v-FORTH 1.5

ZX Spectrum Next version

1990-2020 Matteo Vitturi

Technical Info

Introduction

This document introduces some technical details about this Forth implementation and the glossary of all core words.

This is a straight FIG-Forth I ported to the new **Sinclair ZX Spectrum Next** based on my previous work "vForth 1.413" available at https://github.com/mattsteeldue/vforth.

This version "vForth 1.5" is available on GitHub repository too at https://github.com/mattsteeldue/vforth-next.
The first main big difference from the previous version is that it uses a dedicated file on SD instead of on a ZX Microdrive cartridge.

Even if this is a working piece of software, the porting is still a work-in-progress, there are many things to do.

Disclaimer

Copying, modifying and distributing this software is allowed provided this copyright notice is kept.
This work is available "as-is" with no whatsoever warranty.

The author – me – is not a native English speaking and, for certain, you will find grammatical errors. In case, it would be very appreciated if you could drop me a line with any suggestion and/or correction. I am not able to write a longer disclaimer than the above.

Contents

Technical specifications

CPU Registers

This is a straight FIG-Forth Z80 implementation. Registers are used in the in the following way:

AF – Used for normal operations.
BC – **Instruction Pointer**: should be preserved on enter-exit a definition and during ROM/OS calls.
DE – Free (Low part when used for 32-bit manipulations)
HL – **Work Register** (High part when used for 32-bit manipulations)
AF'– Not used, somewhere used for backup purpose
BC'– Not used: available in fast Interrupt via EXX
DE'– Not used: available in fast Interrupt via EXX
HL'– Not used: available in fast Interrupt via EXX (saved at startup from Basic)
SP – Calculator Stack Pointer
IX – Used to point to the Forth "inner-interpreter".
IY – Used by ZX System

Much care has been taken to avoid any use of alternate registers (at least with interrupts enabled). This should allow users to create their own fast-response interrupt routine with EXX instead of pushing away all registers.

16 bits Number Encoding

A 16 bits *integer* represents an integer number between –32768 and +32767 inclusive. The sign is kept in the most significant bit. Alternatively, the it represents an *unsigned integer* between 0 and +65535.

		H	L
16 bit:	HL:	| **s**bbb bbbb |	bbbb bbbb |

In the CPU registers, an *integer* is kept in H and L where H is the most significant part.
In memory, an *integer* is stored in two contiguous bytes in "little-endian" way, that is, the lower address has the least significant part, the in L register. The byte at higher address has the most significant part, the one in H register, as usual for Zilog Z80.

32 bits Number Encoding

The second integer format requires two *integers* that form a 32 bits number said *double* or *long* that allows an integer between –2.147.483.648 and +2.147.483.647, and the sign is kept on the most significant bit of the first *integer*.

Imagine a *double integer* kept in CPU register in the in this way:

	H	L	D	E
32 bits:	**s**bbb bbbb bbbb bbbb	bbbb bbbb bbbb bbbb	bbbb bbbb bbbb bbbb	bbbb bbbb bbbb bbbb

using register H, L, D and E, with the most significant part in H, and the least in E.

Then, on Calculator Stack the *double integer* requires four contiguous bytes split in the two *integers* that forms it with the most significant integer (HL) on top of Calculator Stack (i.e. in the lower addresses), and the least significant integer (DE) the second element from top is in the higher address, that is the second element from top. so it appears as L H E D,

CPU	Calculator Stack
D	SP + 3
E	SP + 2
H	SP + 1
L	SP + 0 (Top Of Stack)

More confusingly, in RAM it is kept as E D L H. see how `2VARIABLE` is defined to understand this fact.

CPU	2VARIABLE
H	Address + 3
L	Address + 2
D	Address + 1
E	Address + 0

Floating-Point Number Encoding

There is a third optional format that use 32 bits as a *double integer*, but all bits are used in a different way to allows to represent a *floating point number* approximately between –1.7 * 10^38 and +1.7 * 10^38 with 6-7 precision digits.
The sign is kept in the most significant bit, the same way as a *double integer*; then eight bits follow as the exponential part, then 23 bits of mantissa. The sign in this position allows (IMO) using most of the same semantics of *double integers* as per the sign of the number.

	H	L	D	E
32 bits f.p.:	s**xxx xxxx**	**x**bbb bbbb bbbb bbbb	bbbb bbbb bbbb bbbb	bbbb bbbb bbbb bbbb

Far-Pointer Address Encoding

There is a third optional format that use 32 bits as a *double integer*, but all bits are used in a different way to allows to represent a *floating point number* approximately between –1.7 * 10^38 and +1.7 * 10^38 with 6-7 precision digits.
The sign is kept in the most significant bit, the same way as a *double integer*; then eight bits follow as the exponential part, then 23 bits of mantissa. The sign in this position allows (IMO) using most of the same semantics of *double integers* as per the sign of the number.

	H	L	D	E
32 bits f.p.:	s**xxx xxxx**	**x**bbb bbbb bbbb bbbb	bbbb bbbb bbbb bbbb	bbbb bbbb bbbb bbbb

Dictionary

Legenda

In this list:

a	address: memory address	16 bits
b	byte: unsigned integer	8 bits
c	character	8 bits but often only lower 7 are significant.
d	signed double integer	32 bits
fp	floating point number.	32 bits
ha	heap-address	16 bits.
n	signed integer (see >FAR)	16 bits
u	unsigned integer	16 bits
ud	unsigned double integer	32 bits
f	flag: a number evaluated as a boolean	16 bits
ff	false flag: zero	16 bits
tf	true flag: non-zero	16 bits
nfa	name field address	16 bits
lfa	link field address	16 bits
cfa	code field address	16 bits
pfa	parameters field address	16 bits
xt	execution token – same as cfa	16 bits

cccc	character string or word name available in the vocabulary
...	a list of words
TOS	top of calculator stack

Core dictionary

'null' --- (immediate)

This is a "ghost" word executed by `INTERPRET` to go back to the caller once the text to be interpreted ends. This word allows you to use a 0x00 (NULL ASCII) as the end-of-text indicator in the input text stream.

! n a ---

It stores the integer `n` in the memory cell at address `a` and `a + 1`. Pronounced "store"
Zilog Z80 is a little-endian CPU that holds the high byte in the high address.

!CSP ---

It saves the value of SP register in `CSP` user variable. It is used by `:` and `;` for syntax checking.

 # d1 --- d2

From a double number `d1` it produces the next ASCII character to be put in an output string using `HOLD`. The number `d2` is `d1 / BASE` and is kept for subsequent elaborations. It is used between `<#` and `#>`. See also `#S`.

#> **d** **---** **a b**

It terminates a numeric conversion started by <# . It removes d and leaves the values suitable for TYPE.

#BUFF **---** **n**

This is a constant that gives the number of available buffers. This build has 3 buffers located at address between FIRST @ and LIMIT @.

#S **d1** **---** **d2** **CORE**

This word is equivalent of a series of # that is repeated until d2 becomes zero. It is used between <# and #>.

#SEC **---** **n**

This is a constant that gives the number of available screens/blocks.

' **---** **cfa**

Pronounced “tick”. Used in the form

```
' cccc
```

it leaves the cfa of word cccc, that is its xt or value to be compiled of passed to EXECUTE. If the word cccc is not found after the CURRENT and CONTEXT search phases, then an error is raised.
In a previous version of this Forth, this word returned pfa: we changed this previous standard to return cfa.

(**---** **(immediate)**

Used in the form

```
( cccc )
```

it ignores what is between brackets. The space after (is not considered in cccc. The comment must be delimited in the same row with a closing) followed by a space or an end-of-line.

(+LOOP) **n** **---**

This is the primitive compiled by +LOOP.

(.") **---**

This is the primitive compiled by **."** and **.(**. It executes TYPE.

(;CODE) **---**

This is the primitive compiled by ;CODE. It rewrites the cfa of LATEST word so that it points to the machine code starting from the following cell.

(?DO) **---**

This is the primitive compiled by ?DO.
At compile-time compiles the cfa of (?DO) followed by an offset like BRANCH does that is used to jump after the whole ?DO **...** LOOP structure if the limit equals the initial index, otherwise it is equivalent to (DO).

(?EMIT) c1 --- c2

It decodes the character c1 using the following table. It is used internally by EMIT.

HEX 06 → print-comma
HEX 07 → bell rings
HEX 08 → back-space
HEX 09 → tabulator
HEX 0D → carriage return

For not listed character, c2 is equal to c1.

(ABORT) ---

Word executed in case of error issued by ERROR when WARNING contains a negative number. This word usually executes ABORT but can be patched with a user defined word at the pfa of (ABORT).

(COMPARE) a1 a2 n -- b

This word performs a lexicographic compare of n bytes of text at address a1 with n bytes of text t address a2. It returns numeric a value

0 : if strings are equal
+1 : if string at a1 greater than string at a2
-1 : if string at a1 less than string at a2

(DO) ---

This is the primitive compiled by DO.

(FIND) a1 a2 --- cfa b tf
--- ff

It searches in the dictionary starting from address a2 a word which text name is kept at address a1; it returns a cfa, the first byte b of nfa and a tf on a successful search; elsewhere a ff only.
Address a2 must be the nfa of the first word involved in the search in the vocabulary.
In previous version of this Forth, it returned a pfa, we change our mind.
Byte b keeps the length of the found word in the least significant 5 bits, bit 6 is the IMMEDIATE flag. Bit 5 is the SMUDGE bit. Bit 7 is always set to mark the beginning or end of the nfa.

(LINE) n1 n2 --- a b

It retrieves line n1 of block n2 and send it to buffer. It returns the address a within the buffer and a counter b that is C/L (=32) to mean a whole line.

(LOOP) ---

This is the primitive compiled by LOOP. See also DO and +LOOP.

(NEXT) --- a

Constant. It is the address of "next" entry point for the Inner Interpreter. When creating word using machine code, the last op-code should be an unconditional jump to this address. If the created word wants to return an *intger* value on TOS, it should jump to the previous address; and if it wants to return a *double integer* value, it should jump to the next previous one. For example, to create a word to disable interrupts, without an ASSEMBLER, you could use te following snippet:

```
CREATE INT-DI HEX
```

```
F3 C,           \  di
C3 C, (NEXT) , \  jp (NEXT)
SMUDGE          \  now a dictionary search will find this word
```

(NUMBER) d a --- d2 a2

It converts the ASCII text at address `a + 1` in a double integer using the current `BASE` . Number `d2` is left for the subsequent elaborations, `a2` is the address of the first non-converted character. A double integer is kept in CPU registers as HLDE. On the stack is treated as two distinct integers where HL is on TOS and DE is the second from top, so that in memory it appears as LHED. Instead, in a variable declared with 2VARIABLE is stored as EDHL.
Used by NUMBER.

(SGN) a --- a2 f

It determines if the character at address `a` is a sign (+ o –) and if found increments `a`. The flag f indicates the sign: `ff` when it founds a positive sign + or no sign at all, `tf` for a negative sign – . If a is incremented then variable `DPL` is incremented aswell. Used by da `NUMBER` and `(EXP)` in the floatin-point option.

*** n1 n2 --- n3**

It leaves the product of two integers.

***/ n1 n2 n3 --- n4**

It executes (`n1` · `n2`) / `n3` using an intermediate double integer to avoid precision loss.

***/MOD n1 n2 n3 --- n4 n5**

It leaves the quotient `n5` and the reminder `n4` of the operation (`n1`· `n2`) / `n3` using an intermediate double integer to avoid precision loss.

+ n1 n2 --- n3

It leaves the sum of two integer.

+! n a ---

It adds to the cell at address `a` the number `n`. It is the same as the sequence `a @ n + a !`

+- n1 n2 --- n3

It leaves `n3` as `n1` with the sign of `n2`. If `n2` is zero, it means positive.

+BUF a1 --- a2 f

It advances the address of the buffer from `a1` to `a2`, that is the next buffer. The flag `f` is false if `a2` is the buffer pointed by `PREV`.

+LOOP **n1** **---** **(run time)**
a n2 **---** **(compile time)**

Used in colon definition in the form
`DO ... n1 +LOOP`

At run-time `+LOOP` checks the return to the corresponding DO, `n1` is added to the index and the total compared with the limit. The jump back happens :

a) while index < limit if `n1` > 0;
b) while index > limit if `n1` < 0.

Otherwise the execution leaves the loop. On leaving the loop, the parameters are discarded and the execution continues with the following word.
At compile-time `+LOOP` compiles `(+LOOP)` and a jump is calculated from `HERE` to `a` which is the address left on the stack by `DO`. The value `n2` is used internally for syntax checking.

+ORIGIN **n** **---** **a**

It gives the address n bytes after the "origin". In this build the origin is 6400h. Used rarely to modify the boot-up parameters in the origin area.

, **n** **---**

It puts n in the following cell of the dictionary and increments `DP` (dictionary pointer) of two locations.

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

It leaves `n3` = `n1` – `n2` as the difference from the penultimate and the last number on the stack.

--> **---**

It continues the interpretation in the next Screen during a `LOAD` .

-1 **---** **n**

This is the constant value `-1` that in this implementation is 0FFFFh. Compiling a constant result in a faster execution than a literal.

-DUP **n** **---** **n n (non zero)**
n **---** **n (zero)**

It duplicates `n` if it is non zero.

-FIND **---** **cfa b tf** **(ok)**
--- **ff** **(ko)**

Used in the form `-FIND cccc`.
It accepts a word (delimited by spaces) from the current input stream, storing it at address `HERE`. Then, it run a search in the `CONTEXT` vocabulary first, then in the `CURRENT` vocabulary. If the word is found, it leaves the `cfa` of the word, its length-byte `b` and a `tf`. Otherwise only a `ff`.

-TRAILING a1 n1 --- a2 n2

It assumes that a string `n1` characters long is already stored at address `a1` that contains a word right-delimited with spaces. It determines `n2` as the position of the first delimiter after the word.

. n ---

It prints the integer `n` followed by a space.

." --- (immediate)

Used in the form

```
." cccc "
```

At compile-time, within a colon-definition, compiles the primitive to output the text followed by the string `cccc` (delimited by "). The text `cccc` is prepended by a length-counter that `TYPE` will use at run-time.
When interpreted, i.e. outside a colon-definition, immediately sends the text to output.

.(--- (immediate)

Used in the form

```
.( cccc )
```

it acts as `." cccc "` but the string is delimited in a different way

.C c --- (immediate)

Used in the form

```
c .C xxxx C
```

Acts as `." xxxx "` but the string is delimited by character `c`. It is a more generic form of `.(` and `."` that, in fact, use this word as their primitive.

.LINE n1 n2 ---

It sends line `n1` of block `n2` to the current peripheral ignoring the trailing spaces.

.R n1 n2 ---

It prints a number `n1` right aligned in a field `n2` character long, with no following spaces. If the number needs more than `n2` characters, the excess protrudes to the right.

/ n1 n2 --- n3

It leaves `n3` = `n1`/ `n2`, the quotient of the integer division.

/MOD n1 n2 --- n3 n4

It leaves the quotioent `n4` and the reminder `n3` of the integer division `n1`/ `n2`. The reminder has the sign of `n1`.

0 --- n

This is a constant value zero. Compiling a constant results in a faster execution than a literal.

0< **n** **---** **f**

It leaves a `tf` if `n` is less than zero, `ff` otherwise.

0= **n** **---** **f**

It leaves a `tf` if `n` is not zero, `ff` otherwise. It is like a NOT `n`.

0> **n** **---** **f**

It leaves a `tf` if `n` is greater than zero, `ff` otherwise.

0BRANCH **f** **---**

Direct procedure that executes a conditional jump. If `f` is zero the offset in the cell following `0BRANCH` is added to the Instruction Pointer to jump forward of backward.
It is compiled by `IF`, `UNTIL` and `WHILE`.

1 **---** **n**

Constant value 1. Compiling a constant results in a faster execution than a literal.

1+ **n1** **---** **n2**

It increments by one the number on TOS.

1- **n1** **---** **n2**

It decrements by one the number on TOS.

2 **---** **n**

Constant value 2. Compiling a constant results in a faster execution than a literal.

2! **d a** **---**
n-lo n-hi a **---**

It stores the double integer held on TOS to address a.

2* **n1** **---** **n2**

It doubles the number on TOS.

2+ **n1** **---** **n2**

It increments by two the number on TOS.

2/ **n1** **---** **n2**

It halves the number on TOS.

2@ **a** **---** **d**
a **---** **n-lo** **n-hi**

It fetches the double integer at address a. to TOS.

2DROP **d** **---**
n1 **n2** **---**

It discards a double integer from the TOS, i.e. discards the top two integer.

2DUP **d** **---** **d** **d**

It duplicates the double integer on TOS, i.e. duplicates in order the two top integer.

2OVER **d1** **d2** **---** **d1** **d2** **d1**
n1 **n2** **n3** **n4** **---** **n1** **n2** **n3** **n4** **n1** **n2**

It copies to TOS the second double integer from top.

2ROT **d1** **d2** **d3** **---** **d2** **d3** **d1**
n1 **n2** **n3** **n4** **n5** **n6** **---** **n3** **n4** **n5** **n6** **n1** **n2**

It rotates the three top double integers, taking the third an putting it on top. The other two double integer are pushed down from top by one place.

2SWAP **d1** **d2** **---** **d2** **d1**

It swaps the two double integers on TOS.

3 **---** **n**

Constant value 3. Compiling a constant results in a faster execution than a literal.

: **---** **(immediate)**

This is a defining word that creates and begins a colon-definition. Used in the form

`: cccc ... ;`

creates in the dictionary a new word `cccc` so that it executes the sequence of already existing words '**...**'.
The `CONTEXT` vocabulary is set to be the `CURRENT` and compilation continues while `STATE` is not zero. Words having the bit 6 of its length-byte set are immediately executed instead of being compilated.

; **---** **(immediate)**

It ends a colon definition and stops compilation.. It compiles ;S and execute `SMUDGE` to make the word findable.

;CODE **---** **(immediate)**

Used in the form

`: cccc ... ;CODE`

it terminates a colon definition stoppin copilation of word `cccc` and compiling `(;CODE)`. Usually `;CODE` is followed

by suitable machine code sequence..

;S **---** **(immediate)**

This is usually the last word compiled in a colon definition by ; it does the action of returning to the calling word. It is used to force the immediate end of a loading session started by LOAD.

< **n1 n2** **---** **f**

It leaves a tf if n1 is less than n2, ff otherwise.

<# **---**

It sets HLD to the value of PAD. It is used to format numbers using #, #S, SIGN and #>. The conversion is performed using a double integer, and the formatted text is kept in PAD.

<BUILDS **---**

Used in a colon definition in the form

```
: cccc ... <BUILDS ... DOES> ... ;
```

Subsequent execution of cccc in the form

```
cccc nnnn
```

creates a new word nnnn with an high-level procedure that at run-time calls the DOES> part of cccc. When nnnn is executed, the pfa of nnnn is put on TOS and the executed the following DOES>.
<BUILD and DOES> allow writing high-level procedures instead of using machine code as ;CODE would require.

<FAR **a n** **---** **ha**

Given an address a (to be intended as an address between E000h and FFFFh) and a page number n for an 8K-page between 64 and 71 (or 40h – 47h) this definition encodes a number between 0 and 7 in the three most significant bits of ha and a 13-bits offset in the remaining bits. It does not change MMU7 page. See >FAR, MMU7!.

<NAME **cfa** **---** **nfa**

It converts a cfa in its nfa. It is the same as the sequence >BODY NFA.
See also: CFA, LFA, NFA, PFA, >BODY.

= **n1 n2** **---** **f**

It leaves a tf if n1 equals to n2, ff otherwise.

> **n1 n2** **---** **f**

It leaves a tf if n1 is greater than n2, ff otherwise.

>BODY **cfa** **---** **pfa**

Converts a cfa in its pfa.
See also: CFA, LFA, NFA, PFA, <NAME.

>FAR **ha** **--- a n**

Given a heap-encoded pointer `ha` this definition decodes top three bits as one of the 8K-page number between 64 and 71 or (40h – 47h) and lower bits as the offset from E000h. It does not change the MMU7 page. See `<FAR`, `MMU7!`.

>R **n** **---**

It takes an integer from TOS and puts it on top of the Return Stack. It should be used only within a colon definition and the use of `>R` should be balanced with a corresponding `R>`.

? **a** **---**

It prints the content of cell at address `a`. It is the same as the sequence: `a @` .

?COMP **---**

It raises an error message #17 if the current `STATE` is not compile state.

?CSP **---**

It raises an error message #20 if the value of `CSP` is different from the current value of SP register. It is used to check the compilation in a colon definition.

?DO **n1 n2** **---** **(immediate)** **(run time)**
--- a n **(compile time)**

Used in a colon definition in the form

```
?DO ... LOOP
?DO ... n3 +LOOP
```

It is used as `DO` to put in place a loop structure, but at run-time it first checks if `n1` = `n2` and in that case the loop is skipped. At run-time `?DO` starts a sequence of words that will be repeated under control of an initial-index `n2` and a limit `n1`. `?DO` consumes these two value from stack and the corresponding `LOOP` increments the index. If the index is less than the limit, the executions returns to the corresponding `?DO`, otherwise the two parameters are discarded and the execution continues after the `LOOP`.
The limt `n1` and the initial value `n2` are determined during the execution and can be the result of other previous operations. Inside a loop the word `I` copies to TOS the current value of the index.
Se also: `I`, `DO`, `LOOP`, `+LOOP`, `LEAVE`. In particular `LEAVE` allows leaving the loop at the first opportunity.
At compile-time `?DO` compiles `(?DO)` followed by an offset like `BRANCH` and leaves the address of the following location and the number `n` to syntax-check

?DUP **n** **--- n n (non zero)**
n **--- n (zero)**

It duplicates the value on TOS if it is not qual to zero. This is the same as `-DUP`.

?ERROR **f n** **---**

It raises an error message #`n` if `f` is true.

?EXEC **---**

It raises an error message #18 if we aren't compiling.

?LOADING **---**

It raises an error message #22 if we aren't loading. It show the illegal use of `-->`.

?PAIRS **n1 n2** **---**

It raises an error message #19 if n1 is different from n2. It is used for syntax checking by the words that completes the construction of structures `DO`, `BEGIN`, `IF`, `CASE`.

?STACK **---**

It raises an error message #1 if the stack is empty and we tried to consume an element from the calculator stack.
It raises an error message #7 if the stack is full.

?TERMINAL **--- f**

It tests the keyboard. Leaves a `tf` if the [BREAK] key is pressed, `ff` oherwise.

@ **a** **--- n**

It puts on TOS the integer currently held in the cell ad address `a`.

ABORT **---**

It clears the stack and pass to prompt command, prints the copyright message and returns the control to the human operator executing `QUIT`.

ABS **n** **--- u**

It leaves the absolute value of `n`.

ACCEPT **a n1** **--- n2**

It transfers characters from the input terminal to the address `a` for `n1` location or until receiving a 0x13 "CR" character. A 0x00 "null" character is added. It leaves on TOS `n2` as the actual length of the received string. More, `n2` is also copied in `SPAN` user variable. See also `ACCEPT`.

ACCEPT- **a n1** **--- n2**

As for `ACCEPT`, but it reads at most `n1` characters text from current channel/stream via `INKEY` one character at a time, It stores the text at address `a`. Not so efficient, but it allows to compile an external souce-file attached to the stream. It does not modify `SPAN`.

AGAIN **--- (immediate)** **(run time)**
a n **--- (compile time)**

Used in colon definition in the form

```
BEGIN ... AGAIN
```

At run-time `AGAIN` forces the jump to the corresponding `BEGIN` and has no effect on the calculator stack. The execution

cannot leave the loop (at least until a R> is executed at a lower level).
At compile-time AGAIN compiles BRANCH with an offset from HERE to a. The number n is used for syntax-check.

ALLOT n ---
It adds the signed integer n to DP (Dictionary Pointer). It is used to reserve some space in the dictionary or to free memory.

AND n1 n2 --- n3
It executes an AND binary operation between the two integers. The operation is performed bit by bit.

AUTOEXEC ---
This word is called the first time the Forth system boot to load Screen# 1. Once called it patches itself to prevent furhter runs.

B/BUF --- n
Constant that is the number of bytes per buffer. In this implementation is 512.

B/SCR --- n
Constant that indicates the number of Blocks per Screen. In this implementation is 1.

BACK a ---
It calculates and compiles a relative offset from a to HERE. Used by AGAIN, UNTIL, LOOP, +LOOP.

BACK- [a1 n1] a n ---
It calculates and compiles a relative offset from a to HERE and in case it completes the BRANCH part previously compiled by ?DO that left a1 and n1. It is used by LOOP, +LOOP. If the loop begin with DO then a1 and n1 aren't there.

BASE --- a
User variable that indicates the current numbering base used in input/output conversions. It is changed by DECIMAL that put ten, HEX that put sixteen, and with some extensions BINARY that put two and OCTAL that put eight.

BASIC u ---
It quits Forth and returns to Basic returning to the caller USR the unsigned integer on TOS.

BEGIN --- (immediate) (run time)
--- a n (compile time)
Used in colon definition in the forms

```
BEGIN ...   AGAIN            or
BEGIN ... f UNTIL            or
BEGIN ... f WHILE ... REPEAT or
```

```
BEGIN ... f END
```

At compile-time, it starts one of these structures.
At run-time BEGIN marks the beginning of a words sequence to be repeatedly executed and indicates the jump point for the corresponding AGAIN, REPEAT, UNTIL or END.
With UNTIL, the jump to the corresponding BEGIN happens if on TOS there is a ff, otherwise it quits the loop.
With AGAIN and REPEAT, the jump to the corresponding BEGIN always happens.
The WHILE part is executed if and only if on TOS there is a tf, otherwise it quits the loop.

BL --- c

Constant for "Blank". This implementation uses ASCII and BL is 32.

BLANKS a n ---

It fills with "Blanks" n location starting from address a.

BLK --- a

User variable that indicates the current block to be interpreted. If zero then the input is taken from the terminal buffer TIB.

BLOCK n --- a

It leaves the address of the buffer that contains the block n. If the block isn't already there, it is fetched from disk. If in the buffer there was another buffer and it was modified, then it is re-written to disk before reading the block n.
See also BUFFER, R/W, UPDATE, FLUSH.

BRANCH ---

Direct procedure that executes an unconditional jump. The memory cell following BRANCH has the offset to be relatively added to the Instruction Pointer to jump forward or backward. It is compiled by AGAIN, ELSE, REPEAT.

BUFFER n --- a

It makes the next buffer available assigning it the block number n. If the buffer was marked as modified (by UPDATE), such buffer is re-written to disk. The block is not read from disk. The address point to the first character of the buffer.

BYE ---

It executes FLUSH and EMPTY-BUFFERS, then quits Forth and returns to Basic returning to the caller USR the value of 0 +ORIGIN. See also BASIC.

C! b a ---

It stores a byte b to address a.

C, b ---

It puts a byte b in the next location available in the dictionary and increments DP (dictionary pointer) by 1.

C/L **---** **c**

Constant that indicate the number of characters per screen line. In this implementation it is 32.

C@ **a** **---** **b**

It puts on TOS the byte at address `a`.

CASEOFF **---**

It sets case-sensitive search OFF. changes the system behavior so that (FIND) can search the dictionary ignoring case.

CASEON **---**

It sets case-sensitive search ON. It changes the system behavior so that (FIND) will search the dictionary case sensitive.

CELL+ **n1** **---** **n2**

It increments `n1` by 1 “cell”, that is two units. In this implementation a cell is two bytes.

CELL- **n1** **---** **n2**

It decrements `n1` by 1 “cell”, that is two units. In this implementation a cell is two bytes.

CELLS **n1** **---** **n2**

It doubles the number `n1` on TOS giving the number of bytes equialent to `n1` “cells”. In this implementation a cell is two bytes.

CFA **pfa** **---** **cfa**

It converts a `pfa` in its `cfa`. See also `LFA`, `NFA`, `PFA`, `>BODY`, `<NAME`.

CHAR **---** **c**

Used in the form

```
CHAR c
```

determines the first character of the next word in the input stream.

CLS **---**

It clears the screen using the ZX Spectrum ROM routine 0DAFh.

CMOVE **a1** **a2** **n** **---**

It copies the content of memory starting at address `a1` for `n` bytes, storing them from address `a2`. The content of address `a1` is moved first. See also `CMOVE>`.

CMOVE> **a1** **a2** **n** **---**

The same as `CMOVE` but the copy process starts from location `a1 + n - 1` proceding backward to the location `a1`.

COLD **---**

This word executes the Cold Start procedure that restore the system at its startup state.
It sets DP to the minimum standard and executes ABORT.

COMPILE **---**

At compile-time, it determines the cfa of the word that follows COMPILE and compile it in the next dictionary cell.

CONSTANT **n** **---** **(immediate)** **(compile time)**
--- **n** **(run time)**

Defining word that creates a constant. Used in the form

```
n CONSTANT cccc
```

it creates the word cccc and pfa holds the number n. When cccc is later executed it put n on TOS.

CONTEXT **---** **a**

User variable that points to the vocabulary address where a word search begins.

COUNT **a1** **---** **a2** **b**

It leaves the address of text a2 and a length b. It expects that the byte at address a1 to be the length-counter and the text begins to the next location.

CR **---**

It transmits a 0x0D to the current output peripheral.

CREATE **---** **a**

Defining word used in the form

```
CREATE cccc
```

it creates a new dictionary entry for the definition cccc. The cfa of such a definition points to its pfa that is empty for the moment. HERE points this location.
The new word is created in the CURRENT vocabulary but won't be found by (FIND) because it has the SMUDGE bit set.
Once the word construction is complete, it is a programmer responsibility to execute SMUDGE.
Used by : and CONSTANT.

CSP **---** **a**

User variable that temporarily holds the value of SP register during a compilation syntax error check.

CURRENT **---** **a**

User variable that points to the address in the Forth vocabulary where a search continues after a failing search executed in the CONTEXT vocabulary. See also LATEST.

D+ **d1 d2 --- d3**

It leaves `d3` as the sum of `d1` and `d2`. This is a 32 bits sum.

D+- **ud n --- d**

It leaves `d` that is `ud` with the sign of `n`.

D. **d ---**
n-lo n-hi ---

It prints a double integer followed by a space. The double integer is kept on stack in the format n-lo n-hi and the integer on TOS is the most significant.

D.R **d n ---**

It prints a double integer rigth aligned in a field `n` character wide. No space follows. If the field is not large enough, then the excess protrudes to the right.

DABS **d --- ud**

It leaves the absolute value of a double integer.

DECIMAL **---**

It sets `BASE` to 10, that is the decimal base.

DEFINITIONS **---**

To be used in the form

```
cccc DEFINITIONS
```

sets the `CURRENT` vocabulary to be the `CONTEXT` vocabulary and this allows adding new definitions to `cccc` vocabulary. For example: `FORTH DEFINITIONS` or `ASSEMBLER DEFINITIONS`.
In this implementation an `ASSEMBLER` vocabulary is available as an extra-option that can be LOADed from screens 100 - 160.

DEVICE **--- a**

Variable that holds the number of current channel: 2 for video, 3 for printer, 4 for the file open to “!Blocks.bin”, etc.

DIGIT **c n --- u tf (ok)**
c n --- ff (ko)

It converts the ASCII character `c` in the equivalent number using the base `n`, followed by a a `tf`. If the conversion fails it leaves a `ff` only.

DL **--- a**

User variable that keeps the data-stream number used in a `LOAD` from stream using a negative screen number.

DLITERAL **d** **---** **d** **(immediate)** **(run time)**
d **---** **(compile time)**

Same as LITERAL but a 32 bits number is compiled. DLITERAL is an immediate word that is executed and not compiled.

DMINUS **d1** **---** **d2**

It leaves the opposite double number.

DO **n1 n2** **---** **(immediate)** **(run time)**
--- **a n** **(compile time)**

Used in colon definition in the form

```
DO ... LOOP        or
DO ... n +LOOP
```

It is used to put in place a loop structure: The execution of DO starts a sequence of words that will be repeated, under control of an initial-index n2 and a limit n1. DO drops these two value from stack and the corresponding LOOP increments the index. If the index is less than the limit, the executions returns to the corresponding DO, otherwise the two parameters are discarded and the execution continues after the LOOP.
The limt n1 and the initial value n2 are determined during the execution and can be the result of other previous operations. Inside a loop the word I copies to TOS the current value of the index.
See also: I, DO, LOOP, +LOOP, LEAVE. In particular LEAVE allows leaving the loop at the first opportunity.
At compile-time DO compiles (DO) and leaves the address of the following location and the number n to syntax-check.

DOES> **---**

Word that defines the execution action of a high-level defining word. DOES> changes the pfa of the word being defined to point the words sequence compiled after DOES>. It is used in conjunction with <BUILDS. When the machine-code part of DOES> is executed, it leaves on TOS the pfa of the new word, this allows the interpreter to use this area. Obvious use are new vocabularies (Assembler), multidimensional array and other compiling operations.

DOSSCALL n1 n2 n3 n4 a --- n4 n5 n6 n7

This is the NZXOS call wrapper. Parameters passed on stack are used as follow:

n1 = input parameter value for hl registers pair
n2 = input parameter value for de registers pair
n3 = input parameter value for bc registers pair
n4 = a register input parameter value
a = service routine address
n5 = hl returned value
n6 = de returned value
n7 = bc returned value
n8 = a register, i.e. error code or zero when everything is OK.

This word takes care of paging in and out RAM and ROM, and calls the specified routine.
Value returned on register a is also stored at DECIMAL 40 +ORIGIN or HEX 28 +ORIGIN.

DP **---** **a**

User variable (Dictionary Pointer) that holds the address of next available memory location in the dictionary. It is read by HERE and modified by ALLOT.

DPL **---** **a**

User variable that holds the number of digits after the decimal point during the interpretation of double integer. It can be used to keep track of the column of the decimal point during a number format output. For 16 bit integer it defaults to –1. It takes into account the exponential part and its sign for floating point numbers.

DROP **n** **---**

It drops the value on TOS. See also OVER, NIP, TUCK, SWAP, DUP, ROT.

DUP **n** **---** **n n**

It duplicates the value on TOS. See also OVER, DROP, NIP, TUCK, SWAP, ROT.

ELSE **a1 n1** **---** **a2 n2** **(immediate)** **(compile time)**
--- **(run time)**

Used in colon definition in the form

```
IF ... ELSE ... ENDIF
IF ... ELSE ... THEN
```

At run-time ELSE forces the execution of the false part of an IF-ELSE-ENDIF structure. It has no effects on the stack.
At compile-time ELSE compiles BRANCH and prepares the following cell for the relative offset, stores at a1 the previous offset from HERE; then it leaves a2 and n2 for syntax checking.

EMIT **c** **---**

It sends a printable ASCII character to the current output peripheal. OUT is incremented. 7 EMIT activates an acoustic signal. The 'null' 0x00 ASCII character is not transmitted.

EMITC **b** **---**

It sends a byte b character to the current output peripheal selected with SELECT. See also DEVICE.

EMPTY-BUFFERS **---**

It erases all buffers. Any data stored to buffers after the previous FLUSH is lost.

ENCLOSE **a c** **---** **a n1 n2 n3**

Starting from address a, and using a delimiter character c, it determines the offset n1 of the first non-delimiter character, n2 of the first delimiter after the text, n3 of first character non enclosed.
This word doesn't go beyond a 'null' ASCII that represent a unconditional delimiter. For example:

```
1:    c  c  x  x  x  c  x      →    2  5  6
2:    c  c  x  x  x  'null'   →    2  5  5
3:    c  c  'null'            →    2  3  2
```

END **a n** **---** **(immediate)** **(compile time)**
f **---** **(run time)**

Synonym of UNTIL.

ENDIF a n --- (immediate) (compile time)

At run-time, ENDIF indicates the destination of the forward jump from IF or ELSE. It marks the end of a conditional structure. THEN is a synonym of ENDIF.
At compile-time ENDIF calculates the forward jump offset from a to HERE and store it at a. The number n is used for syntax checking.

ERASE a n ---

It erases n memory location starting from a, filling them with 0x00 'null' characters.

ERROR b --- n1 n2
--- ff

It notifies an error b and resets the system to command prompt. First of all, the user variable WARNING is examined.
If WARNING is 0 then the offending word is printed followd by a "?" character and a short message "MSG#n".
If WARNING is 1, instead of the short message, the text available on line b of block 4 (of drive 0) is displayed. Such a number can be positive or negative and lay beyond block 4.
If WARNING is –1 then ABORT is executed, which resets the system to command prompt. The user can (with care) modify this behavior of that by altering (ABORT).
If BLK is non zero, then ERROR leaves on the stack n1 that is the value of IN and n2 that is the value of BLK at the error moment. These numbers can then be used by WHERE to determine and show the exact error position.
In any case, the final action is QUIT .
If BLK is zero, then only a ff is left on TOS.

EXECUTE cfa ---

It executes the word which cfa is held on TOS.

EXP --- a

User variable that holds the exponent in a floating-point conversion.

EXPECT a n ---

It transfers characters from the input terminal to the address a for n location or until receiving a 0x13 "CR" character. A 0x00 "null" character is added in the following location. The actual length of the received string is kept in SPAN user variable. See also ACCEPT.

FAR ha --- a

This definition converts an heap-pointer ha into an offset a (at E000h) and perform the correct 8K paging on MMU7.

FENCE --- a

User variable that holds the (minimum) address to where FORGET can act.

FILL a n b ---
It fills `n` memory location starting from address `a` with the value of `b`.

FIRST --- a
User variable that holds the address of the first buffer. See also `LIMIT`.

FLD --- a
User variable that holds the width of output field.

FLUSH ---
It executes `SAVE-BUFFERS`. It saves to disk the buffers marked "modified" by `UPDATE`.

FORGET ---
Used in the form

```
FORGET cccc
```

removes from the dictionary the word `cccc` and all the preceding definitions. Care must be put when more than one vocabulary is involved.

FORTH --- (immediate)
This is the name of the first vocabulary. Executing `FORTH` sets this to be the CONTEXT vocabulary. As soon as no new vocabulary is defined, all new colon definitions became part of `FORTH` vocabulary. `FORTH` is immediate, so it is executed during the creation of a colon definition to select the needed vocabulary. See also `ASSEMBLER` (optional vocabulary).

HERE --- a
It leaves the address of next location available on the dictionary.

HEX --- a
It changes the base to hexadecimal, setting `BASE` to 16.

HLD --- a
User variable that holds the address of last character used in a numeric conversion output.

HOLD c ---
Used between `<#` and `#>` to put a ASCII character during a numeric format.

I --- n
Used between `DO` and `LOOP` (or DO and `+LOOP`, `?DO` and `LOOP`, `?DO` and `+LOOP`) to put on TOS the current value of the loop index.

ID. **nfa** **---**

It prints the definition name whose `nfa` is on TOS.

IF **f** **---** **(immediate)** **(run time)**
--- **a** **n** **(compile time)**

Used in colon definition in the form

```
IF ...  ENDIF
IF ... ELSE ... ENDIF
```

At run-time `IF` selects which words sequence to execute based on the flag on TOS:
If f is true, the execution continues with the instruction that follows IF ("true" part).
If f is false, the execution continues after the `ELSE` ("false" part).
At the end of the two parts, the executions always continues after `ENDIF`.
`ELSE` and its "false" part are optional and if omitted no "false part" will be executed and execution continues after ENDIF.
At compile time IF compiles `0BRANCH` reserving a cell for an offset to the point after the corresponding ELSE or ENDIF .
The integer `n` is used for syntax checking.

IMMEDIATE **---**

It marks the latest defined word such that at compile-time it is always executed instead of being compiled. The bit 6 of the length byte of the definition is set. This allows such definitions to handle complex compilation situation instead of burdening the main compiler.
The user can force the compilation of an immediate definition prepending a `[COMPILE]` to it.

IN **---** **a**

User variable that keeps track of text position within an input buffer. `WORD` uses and modifies the value of `IN` that is incremented when consuming input buffer.

INDEX **n1** **n2** **---**

It prints the first line of screen between `n1` and `n2`. Handy to quick check the content of a series of screens.

INKEY **---** **b**

It reads the next character available from current stream and previously selected with `SELECT` leving it on TOS. It is the opposite of `EMITC`.

INTERPRET **---**

This is the text interpreter. It executes or compiles, depending on the value of `STATE`, text from input buffer a word at a time. It first searches on `CONTEXT` and `CURRENT` vocabularies; if these fail, the text is interpreted as a numeric value, converted using the current `BASE`, and put on TOS. If that numeric conversion fails too, an error is notified with the symbol "?" followed by the word that caused the error. `INTERPRET` executes `NUMBER` and the presence of a decimal point "." indicates that the number is assumed as double integer instead of a simple integer.
After execution of the word found, the control is given back to the caller procedure.

KEY **---** **b**

It shows a (flashing) cursor on current video position and waits for a keypress. It leaves the ASCII code b of the character read from keyboard without printing it to video. In this implementation some SYMBOL-SHIFT key combinations are

decoded as follow:

E2	STOP	→	7E	~
C3	NOT	→	7C	|
CD	STEP	→	5C	\
CC	TO	→	7B	{
CB	THEN	→	7D	}
C6	AND	→	5B	[
C5	OR	→	5D	]
AC	AT	→	7F	©
C7	<=	→	20	space
C8	>=	→	20	space
C9	<>	→	06	as CAPS-SHIFT + 2 and toggles CAPS-SHIFT On and Off,

L/SCR --- n

Constant that indicates the number of lines per Screen. In this implementation is 16.

LATEST --- nfa

It leaves the `nfa` of the latest word defined in CURRENT vocabulary.

LEAVE ---

It forces the conclusion of a `DO ... LOOP` setting the limit at the current index I, inducing an exit at the first occasion. The index remains unaltered and the execution continues normally up to the following `LOOP` or `+LOOP`.

LFA pfa --- lfa

It converts a pfa in its lfa. See also `CFA`, `NFA`, `PFA`, `>BODY`, `<NAME`.

LIMIT --- a

User variable that points to the first location above the last buffer. Normally it is the top of RAM, but not always. In this implementation, it can be set at E000h to allow MMU7 as a general purpose 8K RAM bank. See also: `FIRST`.

LIST n ---

It prints screen number n. Sets `SCR` to `n`.

LIT --- n

It puts on TOS the value hold in the following location. It is automatically compiled a before each literal number.

LITERAL n --- n (immediate) (run time)
n --- (compile time)

At compile-time, `LITERAL` compiles `LIT` followed by the value `n` in the following cell. This is an immediate word and, a colon definition, it will be executed.
It is used in the form

```
: cccc ... [ calculations ] LITERAL ... ;
```

the compilation is suspended during the calculations and, when compilation resumes, `LITERAL` compiles the value put

on TOS during the previous calculations.

LOAD **n** **---**

It starts interpretation of screen `n`. The loading phase ends at the end of the screen or at the first occurrence of `;S`.
If `n` is negative, instead of loading from Screen# `n`, it loads text directly from Basic's OPEN# `n` stream using `ACCEPT-`.
See also `-->`

LOAD- **n** **---**

It starts interpretation of screen `n`. The loading phase ends at the end of the screen or at the first occurrence of `;S`.
See also `-->`

LOOP **a n ---** **(immediate)** **(run time)**
n --- **(compile time)**

Used in colon defintion in the form

```
DO ... LOOP
?DO ... LOOP
```

At run-time `LOOP` checks the jump to the corresponding DO. The index is incremented and the total compared with the limit; the jump back happens while the index is less than the limit. Otherwise the execution leaves the loop. On loop leaving, the parameters are discarded and the execution continues with the following word.
At compile-time `LOOP` compiles `(LOOP)` and the jump is calculated from `HERE` to a which is the address left by `DO` on the stack. The value `n2` is used internally for syntax checking.

LP **---** **a**

User variable for printer purposed. Not used.

LSHIFT **n1 u** **---** **n2**

Shifts left an integer `n1` by `u` bit.

M* **n1 n2** **---** **d**

Mixed operation. It leaves the product of `n1` and `n2` ad a double integer.

M/ **d n1** **---** **n2 n3**

Mixed operation. It leaves the remainder `n2` and the quotient `n3` of the integer division of a double integer `d` by the divisor `n1`. The sign of the reminder is the same as d.

M/MOD **ud1 u1** **---** **u2 ud3**

Mixed operation. Leaves the remainder `u2` and the quotient `ud3` of the unsigned integer division of a double integer d by the divisor `n1`.

MARKER **---** **(immediate)** **(run time)**

Used outside from a colon defintion in the form

```
MARKER cccc
```

this creates a new definition cccc that once executed restores the dictionary to the status before cccc created. This removes cccc and all subsequent definitions. This word allows forgetting across vocabularies

MAX n1 n2 --- n3
It leaves the maximum between n1 and n2.

MESSAGE n ---
It prints to the current device the error message identified by n. If WARNING is zero, a short MSG#n is printed. If WARNING is non zero 1, line n of screen 4 (of drive 0) is displayed. Such a number can be positive or negative and lay beyond block 4. See also ERROR.

MIN n1 n2 --- n3
It leaves the minimum between n1 and n2.

MINUS n1 --- n2
It changes the sing of n1

MMU7! n ---
This word accepts n between 0 and 223 and map the corresponding 8K-page at E000-FFFh addresses. It is coded in Assembler and uses NEXTREG A,n Next's peculiar op-code (ED 92). See MMU7@.

MMU7@ --- n
This word returns a number n between 0 and 223 by asking the hardware which 8K-page is currently fitted in MMU7. See MMU7!.

MOD n1 n2 --- n3
It divides n1 by n2 and leaves the remainder n3. The sign is the same as n1.

NFA pfa --- nfa
It converts a word's pfa into its nfa. See also CFA, LFA, PFA, >BODY, <NAME.

NIP n1 n2 --- n2
It removes the second element from TOS. See also: OVER, DROP, TUCK, SWAP, DUP, ROT.

NMODE --- a
User variable that indicates how double numbers are interpreted. During the input, numbers can be read as double integers or as floating-point numbers. This variable is modified by the optional words INTEGER that sets it to 0 and FLOATING that sets it to 1.

NOOP **---**

This token does nothing. Useful as a placeholder or to prevent crashes in INTERPRET.

NUMBER **a** **---** **d**
a **---** **fp** **(compile time)**

It converts a counted string at address a with a in a double number. If NMODE is 0, the string is converted to double integer. Position of the last decimal point encountered is kept in DPL.
If NMODE is 1, a floating-point number conversion is tried.
If no conversion can be done, and error #0 is raised.

NXTDRV **n1** **---** **n2**

Takes STRM to serve to NextZXOS call. See also DOSCALL.

NXTRD **a** **n** **---**

Variable dedicated to NextZXOS. It calls DOS_READ NextZXOS / +3e API.
See also DOSCALL.

NXTSTP **n** **---**

Variable dedicated to NextZXOS. It sets position on blocks-file calling DOS_SET_POSITION NextZXOS / +3e API.
See also DOSCALL.

NXTWR **a** **n** **---**

Variable dedicated to NextZXOS. It calls DOS_WRITE +3e API. See also DOSCALL.

OFFSET **---** **a**

User variable that states the beginning of “blocks area”. The content of OFFSET is added by BLOCK to the number on TOS to determine the right offset to read from file open to “!Blocks.bin”. Messages issued by MESSAGE are independent from OFFSET.

OR **n1** **n2** **---** **n3**

It executes an OR binary operation between the two integers. The operation is performed bit by bit.

OUT **---** **a**

User variable incremented by EMIT. The user can examine and alter OUT to control the video formatting.

OVER **n1** **n2** **---** **n1** **n2** **n1**

It copies the second number from TOS and put it on the top. See also DROP, NIP, TUCK, SWAP, DUP, ROT.

P! **u** **b** **---**

It sends to port u a byte b. Note: u is a 16 bit port address and an OUT (C) op-code is internally executed.

P@ n --- b

It accepts the byte `b` from port `u`. Note: u is a 16 bit port address and an IN(C) op-code is internally executed.

PAD ---

It leaves on TOS the address of text output buffer. It is at a fixed distance of 68 byte over `HERE`.

PFA nfa --- pfa

It converts a word's `nfa` to its `pfa`. See also `CFA`, `LFA`, `NFA`, `>BODY`, `<NAME`.

PLACE --- a

User variable that holds the number of places after the decimal point to be shown during a numeric output conversion. See also `PLACES`.

PREV --- a

User variable that points to the last referred buffer. `UPDATE` marks that buffer so that it is later written to disk.

QUERY ---

It awaits from terminal up to 80 characters or until a `CR` is received. The text is stored in `TIB`. User variable `IN` is set to zero.

QUIT ---

It clears the Return-Stack, stops any compilations and return the control to the operator terminal. No message is issued.

R --- n

It copies to TOS the value on top of Return Stack without alter it.

R# --- a

User variable that holds the position of the editing cursor or other function relative to files.

R/W a n f ---

Standard FIG-FORTH read-write facility. Address `a` specifies the buffer used as source or destination; `n` is the sequential number of the block; f is a flag, 0 to Write, 1 to Read. `R/W` determines the location on mass storage, performs the transfer and error checking.

R0 --- a

User variable that holds the initial value of the Return Stack Pointer. See also `RP!` and `RP@`.

R> **---** **n**

It removes the top value from Return Stack and put it on TOS. See also >R, R and RP!.

RECURSE **---**

Used only at compile-time inside a colon-definitions, It compiles the word being created to put in place a recursion call.

REG! **b** **n** **---**

Used only at compile-time inside a colon-definitions, It compiles the word being created to put in place a recursion call.

REG@ **n** **---** **b**

Used only at compile-time inside a colon-definitions, It compiles the word being created to put in place a recursion call.

REG-DATA **---** **c**

Constant 253Bh for the Register-Data port.

REG-SELET **---** **c**

Constant 243Bh for the Register-Select port.

RENAME **---**

Used in the form:

```
RENAME cccc xxxx
```

Searches the word cccc in the CONTEXT vocabulary and changes its name to xxxx. The two word-names cccc and xxxx must have the same length.

REPEAT **a1 n1 a2 n2** **---** **(immediate)** **(compile time)**
--- **(run time)**

Used in colon defintion in the form:

```
BEGIN ... WHILE ... REPEAT
```

At run-time REPEAT does an inconditional jumt to the corresponding BEGIN.
At compile-time REPEAT compiles BRANCH and the offset from HERE to a1 and resolves the offset from a1 to the location after the loop; n1 and n1 are used for sysntax check.

ROT **n1 n2 n3** **---** **n2 n3 n1**

It rotates the three top integers, taking the third an putting it on top. The other two integer are pushed down from top by one place. See also OVER, DROP, NIP, TUCK, SWAP, DUP.

RP! **a** **---**

System procedure to initialize the Return Stack Pointer to the value passed on TOS that should be the address held in R0 user variable.

RP@ **---** **a**

It leaves the current value of Return Stack Pointer.

RSHIFT **n1 u** **---** **n2**

It shifts right an integer `n1` by `u` bit.

S->D **n** **---** **d**

It converts a 16 bit integer into a 32 bit double integer, sign is preserved.

S0 **---** **a**

User variable that holds the initial value of che SP register. See also: `SP!` and `SP@`.

SCR **---** **a**

User variable that hold the number of the last screen retrieved with `LIST`.

SELECT **n** **---**

It selects the current channel. As usual for ZX Spectrum, `n` is 0 and 1 for lower part of screen, 2 for the upper part, 3 for printer, 4 for "!Blocks.bin" stream. Note: `KEY` always select chanle 2 to display the (flashing) cursor.

SIGN **n d** **---** **n**

If `n` is negative, it puts an ASCII "-" at the beginning of the numeric string converted in the text buffer. Then, `n` is discarded while d is kept. Used between `<#` and `#>`.

SMUDGE **---**

Used by the creation word **:** during the definition of a new word; it toggles the smudge-bit of the first byte in the nfa of the `LATEST` defined word. When a word's smudge-bit is set, it prevents the compiler to find it. This is typical for uncomplete or not correctly defined words.
It is also used to remove malformed incomplete words via

```
SMUDGE FORGET cccc
```

SP! **a** **---**

System procedure to initialize the SP register to the address a that should be the address hold in `S0` user variable.

SP@ **---** **a**

It returns the content of SP register before `SP@` was executed.

SPACE **---**

It sends a space to the current output peripheal, usually the video. See also `SELECT`.

SPACES **n** **---**

It sends n spaces.

SPAN **---** **a**

User variable that holds the number of characters got from the last EXPECT.

STATE **---** **a**

User variable that holds the compilator status. A non-zero value indicates a compilation in progress.

STRM **---** **a**

Variable containing the stream number used by the Screens/Blocks facility. Used by NextZXOS calls.
See also NXTDRV, NXTSTP, NXTRD, NXTWR.

SWAP **n1 n2** **---** **n2 n1**

It swaps the two top element at the TOS. See also OVER, DROP, NIP, TUCK, DUP, ROT.

THEN **a n** **---** **(immediate)**
--- **(compile time)**

Synonym of ENDIF.

TIB **---** **a**

User variable that holds the address of the Terminal Input Buffer.

TO **n** **---**

Used in the form:

```
TO cccc
```

It ssigns the value n to the variable cccc previously defined via VALUE.

TOGGLE **a b ---**

The byte at location address a is XOR-ed with the model b.

TRAVERSE **a1 n** **---** **a2**

It spans through the name-field of a definition depending on the value of n.
If n = 1, then a1 must be the beginning of the name-field, i.e. nfa itself; a2 is the address of the last byte of the name field.
If n = –1, then a1 must be the last byte of name-field and a2 will be the nfa.
Used by da NFA and PFA.

TUCK **n1 n2** **---** **n2 n1 n2**

It takes the top element of calculator stack and copies after the second. See also OVER, DROP, NIP, SWAP, DUP, ROT.

TYPE **a n** **---**

It sends to the current output peripheal `n` characters starting from address `a`.

U. **u** **---**

It prints an unsigned integer followed by a space.

U< **u1 u2** **--- f**

It leaves a `tf` if `u1` is less than `u2`, a `ff` otherwise.

UM* **u1 u2** **--- ud**

Unsigned product of the two integers `u1` and `u2`. The result is a double integer.

UM/MOD **ud u1** **--- u2 u3**

It leaves the quotient `u3` and the reminder `u2` of the integer division of `ud` / `u1`.

UNTIL **a n --- (immediate) (compile time)**
f --- (run time)

Used in colon definition in the forms

```
BEGIN ... UNTIL
```

At run-time `UNTIL` controls a conditional jump to the corresponding `BEGIN` when `f` is false; the exit from the loop happens if `f` is true.
At compile-time `UNTIL` compiles `0BRANCH` and an offset from `HERE` to a; n is used for syntax checking.

UPDATE **---**

It marks as modified the most recent used buffer, the one pointed by `PREV`. The block contained in the buffer will be transferred to disk when that buffer is requested for another block.

UPPER **c1** **--- c2**

This word converts a character to upper-case. If `c1` is not between “a” and “z”, then `c1` is left unchanged.

USE **--- a**

User variable that holds the buffer address of the block to be read from disk or that has just been written to.

USER **n** **---**

Defining word used in the form

```
n USER cccc
```

creates an user variable `'cccc'`. The first byte of pfa of `cccc` is a fixed offset for the User Pointer, that is the pointer for the user area. In this implementation there is only one User Area and a fixed User Pointer.
When `cccc` is later executed, it put on TOS the sum of offset and User Pointer, sum to be used ad the address for that

specific user variable. The user variable are: TIB, WIDTH, WARNING, FENCE, DP, VOC-LINK, FIRST, LIMIT, EXP, NMODE, BLK, IN, OUT, SCR, OFFSET, CONTEXT, CURRENT, STATE, BASE, DPL, FLD, CSP, R#, HLD, USE, PREV, LP, PLACE, DL.

VALUE **n** **---**

Defining word used in the form:

```
n VALUE cccc
```

Creates the word cccc that acts as a variable. To store a value in such a variable you have to use TO.
When cccc is later executed it directly returns the value of the variable without the need to access its address using @.

VARIABLE **n** **---**

Defining word used in the form:

```
n VARIABLE cccc
```

creates the word cccc with the pfa containing the initial value n. When cccc is executed, it puts on TOS the pfa of cccc that is the address that holds the value n.
When used in the form

```
cccc @
```

the content of the variable cccc is left on TOS.
When used in the form

```
n cccc !
```

the value on TOS is stored to the variable cccc.

VIDEO **---**

It sets DEV\ICE to 2 select the video as current output peripheral.

VOC-LINK **---** **a**

User variable that holds the address of a field in the definition of the last vocabulary. Each vocabulary is part of a linked-list that use that field as pointer-chain.

VOCABULARY **---**

Defining word used in the form

```
VOCABULARY cccc
```

creates the word cccc that gives the name of a new vocabulary.
Later execution of

```
cccc
```

makes such vocabulary the CONTEXT vocabulary, so that it is possible to search for words defined in this vocabulary first and execute them.
Used in the form

```
cccc DEFINITIONS
```

makes such vocabulary the CURRENT vocabulary, so that it is possible to insert new definitions in it.

WARM **---**

It executes a warm system restart. Executes EMPTY-BUFFERS and ABORT.

WARNING **---** **a**

User variable that determines the way an error message is reported. If zero, only a short “MSG#n” is reported. If non zero, a long message is reported. See also ERROR.

WHILE **f** **---** **(immediate)** **(run time)**
a n --- **a1 n1 a2 n2** **(compile time)**

Used in colon defintion in the form:

```
BEGIN ... WHILE ... REPEAT
```

At run-time WHILE does a conditional execution based on f. If f is true, the execution continues to a REPEAT which will jump to the corresponding BEGIN. If f is false, the execution continues after the REPEAT quitting the loop.
At compile-time WHILE compiles 0BRANCH leaving a2 for the offset; a2 will be comsumed by a REPEAT. The address a1 and the number n1 was left by a BEGIN.

WIDTH **---** **a**

User variable that indicates the maximum number of significant characters of the words during compilation of a definition. It must be between 1 and 31.

WORD **c** **---** **a**

It reads characters from the current input stream up to a delimiter c and stores such string at HERE that is left on TOS. WORD leaves, as the first byte, the length of the string and ends everything with at least two spaces. Further occurrences of c will be ignored.
If BLK is zero, the text is taken from the terminal input buffer TIB. Otherwise the text is taken from the disk block held in BLK. User variable IN is added with the number of character read, the number ENCLOSE return.

WORDS **---**

It lists the words of CONTEXT vocabulary. Pressing Break stops.

X **---**

It show the splash screen.

XOR **n1 n2** **---** **n3**

It executes a XOR binary operation between the two integers. The operation is performed bit by bit.

[**---** **(immediate)**

Used in colon defintion in the form:

```
: cccc [ ... ] ... ;
```

it suspends compilation. The words that follows [will be executed instead of being compiled. This allows to perform some calculations or start other compilers before resuming the original compilation with]. See also LITERAL.

[CHAR] **---** **(immediate)** **(compile time)**

It is the same as the sequence [CHAR c] LITERAL.
It is used in colon defintion in the form:

```
: cccc ... [CHAR] c  ... ;
```

At compile time, [CHAR] compiles LIT and the numeric value of ASCII character c in the following cell.

[COMPILE] --- (immediate)

Used in colon defintion in the form:

```
: cccc ... [COMPILE] wwww ... ;
```

[COMPILE] forces the compilation of a definition wwww that is immediate. Normally immediate words aren't compiled but executed and to compile an immediate word it is not possible to use the sequence COMPILE wwww but it is necessary to use the sequence [COMPILE] wwww.

\ ---

Used in the from:

```
\ ...
```

Any character that follow \ until the end of line are treated as a comment.

] ---

It resumes the compilation suspended by [so it is possible to complete the definition.

Line Editor

The following definitions are available after you give `10 LOAD`.

Line Editor is formed by a dozen words that can operate on a single line of a given Screen and helps inspect things around.

An edit session normally starts with a `LIST` on the desired Screen, this sets `SCR` user variable to the passed Screen number. `LIST` is a word already available in the "core" dictionary. To clear a Screen I foreseen a `BCLEAR` word, but I left it commented in Screen# 13 for now, deeming it too dangerous for my tastes; instead I usually use `BCOPY` from an actually empty Screen.

The word `FLUSH` flushes to disk any modification you've done on any Screen. Beware, a Screen is re-written to disk as soon as the `BUFFERS` containing it are modified. To save space, this implementation has only three `BUFFERS`.

`EMPTY-BUFFERS` is another vital word: it empties all buffers. It is very useful if you mistakenly overwrite or spoil a Screen during an edit operation, with it, you have the chance to "rollback" the things before the anything is written to disk.

To write a line from scratch or to overwrite line, you can use `P` to "put" the following text to the given line on current screen. For example:

```
1000 LIST
0 P \ One thousand screens
L
```

This sequence selects Screen#1000 and put a text "One thousand screens" on the first line of it. The word `L` repeat the `LIST` of current screen.

To move or copy a line around, you can use `H` to "hold in `PAD`" a given line on current screen, you can change Screen if you wish, then you can complete this **copy-and-paste** operation with `INS` to "insert" or `RE` to "replace" the line you copied in advance with H. None of above words, but `H`, modify PAD content, so you can repeat the operation. There is also a way to **cut-and-paste** a line using `D` to "delete and copy to `PAD`" instead of `H`.

See also `BLOCK`, `INDEX`, `L/SCR`, `LIST`, `LOAD`, `MESSAGE`, `PAD`, `SCR`, `STRM.`, `TIB`.

This is a quick reference of involved memory areas and words that work on them.

Text Input Buffer (keyboard) TIB	Parsing Operation	 PAD	Edit Operations	One BLOCK	Blanking Operations
	TEXT →		← H RE →		← E
			← D INS →		← S
			P →		

-MOVE a n ---

“Line move”. It moves a line, C/L bytes length, from address a to the line n of current screen, then it does an UPDATE. Current screen is the one kept by SCR.

.PAD ---

“Show PAD”. It prints the current PAD content.

B ---

“Back” one Screen. This word set to previous Screen by decreasing SCR and prints it using LIST.

D n ---

“Delete” a row. It deletes line n of current screen (the one indicated by SCR), the following lines are moved up and the last one will be blanked. D executes H so that it can be followed by an INS to perform a line move.

BCOPY n1 n2 ---

“Block-Copy” utility that copies Screen n1 to Screen n2. SCR will contain n2.

E n ---

“Erase” a row. This word fills line n with spaces. It does UPDATE.

H n ---

“Hold” a row in PAD. This word put line n of current Screen to PAD without altering the block on disk. Current Screen is the one kept in SCR.

INS n ---

“Insert” from PAD. This word inserts line n using text in PAD. The original line n and the following ones are moved down and the last is lost.

INVV ---

“Inverse video”. It enables Inverse-Video attribute mode. See also TRUV.

L ---

“List” current Screen. This word does SCR @ LIST.

LINE n --- a

It leave the address a of line n of current screen, the one kept in SCR. Such a screen is currently held in a buffer.

N **---**

"Next" Screen. This word sets to next Screen by increasing SCR and prints it using LIST.

P **n** **---**

"Put" a line. This word accepts the following text (delimited by a tilde character ~) as the text of line n of current Screen. Text is taken from TIB and sent to the current Screen

RE **n** **---**

"Replace". This word takes text currently in PAD and put it to line n.

S **n** **---**

"Space" one row. This word frees line n moving the following lines down by one. The last line is lost

SAVE **---**

It does UPDATE and FLUSH saving this Screen and all previously modified Screens back to disk.

ROOM **---**

This word shows the room available in the dictionary, that is the difference between SP@ and PAD addresses.

TEXT **c** **---**

This word accepts the following text and stores it to PAD. c is a text delimiter. TEXT does not go beyond a 0x00 [null] ASCII.

TRUV **---**

"True video". It disables Inverse-Video attribute mode. See also INVV.
Questa definizione è disponibile solo dopo il caricamento del "Line Editor" tramite 10 LOAD.

UNUSED **---** **n**

It returns the number of byte available in dictionary.

WHERE **n1** **n2** **---**

Usually executed after an error has been reported during a LOAD session. Maybe, this word should be included in "core" dictionary. n1 is the value of IN and n2 the value of BLK as were left by ERROR.
WHERE shows on screen the block number, the line number, the very same line highlighting in "inverse video" the word that caused the error.

Case -Of structure

The following definitions are available after you gave `17 LOAD`.

`CASE` **`n0`** **`---`** **`(immediate)`** **`(run time)`**
`---` **`a n`** **`(compile time)`**

Used in colon definition in the form

```
n0 CASE
    n1  OF  ... ENDOF
       ...
    nz  OF  ... ENDOF
    ... ( else )
ENDCASE
```

The word `CASE` marks the beginning of Case-Of structure i.e. a set of branches where only one is performed based on the value of `n0`. If none of the "OF clause" values matches, the `ELSE` part is performed.
At compile time `CASE` leaves previous `CSP` address `a` and a number `n` for syntax checking.
`CASE` has to be balanced by a corresponding `ENDCASE`.

`OF` **`n0 nk`** **`---`** **`(immediate)`** **`(run time)`**
`n1` **`---`** **`a n2`** **`(compile time)`**

This word is used in colon-definition within a Case-Of structure.
At run-time it compares the value now on TOS `nk` with the value `n0` that was on TOS just before the beginning of the Case-Of structure.
At compile-time, it compiles `(OF)` and `0BRANCH` using `n1` and `n2` for syntax checking and leaving `a` to be used by `ENDCASE` to resolve `0BRANCH`.
See also `CASE`.

`ENDOF` **`---`** **`(immediate)`** **`(run time)`**
`a1 n1` **`---`** **`a n2`** **`(compile time)`**

This word ends an "Of-EndOf" clause started with `OF`.
At compile-time it acts like a `THEN`, first compiling a BRANCH that will be resolved by `ENDCASE` to skip any subsequent "Of-End-Of" clauses and resolving the `0BRANCH` compiled by the corresponding previous `OF` to continue the Case-Of structure.
See also `CASE`.

`ENDCASE` **`---`** **`(immediate)`** **`(run time)`**
`a a1 … az` **`---`** **`(compile time)`**

This word ends a Case-Of structure started with `CASE`.
At compile-time it compiles a `DROP` to discard the value `n0` put on TOS before `CASE` and resolves all OF-ENDOF clauses to jump after the `ENDCASE`. Finally, it restores previous content of `CSP`.
See also `CASE`.

`(OF)` **`n0 nk`** **`---`** **`(run time)`**

This word represents the run-time semantic compiled by `OF` word. At run-time, it compares the value now on TOS `nk`

with the value `n0` that was on TOS just before the beginning of the Case-Of structure and leave a flag to be used by the following 0BRANCH (that was compiled by `OF`). When `n0` equals `nk`, the definitions between `OF` and `ENDOF` will be executed, otherwise a jump to the word after `ENDOF` is performed.

Debugger facility

The following definitions are available after Line Editor loading via `20 LOAD`.

`.S` **`---`**

Prints the current status of the Calculator Stack. For example, supposing to start with an empty stack,

```
0 1 2 3 .S
```

will print

```
0 1 2 3 ok
```

`.WORD` **`a`** **`---`**

Given a CFA, this word prints the ID. It is used by SEE to perform some word "de-compile"

`DEPTH` **`---`** **`n`**

It leaves the depth of the Calculator Stack before it was executed. For example, supposing to start with an empty stack,

```
0 1 2 DEPTH .
```

will print

```
3 ok
```

`DUMP` **`a`** **`---`**

Performs a "dump" of a memory area from address `a` for 128 bytes or until BREAK is pressed.
Visualization is always in hexadecimal, current base is maintained. For example:

```
448 DUMP
```

will print the Standard ROM content starting from address 448 (01C0h):

```
01C0  4C 49 53 D4  4C 45 D4 50  LISTLETP
01C8  41 55 53 C5  4E 45 58 D4  AUSENEXT
01D0  50 4F 4B C5  50 52 49 4E  POKEPRIN
01D8  D4 50 4C 4F  D4 52 55 CE  TPLOTRUN
01E0  53 41 56 C5  52 41 4E 44  SAVERAND
01E8  4F 4D 49 5A  C5 49 C6 43  OMIZEIFC
01F0  4C D3 44 52  41 D7 43 4C  LSDRAWCL
01F8  45 41 D2 52  45 54 55 52  EARRETUR
```

`SEE` **`---`** **`n`**

Used in the form

```
SEE cccc
```

it will print how the word `cccc` is defined along with NFA, CFA, PFA data. If `cccc` is a "colon-definition" the result will show

For example, the word **`TYPE`** is a "colon-definition" and is defined as follow:

```
: type   ( a n -- )
    over + swap
    ?Do
        i c@ emit
    Loop
```

;

If you give

```
SEE TYPE
```

it will print

```
Nfa: 7181 84
Lfa: 7186 COUNT
Cfa: 7188 6BB0
OVER + SWAP (?DO) 12 I C@ EMIT (LOOP) -8
```

The first line shows **TYPE**'s CFA (7153h in this case) followed by 84h, the counter byte, that indicates a 4-bytes length word name. The second line is **TYPE**'s LFA (7158h) that is a pointer to COUNT's NFA, that is the previous word defined in the dictionary. The third line is **TYPE**'s CFA: this is a pointer to the ;CODE machine code of : that it's the entry-point of every "colon-definition". The fourth line represents the PFA and is somehow the "decompilation" of the above definition. Literals and offsets are shown in "inverse video" mode.

Another example the word NIP isn't a colon-definition, instead it is coded directly in machine-code as follow:

```
CODE nip  ( n1 n2 -- n2 )
        POP     HL|
        EX(SP)HL
        Next
        C;
```

and if you give

```
SEE NIP
```

it will print

```
Nfa: 69FA 83
Lfa: 69FE DROP
Cfa: 6A00 6A02
6A02 E1 E3 DD E9  84 54 55 43  ac]I.TUC
6A0A CB FA 69 0F  6A E1 D1 E5  Kzi.jaQe
...
```

In this case, since **NIP** is not a colon-definition, the PFA part is just a **DUMP** you can BREAK at any time.

Again, the first line shows **NIP**'s CFA (69FAh in this case) followed by 83h, the counter byte, that indicates a 3-bytes length word name. The second line is **NIP**'s LFA (69FEh) that is a pointer to DROP's NFA, that is the previous word defined in the dictionary. The third line is **NIP**'s CFA that points to the following cell address 6A00h, that is **NIP**'s PFA where the small piece of machine-code lies. We should be able to see **E1** for POP HL, **E3** for EX (SP),HL and **DDE9** for JP (IX) to "Next" inner interpreter address **6434h**.

The following bytes 84 45 55 43 are the beginning of the next word in dictionary (**TUCK** in this case).

Error messages.

Code	Message
#0	?
#1	Stack is empty.
#2	Dictionary full.
#3	No such line.
#4	has already been defined.
#5	Invalid stream.
#6	No such block.
#7	Stack is full!
#8	Old dictionary is full.
#9	Tape error.
#10	Wrong array index.
#11	Invalid floating point.
#17	Can't be executed.
#18	Can't be compiled.
#19	Syntax error.
#20	Bad definition end.
#21	is a protected word.
#22	Aren't loading now.
#23	Forget across vocabularies.
#24	RS loading error.
#25	Cannot open stream.
#26	Error at postit time.
#27	Inconsistent fixup.
#28	Unexpected fixup/commaer.
#29	Commaer data error.
#30	Commaer wrong order.
#31	Programming error.
#33	Programming error.
#45	NextZXOS pos error.
#46	NextZXOS read error.
#47	NextZXOS write error.