Reference manual for Z79FORTH

version 1.2 - vendredi 4 octobre 2024

Author

- François LAAGEL
- Marc PETREMANN

Collaborator

- ….

Contents

All words

I, Francois Laagel, hereby testify that, to the best of my knowledge, Z79Forth/A is an ANS Forth System matching the minimum requirements as stated by ANSI-X3.215-1994.

The standard specification is available at http://lars.nocrew.org/dpans/

Implementation Status Report :

- - Not supported
- E EEPROM resident
- C CompactFlash resident

Core word set

E	<	*less-than*	CORE
E	<#	*less-number-sign*	CORE
E	=	*equals*	CORE
E	>	*greater-than*	CORE
E	-	*minus*	CORE
E	,	*comma*	CORE
E	;	*semicolon*	CORE
E	:	*colon*	CORE
E	!	*store*	CORE
E	/	*slash*	CORE
E	.	*dot*	CORE
E	."	*dot-quote*	CORE
E	'	*tick*	CORE
E	(	*paren*	CORE
E	[	*left-bracket*	CORE
E	[']	*bracket-tick*	CORE
E	]	*right-bracket*	CORE
E	@	*fetch*	CORE
E	*	*star*	CORE
E	*/	*star-slash*	CORE
E	#	*number-sign*	CORE
E	#>	*number-sign-greater*	CORE
E	+	*plus*	CORE
E	+!	*plus-store*	CORE
E	0<	*zero-less*	CORE
E	0=	*zero-equals*	CORE
E	1-	*one-minus*	CORE
E	1+	*one-plus*	CORE
E	2!	*two-store*	CORE
E	2/	*two-slash*	CORE
E	2@	*two-fetch*	CORE

```
E     2*           *two-star*               CORE
E     2DROP        *two-drop*               CORE
E     2DUP         *two-dupe*               CORE
E     2OVER        *two-over*               CORE
E     2SWAP        *two-swap*               CORE
E     ABORT        -                        CORE
C     ABORT"       *abort-quote*            CORE
E     ABS          *abs*                    CORE
E     ACCEPT       -                        CORE
E     ALIGN        -                        CORE
E     ALIGNED      -                        CORE
E     ALLOT        -                        CORE
E     AND          -                        CORE
E     BASE         -                        CORE
E     BEGIN        -                        CORE
E     BL           *b-l*                    CORE
C     >BODY        *to-body*                CORE
E     C,           *c-comma*                CORE
E     C!           *c-store*                CORE
E     C@           *c-fetch*                CORE
E     CELL+        *cell-plus*              CORE
E     CELLS        -                        CORE
E     CHAR         *char*                   CORE
E     [CHAR]       *bracket-char*           CORE
E     CHAR+        *char-plus*              CORE
E     CHARS        *chars*                  CORE
E     CONSTANT     -                        CORE
E     COUNT        -                        CORE
E     CR           *c-r*                    CORE
E     CREATE       -                        CORE
E     DECIMAL      -                        CORE
E     DEPTH        -                        CORE
E     DO           -                        CORE
E     DOES>        *does*                   CORE
E     DROP         -                        CORE
E     DUP          *dupe*                   CORE
E     ?DUP         *question-dupe*          CORE
E     ELSE         -                        CORE
E     EMIT         -                        CORE
C     ENVIRONMENT? *environment-query*      CORE
E     EVALUATE     -                        CORE
E     EXECUTE      -                        CORE
E     EXIT         -                        CORE
E     FILL         -                        CORE
E     FIND         -                        CORE
E     FM/MOD       *f-m-slash-mod*          CORE
E     HERE         -                        CORE
E     HOLD         -                        CORE
```

E	I	-	CORE
E	IF	-	CORE
E	IMMEDIATE	-	CORE
E	>IN	*to-in*	CORE
E	INVERT	-	CORE
E	J	-	CORE
E	KEY	-	CORE
E	LEAVE	-	CORE
E	LITERAL	-	CORE
E	LOOP	-	CORE
E	+LOOP	*plus-loop*	CORE
C	LSHIFT	*l-shift*	CORE
E	M*	*m-star*	CORE
E	MAX	-	CORE
E	MIN	-	CORE
E	MOD	-	CORE
E	/MOD	*slash-mod*	CORE
E	*/MOD	*star-slash-mod*	CORE
E	MOVE	-	CORE
E	NEGATE	-	CORE
E	>NUMBER	*to-number*	CORE
E	OR	-	CORE
E	OVER	-	CORE
E	POSTPONE	-	CORE
E	QUIT	-	CORE
E	>R	*to-r*	CORE
E	R>	*r-from*	CORE
E	R@	*r-fetch*	CORE
E	RECURSE	-	CORE
E	REPEAT	-	CORE
E	ROT	*rote*	CORE
C	RSHIFT	*r-shift*	CORE
E	#S	*number-sign-s*	CORE
E	S"	*s-quote*	CORE
E	S>D	*s-to-d*	CORE
E	SIGN	-	CORE
E	SM/REM	*s-m-slash-rem*	CORE
E	SOURCE	-	CORE
E	SPACE	-	CORE
E	SPACES	-	CORE
E	STATE	-	CORE
E	SWAP	-	CORE
E	THEN	-	CORE
E	TYPE	-	CORE
E	U<	*u-less-than*	CORE
E	U.	*u-dot*	CORE
E	UM*	*u-m-star*	CORE
E	UM/MOD	*u-m-slash-mod*	CORE

E	UNLOOP	-	CORE
E	UNTIL	-	CORE
E	VARIABLE	-	CORE
E	WHILE	-	CORE
E	WORD	-	CORE
E	XOR	*x-or*	CORE

Core Extension word set

E	<>	*not-equals*	CORE_EXT
C	.(	*dot-paren*	CORE_EXT
E	\	*backslash*	CORE_EXT
E	0<>	*zero-not-equals*	CORE_EXT
E	0>	*zero-greater*	CORE_EXT
C	2>R	*two-to-r*	CORE_EXT
C	2R>	*two-r-from*	CORE_EXT
C	2R@	*two-r-fetch*	CORE_EXT
E	AGAIN	-	CORE_EXT
-	C"	*c-quote*	CORE_EXT
-	CASE	-	CORE_EXT
-	[COMPILE]	*bracket-compile*	CORE_EXT
E	COMPILE,	*compile-comma*	CORE_EXT
-	CONVERT	-	CORE_EXT
E	?DO	*question-do*	CORE_EXT
-	ENDCASE	*end-case*	CORE_EXT
-	ENDOF	*end-of*	CORE_EXT
C	ERASE	-	CORE_EXT
-	EXPECT	-	CORE_EXT
E	FALSE	-	CORE_EXT
E	HEX	-	CORE_EXT
E	MARKER	-	CORE_EXT
E	NIP	-	CORE_EXT
E	:NONAME	*colon-no-name*	CORE_EXT
-	OF	-	CORE_EXT
E	PAD	-	CORE_EXT
-	PARSE	-	CORE_EXT
E	PICK	-	CORE_EXT
-	QUERY	-	CORE_EXT
E	.R	*dot-r*	CORE_EXT
E	REFILL	-	CORE_EXT
-	RESTORE-INPUT	-	CORE_EXT
E	ROLL	-	CORE_EXT
-	SAVE-INPUT	-	CORE_EXT
-	SOURCE-ID	*source-i-d*	CORE_EXT
-	SPAN	-	CORE_EXT
-	TIB	*t-i-b*	CORE_EXT
-	#TIB	*number-t-i-b*	CORE_EXT
C	TO	-	CORE_EXT
E	TRUE	-	CORE_EXT

E	TUCK	-	CORE_EXT
E	U>	*u-greater-than*	CORE_EXT
C	UNUSED	-	CORE_EXT
E	U.R	*u-dot-r*	CORE_EXT
C	VALUE	-	CORE_EXT
C	WITHIN	-	CORE_EXT

Block word set

E	BLK	*b-l-k*	BLOCK
E	BLOCK	-	BLOCK
E	BUFFER	-	BLOCK
E	EVALUATE	-	BLOCK
E	FLUSH	-	BLOCK
E	LOAD	-	BLOCK
E	UPDATE	-	BLOCK
E	SAVE-BUFFERS	-	BLOCK

Block Extension word set

E	EMPTY-BUFFERS	-	BLOCK_EXT
E	LIST	-	BLOCK_EXT
E	REFILL	-	BLOCK_EXT
E	SCR	*s-c-r*	BLOCK_EXT
E	THRU	-	BLOCK_EXT

Double-Number word set

-	2CONSTANT	*two-constant*	DOUBLE
-	2LITERAL	*two-literal*	DOUBLE
-	2VARIABLE	*two-variable*	DOUBLE
E	D<	*d-less-than*	DOUBLE
-	D=	*d-equals*	DOUBLE
E	D-	*d-minus*	DOUBLE
-	D.	*d-dot*	DOUBLE
E	D+	*d-plus*	DOUBLE
-	D0<	*d-zero-less*	DOUBLE
E	D0=	*d-zero-equals*	DOUBLE
-	D2/	*d-two-slash*	DOUBLE
-	D2*	*d-two-star*	DOUBLE
-	DABS	*d-abs*	DOUBLE
-	DMAX	*d-max*	DOUBLE
-	DMIN	*d-min*	DOUBLE
E	DNEGATE	*d-negate*	DOUBLE
-	D.R	*d-dot-r*	DOUBLE
-	D>S	*d-to-s*	DOUBLE
-	M*/	*m-star-slash*	DOUBLE
-	M+	*m-plus*	DOUBLE

Facility word set

-	AT-XY	*at-x-y*	FACILITY
E	KEY?	*key-question*	FACILITY
E	PAGE	-	FACILITY

Facility Extension word set

-	EKEY	*e-key*	FACILITY_EXT
-	EKEY?	*e-key-question*	FACILITY_EXT
-	EKEY>CHAR	*e-key-to-char*	FACILITY_EXT
-	EMIT?	*emit-question*	FACILITY_EXT
E	MS	-	FACILITY_EXT
-	TIME&DATE	*time-and-date*	FACILITY_EXT

Programming-Tools word set

E	.S	*dot-s*	TOOLS
E	?	*question*	TOOLS
C	DUMP	-	TOOLS
C	SEE	-	TOOLS
E	WORDS	-	TOOLS

Programming-Tools Extension word set

-	;CODE	*semicolon-code*	TOOLS_EXT
-	[ELSE]	*bracket-else*	TOOLS_EXT
-	[IF]	*bracket-if*	TOOLS_EXT
-	[THEN]	*bracket-then*	TOOLS_EXT
E	AHEAD	-	TOOLS_EXT
-	ASSEMBLER	-	TOOLS_EXT
E	BYE	-	TOOLS_EXT
-	CODE	-	TOOLS_EXT
-	CS-PICK	*c-s-pick*	TOOLS_EXT
-	CS-ROLL	*c-s-roll*	TOOLS_EXT
-	EDITOR	-	TOOLS_EXT
-	FORGET	-	TOOLS_EXT
E	STATE	-	TOOLS_EXT

Note: BYE will reset the system. Dirty (UPDATEd) buffers, if any, will not be flushed to mass storage.

String word set

C	-TRAILING	*dash-trailing*	STRING
C	/STRING	*slash-string*	STRING
E	BLANK	-	STRING

```
E     CMOVE        *c-move*                 STRING
E     CMOVE>       *c-move-up*              STRING
C     COMPARE      -                        STRING
-     SLITERAL     -                        STRING
```

Forth2012 words

See https://forth-standard.org for more information.

```
ACTION-OF DEFER DEFER! DEFER@ IS
```

79-STANDARD words

See *SW/reference/FORTH-79.TXT* for more information.

```
I' INDEX INTERPRET K LAST LINE NOT SHIFT
```

forth

! n addr --

Store n to address.

```
variable TEMPERATURE
32 TEMPERATURE !
```

# ud1 -- ud2

COMPILATION ONLY

Convert 1 digit to formatted numeric string.

This word must be used between <# and #>.

The conversion to digit is done relative to the current numeric base.

#> ud1 -- c-addr u

COMPILE_ONLY

Leave address and count of formatted numeric string.

#s ud1 -- ud2

COMPILE_ONLY

Convert remaining digits to formatted numeric output

' -- cfa

Parse word and find it in dictionary. If found, leave the cfa of this word.

```
' words     \ leave cfa of words on stack
execute     \ execute words
```

(-TEXT) c_addr1 u c_addr2 -- n

Compare the string specified by c-addr1 u1 to the string specified by c-addr2 u2. The strings are compared, beginning at the given addresses, character by character, up to the length of the shorter string or until a difference is found.

- If the two strings are identical, n is zero.

- If the two strings are identical up to the length of the shorter string, n is minus-one (-1) if u1 is less than u2 and one (1) otherwise.

- If the two strings are not identical up to the length of the shorter string, n is minus-one (-1) if the first non-matching character in the string specified by c-addr1 u1 has a lesser numeric value than the corresponding character in the string specified by c-addr2 u2 and one (1) otherwise.

(SGN) n -- -1|0|1

Stack a flag indicating whether the number n is positive, zero, or negative.

```
4 (SGN) .   \ display: 1
-3 (SGN) .   \ display:-1
 0 (SGN) .   \ display:0
```

* n1 n2 -- n1*n2

Integer multiplication of two numbers.

```
6 3  *   \ push 18 operation 6*3
7 3  *   \ push 21 operation 7*3
-7 3 *   \ push -21
7 -3 *   \ push -21
-7 -3 *  \ push 21
```

*/ n1 n2 n3 -- n4=(n1*n2)/n3

Multiply n1 by n2 producing the intermediate double-cell result d. Divide d by n3 giving the single-cell quotient n4.

```
5000 1000 4000 */ .    \ display    1250
```

*/MOD n1 n2 n3 -- rem quot

Multiply n1 with n2 and divide with n3 via 32-bit intermediate result.

```
50000 10 4001 */MOD .    \ display    124 3876
```

+ n1 n2 -- n3=n1+n2

Addition of two signed single precision integers.

```
6 3  + .    \ display 9
-6 3 + .    \ display -3
```

+! n addr --

Increments the content of a variable by the value n

```
variable counter
15 counter +!
```

```
counter @ .  \ display 15
```

+LOOP   n --

Increment index loop with value n.

Mark the end of a loop `n1 0 do ... n2 +loop`.

```
: loopTest
   100 0 do
       i .
   5 +loop
  ;
loopTest  \ display 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
95
```

,   x --

Append x to the current data section.

```
create datas
    126 ,   352 ,   447 ,
datas 0 cells + @ .     \ display 126
datas 2 cells + @ .     \ display 447
```

-   n1 n2 -- n3=n1-n2

Subtract two integers.

```
6 3  -   \ push 3  operation 6-3
-6 3 -   \ push -9 operation -6-3
```

-ROT   n1 n2 n3 -- n3 n1 n2

Same as `ROT ROT`.

-TRAILING   addr n1 -- addr n2

Adjust the character count n1 of a text string beginning at addr to exclude trailing blanks, i.e., the characters at addr+n2 to `-TRAILING` addr+n1-1 are blanks. An error condition exists if n1 is negative.

```
: myTxt
    s" this is my example   "
  ;
mytxt       type \ display: this is my example    ok
mytxt
  -trailing type \ display: this is my example ok
```

. n --

Remove the value at the top of the stack and display it as a signed single precision integer.

```
1 .                         \ display 1
1 2 .                       \ display 2  leave 1 on stack
1 2 + .                     \ display 3  addition 1 and 2, leave nothing on the
stack
6 3  * .                    \ display 18
7 3  * 6 3 * + .            \ display 39 operation (7*3)+(6*3)
```

." -- <string>

The word ." can only be used in a compiled definition.

At runtime, it displays the text between this word and the delimiting **"** character end of string.

```
: TITLE
    ."      GENERAL MENU" cr
    ."      ============" ;
: line1
    ." 1.. Enter datas" ;
: line2
    ." 2.. Display datas" ;
: last-line
    ." F.. end program" ;
: MENU ( -- )
    title cr cr cr
    line1 cr cr
    line2 cr cr
    last-line ;
```

.' addr --

A SwiftForth word which displays the name of the nearest definition before addr, and the offset of addr from the beginning of that definition. "dot-tick"

.("ccc" --

Parse and display ccc delimited by) (right parenthesis). .(is an immediate word.

.R n1 n2 --

Display n1 right aligned in a field n2 characters wide.

If the number of characters required to display n1 is greater than n2, all digits are displayed with no leading spaces in a field as wide as necessary.

```
3 6 .r      \ display        3
254 6 .r      \ display      254
```

.S --

Displays the content of the data stack, with no action on the content of this stack.

.TAB --

Display TAB character (tabulation).

/ n1 n2 -- n3

Divide n1 by n2, giving the single-cell quotient n3.

```
6 3  / .  \ display 2 opération 6/3
7 3  / .  \ display 2 opération 7/3
8 3  / .  \ display 2 opération 8/3
9 3  / .  \ display 3 opération 9/3
```

/MOD n n -- rem quot

16/16 -> 16-bit signed division.

/STRING c-addr1 u1 n -- c-addr2 u2

Adjust the character string at c-addr1 by n characters. The resulting character string, specified by c-addr2 u2, begins at c-addr1 plus n characters and is u1 minus n characters long.

0< n -- fl

Leave true flag if n is less than zero.

0<> n -- fl

Leave -1 if n <> 0

```
1 0<>      \ push TRUE  on stack
0 0<>      \ push FALSE on stack
```

0= x -- fl

flag is true if and only if x is equal to zero.

```
5 0=      \ push  FALSE on stack
0 0=      \ push  TRUE  on stack
```

0> x1 -- fl

Tests if x1 is greater than zero.

```
3 0> .    \ display: -1
-5 0> .   \ display: 0
```

1+ n -- n+1

Increments the value at the top of the stack.

```
17 1+ .    \ display 18
```

1- n -- n-1

Decrements the value at the top of the stack.

```
17 1- .    \ display 16
```

2! d addr --

Store double precision value in memory address addr

2* n -- n*2

Multiply n by two.

```
7 2* .     \ display 14
```

2+ n -- n+2

Increments 2 the value at the top of the stack.

```
35 2+ .      \ display 37
```

2- TODELETE n -- n-2

Decrements 2 the value at the top of the stack.

```
5 2- .    \ display 3
```

2/ n -- n/2

Divide n by two.

```
6 2/ .     \ display 3
7 2/ .     \ display 3
```

2>R   S: d -- R: d

Transfers d to the return stack.

This operation must always be balanced with `2r>`

2@   addr -- d

Fetch two cells

2CONSTANT   comp: d -- <name> | exec: -- d

Creation word. Defines a double precision constant.

```
3.141529 2constant PI
```

2DROP   d --

Removes the double-precision integer that was there from the top of the data stack.

```
2. 5. 8.  2DROP   \ leave 2 and 5 in double precision on stack
```

2DUP   d -- d d

Copies the double-precision number at the top of the data stack. Equivalent to n1 n2 --- n1 n2 n1 n2.

```
5.  2DUP   \ leave 5. et 5. on stack
           \ 5. is a double precision number
```

2NIP   d1 d2 -- d2

Drop d1 and leave d2 on stack.

```
1. 2. 2nip d.   \ display:  2
```

2OVER   d1 d2 -- d1 d2 d1

Duplicate d1 on stack.

```
1. 2. 2over d.   \ display:  1
```

2R>   R: d -- S: d

Transfers d from the return stack.

This operation must always be balanced with `2>r`

2R@ R: d -- S: d

Retrieves a double-precision integer that was previously stored on the return stack by `2>r`

The double-precision integer saved on the return stack by `2>r` must be retrieved by `2r>` before leaving the word.

```
: test ( d -- d d )
    2>r  2r@  2r>
  ;
```

2SWAP d1 d2 -- d2 d1

Exchange the top two cell pairs.

2VARIABLE comp: -- <name> | exec: -- addr

Creation word. Defines a double precision variable.

```
2variable resistance
```

: comp: --

The word `:` is the most used creation word in FORTH.

Subsequent execution of `NOM` performs the execution sequence words compiled in his "colon" definition.

After `:` `NOM`, the interpreter enters compile mode. All non-immediate words are compiled in the definition, the numbers are compiled in literal form. Only immediate words or placed in square brackets (words `[` and `]`) are executed during compilation to help control it.

A "colon" definition remains invalid, ie not inscribed in the current vocabulary, as long as the interpreter did not execute `;` (semi-colon).

```
: NAME  nomex1 nomex2 ... nomexn ;
NAME  \ execute NAME
```

:NONAME -- cfa

Define headerless forth code. cfa-addr is the code execution of a definition.

```
:noname s" Saterday" ;
:noname s" Friday" ;
:noname s" Thursday" ;
:noname s" Wednesday" ;
:noname s" Tuesday" ;
:noname s" Monday" ;
:noname s" Sunday" ;
```

```
create (ENday) ( --- addr)
  , , , , , , ,

:noname s" Samedi" ;
:noname s" Vendredi" ;
:noname s" Jeudi" ;
:noname s" Mercredi" ;
:noname s" Mardi" ;
:noname s" Lundi" ;
:noname s" Dimanche" ;

create (FRday) ( --- addr)
  , , , , , , ,

defer (day)

: ENdays
    ['] (ENday) is (day) ;

: FRdays
    ['] (FRday) is (day) ;

3 value dayLength
: day
    (day)
    swap cells +
    @ execute
    dayLength ?dup if
        min
    then
    type
;
ENdays
0 day   \ display Sun
1 day   \ display Mon
2 day   \ display Tue
FRdays  ok
0 day   \ display Dim
1 day   \ display Lun
2 day   \ display Mar
```

; --

Immediate execution word usually ending the compilation of a "colon" definition.

```
: NAME
    nomex1 nomex2 ... nomexn ;
```

<   x1 x2 -- fl

Test if x1 is less than x2.

```
3 5 < .      \ display -1
5 -1 < .     \ display 0
-5 -1 < .    \ display -1
```

<#   --

Compile Only

Begin numeric conversion

```
\ display byte in binary format
: x. ( c -- )
    0 base @ >r
    2 base !
    s>d
    <# # # # # # # # # #> type
    r> base !
  ;
```

<=   n1 n2 -- fl

Leave fl true if n1 <= n2

```
4 10 <=    \ leave -1 on stack
4 4  <=    \ leave -1 on stack
4 3  <=    \ leave  0 on stack
```

<>   x1 x2 -- fl

flag is true if and only if x1 is different x2.

```
5 5 <>      \ push  FALSE on stack
5 4 <>      \ push  TRUE  on stack
```

=   x1 x2 -- fl

flag is true if and only if x1 is equal x2.

```
5 5 =      \ push  TRUE  on stack
5 4 =      \ push  FALSE on stack
```

>   x1 x2 -- fl

Test if x1 is greater than x2.

```
3 5 > .      \ display  0
```

```
5 -1 > .      \ display -1
-1 -5 > .     \ display -1
```

>BODY xt -- a-addr

a-addr is the data-field address corresponding to xt.

>IN -- a-addr

a-addr is the address of a cell containing the offset in characters from the start of the input buffer to the start of the parse area.

>NUMBER ud1 c-addr1 u1 -- ud2 c-addr2 u2

ud2 is the unsigned result of converting the characters within the string specified by c-addr1 u1 into digits, using the number in BASE, and adding each into ud1 after multiplying ud1 by the number in BASE.

Conversion continues left-to-right until a character that is not convertible, including any "+" or "-", is encountered or the string is entirely converted. c-addr2 is the location of the first unconverted character or the first character past the end of the string if the string was entirely converted. u2 is the number of unconverted characters in the string.

>R x -- | R: -- x

COMPILE_ONLY.

Push x from the parameter stack to the return stack.

```
\ display n in binary format
: b. ( n -- )
    base @ >r
    2 base !   .
    r> base !
  ;
```

? a-addr --

Display the value stored at a-addr.

?DO n1 n2 --

Executes a `do loop` or `do +loop` loop if n1 is strictly greater than n2.

```
DECIMAL
: qd ?DO I LOOP ;
   789   789 qd \
 -9876 -9876 qd \
     5     0 qd \  display: 0 1 2 3 4
```

?DUP   n -- n | n n

Duplicate n if n is not nul.

@   addr -- n

Retrieves the integer value n stored at address addr.

```
variable SCORE
36 SCORE !
1 SCORE +!
SCORE @ .    \ display 37
```

ABORT   --

Reset stack pointer and execute `quit`.

ABORT"   comp: -- <error message>

Displays an error message and aborts any FORTH execution in progress.

```
: abort-test
    if
        abort" stop program"
    then
    ." continue program"
  ;

0 abort-test   \ display: continue program
1 abort-test   \ display: stop program ERROR
```

ABS   n -- n'

Return the absolute value of n.

```
-7 abs .    \ display 7
```

ACCEPT   c-addr +n -- +n'

Get line from terminal.

AGAIN   addr --

End structure `begin ... again`.

ALIGN   --

Align the current data section dictionary pointer to cell boundary.

ALIGNED addr -- a-addr

Align addr to a cell boundary.

ALLOT n --

Adjust the current data section dictionary pointer.

```
create myScores
    4 cells allot   \ allot 8 bytes in memory
```

AND n1 n2 -- n3

Execute logic AND.

```
false  false and .  \ display 0
false  true  and .  \ display 0
true   false and .  \ display 0
true   true  and .  \ display -1
```

BASE -- addr

Single precision variable determining the current numerical base.

The **BASE** variable contains the value 10 (decimal) when FORTH starts.

```
DECIMAL      \ select decimal base
2 BASE !     \ select binary base
```

BEGIN -- addr

COMPILE_ONLY

begin ... again

begin ... until

begin ... while ... repeat

BL -- 32

Constante. Empile 32 qui est le code ascii du caractère 'espace'.

BLANK addr len --

If len is greater than zero, store byte $20 (space) in each of len consecutive characters of memory beginning at addr.

BLK -- a-addr

a-addr is the address of a cell containing zero or the number of the mass-storage block being interpreted.

BLOCK u -- a-addr

a-addr is the address of the first character of the block buffer assigned to mass-storage block u.

BUFFER u -- a-addr

a-addr is the address of the first character of the block buffer assigned to block u. The contents of the block are unspecified.

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, an error condition exists. The address left is the first byte within the buffer for data storage.

n is an unsigned number.

BYE --

Reset FORTH

```
bye
\ display:
Z79Forth/AI 6309 ANS Forth System
20240628 (C) Francois Laagel 2019

RAM OK: 32 KB
SanDisk SDCFJ-128
Block #1 loaded
```

C! c addr --

Stores an 8-bit c value at address addr.

C, c --

Append c to the current data section.

```
create myPresets
    $04 c,  $2f c,  $40 c,
```

C@ addr -- c

Retrieves the 8-bit c value stored at address addr.

```
35 constant PINB   \ adresse registre données PIN de PORT B sur Arduino
PINB c@            \ empile contenu registre pointé par PINB
```

CELL+ a-addr1 -- a-addr2

Add the size in address units of a cell to a-addr1, giving a-addr2.

CELLS n1 -- n2

n2 is the size in address units of n1 cells.

```
CREATE NUMBERS
    100 CELLS ALLOT
```

CHAR -- <string>

Word used in interpretation only.

Leave the first character of the string following this word.

```
char v .           \ display: 118 (ascii code for "v")
char house .       \ display: 104 - code for "h"
```

CHAR+ c-addr1 -- c-addr2

Add the size in address units of a character to c-addr1, giving c-addr2.

CHARS n1 -- n2

n2 is the size in address units of n1 characters.

CMOVE addr1 addr2 u --

Move u chars from addr1 to addr2.

COMPARE c-addr1 u1 c-addr2 u2 -- n

Compare the string specified by c-addr1 u1 to the string specified by c-addr2 u2.

The strings are compared, beginning at the given addresses, character by character, up to the length of the shorter string or until a difference is found. If the two strings are identical, n is zero. If the two strings are identical up to the length of the shorter string, n is minus-one (-1) if u1 is less than u2 and one (1) otherwise. If the two strings are not identical up to the length of the shorter string, n is minus-one (-1) if the first non-matching character in the string specified by c-addr1 u1 has a lesser numeric value than the corresponding character in the string specified by c-addr2 u2 and one (1) otherwise.

CONSTANT comp: n -- <name> | exec: -- n

Creation word. Defines a simple precision constant.

```
\ definition of constant
130 constant speed-max
speed-max .      \ display 130
```

COUNT c-addr1 -- c-addr2 u

Return the character string specification for the counted string stored at c-addr1. c-addr2 is the address of the first character after c-addr1. u is the contents of the character at c-addr1, which is the length in characters of the string at c-addr2.

CR --

Show a new line return.

```
: .result ( ---)
    ." Port analys result" cr
    . "pool detectors" cr ;
```

CREATE comp: -- | exec: -- addr

Create a new word.

The word `CREATE` can be used alone.

```
CREATE DATAS  ( --- addr)
    25 c, 32 c, 44 c, 17 c,
```

D+ d1 d2 -- d3

Add double numbers.

```
35. 7. d+      \ leave 35. + 7. on the stack
```

D- d1 d2 -- d3

Subtract double numbers.

```
35. 7. d-   \ leave result of 35-7 in double precision
```

D. d --

Display d.

This word is not defined in Z79Forth dictionnary.

```
\ leave absolute value of d if d<
```

```
: DABS ( d -- d' )
    dup 0< if
        dnegate
    then
;

\ leave addr len on stack resulting of d conversion in string
: (D.)   ( d -- a l )
    tuck dabs <# #s rot sign #>
;

\ display d
: D.   ( d -- )
  (d.) type space
;
```

D0= d -- fl

True if d equals zero.

D< d1 d2 -- flag

flag is true if and only if d1 is less than d2.

DECIMAL --

Selects the decimal number base. It is the default digital base when FORTH starts.

```
255 hex . decimal   \ display FF
```

DEFER -- <vec-name>

Define a deferred execution vector.

`vec-name` execute the word whose execution token is stored in vec-name's data space.

```
defer xEmit
: vxEmit ( c ---)
    1+ emit ;
' vxEmit is xEmit
```

DEFER! xt2 xt1 --

Set the word xt1 to execute xt2.

DEFER@ xt1 -- xt2

xt2 is the execution token xt1 is set to execute.

DEPTH -- n

n is the number of single-cell values contained in the data stack before n was placed on the stack.

```
depth .     \ display 0
55 22 4     depth .     \ display 3
```

DNEGATE d -- -d

Negate double number.

```
4. dnegate d.    \ display -4
```

DO n1 n2 --

Set up loop control parameters with index n2 and limit n1.

```
: testLoop
    256 32 do
        I emit
    loop
  ;
```

DOES> comp: -- | exec: -- addr

The word `CREATE` can be used in a new word creation word...

Associated with `DOES>`, we can define words that say how a word is created then executed.

DROP n --

Removes the single-precision integer that was there from the top of the data stack.

```
2 5 8  drop   \ leave 2 and 5 on stack
```

DUMPLOAD --

Compile the contents of blocks 150 to 154.

These blocks contain the definitions for performing a memory dump.

DUP n -- n n

Duplicates the single-precision integer at the top of the data stack.

```
: SQUARE ( n --- nE2)
    DUP * ;
 5 SQUARE  .    \ display 25
```

```
10 SQUARE  .    \ display 100
```

ELSE --

Word of immediate execution and used in compilation only. Mark a alternative in a control structure of the type:

`(condition) IF ... ELSE ... THEN ...`

At runtime, if the condition on the stack before `IF` is false, there is a break in sequence with a jump following `ELSE`, then resumed in sequence after `THEN`

```
: TEST ( ---)
    cr ." Press a key " key
    dup 65 122 between
    if
        cr 3 spaces ." c'est une lettre "
    else
        dup 48 57 between
        if
            cr 3 spaces ." is a digit "
        else
            cr 3 spaces ." is a special character "
        then
    then
    drop ;
```

EMIT c --

Display character code c.

If x is a graphic character in the implementation-defined character set, display x. The effect of `emit` for all other values of x is implementation-defined.

When passed a character whose character-defining bits have a value between hex 20 and 7E inclusive, the corresponding standard character is displayed. Because different output devices can respond differently to control characters, programs that use control characters to perform specific functions have an environmental dependency. Each `emit` deals with only one character.

```
65 emit    \ display A
66 emit    \ display B
```

EMPTY-BUFFERS --

Unassign all block buffers. Do not transfer the contents of any UPDATEd block buffer to mass storage.

ERASE   addr u --

If u is greater than zero, clear all bits in each of u consecutive address units of memory beginning at addr.

EVALUATE   addr-c u --

Save the current input source specification. Make the string described by c-addr and u both the input source and input buffer, set `>IN` to zero, and interpret. When the parse area is empty, restore the prior input source specification.

```
: GE1 S" 123" ;
: GE2 S" 123 1+" ;
GE1 EVALUATE .   \ display: 123
GE2 EVALUATE .   \ display: 124
```

EXECUTE   cfa --

Execute word from his cfa address.

Take the execution address from the data stack and executes that token.

```
' words
execute      \ execute WORDS from his execution code address
```

EXIT   --

Aborts the execution of a word and gives back to the calling word.

Typical use: `: X ... test IF ... EXIT THEN ... ;`

FALSE   -- 0

Preset constant. Push 0 on stack.

```
false .     \ display: 0
```

FILL   addr len c --

If len is greater than zero, store c in each of len consecutive characters of memory beginning at addr.

FIND   c-addr -- c-addr 0 | xt 1 | xt -1

Find the definition named in the counted string at c-addr.

If the definition is not found, return c-addr and zero. If the definition is found, return its execution token xt. If the definition is immediate, also return one (1), otherwise also return minus-one (-1). For a given string, the values returned by FIND while compiling may differ from those returned while not compiling.

FLUSH --

A synonym for `SAVE-BUFFERS`.

FM/MOD d1 n1 -- n2 n3

Divide d1 by n1, giving the floored quotient n3 and the remainder n2. Input and output stack arguments are signed. An ambiguous condition exists if n1 is zero or if the quotient lies outside the range of a single-cell signed integer.

H. n --

Display n in hexadecimal format.

```
decimal
16 h.       \ display 10
35 h.       \ display 23
```

HERE -- addr

Leave the current data section dictionary pointer.

HEX --

Selects the hexadecimal digital base.

```
255 HEX .    \ display FF
DECIMAL      \ return to decimal base
```

HOLD c --

Inserts the ASCII code of an ASCII character into the character string initiated by `<#`.

I -- n

n is a copy of the current loop index.

```
: mySingleLoop ( -- )
    cr
    10 0 do
        i .
    loop
  ;
mySingleLoop
\ display 0 1 2 3 4 5 6 7 8 9
```

IF   fl --

The word IF is of immediate execution.

IF marks the beginning of a control structure of type IF..THEN or IF..ELSE..THEN.

At runtime, the definition part between IF and THEN or between IF and ELSE is executed if the boolean flag at the top of the data stack is true (f<>0).

Otherwise, if the boolean flag is false (f=0), the definition part located between ELSE and THEN will be executed. If there is no ELSE, execution continues after THEN.

```
: Nice? ( fl ---)
    if
        ." Nice weather "
    else
        ." Cloudy weather "
    then
  ;
1 Nice?    \ display: Nice weather
0 Nice?    \ display: Cloudy weather
```

IMMEDIATE   --

Make the most recent definition an immediate word.

INDEX   n1 n2 --

Print the first line of each screen over the range {n1..n2}.

This displays the first line of each screen of source text, which conventionally contains a title.

INTERPRET   addr len --

Interpret the buffer

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

INVERT   x1 -- x2

Complement to one of x1.

```
1 invert .   \ display -2 (%1111111111111110)
```

IS   --

Affecte le code d'exécution d'un mot à un mot d'exécution vectorisée.

```
defer xEmit
: vxEmit ( c ---)
    1+ emit ;
' vxEmit is xEmit
```

J   -- n | u

n | u is a copy of the next-outer loop index.

```
: twoLoops ( -- )
  cr
  5 0 do
    5 0 do
      j i . . cr
    loop
  loop ;
twoLoops  \ display:
0 0
1 0
2 0
3 0
4 0
0 1
1 1
2 1
3 1.....
```

J'   -- n

Returns the fourth item from the return stack.

K   -- n

Within a nested `DO`..`LOOP`, return the index of the second outer loop.

KEY   -- c

Waits for a key to be pressed. Pressing a key returns its ASCII code.

```
key .    \ display 97 if key "a" is active
key .    \ affiche 65 if key "A" is active
```

KEY?   -- fl

Returns *true* if a key is pressed.

```
: keyLoop
    begin
    key? until
```

```
  ;
```

LAST   -- addr

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

LIST   n --

Displays the contents of block n.

LITERAL   x --

Compiles the value x as a literal value.

```
: valueReg ( --- n)
   [ 36 2 * ] literal
 ;
```

LOOP   --

Add one to the loop index. If the loop index is then equal to the loop limit, discard the loop parameters and continue execution immediately following the loop. Otherwise continue execution at the beginning of the loop.

```
: myLoop ( -- )
    10 0 do
        i .
    loop
  ;
myLoop \ display: 0 1 2 3 4 5 6 7 8 9
```

LSHIFT   x1 n -- x2

Shift x1 left by n bits.

```
2 3 lshift .   \ display: 16
```

M*   n1 n2 -- d

d is the signed product of n1 times n2.

MARKER   comp: -- <name> | exec: --

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below.

The execution of restore all dictionary allocation and search order pointers to the state they had just prior to the definition of name. Remove the definition of name and all subsequent definitions. Restoration of any structures still existing that could refer to deleted definitions or deallocated data space is not necessarily provided. No other contextual information such as numeric base is affected.

```
marker --wx
: w1 ;
: w2 ;
--wx  \ delete words --wx w1 w2 in dictionnary
```

MAX   n1 n2 -- n1|n2

Leave max of n1 and n2.

MIN   n1 n2 -- n1|n2

Leave min of n1 and n2.

MOD   n1 n2 -- n3

Divide n1 by n2, giving the single-cell remainder n3.

```
21 7 mod . \ display 0
22 7 mod . \ display 1
23 7 mod . \ display 2
24 7 mod . \ display 3

\ The modulo function can be used to determine the
\ divisibility of one number by another:
: div? ( n1 n2 ---)
    over over mod cr
    if
        swap . ." is not "
    else
        swap . ." is "
    then
    ." divisible by " .
  ;
```

MOVE   c-addr1 c-addr2 u --

If u is greater than zero, copy u consecutive characters from the data space starting at c-addr1 to that starting at c-addr2, proceeding character-by-character from lower addresses to higher addresses.

MS n --

Waiting in milliseconds.

b class="alert"> CAUTION the word `ms` blocks all others process. For long expectations, it is advisable to split the wait long in a succession of short expectations in a loop-type **begin..until** for example.

```
500 ms \ delay for 1/2 second
```

NCLR i*x --

Clears the data stack.

NEGATE n -- n'

Two's complement of n.

```
5 negate .    \ display -5
```

NIP n1 n2 -- n2

Remove n1 from the stack.

NOT x1 -- x2

Invert all bits.

OR n1 n2 -- n3

Execute logic OR.

```
false  false or  .  \ display 0
false  true  or  .  \ display -1
true   false or  .  \ display -1
true   true  or  .  \ display -1
```

OVER n1 n2 -- n1 n2 n1

Place a copy of n1 on top of the stack.

```
2 5 OVER  \ duplicate 2 on top of the stack
```

PAGE --

Erases the screen.

PAYLOAD -- nbytes

This primitive output is only relevant after an invokation of `'` or `FIND`.

It will retrieve a word's definition length (code section) corresponding to the latest dictionary search. This word is of marginal importance. However, it facilitates an implementation of the disassembler in which that payload does not have to be specified at DIS's invokation time (see SW/examples/dis.4th for a minimal disassembler implementation).

PICK   xu ... x1 x0 u -- xu ... x1 x0 xu

Remove u. Copy the xu to the top of the stack.

POSTPONE   -- <name>

Skip leading space delimiters. Parse *name* delimited by a space. Find *name*. Append the compilation semantics of *name* to the current definition.

`POSTPONE` replaces most of the functionality of `COMPILE` and `[COMPILE]`. `COMPILE` and `[COMPILE]` are used for the same purpose: postpone the compilation behavior of the next word in the parse area. `COMPILE` was designed to be applied to non-immediate words and `[COMPILE]` to immediate words.

```
: ACTION-OF
   STATE @ IF
     POSTPONE ['] POSTPONE DEFER@
   ELSE
     ' DEFER@
   THEN ; IMMEDIATE
```

R>   R: n -- S: n

Transfers n from the return stack.

This operation must always be balanced with `>r`

```
\ display n in binary format
: b. ( n -- )
    base @ >r
    2 base !
    .
    r> base !
  ;
```

R@   -- x | R: x -- x

Copy x from the return stack to the data stack.

Warning: does not work in a `do..loop` loop.

RCLR R: i*x --

Clears the return stack.

RECURSE --

Append the execution semantics of the current definition to the current definition.

The usual example is the coding of the factorial function.

```
: FACTORIAL ( +n1 -- +n2)
   DUP 2 < IF DROP 1 EXIT THEN
   DUP 1- RECURSE *
;
```

REPEAT --

End a indefinite loop `begin.. while.. repeat`

RESTRICT --

Makes the latest non-anonymous definition available in compilation mode only.

ROT n1 n2 n3 -- n2 n3 n1

Rotate three top stack items.

RSHIFT x1 u -- x2

Right shift of the value x1 by u bits.

```
64 2 rshift .  \ display 16
```

RTC! rtcbyteval rtcregoffset --

Writes to an RTC register.

Please note that RTC support in Z79Forth/A is not official.

RTC@ rtcregoffset -- rtcbyteval

Retrieves the contents of an RTC register.

Please note that RTC support in Z79Forth/A is not official.

S -- sreg

Returns the contents of the system stack pointer (S).

S" comp: -- <string> | exec: addr len

In interpretation, leaves on the data stack the string delimited by "

In compilation, compiles the string delimited by "

When executing the compiled word, returns the address and length of the string...

```
\ header for DUMP
: headDump
    s" --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F"
  ;
headDump         \ push addr len on stack
headDump type    \ display: --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B
0C 0D 0E 0F
```

S>D n -- d

Sign extend single to double precision number.

```
57 s>d . .   \ display 0 57
```

S@ -- retaddr

Returns the return address to the caller's.

This is most useful for debugging purposes when used in conjunction with `.'`. Please note that for `.'` ti spuuly useful symbolic information, **RELFEAT** should have been enabled at compilation time (default).

SEE -- <name>

Decompile/disassemble the word <name>

```
see dup    \ display:
FC85 BDE9D7     jsr     CKDPTRA
FC88 AE         fcb     $AE
FC89 C4         fcb     $C4
FC8A 7EE7DB     jmp     NPUSH ok
```

SHIFT n1 n2 --

Shift n1 by n2 bits to the left if n2 is positive, otherwise shift to the right if n2 is negative.

```
4  1 shift .   \ display: 8
 4  2 shift .   \ display: 16
 2  0 shift .   \ display: 2
16 -1 shift .   \ display: 8
```

SIGN n --

If n is negative, add a minus sign to the beginning of the pictured numeric output string.

SPACE --

Display one space.

SPACES n --

If n is greater than zero, display n spaces.

STATE -- fl

Compilation state. State can only be changed by `[` and `]`.

SWAP n1 n2 -- n2 n1

Swaps values at the top of the stack.

```
2 5 swap
.   \ display 2
.   \ display 5
```

TAB -- 9

Push constant 9 on stack.

Cette constante permet d'utiliser le caractère *tabulation* qui ne peut pas être récupéré par `char` ou `[char]`.

THEN --

Immediate execution word used in compilation only. Mark the end a control structure of type `IF..THEN` or `IF..ELSE..THEN` .

THRU n1 n2 --

Loads the contents of a block file, from block n1 to block n2.

TICKS -- ud

System ticks. 64 ticks per millisecond.

Stacks up a double cell containing the number of ticks since boot time. There are 64 ticks per second. Please note that RTC support in Z79Forth/A is not official.

TO n --- <valname>

`to` assign new value to *valname*

TOUPPER c -- c'

Transforms the ASCII code of a character in [a..z] into its equivalent code in [A..Z].

```
char e toupper emit  \ display E
```

TRUE -- -1

Preset constant. Push -1 on stack.

```
true .    \ display: -1
```

TUCK n1 n2 -- n2 n1 n2

Insert n2 below n1 in the stack.

TYPE addr len --

Display the string characters over len bytes.

```
: hello ( --- addr c)
  s" Hello world" ;
hello type
```

U. x --

Removes the value from the top of the stack and displays it as an unsigned single precision integer.

```
1 u.                         \ display 1
-1 u.                        \ display 65535
```

U.R u n --

Display u right aligned in a field n characters wide.

If the number of characters required to display u is greater than n, all digits are displayed with no leading spaces in a field as wide as necessary.

U< u1 u2 -- flag

flag is true if and only if u1 is less than u2.

U> u1 u2 -- flag

flag is true if and only if u1 is morethan u2.

UM* u1 u2 -- ud

Unsigned 16x16 -> 32 bit multiply

UM/MOD   ud u1 -- u2 u3

Divide ud by u1, giving the quotient u3 and the remainder u2. All values and arithmetic are unsigned. An ambiguous condition exists if u1 is zero or if the quotient lies outside the range of a single-cell unsigned integer.

UNLESS   --

The code block covered by this conditional execution primitive should be terminated by a matching `THEN`.

Perl inspired. Functionally equivalent to:

```
: UNLESS

  ['] 0= COMPILE, POSTPONE IF

  ; IMMEDIATE RESTRICT
```

UNLOOP   --

Discard the loop-control parameters for the current nesting level. An `UNLOOP` is required for each nesting level before the definition may be EXITed.

UNMONITOR   --

Marks the latest non-anonymous definition as a non-target for integrity checks by `ICHECK`. The reason for this word to exist is to support VALUEs (see SW/examples/ansiextern.4th).

UNTIL   fl --

COMPILE ONLY

begin..until

UNUSED   -- u

u is the amount of space remaining in the region addressed by `HERE`, in address units.

VALUE   comp: n -- <valname> | exec: -- n

Define value.

valname leave value on stack.

A Value behaves like a Constant, but it can be changed.

```
12 value APPLES      \ Define APPLES with an initial value of 12
34 to APPLES         \ Change the value of APPLES. to is a parsing word
APPLES               \ puts 34 on the top of the stack
```

VARIABLE comp: -- <name> | exec: -- addr

Creation word. Defines a simple precision variable.

```
\ define variable speed
variable speed
35 speed !          \ store 32 in speed
10 speed +!         \ increment content of speed
speed @ .           \ display 45
```

WHILE fl --

Mark the conditionnal part execution of a structure `begin..while..repeat`

WITHIN n1 | u1 n2 | u2 n3 | u3 -- flag

Perform a comparison of a test value n1 | u1 with a lower limit n2 | u2 and an upper limit n3 | u3, returning true if either (n2 | u2 < n3 | u3 and (n2 | u2 <= n1 | u1 and n1 | u1 < n3 | u3)) or (n2 | u2 > n3 | u3 and (n2 | u2 <= n1 | u1 or n1 | u1 < n3 | u3)) is true, returning false otherwise.

WORD char "ccc" -- c-addr

Skip leading delimiters. Parse characters ccc delimited by char.

c-addr is the address of a transient region containing the parsed word as a counted string. If the parse area was empty or contained no characters other than the delimiter, the resulting string has a zero length. A program may replace characters within the string.

WORDS --

List the definition names in the first word list of the search order. The format of the display is implementation-dependent.

XOR n1 n2 -- n3

Execute logic eXclusif OR.

```
false false xor .  \ display 0
false true  xor .  \ display -1
true  false xor .  \ display -1
true  true  xor .  \ display 0
```

[--

Enter interpretation state. `[` is an immediate word.

['] -- <name>

COMPILE ONLY.

Compile xt of name as a literal.

[CHAR] comp: -- <spaces>name | exec: -- xchar

Place xchar, the value of the first xchar of name, on the stack.

```
: GC1  [CHAR] X      ;
: GC2  [CHAR] HELLO ;
GC1 \ push    58
GC2 \ push    48
```

\ --

Skip rest of line.

The word `\` must be followed by at least one space character and completed by the line comment.

```
2 3 + .   \ display: 5
```

] --

Return to compilation. `]` is an immediate word.

79Forth Repository Organization

Lien : https://github.com/frenchie68/Z79Forth

The sample files for 779Forth

Some of these have been "blockified" and are present in the CompactFlash image that can be found in SW/util/*.img, in which case credit to the original author can be found at the beginning of each block and also in SW/util/cfinit.sh.

100fact.4th

Michel Jean

Computes 100!. This came up as an SVFIG challenge. The file has to be sourced from the console.

bernd-oof.4th

Bernd Paysan

Bernd's mini objectr oriented Forth. This must be sourced from the console.

blkins.4th

Francois Laagel

Inserts one blank clock in front of a specified block range. Absolutely not guaranteed to do the right thing.

cftest.4th

Francois Laagel

Original prototype implementation for low level CompactFlash support. Kept for historical reference. Do not use this, as it is now implemented in the default EEPROM image!

coop-mtask.4th

Matthias Koch

A sample cooperative multitasker. A sample demo can be loaded from CF storage by resorting to COOPMTLOAD

dis.4th

Francois Laagel

The standard disassembler. It is normally loaded automatically from CompactFlash when the system comes up. Standard invokation is via:

```
SEE <word>
```

dump.4th

Francois Laagel

An hexadecimal DUMP utility. It can be loaded from CF by resorting to DUMPLOAD Usage is:

```
<addr> <bytecount> DUMP
```

enigma-f.4th

Bill Ragsdale

The current SVFIG's boss's take on the Enigma encryption system. The code has to be sourced from the console.

expsys.4th

Demitri Peynado

An expert system that tries to diagnose various forms of COVID-19/common flu. This is very interesting code in itself. It can be loaded from CF storage by resorting to: EXPSYSLOAD

Side note: the added code supports the functionality of Forthwin's WORDBYADDR

fsformat.4th

F+L

A Forth source code formatter. This comes from the archives of "Forth Dimension" that can be reached at http://forth.org/fd/FDcover.html

The code has to be sourced from the console.

fsh.4th

Francois Laagel

An alternative interpreter based on Peter Jakacki's Tachyon Forth "COMPEX CLI" concept.

After INTERP is invoked, one will enter a subshell which allows the user to invoke constructs that are normally available in compilation mode only, loops and conditionals fall under this category. Return stack contents may also be accessed interactively.

The feature supports input spread across multiple lines.

However, please be aware that this shell will be aborted on the the first error encountered.

WARNING: INTERP has to be invoked on a single command line, words following it will be ignored. The code has to be sourced from the console.

hanoi.4th

Peter Midnight

The classic towers of Hanoi recursion demo. This also comes from the archives of "Forth Dimension", however it also benefits from a bug fix found in the VfxForth distribution.

It can be loaded from CF storage by resorting to **`HANLOAD`** The application is started by issuing **`<nn> TOWERS`** where nn is a number between 2 and 10.

life.4th

Paul E. Bennett

An implementation of John Horton Comway's "Game of Life" and my first brush with ANSI Forth source code. The application can be started from CF storage by resorting to **`LIFELOAD`** It can be started over and over again by calling **`LIFE`**.

lwvi.4th

Francois Laagel

A lightweight implementation of the Berkeley VI editor, especially tuned for block editing.

This utility is not entirely finished and is partially buggy! It has its own documentation in *SW/doc/LWVI-UserManual.txt*. It can be loaded from CF storage by resorting to **`VILOAD`**.

After that any block can be edited via:

```
<nn> EDIT
```

mandel.4th

Martin H.

An ASCII based Mandelbrot fractal generation program (source is https://github.com/Martin-H1). The code has to be sourced from the console.

Meta2

Demitri Peynado

A syntax directed compiler generator for ValgolI. This is a remarkable program that takes a syntax specification from blocks 600-602 and uses Forth as an intermediary target language. Block 603 contains a sample program expressed in ValgolI. Further documentation can be found in SW/examples/meta2/README.txt

The program has to be sourced from the console. It makes extensive use of the following ANSI primitives: `DEFER IS :NONAME`

palflt.4th

Francois Laagel

A palindromic number generator that operates on double cells. It is a bit CPU intensive since it a brute force approach. It also uses `<# #S #>` to produce a string representation of every number under consideration. The application can be launched from CF storage by resorting to `PALFLTLOAD`

phrpal.4th

Pablo Hugo Reda

A palindromic number generator that operates one single cells. This is a much smarter approach that does not require number to string convertion. That application can be launched from CF storage by resorting to `PHRPALLOAD`

rc4.4th

Wikipedia's

An implementation of the RC4 encryption Forth page algorithm. This needs to be sourced from the console. Interesting is the use of `ANSI VALUE TO` primitives.

rtc.4th

Francois Laagel

Experimental application level support for the Motorola MC146818 real-time clock. This relies on hardware support that I do not endorse. There are better solutions to this then what I had initially envisioned.

sapin.4th

Michel Jean

Michel's obfuscated Forth code Christmas gift from a few years ago. The application can be launched from CF storage by resorting to `SAPINLOAD`

test.4th

Francois Laagel

Vestigial code used for validation purposes at some point. This should probably go...

vt100.4th

Francois Laagel

Vestigial code of little interest besides folks interested in terminal based editor development.