Appendix A

Glossary of TILE Words

This glossary includes a complete description of the entire TILE word set together with information regarding stack arguments, source code (where applicable) and any other relevant information.

The following abbreviations are used to characterise the words

I	An immediate word.
C	May only be used in compilation mode.
E	May only be used in execute mode.
P	Code primitive.
PRE	Defined in the 'prelude' file.

In addition to the abbreviations specified above, there are also a number of stack related abbreviations used in describing the type of stack arguments consumed or produced by a word. The conventional format of stack abbreviations is (**stack before -- stack after**) thus a word specified by the abbreviation (**n addr -- c**) consumes a single length signed number and an address from the parameter stack respectively, and produces a 7-bit character value.

n	Single-length signed number.
d	Double-length signed number.
u	single-length unsigned number.
ud	Double-length unsigned number.
c	7-bit character value.
b	8-bit byte.
f	Boolean flag; or; True (-1) False (0)
addr	Address
nfa	Name field address.
lfa	Link field address.
cfa	Code field address.
pfa	Parameter field address.

The glossary is ordered according to **ASCII** value of the words with the smallest value first and the largest value last. To aid in the location of a word the **ASCII** alphabet is printed at the bottom of each page.

! "store" P
(n addr --)
Stores n at addr.

"sharp" PRE
(d1 -- d2)
The rightmost digit of d1 is converted to an ASCII character according to the value of **BASE** and appended to a pictured (formatted) output string (for subsequent output by **type**). d2 is a number with the remaining digits and is maintained for further processing.Used between <# and #>. A word to display a signed double-length number of pence as pounds and pence could be defined thus :-

```
: .pounds ( d -- )
    swap over dabs <# # # 46 hold #s 36 hold rot sign #> type ;
```

See also <# , # , #s , **hold** , **sign** , #> .

#> "sharp-greater" PRE
(d -- addr n)
Ends conversion of a number to a pictured (formatted) output string. addr is the address of the resulting output string and n is the number of characters in it. addr and n together are suitable for use by **type**. See # for an example.See also <# , #, #s , **hold** , **sign**.

#s "sharp-s" PRE
(d -- 0 0)
Converts d to a pictured (formatted) output string, digit by digit, according to the current value in **BASE**. See # for an example. A single zero is placed in the output string if d was initially zero. Used between <# and #>.

' "tick" PRE , I
(-- cfa)
Used in the form :-

```
' <name>
```

returns the code filed address of <name>, or false if <name> is not in the vocabulary list.

(+loop) "brackets-plus_loop" P
(n --)
When used in the form :-

```
: < name > ... compile (+loop) ... ;
```

a loop termination operation is compiled into a colon-definition. When executed, the top single-length value of the return stack is incremented by n, and compared with the second single-length value on the return stack. If n is positive and the second is greater than the first, or n is negative and second is smaller than the first then a relative backward jump is made, otherwise execution continues with the word compiled immediately after the branch address that follows **(+loop)**. Each occurrence of **(+loop)** should be accompanied with a **(do)** to prevent the return stack from becoming corrupted.

(.") "brackets-dot-quote" P
(--)
Displays the counted string that follows immediately after **(.")** and continues execution with the word that follows immediately after the last character.

(:) "bracket-colon" P , E , I
(--)
A defining word that creates a colon-word. Used only in execution mode, in the form :-

(:) < name > ... (;)

creates a dictionary entry for <name> in the current vocabulary and sets **STATE** to compile mode, causing the following words and numbers from the input stream to be compiled. < name > cannot be found in the dictionary until a corresponding **(;)** or **;code** is successfully processed. **(:)** sets the search order so that the first vocabulary to be searched is replaced by the compilation vocabulary. The compilation vocabulary is unchanged. Used in the definition of **:** but contains no error checking.

(;) "brackets-semi-colon" P , I , C
(--)
Stops compilation of a colon-definition. Used in the form :-

(:) < name > ... (;)

enables < name > to be found in the dictionary and sets **STATE** to execute mode. Used in the definition of **;** but contains no error checking.

(do) "brackets-do" P
(n1 n2 --)
When used in the form :-

: < name > ... compile (do) ... ;

a loop initialisation operation is compiled into a colon-definition. When executed, n1 and n2 are transferred to the return stack and used as a loop counter and loop delimiter respectively. Each occurrence of a **(do)** must be accompanied with either a **(loop)** or **(+loop)** to prevent the return stack from becoming unbalanced.

(emit) "brackets-emit" P
(n --)
Sends the character represented by the least significant byte of n to the console but does not increments **OUT.**

(endif) "brackets-end-if" PRE , C
(addr 2 --)
Calculate the relative difference between addr and the next available dictionary location (the value returned by **here**) and store the result at addr. The 2 is used to trap errors within constructs such as **if , else , then** which use **(endif).**

(find) "brackets-find" P
(addr1 addr2 -- addr2 False) or (addr1 addr2 -- addr2 nfa True)
Searches the vocabulary starting at addr2 for the counted string at addr2. If no match is found then the address of the counted string together with False (0) are returned. If a match is found then the address of the counted string

together with the name-field address of the matching dictionary entry and True (-1) are retuned.

(loop) "brackets-loop" P
(--)
When used in the form :-

: < name > ... compile (loop) ... ;

a loop termination operation is compiled into a colon-definition. When executed, the top single-length value of the return stack is incremented and compared with the second single-length value on the return stack. If the second is greater than the first then a relative backward jump is made, otherwise execution continues with the word compiled immediately after the branch address that follows **(loop)**. Each occurrence of **(loop)** should be accompanied with a **(do)** to prevent the return stack from becoming corrupted.

(toggle) "brackets-toggle" PRE
(n1 n2 -- n3)
Performs the bit-by-bit toggle of n1 with the bits specified by the single-length mask n2 to produce n3. See also **toggle**.

***** "times" PRE
(n1 n2 -- n3)
Multiplies n1 by n2, giving the product n3. n3 will contain the least significant 16 bits of the product even if arithmetic overflow occurs.

***/** "times-divide" PRE
(n1 n2 n3 -- n4)
Multiplies n1 by n2, divides the result by n3, and leaves the quotient n4. The product of n1 times n2 is maintained as an intermediate double-length number, giving greater precision than the otherwise equivalent sequence.

***/mod** "times-divide-mod" PRE
(n1 n2 n3 -- n4 n5)
Multiplies n1 by n2, divides the result by n3, and leaves the remainder n4 and quotient n5. A double-length intermediate product is used as for */.

+ "plus" P
(n1 n2 -- n3)
Adds n2 to n1 to give the sum n3. n3 contains the least significant 16 bits of the sum even if arithmetic overflow occurs.

++ "plus-plus" PRE
(addr --)
Increment the variable at addr by one.

+ - "plus-minus" PRE
(n1 n2 -- n3)
Returns the absolute value of the signed single-length number n1 if n2 is negative.

+! "plus-store" PRE
(n1 addr --)
Adds n1 to the single-length number at addr, replacing the original value at that address with the sum.

+C! "plus-c-store" PRE
(n1 addr --)
Adds the least significant byte of n1 to the 8 bit byte at addr, replacing the original value at that address with the sum.

+loop "plus-loop" PRE
(n --)
Terminates a do-loop, allowing the loop index to be incremented (or decremented) with values other than 1. Used in the form :-

: <name> ... do ... +loop ;

compiles a do-loop structure. When <name> is executed, **+loop** expects a number(n) on the stack that is used to determine by now much to increment (or decrement) the loop index. The loop is terminated when the index becomes greater than or equal to the limit (in a signed sense). Execution will jump to just after the corresponding **do** if the loop is not terminated an to just after **+loop** if it is. Contrast with **loop**. See also **do**.

, "comma" PRE
(n --)
Stores n into the next available dictionary location (the address returned by **here**) and increments the dictionary pointer by 2 to allocate space for the storage of n. It is said that n is "compiled" at **here**. See also **C,**

- "minus" PRE
(n1 n2 -- n3)
Subtracts n2 from n1, leaving the remainder n3.

-- "minus-minus" PRE
(addr --)
Decrements the variable at addr by one.

-dup "minus-dupe" PRE
(n1 -- n1 n1) or (n1 -- n1)
Duplicates the single-length number n1 only if n1 is not zero, otherwise the stack remains unchanged.

-find "minus-find" PRE
(-- addr False) or (-- addr nfa True)
Used in the form :-

-find < name >

Searches the dictionary chain for <name>. If <name> is not found then the address of <name> in the text-input buffer together with False (0) is returned. If <name> is found the the address of <name> in the text-input buffer, the name-field address of <name> together with True (-1) is returned.

. "dot" PRE
(n --)
Displays the value of n (including a leading minus sign if n is negative), according to the value of **base**, at the current display position, followed by a space.

." "dot-quote" PRE , I , C , E
(--)

Compiles and/or displays text. Used in the form

: <name> ... ." ccc" ... ;

compiles the text ccc following ." up to the delimiter" (double-quote) into the definition of <name>. A blank must follow ." (and is not included in ccc) but a blank need not precede "double-quote",which is a delimiter, not a TILE word. The compiled text will then be displayed when <name> is executed. ." may also be used in execution mode, in which case the text is immediately displayed.

.r "dot-r" PRE
(n1 n2 --)
Displays the value of n1 right justified in a field n2 characters wide according to the value of **BASE**. A leading minus sign is displayed if n1 is negative. It is not an error if the number characters required to display n1 is greater than n2. The results in the number being displayed without a leading space. Also if n2 is less than 1 no leading blanks will be supplied.

/ "divide" PRE
(n1 n2 -- n3)
Divides n1 by n2 and leaves the quotient n3. n3 is rounded toward zero. An error is generated if divisor is zero.

/mod "divide-mod" PRE
(n1 n2 -- n3 n4)
Divides n1 by n2 and leaves the remainder n3 and quotient n4. n4 is rounded toward zero. An error occurs if the divisor is zero.

0< "zero-less" PRE
(n -- f)
Compares n with 0 and returns a true flag if n is less than zero.

0= "zero-equals" PRE
(n -- f)
Compares n with 0 and returns a True (-1) flag if n is zero. See also **not**.

0> "zero-greater" PRE
(n -- f)
Compares n with 0 and returns a True (-1) flag if n is greater than zero.

0branch "zero-branch" P
(f --)
When used in the form :-

: < name > ... compile 0branch ... ;

a conditional branch operation is compiled into a colon-definition. When executed, if a flag on the stack is False (0), the branch is performed as with **branch**. If the flag is True (-1), execution continues with the word compiled immediately after the branch address that follows **0branch**. Not normally used for routine programming, used in the definition of **if** , **while**, and **until**.

1+ "one-plus" PRE
(n1 -- n2)
Adds 1 to n1 to give n2.

1- "one-minus" PRE
(n1 -- n2)
Subtracts 1 from n1 to give n2.

2+ "two-plus" PRE
(n1 -- n2)
Adds 2 to n1 to give n2.

2- "two-minus" PRE
(n1 -- n2)
Subtracts 2 from n1 to give n2.

2dup "two-dupe" PRE
(d -- d d) or (n1 n2 -- n1 n2 n1 n2)
Duplicates the double-length number on the stack or the first and second single-length numbers.

2drop "two-drop" PRE
(d --) or (n1 n2 --)
Removes a double-length number (or two single-length numbers) from the stack.

2over "two-over" PRE
(d1 d2 -- d1 d2 d1) or (n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2)
Copies the second double-length number on the stack to the top of the stack or copies the third and fourth single-length numbers to the top of the stack.

2swap "two-swap" PRE
(d1 d2 -- d2 d1) or (n1 n2 n3 n4 -- n3 n4 n1 n2)
Swaps the top two double-length numbers on the stack or swaps the first and the second second single-length numbers with the third and fourth.

: "colon" PRE , I , E
(--)
A defining word that creates a colon-word. Used only in execution mode, in the form :-

: < name > ... ;

creates a dictionary entry for <name> in the current vocabulary and sets **STATE** to compile mode, causing the following words and numbers from the input stream to be compiled. <name> cannot be found in the dictionary until a corresponding **;** or **;code** is successfully processed. **:** sets the search order so that the first vocabulary to be searched is replaced by the compilation vocabulary. The compilation vocabulary is unchanged. Colon-words are the main means for TILE programming. See also **; , ;code.**

; "semi-colon" PRE , I , C
(--)
Stops compilation of a colon-definition. Used in the form :-

: < name > ... ;

enables < name > to be found in the dictionary and sets **STATE** to execute mode.

;code "semi-colon-code" PRE , I , C
(--)
A defining-word terminator that may only be used in compilation mode. Used in the form :-

: < namex > ... create ... ;code ... **machine code** ...

stops compilation, terminates the defining word <namex> and compiles **scode**. When <namex> is executed in the form :-

<namex> < name >

to create <nam >, the code-field contents of <name> will be the address of the machine code following the **;code** in <namex>. Thus **;code** permits the create of parent words that create child words whose action is defined by the machine code after **;code** in the definition of <namex>. Analogous in action to **does>**. See also **scode , create , does>**.

< "less-than" P
(n1 n2 -- f)
Compares n1 with n2 and returns a True (-1) flag if n1 is less than n2.

<# "less-sharp" PRE
(--)
Starts conversion of a number to a pictured (formatted) output string. See # for an example. See also **#> , #s , hold , sign .**

<> "not-equal" PRE
(n1 n2 -- ?)
Compares n1 with n2 and returns a True (-1) flag if n1 and n2 are not equal.

<builds "builds" PRE , C
(--)
A defining word to create a dictionary entry. Used in the form :-

<builds < name >

creates a dictionary entry for <name> and allocates space in <names>'s parameter field. When <name> is executed, the value 0 is left on the stack. <builds is used within a colon definition in the form :-

: < name > ... <builds ... does> ... ;

to specify the compile-time action of a defining word. See also **does>**.

= "equals" P
(n1 n2 -- f)
Compares n1 with n2 and returns a True (-1) flag if n1 is equal to n2.

> "greater-than" P
(n1 n2 -- f)
Compares n1 with n2 and returns a True (-1) flag if n1 is greater than n2.

>R "to-r" P
(n --)
Transfers n from the stack to the return stack. Since **>R** changes the return stack, it normally must be paired with a corresponding **R>** before a colon-definition is exited.

?comp "question-comp" PRE
(--)
Generates an error if the compilation state (the value of the **STATE** variabl) is False (execution mode).

?exec "question-execution" PRE
(--)
Generates an error if the compilation state (the value of the **STATE** variable) is True (compilation mode).

?pairs "question-pairs" PRE
(n1 n2 --)
Generates an error if the top two stack items are not equal.

@ "fetch" P
(addr -- n)
Returns a single-length number stored at addr. See also **c@**.

BASE "base" PRE
(-- addr)
A variable containing the number base being used for numerical input and output. Thus :-

```
2 BASE !
```

would select binary input and output. See also **decimal** , **hex** , **binary**.

BL "b-l" PRE
(-- 32)
Puts the ASCII code for a space or blank (decimal 3) on the stack.

C! "c-store" P
(n addr --)
Stores the least significant byte of n into a byte at addr. See also **!** .

C, "c-comma" P
(n --)
Stores the least significant byte of n into a byte at the next available dictionary location (the address returned by **here**), and increments the dictionary pointer by 1 to allocate space for the compiled byte. See also , (comma) .

C/L "c-slash-l" PRE
(-- addr)
A variable containing the maximum number of characters per line.

C@ "c-fetch" P
(addr -- n)
Fetches a byte at addr onto the stack. The result is a number whose most significant byte is 0 and whose least significant byte is the fetched byte. See also **@**.

CONTEXT "context" PRE
(-- addr)
Returns the address of a variable that determines the vocabulary to besearched first in the dictionary. See also **CURRENT , definitions , vocabulary , seal .**

CURRENT "current" PRE
(-- addr)
Returns the address of a variable that determines the vocabulary into which new word definitions will be entered See also **CONTEXT , definitions , vocabulary , seal.**

DP "d-p" PRE
(-- addr)
A variable containing the address of the next free location in the dictionary.

FENCE "fence" PRE
(-- addr)
A variable that specifies an address in the dictionary below which words may not be deleted using **forget.**

HLD "h-l-d" PRE
(--)
A variable that contains the byte offset into the text scratch pad where characters are currently being stored.

I "i"
(-- n)
Places the index of a do-loop onto the stack. Can only be used directly and not in a word called from within a do-loop. See also **J.**

IN "in" PRE
(-- addr)
A variable that contains the byte offset into the input stream where interpretation is presently occurring.

INNER "inner" PRE
(-- addr)
A variable containing the code-field address of the inner interpreter.

J "j" P
(-- n)
Used within a nested do-loop in the form, for example :-

do .. do .. J .. loop ... +loop

returns the index of the next outer loop, in this case the loop terminated by **+loop**. Like **I , J** can only be used directly and not in a word called from within the loop.

OUT "out" PRE
(-- addr)
A variable containing a count of the number of characters currently output on a line. See also **C/L , emit.**

R: "r-colon" PRE , E , I
(--)
A defining word that creates a colon-word. Used only in execution mode, in the form :-

:R < name > ... R;

creates a dictionary entry for < name > in the current vocabulary and sets **STATE** to compile mode, causing the following words and numbers from the input stream to be compiled. < name > can be found in the dictionary thus enabling recursion. **R:** sets the search order so that the first vocabulary to be searched is replaced by the compilation vocabulary. The compilationvocabulary is unchanged. See also **R;** , **:** , **;**.

R; "r-semi-colon" PRE , C , I
(--)
Stops compilation of a 'R' colon-definition. Used in the form :-

R: < name > ... R;

Sets **STATE** to execute mode.

R> "r-from" P
(-- n)
Removes n from the return stack and places it on the stack. **R>** generally must be used with a corresponding **>R** to prevent the return stack from becoming unbalanced. See also **R** , **>R**.

R "r" P
(-- r)
Copies n from the return stack to the stack (without changing the return stack). **R** is useful when the return stack is used for temporary storage of numbers within a colon-definition. See also **>R** , **R>**.

STATE "state" PRE
(-- addr)
A variable that contains a value defining the compilation state. A True (-1) value indicates that compilation is occurring and a False (0) value indicates that interpretation is occurring.

TIB "t-i-b" PRE
(-- addr)
returns the address of the start of the text-input buffer (sometimes called the "terminal input buffer"). This is used to receive characters from the keyboard.

WIDTH "width" PRE
(-- addr)
A variable containing the number of characters that will be compiled in the name-field of a definition, thus the maximum length for a word name. **WIDTH** can be used to truncate word names so as to save dictionary space.

["left-bracket" PRE , I
(--)
Sets the system state to execute mode. The text from the input stream is subsequently executed rather than compiled. Used within colon definitions in the form :-

: < name > ... [...] ... ;

allows the words between [and] to be executed when < name > is compiled. See also] , **STATE.**

[compile] "bracket-compile" PRE , I
(--)
Causes an immediate word to be compiled. Used in the form :-

: < namex > ... [compile] < name > ... ;

forces compilation of the immediate word < name >, which would ordinarily be executed even in compile mode. **[compile]** itself is not compiled, in contrast to **compile**, which has a very different function.

\ "backslash" P , I
(--)
Causes the TILE interpreter to ignore the rest of the line of input. Used so that the rest of a line of source code may contain comments. May be used within a colon-definition.

] "right-bracket" PRE , I
(--)
Sets the system state to compile mode. The text from the input stream is subsequently compiled.] is useful for compiling headerless words and for compiling the cfa of words into an array for vectored execution. See also [, **STATE.**

abort "abort" P
(--)
Clears both the parameter and return stacks and sets **STATE** back to execute. No message is displayed, not even " ok ".

abs "absolute" PRE
(n1 -- n2)
Returns the absolute value of the signed number n1, that is, a number of the same magnitude but always with a positive sign.

again "again" PRE , C , I
(--)
Marks the end of an infinite loop. Used in the form :-

: < name > ... begin ... again ... ;

Compiles an unconditional branch back to the corresponding **begin**. When <name> is executed the words between **begin** and **again** willl be repeated without terminating.

allot "allot" PRE
(n --)

Allocates n bytes in the dictionary. That is, the address of the next available dictionary location, the dictionary pointer (returned by **here**), is incremented by n.

and "and" P
(n1 n2 -- n3)
n3 is the bit-by-bit logical "and" of n1 with n2. That is, each bit of n1 is compared with the equivalent bit in n2 and if either bit is 0 a 0 is placed in the corresponding bit position in n3. Thus (in binary) :-

```
110 100 and
```

will return binary 100. See also **or** , **xor**.

begin "begin" PRE
(--)
Marks the start of an indefinite loop structure. Used in the form :-

```
: < name > ... begin ... flag until ... ;
```

or

```
: < name > ... begin ... flag while ... repeat ... ;
```

or

```
: < name > ... begin ... again ... ;
```

compiles an indefinite loop structure. When < name > is executed, the words between **begin** and **until** will be repeated until the flag on the stack is true. The words between **begin** and **repeat** will be repeated while the flag on the stack seen by **while** is true. The words after **until** or **repeat** will be executed when either loop is finished. The words between **begin** and **again** will be repeated without terminating.

binary "binary" PRE
(--)
Selects binary notation for numerical input and output.

```
: binary 2 base ! ;
```

See also **decimal** , **hex** .

branch "branch" PRE
(--)
When used in the form :-

```
: < name > ... compile branch ... ;
```

an unconditional branch operator is compiled. A relative address must be compiled immediately after the compilation address of the unconditional branch operator.

bye "bye" P
(--)
Quit TILE and restore the system to the state just before TILE was invoked.

ca! "c-a-store" PRE
(n --)
Stores n in the code-field address of the last definition in the **CONTEXT** vocabulary.

cfa "c-f-a" PRE
(pfa -- cfa)
Given the parameter-field address of a word, returns the code-field address of that word.

cmove "cmove" PRE
(addr1 addr2 n --)
Copies n bytes beginning at address addr1 to addr2. The move proceeds from low memory toward high memory. If n is zero or negated, nothing is moved. See also **move** .

compile "compile" PRE
(--)
Typically used in the form :-

: < name1> ... compile < name2 > ... ;

When <name1> is compiled, the code-field address of <name2> is compiled in the dictionary following the code-field address of **compile**. <name1> is typically immediate and <name2> is typically not immediate. When <name1> is executed, the code-field address of <name2> is compiled at the top of the dictionary, usually in a colon-definition.

constant "constant" PRE
(n --)
A defining word that creates a single-length constant. Used in the form :-

n constant < name >

creates a dictionary entry for <name> and compiles n into <name>'s parameter field. When <name> is executed, the single-length number n is placed on the stack. See also **variable.**

count "count" PRE
(addr1 -- addr2 n)
Returns the number of characters in a counted string at addr1. addr2 is addr1+1, the start of the text, and n is the length of the text. **count** returns an address and count suitable for use by **type**. thus the phrase

addr count type

will display the counted string stored at addr.

: count dup C@ swap 1+ ;

cr "c-r" PRE
(--)
Sends a carriage return and line feed to the console, moving the cursor to the start of the next line, and resets **OUT**.

create "create" P
(--)
A defining word to create a dictionary entry. Used in the form

create < name >

creates a dictionary entry for <name> without allocating space in <names>'s parameter field, and sets the code-field address to point the parameter-field. Executing <name> without modification to either to code- or parameter-fields will result in an error, thus <name> is normally only used in association with other defining words. See also **<builds , does>**.

d+ "d-plus" P
(d1 d2 -- d3)
Adds d2 to d1 and returns the double-length sum d3.

d+ - "d-plus-minus" PRE
(d1 d2 -- d3)
Returns the absolute value of the double-length signed number d1 if d2 is negative.

d- "d-minus" PRE
(d1 d2 -- d3)
Subtracts d2 from d1 and returns the double-length difference d3.

d. "d-dot" PRE
(d --)
Displays, according to the value of **BASE**, the value of d (including a leading minus sign if negative) at the current display position, followed by a space. See also . (dot).

d.r "d-dot-r" PRE
(d n --)
Displays the value of d right justified in a field n characters wide. A leading minus sign is displayed if d is negative. If n is less than 1, no leading blanks will be supplied and it is not an error if the number of characters required to display d is greater than n.

dabs "d-absolute" PRE
(d1 -- d2)
Returns the absolute value of the signed double-length number d1. That is, d2 will be a double-length number of the same magnitude as d1 but will always have a positive sign.

decimal "decimal" PRE
(--)
Selects decimal notation for input and output. Puts 10 (decimal) into the variable **BASE**.

definitions "definitions" PRE
(--)
Sets the vocabulary into which definitions are compiled to be the same as the first vocabulary in the search order. This is achieved by setting the contents of the variable **CURRENT** equal to that of the variable **CONTEXT**. For example , executing :-

logic definitions

would make subsequent word definitions be placed in the logic vocabulary. See also **vocabulary**.

dminus "d-minus" P
(d1 -- d2)
Reverses the sign of d1 to produce d2. d2 is the twos complement of d1 (i.e. zero minus d1).

do "do" PRE , I , C
(n1 n2 --)
Marks the start of a do-loop. Used in the form :-

: < name > ... n1 n2 do ... loop ... ;

or

: < name > ... n1 n2 do ... increment +loop ... ;

marks the beginning of a do-loop. When < name > is executed, **do** takes both n1 and n2 from the stack and uses n2 as the starting value of the loop index and n1 as the limit that will determine when the do-loop is terminated. See **loop** and **+loop** for the differences in loop termination. Any do-loop is always executed at least once.

does> "does" PRE , I
(-- addr)
Defines the execution-time action of a word created by a defining word. Used in the form :-

: < namex > ... <builds ... does> ... ;

where <builds creates a child dictionary entry, and does> marks the start of the part of <namex>'s definition that determines the execution behaviour of its children. Thus when < namex > is executed in the form :-

< namex > < name >

a dictionary entry is created for the child word <name> and the execution-time action for <name> will be determined by the sequence of words between **does>** and **;** in the definition <namex>. **does>** may be used only within a colon-definition. For example, the definition of **constant** could be specified thus :-

: constant <builds , does> @ ;

See also **<builds , ;code , scode.**

drop "drop" P
(n -- n)
Removes n from the stack.

dup "dupe" P
(n - n n)
Duplicates the single-length number n on the top of the stack.

else "else" PRE , C , I
(--)
Marks the start of an alternative branch. Used in the form :-

: < name > ... flag if ... else ... then ... ;

compiles an unconditional branch operator to guide execution of the word to just after **then**. When < name > is executed, **if** expects a flag on the stack. If the flag is nonzero, the words between **if** and **else** will be executed with a jump to after **then** but if the flag is zero, execution will jump to the words between **else** and **then** with continuation after **then** . See also **if** , **then** .

emit "emit" PRE
(n --)
Sends the character represented by the least significant byte of n to the console and increments **OUT.** see also **cr** , **OUT** .

empty? "empty-question" P
(-- f)
Returns single-length value of True (-1) if the stack is empty otherwise a single-length value of False (0) is returned.

execute "execute" P
(cfa --)
Executes the word whose code-field address is on the stack.

fill "fill" PRE
(addr n1 n2 --)
Fills n1 bytes of memory beginning at addr with the least significant byte of n2. No action is taken if n1 is zero or negative.

forget "forget" PRE
(--)
Used in the form :-

forget <name>

deletes <name> and all words added to the dictionary after <name>, regardless of their vocabulary, if <name> is found in the vocabulary to which definitions are being added. If <name> is a vocabulary link then it is removed and **CONTEXT** and **CURRENT** are reset to point the previous hierarchical vocabulary.

here "here" PRE
(-- addr)
Returns the address of the next available location in the dictionary, the location of the dictionary pointer.

hex "hex" PRE
(--)
Selects hexadecimal notation for input and output.

: hex 16 BASE ! ;

hold "hold" PRE
(n --)
Inserts a character with ASCII value n into a pictured numeric output string. Used between **<#** and **#>**. See **#** for an example.

id. "id." PRE
(nfa --)
Displays the name of definition give its name-field address.

if "if" PRE
(flag --)
Starts a one- or two-way branch structure. Used in the form :-

: < name > ... flag if ... then ... ;

or

: < name > ... flag if ... else .. then ... ;

when <name> is executed **if** expects a flag on the stack. If the flag is nonzero the words between **if** and **else** (or the words between **if** and **then** if **else** is not used) are executed. But if the flag is zero execution branches to the words between **else** and **then** (or to the word after **then** if **else** is not used). In either case execution continues with the word after **then** . **if ... else ... then** constructs may be nested.

immediate "immediate" PRE
(--)
Marks the most recently created dictionary entry as a word that will be executed immediately even if TILE is in compile mode (rather than being compiled) as well as if tile is in execute mode.

inline "inline" P
(--)
Pauses to receive a series of characters from the current input stream, stores them in the text-input buffer and resets **IN** to point to the beginning of the received text. Input is terminated when either a carriage return or end of file character is received. In the case of an end of file character either the stream is reset to accept input form the keyboard if input was previously from a file, or TILE is exited if the input was form the keyboard.

key "key" P
(-- n)
Pauses to wait for a key to be pressed and then places the ASCII value of the key (n) on the stack. Characters received by **key** are not displayed.

last "last" PRE
(-- nfa)
Returns the name-filed address of the last definition in the **CONTEXT** vocabulary. See also **CONTEXT** , **CURRENT**.

lfa "l-f-a" PRE
(pfa -- lfa)
Given the parameter-field address of a word, returns he link-field address of that word.

lit "lit" P
(-- n)
A word that is compiled by **literal** so as to put the 16-bit value following it in the dictionary onto the stack. See also **compile**.

literal "literal" PRE , I
(n --)
Typically used in the form :-

: < name > ... [n] literal ... ;

compiles **lit** and the number (n) on the stack during compilation. When <name> is executed n is placed on the stack.

load "load" P
(--)
Used in the form :-

load < name >

loads and compiles the file specified by the space delimited string <name>.<name> must not contain any spaces and interpretation of the line is terminated after the load.

loop "loop" PRE , C
(--)
Used in the form :-

: < name > ... do ... loop ;

defines the end point of a do-loop. When <name> is executed, **do** sets the initial value of the loop index and the loop limit. Each iteration of the loop causes the loop index to be incremented by 1 when **loop** is encountered. The loop is terminated by whether the index is greater than or equal to the limit. If the loop is not terminated execution returns to just after **do**. See also **+loop**.

m* "m-times" PRE
(n1 n2 -- d)
Multiplies two single-length numbers, n1 by n2, giving the double-length product. Used if an overflow would result with *.

minus "minus" P
(n1 -- n2)
Reverses the sign of n1 to produce n2. n2 is the twos complement of n1 (i.e. zero minus n1)

move "move" PRE
(addr1 addr2 n --)
Copies n single-length (16 bit) words beginning at address addr1 to addr2. The move proceeds from low memory toward high memory. If n is zero or negated, nothing is moved. See also **cmove** .

m/ "m-divide" PRE
(d n1 -- n2)
Divides d by n1 leaving the single-length quotient n2 (rounded toward zero). An error is generated if divisor is zero.

m/mod "m-divide-mod" PRE
(d n1 -- n2 n3)
Divides d by n1 leaving the single-length remainder n2 and single-length quotient n3 (rounded toward zero). An error is generated if divisor is zero.

max "max" PRE
(n1 n2 -- n3)
Returns n3 as the greater of n1 and n2. See also **min**.

min "min" PRE
(n1 n2 -- n3)
Returns n3 as the lesser of n1 and n2. See also **max**.

mod "mod" PRE
(n1 n2 -- n3)
Divides n1 by n2 and returns the remainder n3. The quotient is rounded toward zero. An error occurs if the divisor is zero.

nfa "n-f-a" PRE
(pfa -- nfa)
Given the parameter-field address of a word, returns the name-field address of that word.

not "not" P
(n1 -- n2)
Performs a bit-by-bit negation of n1 to give n2. That is each bit in n1 is reversed in n2. Thus in binary :-

100 not

will return 1111111111111011. See also **or** , **and** , **xor** .

number "number" P
(-- n1 n2) or (-- d n2)
Converts the counted string stored in the next available dictionary location (the address returned by **here**) into a numeric representation according to the current value of **BASE**. If the string contained a '.' then the a double-length representation is returned together with a the value of 2, otherwise a single-length value if returned together with a value of 1. If the string could not be converted then a single-length 0 and False (0)are returned.

or "or" P
(n1 n2 - n3)
Performs a bit-by-bit inclusive-or of n1 with n2 to give n3. That is each bit in n1 is compared with the equivalent bit in n2 and if either or both bits are set(1) the corresponding bit in n3 will be set, otherwise the corresponding bit will be 0. Thus in binary :-

110 100 or

will return binary 110. See also **and** , **xor** .

over "over" P
(n1 n2 -- n1 n2 n1)
Copies the second number on the stack to the top of the stack.

pad "pad" PRE
(-- addr)
Returning the lowest address of a "scratchpad" area that may be used to store data temporarily. The address of **pad** is changed and the data stored there are lost if the address of the next available dictionary location (returned by **here**) is changed, that is, if anything is added to the dictionary.

pfa "p-f-a" PRE
(nfa -- pfa)
Given the name-field address of a word, returns the parameter-field address of that word.

repeat "repeat" PRE , C
(--)
Terminates a **begin** ... **while** ... **repeat** loop. Used in the form :-

: < name > ... begin ... flag while ... repeat ... ;

compiles an unconditional branch to **begin** in the definition of <name>. When <name> executes, the words between **begin** and **end** execute repeatedly while the flag on the stack remains nonzero. If the flag is False (0), execution branches to the word following **repeat**. See also **begin** , **while** , **until**.

rot "rote" P
(n1 n2 n3 -- n2 n3 n1)
Rotates the third number on the stack to the top of the stack.

s->d "single-to-double" PRE
(n -- d)
Converts a single-length number to a double-length number, retaining the correct sign.

scode "s-code" PRE
(n1 --)
Copies the single-length value on the top of the return stack (the code-field address of the word to which execution will return) into the code-field address of the most recent definition in the **CONTEXT** vocabulary. See also **;code** , **ca!**.

seal "seal" PRE
(--)
Seals of the **CONTEXT** vocabulary and prevents all previous definitions from being seen from within the vocabulary. See also **CONTEXT** , **-find**.

sign "sign" PRE
(n --)
Appends an ASCII "-" (minus sign) to the start of a pictured (formatted) numeric output string if n is negative. See **#** for an example.

smudge "smudge" PRE
(--)
Toggles a bit (the smudge bit) in the header of the most recently defined word so as to determine whether or not the word can be found during a dictionary search. If the word can be found **smudge** marks it not to be found. A second execution of **smudge** will make if "findable" again. While a word is being defined the smudge bit is set so that the word cannot be found, **;** executing **smudge** to make it "findable". This is but one of several ways to keep bad definitions from being executed.

space "space" PRE
(--)
Sends a space (blank or ASCII 32) to the console.

spaces "spaces" PRE
(n --)
Sends n spaces (blanks) to the console. Nothing is displayed if n is zero or negative.

stdin? "stdin-question" P
(-- f)
Returns single-length value of True (-1) if the standard input stream is being used otherwise a single-length value of False (0) is returned.

swap "swap" P
(n1 n2 -- n2 n1)
Swaps the top two numbers on the stack.

then "then" PRE , I , C
(--)
Marks the end of a conditional branch. Used in the form :-

: < name > ... flag if ... then ... ;

or

: < name > ... flag if ... else ... else ... then ... ;

marks where execution will continue relative to the corresponding **if** or **else**. When the **if** construct is finished, execution jumps to the word just after **then**.

toggle "toggle" PRE
(addr n1 --)
Performs the bit-by-bit toggle of the least significant byte at addr with the bits specified by the single-length mask n2 and places the least significant byte of the result back into addr. See also **(toggle)**.

type "type" PRE
(addr n --)
Displays a string of n characters from memory starting with the character at addr. Nothing is displayed if n is zero or negative. See also **count**.

u* "u-times" P
(u1 u2 -- ud)
Multiplies u1 by u2, returning the double-length product ud. All values are treated as unsigned.

u. "u-dot" PRE
(u --)
Displays the unsigned value of u at the current display position, followed by a space. See also **.** (dot).

u/ "u-divide" P
(ud1 u2 -- u3)
Divides the double-length number ud1 by u2 returning the single-length remainder u2 and the single-length quotient u3. Floored division is used. All values are treated as unsigned. An error is generated it the divisor is zero.

unlink "unlink" PRE
(--)s
Unlinks the current **CONTEXT** vocabulary and reverts to the previous **CONTEXT** vocabulary. See also **forget.**

until "until" PRE , C , I
(f --)
Marks the end of an indefinite loop. Used in the form :-

: < name > ... begin ... flag until ... ;

compiles a conditional branch back to the corresponding **begin**. When <name> is executed **until** expects a flag on the stack and execution loops back to the corresponding **begin** until the flag is nonzero.

If the flag is nonzero, execution continues with the word following **until**. See also **begin , while , repeat.**

variable "variable" PRE
(--)
A defining word that creates a single-length variable. Used in the form :-

variable < name >

creates a dictionary entry for <name> and allocates storage space for a single-length number. When <name> is executed the address of the parameter-field of <name> is placed on the stack, suitable for use by @ (fetch) and ! (store) .

vocabulary "vocabulary" PRE
(--)
A defining word that creates a vocabulary. Used in the form :-

vocabulary < name >

creates a dictionary entry for <name> that specifies a new ordered list of word definitions. After <name> is executed the vocabulary <name> will be first to be searched (the **CONTEXT** vocabulary). <name> can be made the compilation vocabulary or the vocabulary to receive new definitions (the **CURRENT** vocabulary) with :-

< name > definitions

which will cause new definitions to be appended to < names >'s path.

while "while" PRE , I , C
(f --)
Decides the continuation or termination of a **begin** ... **while** ... **repeat** loop. Used in the form :-

: < name > ... begin ... flag while ... repeat ... ;

compiles a conditional branch operator into the definition of <name>. When <name> is executed **while** expects a flag on the stack, while the flag is nonzero the words between **while** and the corresponding **repeat** are executed, and **repeat** returns execution to immediately after the corresponding **begin**. If the flag is zero, execution jumps to the word following **repeat** .

word "word" P
(n --)
Generates a counted string by nondestructively accepting characters from the input stream until a delimiting character whose ASCII code is n is encountered or until the input stream is exhausted. Does not cause execution to pause. Leading delimiters are ignored. The characters are stored as a counted string at the next available dictionary location (the value returned by **here**).

words "words" PRE
(--)
Displays a list of all words in the **CONTEXT** vocabulary.

xor "x-or" P
(n1 n2 -- n3)
Performs a bit-by-bit exclusive-or of n1 with n2 to give n3. That is, each bit of n1 is compared with the equivalent bit in n2 and if either one or the other bit is set (but not both) the corresponding bit in n3 will be set, otherwise the corresponding bit will be 0. Thus in binary :-

110 100 xor

will return 010. See also **and** , **or**.

Appendix B

TILE Memory Allocation