system variable containing the compilation address of the code used by ERROR. Set to STD-ERROR initially.

XKEY (--- addr) [U]

A system variable containing the compilation address of the code used by KEY.

XNUMBER (--- addr) [U]

A system variable containing the compilation address of code called by the interpreter to attempt to decode a word not found in the dictionary. Set initially to attempt to decode the word as a number in the present base; if the number contains one or more decimal points the number will be taken to be a double number and DPL will be set to the position of the last such decimal point. If the word cannot be decoded as a number, the default code executes 0 ERROR.

XOFF-CHAR (--- addr)

A variable containing -1 or else the ASCII code of the variable used to temporarily pause execution. Typically set to the ASCII character XOFF, viz. control/S. See ?PAUSE.

XOK (--- addr) [U]

A system variable containing the compilation address of the code called to indicate normal return of control to terminal.

XOR (n1 n2 --- n3) [174]

Leave the bitwise exclusive-or of two numbers. "x-or"

XPAGE (--- addr) [U]

A system variable containing the compilation address of the code called by PAGE, usually to blank a screen or cause a paper throw.

XPROMPT (--- addr) [U]

A system variable containing the compilation address of the code always called just before control is returned to terminal.

XRUBOUT (--- addr) [U]

A system variable containing the compilation address of the code used in EXPECT to remove last character typed.

XSIGNON (--- addr) [U]

A system variable containing the compilation address of the code called by ABORT just before control is returned to terminal.

[(---) [I,125]

End the compilation mode. The text from the input stream is subsequently executed. See]. "left-bracket"

[,]VARIABLE (n1 n2 +++)

A defining word used in the form:

n1 n2 [,]VARIABLE <name>

to create a dictionary entry for a two dimensional array of 16 bit integers, with rows numbered 0 to n1 and columns 0 to n2. On later execution, n3 n4 <name> returns the address of the element in row n3 and column n4.

[COMPILE] (+++) [I,C,179]

Used in a colon definition in the form:

[COMPILE] <name>

Force compilation of the following word. This allows compilation of an IMMEDIATE word that would otherwise be executed. "bracket-compile"

[]VARIABLE (n1 +++)

A defining word used in the form:

n1 []VARIABLE <name>

to create a dictionary entry for an array of 16 bit integers numbered 0 to n1. On later execution, n2 <name> returns the address of element n2.

] (---) [126]

Set the compilation mode. The text from the input stream is subsequently compiled. See [and LITERAL. "right-bracket"

^EMIT (byte ---)

Strip the high bit of byte to convert it to a char, then if the result is a printing character, execute EMIT after having first executed CR if the value of OUT was not less than C/L. If the result is not a printing character, take a new line if the value of OUT is not at least 1 less than C/L. Then send ^X to the output device, where X is replaced by the ASCII character formed by adding the value of char to the ASCII code for @. In the special case of ASCII 127, send ^?.

^TYPE (s ---)

Send the string s to the output device, using ^EMIT for each character instead of EMIT. (This will wrap long lines and will print control characters visibly.)

ch-in-str? (char s --- n)

If char is contained in s, return its position in s. Otherwise return zero. Note that for most purposes (e.g. testing with IF) n can be regarded as a true/false flag.

close-files (---)

Close all files that presently own virtual memory segments.

copy (---)

Copy the input spool to the output spool.

count (addr --- addr n)

Return the number of characters in the string starting at addr and terminated by a null (zero byte). Used by EXPECT$.

defdrv (--- n)

Return the presently selected ('default') disc drive. Drive A: is 1, B: is 2 etc.

eol? (--- flag)

Returns TRUE if the present input is exhausted.

ext (--- addr)

A variable containing the extent which will be opened by OPEN when it opens a file. For CP/M1.4 systems only. Not normally for user alteration.

fassign (n file ---)

Allocate the n'th virtual memory segment to the file.

fdelete (addr --- flag)

Attempt to delete the operating system file corresponding to the file control structure at addr, returning TRUE if successful.

fname! (s addr ---)

Let the name of the file control structure at addr be the string s, if the syntax of s is correct; otherwise execute ERROR.

frelease (n ---)

Make virtual memory segment n belong to no file, flushing buffers and closing any file presently owning the segment if necessary.

getc (--- c)

Read the next character from the input spool. Return control/Z if the input is ended. The input spool is set up using the redirection words == and << as described under the appropriate entries.

in-addr (--- addr)

Return the address of the next byte looked at by the interpreter.

in-range? (n1 n2 n3 --- flag)

Return true if n2<=n1<=n3.

maxdrv (--- n)

Return the number of drives xForth thinks the system has. Set by CONFIG.

n-TAB (n ---)

Print enough spaces to make the value of OUT a multiple of n, if this is possible without OUT

equalling or exceeding C/L. Otherwise execute CR.

put$ ($ ---)

Write a string to the output spool. See putc.

putc (c ---)

Write a character to the output spool. If there are any characters on the spool when control returns to the terminal, the spool is typed and left empty.

seg-size (--- n)

A constant returning the size of a virtual memory segment. (Set to 1000 on delivery.)

skip-char (---)

Pass over one character from the input without interpreting it, i.e. increment the input pointer unless the input is exhausted.

skip-until (char ---)

Advance the input pointer by zero or more until char has been reached or the input is exhausted.

skip-while (char ---)

Advance the input pointer by zero or more as long as char is found, stopping if the input is exhausted.

wrap (char --- $)

Advance the input pointer by zero or more until char is found or the input is exhausted. Return the address and length of the string passed over in this

way. Leave the input pointer beyond char.

Used in conjunction with skip-while, this achieves the same effect as WORD but with a normal rather than a packed string returned. One has the additional flexibility that empty strings may be dealt with.

{ (flag ---)

If flag is true, continue execution; if flag is false, advance the input stream pointer until a matching word | or } is found, then continue interpretation as usual. Used in the form:

```
condition { do-if-true ... | do-if-false ...
}
```

where the | and the do-if-false part are optional. This gives a limited form of IF ... ELSE ... ENDIF usable at execution time. { | } may be nested but little error checking is done. Note that all of { | and } must be in the same input stream, i.e. in the same disc block or on the same terminal input line. ERROR is executed if { with a false flag fails to find a matching | or } before the input stream is ended.

| (flag ---)

Used in conjunction with { and }, giving an execution time version of ELSE. See entry for {.

If executed rather than skipped to, | causes the input pointer to be advanced until a matching } is found. If the end of the input stream is reached, ERROR is executed.

} (---)

Used in conjunction with { and (optionally) | to give an execution time version of ENDIF. See entry for {.

If executed rather than skipped to, } has no action.

xFORTH HANDY REFERENCE

Words in xForth in addition to the FORTH-79 words.

STACK MANIPULATION

2DROP	(d ---)	Discard double item on top of stack.
2DUP	(d --- d d)	Duplicate double item on top of stack.
2OVER	(d1 d2 --- d1 d2 d1)	Copy second top stack double item.
2SWAP	(d1 d2 --- d2 d1)	Exchange top two stack double items.
SP!	(any ---)	Discard all items on stack. "stack pointer store"

COMPARISON

<>	(n1 n2 --- flag)	True if n1 not equal to n2. "not-equal"

ARITHMETIC AND LOGICAL

2*	(n --- 2*n)	Double the stack top. "two-times"
DABS	(d --- |d|)	Absolute value of a double number.
M*	(n1 n2 --- d)	Double number signed product of signed single numbers.
S->D	(n --- d)	Extend signed single number to signed double number.

MEMORY

1+!	(addr ---)	Add 1 to number at address. "one-plus-store"
1-!	(addr ---)	Subtract 1 from number at address. "one-minus-store"
2!	(d addr ---)	Store double number d in 4 bytes starting at addr.
2@	(addr --- d)	Fetch double number from 4 bytes starting at addr.
(CMOVE)	(addr1 addr2 u ---)	Move u bytes from addr1 to addr2, if u is nonzero. Overlapping moves are handled correctly.
BLANKS	(addr n ---)	Fill memory with n blanks starting at addr.
ERASE	(addr n ---)	Fill memory with n nulls starting at addr.

CONTROL STRUCTURES

CASE ... OF ... ENDOF ... OF ... ENDOF ... DEFAULT ... ENDCASE (n ---)

Execute the first part between the OF ... ENDOF for which the stack top on entry to CASE matches the stack top on entry to OF . If no match, perform the DEFAULT part if there is one. If no match and no DEFAULT part, remove the stack top. In all cases continue beyond ENDCASE.

ENDIF	(---)	A synonym for THEN

TERMINAL INPUT-OUTPUT

>LINE	(--- addr)	Return the address of a variable that is incremented by CR. May be used for 'paging' a VDU.
?PAUSE	(---)	If XOFF-CHAR contains -1, do nothing. Otherwise execute ?TERMINAL and return at once if no key has been struck. If the key whose ASCII code is in XOFF-CHAR has been struck then wait until any key except control/C is struck before returning. If control/C is struck, either before or during the paused state, execute 6 ERROR. Note: CR calls ?PAUSE so all words that produce output may be paused or interrupted.
?TERMINAL	(--- flag)	Return TRUE if any key has been struck, leaving the actual key to be read if desired from LAST-KEY
BELL	(---)	Send ASCII code 7 to the terminal. This normally sounds a noise-maker.
BL	(--- 32)	Constant returning ASCII code of a blank.
C/L	(--- n)	Constant returning number of characters in a line of output. (Set to 80 on delivery.)
CURSOR	(n1 n2 ---)	Execute the code pointed to by XCURSOR. The code should move the terminal's cursor to row n1 and column n2, relative to 0 0 as the top left corner.
CRS	(n ---)	Perform CR n times if n>0.
EMITP	(c ---)	Send c to the printer only, without incrementing OUT or checking what output is selected.
EMITT	(c ---)	Send c to the terminal only, without incrementing OUT or checking what output is selected.
ENSURE-LINE	(n ---)	If there are less than n character positions remaining on the present output line, execute CR .
L/S	(--- n)	Constant returning number of lines in a VDU screen. (Set to 24 on delivery.)
LAST-KEY	(--- char)	Return the ASCII code of the last key read by KEY or ?TERMINAL.
OUT	(--- addr)	Variable holding present position of output pointer. Incremented by EMIT, reset to 0 by CR and adjusted by all other xForth output words.
PAGE	(---)	Execute the definition pointed to by XPAGE. Usually causes a paper throw or blanks a screen.

PRINTER-ON?	(--- addr)	A variable that if set to 1 causes output to be copied to the printer. Toggled by control/P during EXPECT or may be set and reset under program control.
TAB	(n ---)	Type as many spaces as needed to bring the output pointer to position n, if possible. Otherwise do nothing.
XOFF-CHAR	(--- addr)	A variable containing the character used in ?PAUSE to pause execution. If it contains -1, ?PAUSE will never attempt to read the keyboard. Set to control/S on delivery.
wrap	(char --- s)	For the input presently being interpreted, advance the pointer until char is found or the input is exhausted. Return the address and length of the string passed over in this way. Leave the input pointer beyond char.
^EMIT	(byte ---)	Execute EMIT for the ASCII value formed from the lower 7 bits of byte, taking a new line if the value of OUT exceeds C/L. If the value is 127 then type ^? If the value is less than 32 then EMIT ^ followed by ASCII value 64+value. (So 3 ^EMIT gives ^C)
^TYPE	(s ---)	As TYPE but use ^EMIT to type each character.

NUMERIC CONVERSION

.BASE	(---)	Type the present base for numeric input/output, in decimal.
.R	(n1 n2 ---)	Type a signed number in the current base, right aligned in a field of width n2.
BINARY	(---)	Set the base for numeric input/output to 2.
D.	(d ---)	Type a signed double number in the current base, followed by a blank.
D.R	(d n ---)	Type a signed double number in the current base, right aligned in a field of width n.
HEX	(---)	Set the base for numeric input/output to 16.

MASS STORAGE INPUT/OUTPUT AND VIRTUAL MEMORY

#files	(--- n)	A constant returning 8, the number of virtual memory segments normally accessible to the user.
's-FCB	(addr1 --- addr2)	Given the address addr1 of a file control structure, return the address addr2 of a part of memory used to communicate file control information with the operating system.
's-name	(addr --- s)	Given the address of a file control structure, return the address and length of a string containg the ASCII name of the corresponding CP/M file.
'th-FILE	(n --- addr)	Return the address of the file control structure owning the n'th virtual memory segment, if this segment has been allocated to a file. Otherwise return 0.
-->	(---)	Continue interpretation with the next block. "next block"
<<	(+++)	Read the next word in the input stream as a CP/M file name, and make that file the spooled input for reading with getc. "redirect-in"
==	(---)	Make the previous spooled output for putc the new spooled input for getc. "pipe"
>>	(+++)	Read the next word in the input stream as a CP/M file name, and copy to that file the spooled output written previously by putc. "redirect-out"
COPIES	(n1 n2 n3 ---)	Copy block n1 to n2, n1+1 to n2+1 etc until n3 blocks have been copied. Overlapping shifts are done correctly.
COPY	(n1 n2 ---)	Copy block n1 to n2.
CPM-CALL	(n1 byte --- n2)	Make a call to the CP/M system for function byte with n1 in the DE register. The contents of the HL register on return are left as n2.
CPM-CALLb	(n byte1 --- byte2)	Execute CPM-CALL and mask off all but the low order byte of the result.
DIR	(n ---)	Display the directory of the disc in CP/M drive n, where n=1 gives drive A and so on.
INDEX	(n1 n2 ---)	Type the first 64 characters each of blocks n1 to n2. May be stopped by hitting any key.
INSTALL-$$$	(+++)	Read the next word from the input stream as a CP/M file and make it the CP/M name of a temporary file owning the segment of virtual memory beyond the last user-accessible segment.
LIST-FILE	(+++)	Read the next word from the input stream as a CP/M file name and list the file in LIST form if the file extension is .BLK Otherwise assume the file is an ASCII text file and list it as such.
LOAD-FILE	(+++)	Read the next word from the input stream as a CP/M file name and LOAD the first block of the file if the file extension is .BLK
P!	(byte1 byte2 ---)	Send byte1 to the port address given by byte2.
P@	(byte1 --- byte2)	Return the byte obtained by inputting from the port address given by byte1.
PREV	(--- addr)	System variable containing number of block most recently referenced via BLOCK. High bit of block number is set if block is updated.
SEE	(n ---)	Invoke the screen editor with the cursor pointing to the first character of block n.
SEE-FILE	(+++)	Read the next word from the input stream as a CP/M file name and invoke the screen editor with the cursor pointing to the first character of the file, if the file extension is .BLK

Operand key as for FORTH-79 except: s s1 s2 are all address and count for string.

[illegible]	(--- addr)	Return the address of a file control structure owning virtual memory segment 0. (Blocks 1 to 999 in the standard system).
TRIAD	(n ---)	Type 3 blocks, including block n, in a form suitable for hard copy.
WHERE	(n1 n2 ---)	Invoke the screen editor with the cursor pointing after character n1 of block n2. If used after ERROR, will leave the cursor just after the word where the error was detected.
copy	(---)	Copy the input spool to the output spool.
fassign	(n file ---)	Allocate the n'th virtual memory segment to the file.
frelease	(n ---)	Release the n'th virtual memory segment so that it belongs to no file.
getc	(--- c)	Read the next character from the input spool. Return control/Z if the input is ended.
put$	(s ---)	As putc but writes a string.
putc	(c ---)	Write a character to the output spool.
seg-size	(--- n)	A constant returning the size of a virtual memory segment. (Set to 1000 on delivery.)

DEFINING WORDS

FILE xxx	(+++)	See entry under MASS STORAGE.
STRING xxx	(n +++)	Create a string variable to hold a text string of length up to n characters. If n exceeds 256, the maximum length will be 256.
	xxx: (--- s)	Returns address of first character and current length of string when executed.
[,]VARIABLE xxx	(n1 n2 +++)	Create a two dimensional array of 16 bit integers, with rows numbered 0 to n1 and columns 0 to n2.
	xxx: (n1 n2 --- addr)	Returns address of element in row n1 and colusn n2 when executed.
[]VARIABLE xxx	(n +++)	Create an array of 16 bit integers numbered 0 to n.
	xxx: (n --- addr)	Returns address of element n when executed.

VOCABULARIES

(EDITOR)	(---)	The vocabulary of internal screen editor words.
ASSEMBLER	(---)	The vocabulary of assembler words.
DEBUG	(---)	The vocabulary used for debugging and tracing.
EMPTY	(---)	Remove all unprotected definitions without altering the stacks. See PROTECT.
VLIST	(---)	List all the words in the present context vocabulary, and all vocabularies contained in it. May be stopped by hitting any key.

COMPILER

"	(+++ s)	Return address and length of string terminated by next " If used in a definition, return address and count when executed.
$FIND	(s --- addr)	Like FIND, but uses s instead of reading a word from the current input.
INTERPRET	(---)	Execute (or compile, if STATE is nonzero) the text from the present input. The input is from the keyboard if the value of BLK is nonzero and from the virtual memory block numbered by BLK otherwise.
C,	(byte ---)	Compile a byte into the dictionary.
MYSELF	(---)	Compile a reference to the latest definition (usually the definition MYSELF is contained in). Allows recursion.
WIDTH	(--- addr)	System variable containing maximum length stored in dictionary for a name. The interpreter stores the number of characters in the name and its natural characters up to the value in WIDTH. The default is 31.
eol?	(--- flag)	Returns TRUE if present input is exhausted. "end-of-line?"
in-addr	(--- addr)	Return address of next byte looked at by interpreter.
skip-char	(---)	Pass over one character without interpreting it.
skip-until	(char ---)	Advance the input pointer until char has been reached or the input is exhausted.
skip-while	(char ---)	Advance the input pointer as long as char is found, stopping if the input is exhausted.
wrap	(char --- s)	For the input presently being interpreted, advance the pointer until char is found or the input is exhausted. Return the address and length of the string passed over in this way. Leave the input pointer beyond char.

TEXT STRINGS AND CHARACTERS

"	(+++ s)	Return address and length of string terminated by next " If used in definition, return address and count when executed.
$!	(s1 s2 ---)	Assign the string literal s1 to the string variable s2, truncating s1 at the right if necessary.
$+	(s1 s2 --- s3)	Concatenate strings.
$<	(s1 s2 --- flag)	Return TRUE if s1 is lexically prior to s2.
$->d	(s --- d flag)	Return TRUE if s contained only digits and, optionally, a leading minus sign. If TRUE is returned then d is the value of the number represented by s in the current base.
$=	(s1 s2 --- flag)	Return TRUE if the strings are identical.
ASCII	(+++ char)	Read the next word from the input and leave the ASCII code of its

Operand key as for FORTH-79 except: s s1 s2 are all address and count for string.

first character.

BL	(--- 32)	Return the ASCII code of a blank.
CTRL	(+++ char)	Read the next word from the input and leave the low order 5 bits of the ASCII code of its first character.
EXPECT$	(addr n --- s)	Read up to n characters from the keyboard as for EXPECT, storing them at addr. Return a string denotation, consisting of addr and the actual number of characters typed.
(skip-until)	(addr1 char --- addr2)	Increase addr1 by 0 or more until the byte address pointed to contains char or 0.
(skip-while)	(addr1 char --- addr2)	Increase addr1 by 0 or more until the byte address pointed to does not contain char or does contain 0.

EXECUTION VARIABLES

REPLACED-BY	(addr +++)	Find the execution code of the next word in the input stream, and store it in addr.
XCANCEL	(--- addr)	Points to execution code used in EXPECT to cancel all input.
XCURSOR	(--- addr)	Points to execution code used by CURSOR.
XEMIT	(--- addr)	Points to execution code used by EMIT.
XERROR	(--- addr)	Points to execution code used by ERROR.
XKEY	(--- addr)	Points to execution code used by KEY.
XNUMBER	(--- addr)	Points to execution code used by NUMBER. This is called by INTERPRET to decode a word that cannot be found in the dictionary. By default, it attempts to decode the word as a number (or as a double number if it contains 1 or more decimal points) in the present BASE, and calls 0 ERROR if it fails.
XOK	(--- addr)	Points to execution code called to indicate normal return of control to terminal.
XPROMPT	(--- addr)	Points to execution code always called just before control is returned to terminal.
XRUBOUT	(--- addr)	Points to execution code used in EXPECT to remove last character typed.
XSIGNON	(--- addr)	Points to code called by ABORT just before control is returned to terminal.

CONSTANTS

BL	(--- 32)	Return the ASCII code of a blank.
C/L	(--- n)	Return the number of characters per line in normal input/output. Set by CONFIG. Typically 80.
FALSE	(--- 0)	Return the value of a FALSE flag.
L/S	(--- n)	Return the number of lines per page (i.e. per VDU screen) in normal input/output. Set by CONFIG. Typically 24.
TRUE	(--- 1)	Return the value of a TRUE flag.

ERROR HANDLING

?COMP	(---)	Error if not compiling.
?DEPTH	(n ---)	Error if stack has less than n entries below n itself.
?ERROR	(flag n ---)	If flag is TRUE, call n ERROR. If flag is false, remove it and n and continue normally.
?EXEC	(---)	Error if not executing.
?LOADING	(---)	Error if not loading from virtual memory.
?PAIRS	(n1 n2 ---)	Error if n1 is not equal to n2. Used in checking syntax of conditionals.
?STACK	(---)	Error if stack is out of bounds.
ERROR	(n ---)	Execute the definition whose code field address is in XERROR. Set to STD-ERROR on delivery.
MESSAGE	(n ---)	Type error message n but continue normally.
STD-ERROR	(n ---)	Issue error message n, clear stacks, and obey QUIT.
WARNING	(--- addr)	A system variable used for error handling. If its lowest order bit is set to 1, error messages are read from blocks 4 onwards of FILE-A. If this bit is set to 0, numeric error messages are given. If the high order bit is set to 1 (i.e. the value is negative), warning messages are suppressed whatever the value of the low order bit although true errors are caught as usual.
in-range?	(n1 n2 n3 --- flag)	True if n2<=n1<=n3.

ASSEMBLER

CODE xxx	(+++)	Begin definition of word in assembler code.
END-CODE	(---)	End CODE or ;CODE definition.
LABEL xxx	(+++)	Insert a label in the dictionary and set the CONTEXT vocabulary to ASSEMBLER.

Operand key as for FORTH-79 except: s s1 s2 are all address and count for string.

BYE	(---)	Call SAVE-BUFFERS and then return control to the operating system.
COLD	(---)	Reset the system to the state it had when PROTECT was last called, or to the initial start-up state. This includes removing all new definitions, resetting the filing system without saving buffers or closing files, and resetting all system and execution variables. Finally call ABORT.
CONFIG	(---)	Equivalent to 2 LOAD. Set up on delivery to ask the user for the values of DEL-KEY etc, then allow the user to change the editor key bindings if desired, and finally to execute SYSADAPT with a 'reasonable' number of buffers.
PROTECT	(---)	Set the initialisation sequence so that the present state will be restored by COLD.
SYSADAPT	(n ---)	Call PROTECT then set the initialisation sequence so that n buffers will be used for virtual memory, and all possible memory will be used. Then call COLD.
SYSGEN	(---)	Generate the standard system from the kernel system.
WARM	(---)	Call EMPTY-BUFFERS and then ABORT.

Operand key as for FORTH-79 except: s s1 s2 are all address and count for string.

Stack inputs and outputs are shown; top of stack on right. See operand key at bottom.

STACK MANIPULATION

DUP	(n → n n)	Duplicate top of stack.
DROP	(n →)	Discard top of stack.
SWAP	(n1 n2 → n2 n1)	Exchange top two stack items.
OVER	(n1 n2 → n1 n2 n1)	Make copy of second item on top.
ROT	(n1 n2 n3 → n2 n3 n1)	Rotate third item to top. "rote"
PICK	(n1 → n2)	Copy n1-th item to top. (Thus 1 PICK = DUP , 2 PICK = OVER)
ROLL	(n →)	Rotate n-th item to top. (Thus 2 ROLL = SWAP , 3 ROLL = ROT)
?DUP	(n → n (n))	Duplicate only if non-zero. "query-dup"
>R	(n →)	Move top item to "return stack" for temporary storage (use caution). "to-r"
R>	(→ n)	Retrieve item from return stack. "r-from"
R@	(→ n)	Copy top of return stack onto stack. "r-fetch"
DEPTH	(→ n)	Count number of items on stack.

COMPARISON

<	(n1 n2 → flag)	True if n1 less than n2. "less-than"
=	(n1 n2 → flag)	True if top two numbers are equal. "equals"
>	(n1 n2 → flag)	True if n1 greater than n2. "greater-than"
0<	(n → flag)	True if top number negative. "zero-less"
0=	(n → flag)	True if top number zero. (Equivalent to NOT) "zero-equals"
0>	(n → flag)	True if top number greater than zero. "zero-greater"
D<	(d1 d2 → flag)	True if d1 less than d2. "d-less-than"
U<	(un1 un2 → flag)	Compare top two items as unsigned integers. "u-less-than"
NOT	(flag → ¬flag)	Reverse truth value. (Equivalent to 0=)

ARITHMETIC AND LOGICAL

+	(n1 n2 → sum)	Add. "plus"
D+	(d1 d2 → sum)	Add double-precision numbers. "d-plus"
−	(n1 n2 → diff)	Subtract (n1−n2) "minus"
1+	(n → n+1)	Add 1 to top number. "one-plus"
1−	(n → n−)	Subtract 1 from top number. "one-minus"
2+	(n → n+2)	Add 2 to top number. "two-plus"
2−	(n → n−2)	Subtract 2 from top number. "two-minus"
*	(n1 n2 → prod)	Multiply. "times"
/	(n1 n2 → quot)	Divide (n1/n2). (Quotient rounded toward zero) "divide"
MOD	(n1 n2 → rem)	Modulo (i.e., remainder from division n1/n2). Remainder has same sign as n1. "mod"
/MOD	(n1 n2 → rem quot)	Divide, giving remainder and quotient. "divide-mod"
*/MOD	(n1 n2 n3 → rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate. "times-divide-mod"
*/	(n1 n2 n3 → quot)	Like */MOD , but give quotient only, rounded toward zero. "times-divide"
U*	(un1 un2 → ud)	Multiply unsigned numbers, leaving unsigned double-precision result. "u-times"
U/MOD	(ud un → urem uquot)	Divide double number by single, giving remainder and quotient, all unsigned. "u-divide-mod"
MAX	(n1 n2 → max)	Leave greater of two numbers. "max"
MIN	(n1 n2 → min)	Leave lesser of two numbers. "min"
ABS	(n → |n|)	Absolute value. "absolute"
NEGATE	(n → −n)	Leave two's complement.
DNEGATE	(d → −d)	Leave two's complement of double-precision number. "d-negate"
AND	(n1 n2 → and)	Bitwise logical AND.
OR	(n1 n2 → or)	Bitwise logical OR.
XOR	(n1 n2 → xor)	Bitwise logical exclusive-OR. "x-or"

MEMORY

@	(addr → n)	Replace address by number at address. "fetch"
!	(n addr →)	Store n at addr. "store"
C@	(addr → byte)	Fetch least significant byte only. "c-fetch"
C!	(n addr →)	Store least signficant byte only. "c-store"
?	(addr →)	Display number at address. "question-mark"
+!	(n addr →)	Add n to number at addr. "plus-store"
MOVE	(addr1 addr2 n →)	Move n numbers starting at addr1 to memory starting at addr2, if n>0.
CMOVE	(addr1 addr2 n →)	Move n bytes starting at addr1 to memory starting at addr2, if n>0. "c-move"
FILL	(addr n byte →)	Fill n bytes in memory with byte beginning at addr, if n>0.

CONTROL STRUCTURES

DO . . . LOOP	do: (end+1 start →)	Set up loop, given index range.
I	(→ index)	Place current loop index on data stack.
J	(→ index)	Return index of next outer loop in same definition.
LEAVE	(→)	Terminate loop at next LOOP or +LOOP , by setting limit equal to index.
DO . . . +LOOP	do: (limit start →) +loop: (n →)	Like DO . . . LOOP , but adds stack value (instead of always 1) to index. Loop terminates when index is greater than or equal to limit (n>0), or when index is less than limit (n<0). "plus-loop"
IF . . . (true) . . . THEN	if: (flag →)	If top of stack true, execute.
IF . . . (true) . . . ELSE . . . (false) . . . THEN	if: (flag →)	Same, but if false, execute ELSE clause.
BEGIN . . . UNTIL	until: (flag →)	Loop back to BEGIN until true at UNTIL .
BEGIN . . . WHILE . . . REPEAT	while: (flag →)	Loop while true at WHILE ; REPEAT loops unconditionally to BEGIN . When false, continue after REPEAT .
EXIT	(→)	Terminate execution of colon definition. (May not be used within DO . . . LOOP)
EXECUTE	(addr →)	Execute dictionary entry at compilation address on stack (e.g., address returned by FIND).

Operand key:

n, n1, . . . 16-bit signed numbers	d, d1, . . . 32-bit signed numbers	addr, addr1, . . . addresses	char 7-bit ascii character value
u unsigned	byte 8-bit byte	flag boolean flag

CR	(→)	Do a carriage return and line feed. "c-r"
EMIT	(char →)	Type ascii value from stack.
SPACE	(→)	Type one space.
SPACES	(n →)	Type n spaces, if n>0.
TYPE	(addr n →)	Type string of n characters beginning at addr, if n>0.
COUNT	(addr → addr+1 n)	Change address of string (prefixed by length byte at addr) to TYPE form.
-TRAILING	(addr n1 → addr n2)	Reduce character count of string at addr to omit trailing blanks. "dash-trailing"
KEY	(→ char)	Read key and leave ascii value on stack.
EXPECT	(addr n →)	Read n characters (or until carriage return) from terminal to address, with null(s) at end.
QUERY	(→)	Read line of up to 80 characters from terminal to input buffer.
WORD	(char → addr)	Read next word from input stream using char as delimiter, or until null. Leave addr of length byte.

NUMERIC CONVERSION

BASE	(→ addr)	System variable containing radix for numeric conversion.
DECIMAL	(→)	Set decimal number base.
.	(n →)	Print number with one trailing blank and sign if negative. "dot"
U.	(un →)	Print top of stack as unsigned number with one trailing blank. "u-dot"
CONVERT	(d1 addr1 → d2 addr2)	Convert string at addr1+1 to double number. Add to d1 leaving sum d2 and addr2 of first non-digit.
<#	(→)	Start numeric output string conversion. "less-sharp"
#	(ud1 → ud2)	Convert next digit of unsigned double number and add character to output string. "sharp"
#S	(ud → 0 0)	Convert all significant digits of unsigned double number to output string. "sharp-s"
HOLD	(char →)	Add ascii char to output string.
SIGN	(n →)	Add minus sign to output string if n<0.
#>	(d → addr n)	Drop d and terminate numeric output string, leaving addr and count for TYPE. "sharp-greater"

MASS STORAGE INPUT/OUTPUT

LIST	(n →)	List screen n and set SCR to contain n.
LOAD	(n →)	Interpret screen n, then resume interpretation of the current input stream.
SCR	(→ addr)	System variable containing screen number most recently listed.
BLOCK	(n → addr)	Leave memory address of block, reading from mass storage if necessary.
UPDATE	(→)	Mark last block referenced as modified.
BUFFER	(n → addr)	Leave addr of a free buffer, assigned to block n; write previous contents to mass storage if UPDATEd.
SAVE-BUFFERS	(→)	Write all UPDATEd blocks to mass storage.
EMPTY-BUFFERS	(→)	Mark all block buffers as empty, without writing UPDATEd blocks to mass storage.

DEFINING WORDS

: xxx	(→)	Begin colon definition of xxx . "colon"
;	(→)	End colon definition. "semi-colon"
VARIABLE xxx	(→)	Create a two-byte variable named xxx ; returns address when executed.
	xxx: (→ addr)	
CONSTANT xxx	(n →)	Create a constant named xxx with value n; returns value when executed.
	xxx: (→ n)	
VOCABULARY xxx	(→)	Create a vocabulary named xxx ; becomes CONTEXT vocabulary when executed.
CREATE . . . DOES>	does: (→ addr)	Used to create a new defining word, with execution-time routine in high-level FORTH. "does"

VOCABULARIES

CONTEXT	(→ addr)	System variable pointing to vocabulary where word names are searched for.
CURRENT	(→ addr)	System variable pointing to vocabulary where new definitions are put.
FORTH	(→)	Main vocabulary, contained in all other vocabularies. Execution of FORTH sets context vocabulary.
DEFINITIONS	(→)	Sets CURRENT vocabulary to CONTEXT.
' xxx	(→ addr)	Find address of xxx in dictionary; if used in definition, compile address. "tick"
FIND	(→ addr)	Leave compilation address of next word in input stream. If not found in CONTEXT or FORTH, leave 0.
FORGET xxx	(→)	Forget all definitions back to and including xxx , which must be in CURRENT or FORTH.

COMPILER

,	(n →)	Compile a number into the dictionary. "comma"
ALLOT	(n →)	Add two bytes to the parameter field of the most recently-defined word.
."	(→)	Print message (terminated by "). If used in definition, print when executed. "dot-quote"
IMMEDIATE	(→)	Mark last-defined word to be executed when encountered in a definition, rather than compiled.
LITERAL	(n →)	If compiling, save n in dictionary, to be returned to stack when definition is executed.
STATE	(→ addr)	System variable whose value is non-zero when compilation is occurring.
[	(→)	Stop compiling input text and begin executing. "left-bracket"
]	(→)	Stop executing input text and begin compiling. "right-bracket"
COMPILE	(→)	Compile the address of the next non-IMMEDIATE word into the dictionary.
[COMPILE]	(→)	Compile the following word, even if IMMEDIATE. "bracket-compile"

MISCELLANEOUS

(	(→)	Begin comment, terminated by) on same line or screen; space after (. "paren", "close-paren"
HERE	(→ addr)	Leave address of next available dictionary location.
PAD	(→ addr)	Leave address of a scratch area of at least 64 bytes.
>IN	(→ addr)	System variable containing character offset into input buffer; used, e.g., by WORD. "to-in"
BLK	(→ addr)	System variable containing block number currently being interpreted, or 0 if from terminal. "b-l-k"
ABORT	(→)	Clear data and return stacks, set execution mode, return control to terminal.
QUIT	(→)	Like ABORT , except does not clear data stack or print any message.
79-STANDARD	(→)	Verify that system conforms to FORTH-79 Standard.

FORTH INTEREST GROUP, P.O. Box 1105, San Carlos, CA 94070, USA

Please fill in and sign this agreement and return it with your order. Although the licence only covers a single computer, it may be extended at a reduced price by agreement with us. If you sell your computer, we may transfer the licence to the new owner for a nominal charge. Registering with us means you will be able to get updates and new manuals at specially reduced prices.

A.I.M. Research
20 Montague Road, Cambridge CB4 1BX.

Software Licence Agreement.

A.I.M. Research agrees to grant and the customer signing below (hereinafter "the Customer") agrees to accept on the following conditions a non-transferable and non-exclusive licence to use the software detailed below ("the Software") on the computer system detailed below ("the System").

The Software may be used in any machine-readable form on the System by the Customer or by others under the Customer's personal supervision. Any number of copies of the Software may be made for use on the System under the conditions of this licence. The Software may be modified or adapted for use on the System but neither in its original nor in its modified form may the Software be sold or otherwise transferred without the prior written consent of A.I.M. Research. In this context, transfer includes but is not restricted to any means by which the source code, the executable code or the design of the software may be made available, directly or indirectly, for use by anyone who is not a licence holder.

In the absence of a specific written agreement by A.I.M. Research that the Software is suitable for a given purpose, the liability of A.I.M. Research shall be restricted to the replacement of the Software or, at our option, the refund of its purchase price.

Customer:
Name: Signature:
Date:

Address:

System:
Manufacturer:
Model: Serial number:
Operating system: Serial number:

Software: xForth