vFORTH 1.7

ZX Spectrum Next version

also
Microdrive and MGT back-port annotations

1990-2024 Matteo Vitturi

...oOo...

Introduction
&
Technical Information

...oOo...

Build 2024-08-15

1 Foreword

This document introduces a Forth implementation suitable to run on *Sinclair ZX Spectrum Next* as a "**dot command**". Back-port to Microdrive or MGT is also summarized.

This is more a technical reference and not an introductory book to Forth: To learn Forth language or for a good reference I suggest the book "Starting FORTH" - Leo Brodie (1981, Forth Inc.). The PDF is hopefully available at Forth Inc web-site. The first edition is old enough to stick on 16-bits integer numbers. In this perspective, almost all Forth source described in Leo's book are available within this system in **Screens** from #800 upward (see §**3.1**).

This is a working and functional piece of software, but there are many things still to do to cope with all the new *Sinclair ZX Spectrum Next*'s functionalities – Repository at https://github.com/mattsteeldue/vforth-next.

In essence, this is a *quasi* standard Forth based on my previous work vForth **version 1.413** available somewhere in the Internet (https://sites.google.com/view/vforth/vforth1413). Among the differences is that vForth uses a single dedicated file on SD to provide **`BLOCK`** / **Screen** storage facility, while other versions used ZX Microdrive cartridges or DISCiPLE disks (https://github.com/mattsteeldue/vforth). See **`USE`** definition.

The latest **version 1.7** splits the dictionary structure in two parts, a *name-space* and a *code-space*, by exploiting a eight 8K RAM pages (32 – 39) fitted at MMU7 providing a more efficient memory usage and much more memory available for actual code. This also means there is no more fear in create *long-name definitions*, since the definition's name uses *heap memory* and not *main memory*. For *heap-memory* capability see § 6.3.

From previous **versions 1.6**, vForth runtime is faster than any previous version, thanks to the idea of dedicating more Z80 registers to keep the internal status of Forth's *virtual machine* and a shorter/faster *Inner-Interpreter*, see § 3.7.1.

Starting from **version 1.52**, the behavior of `VARIABLE` is more *standard* and doesn't need an initial value: this means that some older syntax *à la* White Lightning may not work properly, since it leaves some spurious data on the stack that isn't used by `VARIABLE` anymore.

Starting from version **vForth 1.51**, this Forth comes with two flavors: *Direct-Threaded* or *Indirect-Threaded* code, but version 1.7 is only Direct-Threaded, and offers some more speed at the cost of a little more memory allocation for each colon-definition. See § 3.7.1 for technical detail.

Previous versions are available as zip files at https://github.com/mattsteeldue/vforth-next/tree/master/download/older.

1.1 MIT License

...---...

This very same MIT License can be viewed from within vForth environment via **VIEW LICENSE.MD** provided that **VIEW** is made available via **NEEDS VIEW**.

1.2 Typos and suggestions

I, the author am not a native English speaker and you, very likely, will find grammatical errors. In this case, it would be kindly appreciated if you could drop me a line with any suggestion and/or correction at *matteo* -underscore- *vitturi@yahoo.com.* Or you can often reach me on FB.

1.3 Acknowledgment

Many thanks to Rob Probin for his invaluable suggestions and insights.
Special thanks goes to Roland Herrera who helped to edit one of the first issues of this documentation.
Special mention to Albert van der Horst for his 8080 inspirational Assembler library.

1.4 Document structure

Chapter 2 describes how to install, activate and get acquainted with the Forth environment, briefly exposing various basic options and utilities, such as chosing case-sensitive, editing screens & blocks and feeding your source code to build new definitions. It also describes how to create a standalone executable via **ZAP**.
Chapter 3 lists some useful libraries and utilities you can use while in Forth that (memory is a premium) can be imported in your session using **NEEDS**. Here are some details:
Chapter 3.1: The old fashion Screen/Block facility is a very quick way of coding in Forth so **EDIT** the **Full Screen Editor** is available; to handle *large text files* I've coded the **Large file EDitor** aka LED (§ **3.3**) that is able to handle files as large as about 17.000 rows, 85 characters per row; it's a "work-in-progress" though.
Chapter 3.2 introduces **GRAPHICS** library to manage various **Modes** and **Layers** along with **Color** and **Attributes** definitions, with pixel addressing and line or circle drawing, etc.
Chapter 3.4 introduces the **MOUSE** facility relying upon the **Interrupt Service Routine** library (§ **3.5**): see the demo demo/color-picker.f for a nice example of interrupt-driven mouse cursor movement.
Chapter 3.6 provides an old fashion **BLOCK** oriented **Search** and **Locate** Utility, especially for Microdrive/MGT.
Chapter 3.7 explains the inner parts of Forth introducing the **SEE** Debugger Utility.
Chapter 3.8 shows how to exploit the Standard-ROM **FLOATING point** calculator.
Chapter 3.9 shows the obsolete **Line Oriented Editor** that's the foundation for the aforementioned **Full Screen Editor**.
Chapter 3.10 introduces the **ASSEMBLER** vocabulary, an alternative way to code Z80. There are a few examples.
Chapter 3.11 explains how to interact with the Raspberry Pi Zero accelerator.
Chapter 4 gives some deeper insight and technical information about registers and number formalism.
Chapter 5 is a straight list of error messages stored in the first few Screens.
Chapter 6 provides detailed information of **Forth Dictionary** where each definition is presented in a formal way:
Chapter 6.1 is the "core" dictionary, a long list where most definitions are available at COLD start,
Chapter 6.2 introduces the optional set of definitions that provides the **Case-Of structure**.
Chapter 6.3 describes the **Heap Memory Facility** to access the large quantity of memory available: from this **Version 1.7** Forth's dictionary itself relies upon this Heap Memory Facility.
Chapter 6.4 introduces the **Testing Suite**, to show that vForth *wants to comply* to modern Standard.
Chapter 6.5 for other minor utilities.
Chapter 7 briefly shows the memory map and pages used.

1.5 Legend

Much effort was put to adhere to the following conventions:

`Courier New`	used whenever a Forth definition is referenced or to indicate some typed-in source code.
Calibri Bold	to highlight something important, like filenames, messages, or key-strokes.
Calibri	is used elsewhere

All definitions explained in the dictionary pages are introduced in the form:

`EXAMPLE      a1 n1 ... ---  a2 n2 ...`

where "**`a1 n1 ...`**" represents the Stack status before **`EXAMPLE`** is executed, and "**`a2 n2 ...`**" represents the Stack status after **`EXAMPLE`** is executed. Special behavior, such as **`IMMEDIATE`** definitions are explained properly.

`a`	memory address	16 bits
`b`	byte, small unsigned integer	8 bits
`c`	character	8 bits, but often only lower 7 are significant.
`d`	signed double precision integer	32 bits
`fh`	file-handle (NextZXOS)	8 bits
`fp`	floating point number (see **§3.8**)	32 bits
`ha`	heap-pointer address (see **§6.3** and **`>FAR`**)	16 bits.
`n`	signed integer	16 bits
`u`	unsigned integer	16 bits
`ud`	unsigned double precision 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	16 bits
`cccc`	character string or definition-name available in the vocabulary	
`...`	a list of definitions	
TOS	top of calculator stack	

.

2 Getting started

2.1 Installation

The most recent version of this software can be downloaded from GitHub repository as .zip file at

https://github.com/mattsteeldue/vforth-next/tree/master/download

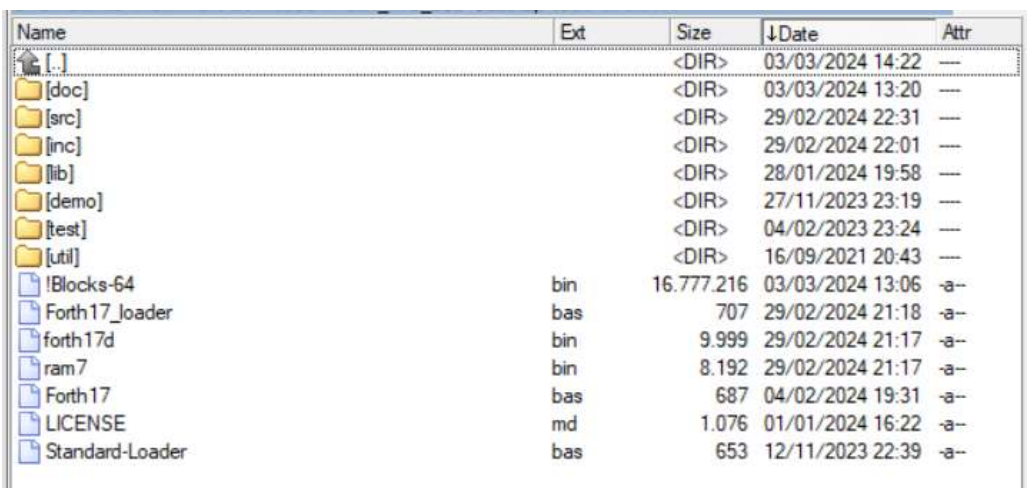
The same executable programs are available in the same repository:

https://github.com/mattsteeldue/vforth-next/tree/master/SD/tools/vforth

The zip file contains two subdirectories **/dot** and **/tools/vForth**.

The **/dot** directory contains the **vforth** *dot-command* that has to be copied to **C:/dot/** directory inside your Next's SD card, the **/tools/vForth** directory contains the standard installation and has to be copied to **C:/tools/vForth** of SD so ithe following sub-directory hierarchy will appear:

doc/ where I keep some help text files and this very same document.
inc/ contains source files of single definitions available via `NEEDS cccc`.
lib/ same as inc/ but these source files are a collection of several definitions that forms a "library utility".
src/ among others, the source file of this Forth System.
test/ contains an adaptation of John Hayes' Test Suite that tries to make this Forth more *standard*.
util/ with some Perl script to manage with !Blocks-64.bin file I collect over the time.
demo/ some useful demos.



Name	Ext	Size	↓Date	Attr
[..]		<DIR>	03/03/2024 14:22	----
[doc]		<DIR>	03/03/2024 13:20	----
[src]		<DIR>	29/02/2024 22:31	----
[inc]		<DIR>	29/02/2024 22:01	----
[lib]		<DIR>	28/01/2024 19:58	----
[demo]		<DIR>	27/11/2023 23:19	----
[test]		<DIR>	04/02/2023 23:24	----
[util]		<DIR>	16/09/2021 20:43	----
!Blocks-64	bin	16.777.216	03/03/2024 13:06	-a--
Forth17_loader	bas	707	29/02/2024 21:18	-a--
forth17d	bin	9.999	29/02/2024 21:17	-a--
ram7	bin	8.192	29/02/2024 21:17	-a--
Forth17	bas	687	04/02/2024 19:31	-a--
LICENSE	md	1.076	01/01/2024 16:22	-a--
Standard-Loader	bas	653	12/11/2023 22:39	-a--

If you wish to use a different directory instead of **C:/tools/vforth**, you would need to modify the paths in the above Basic programs *and* the manually patch the path+filename specified inside the binary files... or recompile the whole thing.

2.1.1 dot-command version

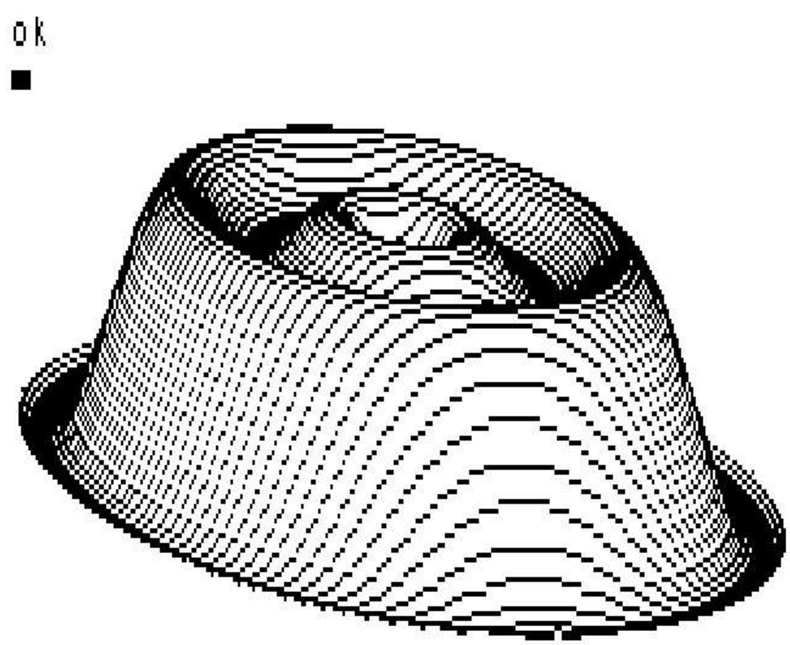
To properly work, the dot-command version relies on the **standard installation part** described in paragraph 2.1: Along with the standard **/tools/vForth** directory, there is a **/dot** directory that contains `vforth` binary dot-command that has to be manually copied to directory **/dot**.

N.B. : Executing **`.vforth`** changes current directory to **/tools/vforth** and screen mode to LAYER 1,2. Screen mode is restored on regular quit to Basic, but the current directory is not restored, maybe in the future I will be able to fix this.

This dot version accepts an optional filename parameter that is taken as the source file to be immediately executed, for example:

```
.vforth demo/fedora.f
```

after a couple of minutes of busy loading, it produces the following result



2.2 Activation / Deactivation and brief tutorial

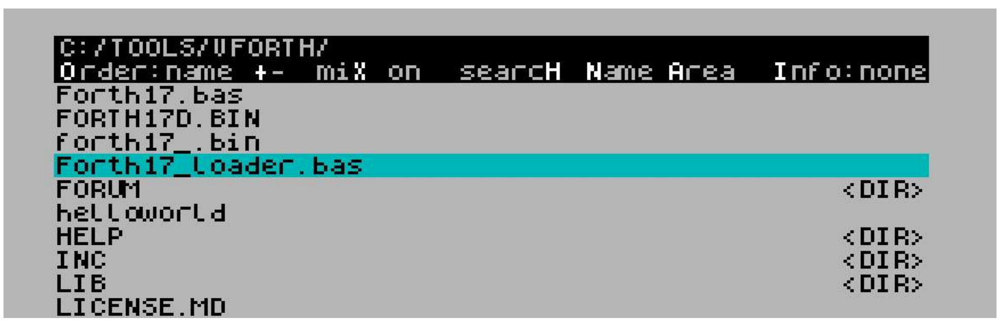
The Forth System can be activated in two ways:

- running dot-command `.vforth` from command line or NextBasic (as seen above)
- running the Basic program `C:/tools/vforth/forth17_loader.bas`.

Both should be equivalent, even though the dot-command loads at address $2000 and has some OS dependencies.

To terminate a Forth session exiting to Basic you can use **BYE**.

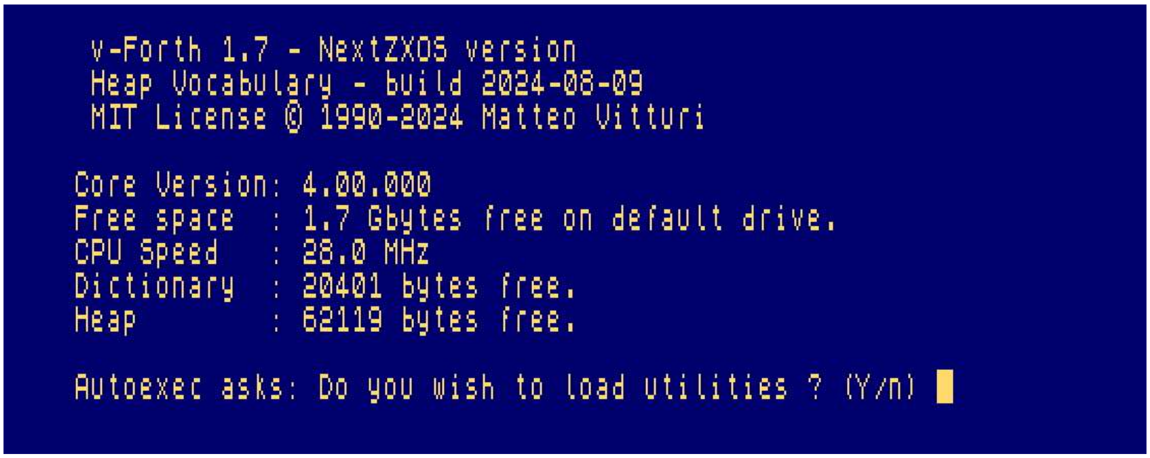
The obvious way can is using the Browser and selecting it, then hitting [ENTER].



The Basic loader **forth17_loader.bas** frees upper memory setting RAMTOP to a very low address 25345 and usually loads **ram7.bin** to BANK 16, then **forth17d.bin** to few bytes above RAMTOP and then it loads a smaller Basic launcher **forth17.bas** that you can customize for your purposes.

Then, a Splash screen displays the Version Number and the Build Date followed by some technical system information that are collected by the Auto-Start sequence of **Screen # 11**: within a few seconds the system will ask if you would like to "**`Run Scr # 11 autoexec`**": the only way to refuse is by using the **[N] key**. However, it's a good idea to allow Forth to continue to LOAD **Screen # 11** which in turn loads a few useful utilities making available, among the other definitions, two particular definitions: **`EDIT`** the "**Screen Editor**" and **`SEE`** the "**Debugger Inspector**". This phase is executed only at *first* startup, but you can run it again using **`AUTOEXEC`**. Dictionary space is a premium, this is why of this choice.

In general, in any source-file available in these directories to be imported, I always put some "print" statement just to show the user what is going on or what is being loaded, this is because often, whenever nothing moving nor blinking on the screen, a person is induced to think the machine just crashed.



The Basic launcher **forth17.bas** usually auto-starts the first time at line `20`, so you usually won't notice, but just in case you `STOP` or the Forth system encounters a ROM Error that forces it to suddenly return to Basic, you have two main choices:

a. type RUN at Basic prompt : This does a **`WARM`** Forth-start, preserving your previous work and buffers status.

b. type RUN 20 at Basic prompt: This does a **`COLD`** start, that should reset all as if you had just loaded from SD card.

In both case, file-handles are not released (and you have to manually close them using **`F_CLOSE`**).

2.3 Case-insensitive and Case-sensitive option

By default, the Forth interpreter is *case-insensitive*, so you can type your commands using lower-case or upper-case or a mix of them with no difference. To enable or disable the case-sensitive interpretation you can use **`CASEON`** and **`CASEOFF`** definitions.

The case-insensitive option applies to the Intepreter's dictionary search only or when using a definition that uses **`(COMPARE)`** primitive such as **`GREP`**. A new definition-name retains the exact case it was coded with.

2.4 Block / Screen system

This Forth System comes with a file named `!Blocks-64.bin` that provides the simplest mass-storage system usually available in old-days Forth systems.

To choose a different file you must the **USE** definition (§ 6.1), for example:

```
USE /tools/filename.bin
```

A **BLOCK** is identified by its *number*, an integer between 1 and 32,767, and can be temporarily kept in RAM in a **BUFFER** identified by the same *number* in a memory area that can be addressed, inspected and (persistently) modified. The dedicated memory area that lies between **FIRST** and **LIMIT**. In this system there are seven buffers.

Using **UPDATE**, the most recent referred **BUFFER** is marked for re-write to file at the moment the system will request the same buffer for a different block or when **FLUSH** is used.

On this Forth system, two **blocks** forms a **Screen** that can be edited using the "Full Screen Editor" (see §**3.1**) utility available after you type:

```
NEEDS EDIT
```

Each **BLOCK** has 512 bytes and each **Screen** is 1 KByte so it can store text in 16 lines and 64 columns . A **BLOCK** can be used as a *virtual-memory* area where you can persistently store anything you like. For instance, you can think a **BLOCK** as an 256-elements *integer array* with persistance.

2.4.1 Block File Format

The file provided as default (`!Blocks-64.bin`) is 16 MBytes long and it is a simple ordered concatenation of all blocks available, so that block # 1 starts at file-offset 0 and ends at file-offset 511, block # 2 starts at offset 512 and ends at offset 1023, and so on. There is no block # 0 and the greatest number available is then 32,767 (16,383 **Screens**).

You can use your own file, but that the first Screens must be reserved for error messages and such things.

In particular, Screens # between 4 and 7 must contain the standard **MESSAGE**, those listed in § 5. If you fail to have them there, the system will become quite foolish, since any **ERROR** message would be something blank or random.

A **Block-number** is always twice the corresponding **Screen-number**, for example **Screen # 11** uses block # **22** and **23**. You may argue that this way **BLOCK** *number* 1 is never accessible, and in fact it is reserved for internal purposes (see **NEEDS**).

2.5 Character visualization size

In this Forth implementation I prefer LAYER 1,2 display mode to allow 64 character per line: this is quite necessary in order to be able to display a whole 1024 characters Screen in a single go.

If you prefer LAYER 1,1 you can add a line 61 in **forth17.bas** wrapper as follow

```
61 LAYER 1,1: PRINT CHR$ 30; CHR$ 4;
```

to switch to LAYER 1,1 and condensed character set. The result is quite poor in my opinion.

You can also change **LAYER** mode using some Layer-related definitions available after **NEEDS GRAPHICS** or **NEEDS LAYERS**.

2.6 Source feeding and output spooling

Before entering Forth, the Basic launcher is allowed to open text-files via OPEN#, for instance

```
OPEN# 13,"o>output.txt"
```

that can be later selected for output in Forth via **13 SELECT** to collect any output you send to this output-channel. To restore sending output to video there is an easy **VIDEO** definition that simply does **2 SELECT**.

Two specific definitions allow you to include (and compile) source from any file i.e. **INCLUDE** and **NEEDS.**
For example,

```
INCLUDE demo/chomp-chomp.f
```

or

```
NEEDS GRAPHICS
```

Moreover, you may edit almost any source text-file using **LED**, the built-in editor available via

```
NEEDS LED
```

See § 3.3 “LED – The Large EDitor” for more details.

2.7 Creation of standalone executable

Available only for non dot-version, the **ZAP** and **ZAP"** definitions (§6.1) allow the creation of an standalone executable of a vForth game or program: The purpose is to create a Basic program that loads (at least) the three binary memory images that contain the current status of the whole vForth system and that can be resumed later. This component is evolving for improvement.

2.8 Numeric literals interpretation

Current **BASE** determines how numbers are displayed or more broadly sent to output, and how they are interpreted from keyboard prompt or during compilation. The definitions **HEX** , **DECIMAL** and **BINARY** respectively change current **BASE** to hexadecimal (base sixteen), decimal (base ten) and binary (base two). The **NUMBER** interpreter accepts an optional prefix “$”, “#” or “%” to temporarily modify the **BASE** for the number being interpreted, “$” for hexadecimal, “#” for decimal, “%” for binary. The side effect is that **$** and **%** alone are somehow equivalent to zero as long as they aren't defined by you; that's not the case of **#** since it is a definition on its own. Octal (base eight) hasn't a prefix, but see **OCTAL** definition.

The “unary-sign” must be the first character interpreted, even before any base-prefix “$”, “#” or “%”.

Double-precision Integer numbers interpretation accepts any of the following five punctuation marks: `,  .  /  -  :`

For example:

```
120,000     23:59:59     1/23/45     3.14159
```

are all legit double-precision integer.

2.9 Back-porting to Microdrive and MGT

Version 1.6 – that is quite the same as version 1.7, but without **HEAP** facility – has been back-ported to **Microdrive** and

MGT suitable to run in 48K or 128K real hardware, but that can be used within an emulator such as Fuse Emulator (see the following repository for any reference: https://github.com/mattsteeldue/vforth).

Due to obvious hardware limitations, there is no **NEEDS** definition to import semantics, instead you have to use **LOAD** from the appropriate Block, and this is why the first 150 Blocks or so are reserved to keep the many utilities loaded during **AUTOEXEC** (see Screen# 13) such as Line-Editor, Full-screen Editor, SEE Debugger, Case-Of structure, Search utility, Graphics, Interrupt Handler, Assembler vocabulary.

For example, you have to type **600 LOAD** instead of **INCLUDE demo/chomp-chomp.f**

To run this Forth system within an emulator, you have to pick one that supports Microdrive and/or DISCiPLE disk drive, such as Fuse (https://fuse-emulator.sourceforge.net) that works well under Windows and Linux.

There are two zip files available:

- **vForth16m_8Microdrives_20240809.zip** that contains **eight** .mdr files.
- **vForth16m_DISCiPLE_20240809.zip** that contains **two** .img files.

suitable to be used within Fuse Emulator, real hardware has not been tested since years... but see next section for a possible real hardware stage.

2.9.1 Microdrive version

This version uses 8 Microdrive units somehow chained together to offer **1778** blocks half-KB each (889 Kbytes total). Emulator such as FUSE provides such 8 Microdrive units: unit number one is used to keep the system loader, and the other seven units are used to store Blocks. This Forth system uses a low-level direct access to sectors, so that the "!Blocks" text-file appears as a single file spread across seven cartridges.

To run under Windows you can use Fuse and to spare some time at start-up, you can specify the switches to enable ZX Interface 1 and insert eight Microdrive cartridges.

```
start fuse.exe ^
      --interface1 ^
      --microdrive-file M1.MDR   ^
      --microdrive-2-file M2.MDR ^
      --microdrive-3-file M3.MDR ^
      --microdrive-4-file M4.MDR ^
      --microdrive-5-file M5.MDR ^
      --microdrive-6-file M6.MDR ^
      --microdrive-7-file M7.MDR ^
      --microdrive-8-file M8.MDR
```

Once the Spectrum is shows the copyright message, you should give the classic **RUN** to load the "run" Basic loader.

This Forth system was born and run on my 48K for years, but to effectively run on real hardware with a single Microdrive Unit, you need to use "**run_HW**" Basic loader instead of the usual "**run**" loader. That loader prompts you to switch cartridges, removing the "Programs" cartridge and inserting the "Blocks" one, and awaits a key-press. To achieve such result, the "Blocks" cartridge must be prepared beforehand using the Basic program "Tap2Mdr.bas" (available in M1.MDR cartridge-file) that reads from tape file !Blocks7.TAP four string-array to be transferred to a single text file that fills the whole cartridge. Usually, such a transfer program stops with "Microdrive full" message after 160 or 170 blocks, depending on the real capacity of a cartridge. At this point the real-hardware single-unit system is ready to run. In particular, the TAP file content was produced by "Mdr2Tap.bas" Basic program that exploits – in Basic – the same technique to achieve a "Random R/W Access" from/to a single text file "!Blocks" present in all seven cartridge-files M2.MDR ... M8.MDR

2.9.2 DISCiPLE version

This version needs two disks unit, the first for DOS system and vForth itself, and the second for data storage to offer 1560 Blocks / Screens (780 KBytes). Again, Fuse emulator works fine.

To spare some time you can specify the suitable switches

```
start fuse.exe ^
      --disciple ^
      --discipledisk Forth1.IMG
```

To start v-Forth system, you have to load the Basic loader, usually **LOAD P6** would be fine. But, I'm not aware of a switch Fuse provides to insert the second floppy disk image at start-up, and you have to select Forth2.IMG via usual Menu bar. If you don't insert the second disk image you'll get an error message "NO DISK in drive" and must redo from start.

2.10 Definitions grouped by category

Here is my personal classification of most definitions available in this system.

2.10.1 Comments

Block oriented	Line oriented	No-operation
(…)	\ …	NOOP

2.10.2 Stack manipulation

Broadly speaking, a *Calculator-Stack* entry is a 16-bits number (§ 4.2) i.e. a **CELL**, while a Double-precision Integer value is a 32-bits number (§ 4.3) which needs two CELLS in the Calculator-Stack, the higher significant part on top of stack. *Return-Stack* is used on entering-exiting phase of a definition *and also* to keep track of **DO** – **LOOP** index and limit. *Floating-Point-Stack* is the standard ZX Spectrum floating-point Calculator Stack that is accessible after loading the Floating-Point-Option (§ 3.8). See "Technical specifications" (§ 4) for more details.

Single Cell	Double cells	Return Stack	Stack Inspection	Floating point Stack
DUP OVER DROP SWAP ROT -ROT NIP TUCK PICK ROLL ?DUP and -DUP	2DUP 2OVER 2DROP 2SWAP 2ROT	>R R> R@ I' DUP>R R>DROP	DEPTH .S	>F F> FOP

2.10.3 Comparison

Comparison involves the top element or the two top elements available on the Calculator Stack. Double-precision Integers (§ 4.3), formed by two elements each, involves twice the elements, obviously.

Zero comparison	Signed	Unsigned	Double-precision
0= 0< 0> NOT	= < > <> MIN MAX	U<	D0= D< D= DU<

2.10.4 Output

Any output is sent to video by default, but the actual device depends on which Stream is chosen via **`SELECT`**.

Single-integer definition Stack Value	**Double-precision integer stack value and Floating point Stack**	**String**	**Other**

`.`
`.R`
`?`
`U.`
	`D.`
	`D.R`
	`F.`
		`."`
		`.(`
		`.C`
		`SPACE`
		`SPACES`
		`EMIT`
		`EMITC`
		`INVV`
		`TRUV`
		`MARK`
		`MESSAGE`
		`TYPE`
			`SPLASH`
			`CLS`
			`CR`
			`DEVICE`
			`SELECT`

2.10.5 Integer Arithmetics

Normally all definitions act upon 16-bits (signed or unsigned) integers (§ 4.2).
Definitions that act upon 32-bits integers (§ 4.3) have names that begin with **D** for *double-precision*.
Mixed definitions, that involve both 16-bits and 32-bits integers begin with **M** for *mixed*.

Arithmetics	**Signed / Unsigned**	**Double-Precision**	**Constants**

`+`
`-`
`*`
`/`
`/MOD`
`MOD`
`2/`
`2*`
	`+-`
	`ABS`
	`NEGATE`
	
	`UM/MOD`
	`UM*`
		`D+`
		`D-`
		`DNEGATE`
		`DABS`
		`D+-`
		`S>D`
			`0`
			`1`
			`2`
			`3`
			`-1`
			
			`PI`

Mixed	**Bitwise**	**Increment/Decre-ment**	**Other**

`M+`
`M*`
`M/`
`M/MOD`
`*/`
`*/MOD`
	`AND`
	`OR`
	`XOR`
	`NOT`
	`RSHIFT`
	`LSHIFT`
	`FLIP`
	`INVERT`
	`SPLIT`
	`UPPER`
		`1+`
		`2+`
		`CELL+`
		`1-`
		`2-`
		`CELL-`
			`FM/MOD`
			`SM/MOD`
			`RANDOM`
			`RND`
			`RANDOMIZE`
			`DSQRT`

2.10.6 Memory

Normally all definitions act upon 16-bits (signed or unsigned) integers (§ 4.2).
Definitions that act upon 32-bits integers (§ 4.3) have names that begin with **2** for *two-cells*.

Store & Fetch	Memory chunks	Pointers & Variables	8K RAM Paging

`!`
`@`
`2!`
`2@`
`C!`
`C@`
`+!`
`TOGGLE`
`TO` (used with `VALUE`)
`+TO`
	`FILL`
	`ERASE`
	`BLANK`
	`CMOVE`
	`CMOVE>`
	`PAD`
	`BUFFER`
	`BLOCK`
	`CELL`
	`CELLS`
	`ALIGNED`
	`DUMP`
		`RP@`
		`RP!`
		`SP@`
		`SP!`
		`S0`
		`R0`
		`USE`
		`PREV`
			`S"`
			`HEAP`
			`FAR`
			`POINTER`
			`H"`
			`+"`
			`+C`
			`>FAR`
			`<FAR`
			`HEAP-INIT`
			`HEAP-DONE`
			`SKIP-PAGE`

2.10.7 Flow control

Counted Loop	Unconted Loop	Conditionals	System related	Exception handling

`DO`
`?DO`
`LOOP`
`+LOOP`
`LEAVE`
`I`
`I'`
`J`
`K`
`WITHIN`
	`BEGIN`
	`WHILE`
	`REPEAT`
	`UNTIL or END`
	`AGAIN`
	`BACK`
	`RECURSE`
	`UNLOOP`
		`IF`
		`THEN or ENDIF`
		`ELSE`
		`CASE`
		`ENDCASE`
		`OF`
		`ENDOF`
		`EXEC:`
		`-?EXECUTE`
			`BYE`
			`AUTOEXEC`
			`COLD`
			`WARM`
			`ABORT`
			`CALL#`
			`EXECUTE`
			`QUIT`
			`BASIC`
				`CATCH`
				`THROW`
				`ERROR`
				`ABORT`
				`ABORT"`

2.10.8 I/O and Hardware

I/O Ports	HW Registers	Keyboard	Other

`P!`
`P@`
	`REG!`
	`REG@`
	`MMU7!`
	`MMU7@`
		`?TERMINAL`
		`KEY`
		`CURS`
		`WAIT-KEY`
			`LOAD2BLOCK`
			`LOAD-BYTES`
			`SAVE-BYTES`

2.10.9 Definition related

Creators	Status & Variables	Compilation / Interpretation	Dictionary Allocation
:	?COMP	COMPILE	ALLOT
:NONAME	?CSP	[COMPILE]	,
CREATE	?ERROR	[CHAR]	C,
VARIABLE	?EXEC	[']	."
CONSTANT	?LOADING	POSTOPNE	.(
CODE	?PAIRS	[	COMPILE,
EXIT	?STACK	]	LITERAL
DOES>	STATE	EVALUATE	DLITERAL
<BUILDS	CSP	INTERPRET	
USER	!CSP	;	
VALUE			

2.10.10 BLOCK / Screen related

Block & Buffer	Input	Block-file primitives	Variables & Constants
.LINE	LOAD	BLK-INIT	TIB
BLOCK	WHERE	BLK-READ	FIRST
EMPTY-BUFFERS	-->	BLK-SEEK	LIMIT
FLUSH	QUERY	BLK-WRITE	SOURCE-ID
INDEX	ACCEPT		BLK
LIST	ENCLOSE		>IN
UPDATE	CHAR		OUT
OPEN<	EXPECT		SCR
SCREEN-TO-FILE	WHERE		OFFSET
SCREEN-FROM-FILE	LOCATE		BLK-FH
USE	GREP		BLK-FNAME
	BSEARCH		#SEC
			#BUFF
			SPAN
			B/BUF
			B/SCR
			C/L

2.10.11 Numbers & strings

Number to string	Base	Interpretation	Variables
<#	BASE	NUMBER	NMODE
#	HEX	(NUMBER)	HLD
#S	DECIMAL	(SGN)	DPL
#>	BINARY		FLD
SIGN	OCTAL		PLACE
HOLD			EXP

2.10.12 Dictionary related

Input Stream	**Vocabulary manipulation**	**Definition data**	**Variables & Constants**
' -FIND INCLUDE MARKER NEEDS CASEOFF CASEON WORDS	FORTH DEFINITIONS ASSEMBLER SMUDGE RENAME FORGET ALIGN ID.	CFA PFA NFA LFA <NAME >BODY TRAVERSE .WORD	WIDTH WARNING FENCE DP VOC-LINK CONTEXT CURRENT BL

2.10.13 Editor

Screen oriented	**Line oriented**	**Memory**	**File oriented**
EDIT B N L SAVE WIPE	H D RE INS S E	.PAD P -MOVE TEXT LINE UNUSED ROOM	LED LED-EDIT LED-SAVE LED-FILE

2.10.14 NextZXOS

File hook	**Directory hook**	**+3DOS hook**	**Other**
F_SEEK F_CLOSE F_SYNC F_FGETPOS F_READ F_WRITE F_OPEN F_FSTAT TOUCH UNLINK	F_OPENDIR F_READDIR DIR PWD CD	M_P3DOS	R# LP HANDLER ZAP VIEW

2.11 Known bugs, workarounds, and improvement needed

INCLUDE The **INCLUDE**d source text file *must end with an empty line*, otherwise the system crashes usually showing some vertical grid. **NEEDS** suffers the same bug since it uses **INCLUDE**.

NEEDS In case of interpretation/compilation error, the file/handle remains open and you have to manually close it using something like **2 F_CLOSE DROP** and you cannot use **REMOUNT** until then.
The same problem arises using other disk related definitions.

LOAD In some cases, interpretation of long structure via **LOAD** cannot cope with **BLOCK**s boundaries within the same Screen. This means, for example, that you cannot start an **ENUMERATED** structure in the first **BLOCK** of a Screen and continue it in the next one.

A 'nul' character (0x00) inside a Screen is completely invisible, but is the cause of most programmer headache because it provokes a sudden stop during **LOAD**. Within a Screen, you can locate such characters using **EDIT** (§ 3.1) that shows the ASCII code of character at cursor position.

OPEN< At the moment, this definition can be used only in interpretation mode.

BCOPY When used repeatedly, it seem to miss half Screen every three, especially when you work backward in block number. Maybe this depends from the number of **BUFFER**s available. A simple workaround is to **FLUSH** often.

LED Pressing [BREAK] will stop any I/O operations: if not correctly used, may produce data-loss.

3 Utilities

WARNING: some of these definitions are still under development and specifications may change in the future. Much effort is put to keep backward compatibility.

3.1 The Full Screen Editor Utility – Screen oriented

The **EDIT** definition is available after you type: **NEEDS EDIT** (or in the old way **190 LOAD** if the source it is still there and you didn't reused these Screens).

On this Forth system, as in many others, a Screen has 1,024 bytes of data spread in 16 lines, 64 bytes each.

This “Full Screen Editor Utility” is invoked using the **EDIT** definition that enters a simple page-editor that allows modifying the current Screen, i.e. the one contained in **SCR** variable. During **EDIT**, you are allowed move to the next Screen or to the previous one using the commands explained below.

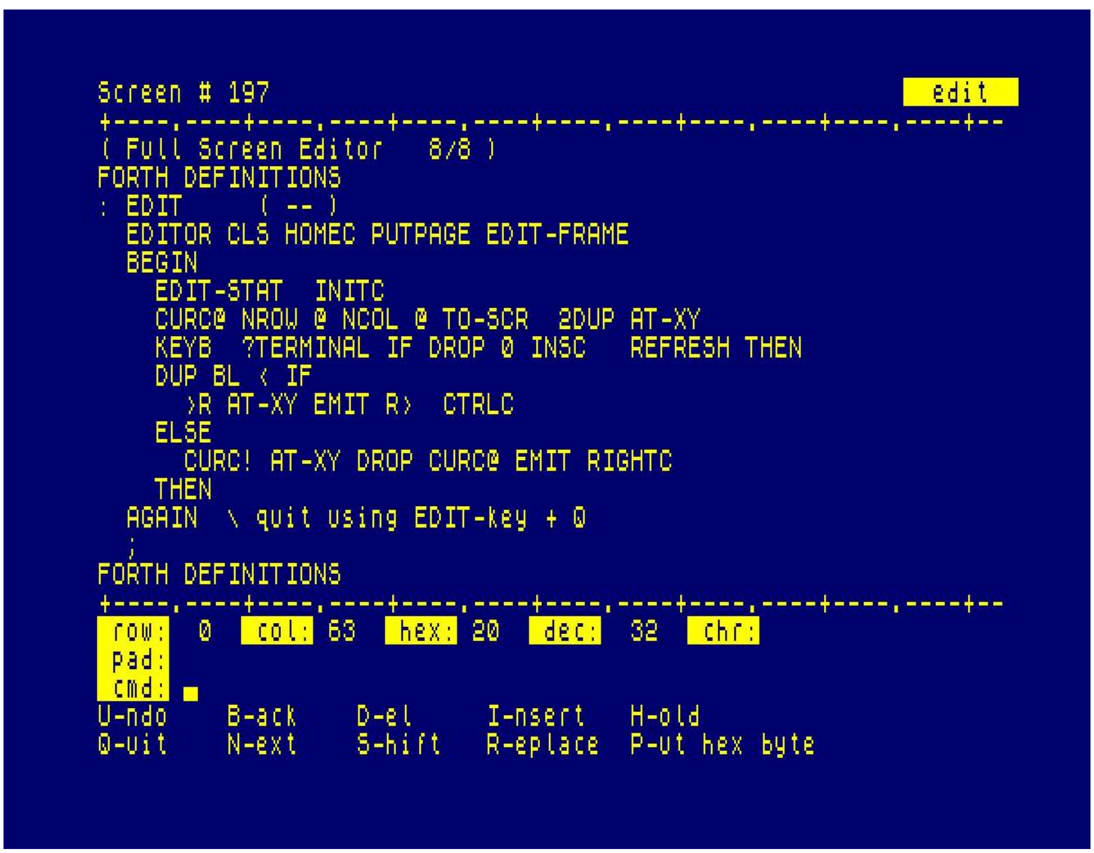
Remember: to *quit* the **EDIT** phase, you have to use [**Edit**] key followed by **[Q]** key, in a way that mimics Unix **vi** editor.

This editor works only if the display-mode allows 64 character per line at least.

EDIT ---

For example, to select, show and edit **Screen # 197** you can type:

DECIMAL 197 LIST	(to set 197 the “current screen”)
EDIT	(to enter the editor on “current screen”)



Previous picture shows a header reporting the Screen number and a line-ruler followed by 16 lines that make up the Screen itself.

A flashing cursor is visible at home position: The cursor has two flashing mode to distinguish **CAPS LOCK** enabled – the higher flashing cursor – (█) or disabled – the half flashing cursor – (▄) .

The cursor keys, or the equivalent **[Shift]** key + **5 / 6 / 7 / 8** keys or using standard PC keyboard, allow the flashing cursor to be moved across the screen to point the current position inside the Screen, so text can be typed at any position on the Screen.

Current cursor positions (**row** number and **col**umn number) are shown on the bottom status bar along with current character, **dec**imal ASCII code and **hex**adecimal code of it.

Pad line shows the current **PAD** content. Line oriented commands handle and work with **PAD**. See the "Line Editor" chapter (§3.9). When **PAD** contains garbage, the whole screen may become corrupted: in this case you can type **[Edit]** + **H** to copy the current line to **PAD** that should fix that issue.

After the **[Edit]** key (**Shift + 1** using standard PC keyboard) the Editor recognizes the subsequent single or double key-stroke commands:

> **[Edit] , Q** : **Q**uit **EDIT** Utility
> **[Edit] , U** : **U**ndo, that is re-read current screen from disk ignoring any modification done since last **FLUSH**. This feature is quite important, since it does for the current Screen buffers what **EMPTY-BUFFERS** in general.
> **[Edit] , H** : take (or **H**old) current line content and keep it in **PAD**
> **[Edit] , R** : **R**eplace current line with the current **PAD** content.
> **[Edit] , S** : make **S**pace at current cursor position shifting lower lines down; last line will be lost.
> **[Edit] , D** : **D**elete current line shifting up lower line, but a copy is copied to **PAD** before deletion, like **H**
> **[Edit] , I** : **I**nsert at current cursor line position the content of **PAD**: it does commands **S** and **R**.
> **[Edit] , N** : go to **N**ext screen
> **[Edit] , B** : go **B**ack to previous screen
> **[Edit] , P** : accepts *two hexadecimal digits* representing a byte and **P**ut it at cursor position. This way, non-printable characters, i.e. ASCII code between 0 and 31 ($00 - $1F) or graphics characters (above $7F), can be stored inside a Screen, but care must be paid to avoid corrupting the display because most of them are *control characters* and some of them are interpreted during display. Characters with ASCII code between $80 and $FF can be stored in a Screen, but they are emitted to video translated to the corresponding codes between $00 and $7F.

Any other key has no meaning and return the flashing cursor back to its position.

[Delete] (that is **Caps-Shift + ZERO**) removes a character at current cursor position, shifting left the rest of the line.

[Break] (that is **Caps-Shift + SPACE**) inserts a space at current cursor position, shifting right the rest of the line.

[Caps-Lock] (that is **Caps-Shift + 2)** accounts for a keystroke, but it is interpreted by the system to change the Caps-Lock state switching between enabled (█) and disabled (▄).

Beware, any modification you make – even for mistake – immediately affects the underlying buffers, so if you mess things too much so that **[Edit] + U** is not enough, there is only a way to recover it, to quit before it's too late and invoke the definition **EMPTY-BUFFERS** to blank all buffers without flushing them to disk.

This "Full Screen Editor" is a work-in-progress and can be improved if needed either acting on Screens between 190 and 197 or editing **./lib/edit.f** source file.

SCREEN-FROM-FILE n ---

Available after **NEEDS SCREEN-FROM-FILE**. Import a single Screen from file.

Used in the form

```
n SCREEN-FROM-FILE filename.f
```

SCREEN-TO-FILE   n   ---

Available after **`NEEDS SCREEN-TO-FILE`**. Export a single Screen to file.

Used in the form

```
n SCREEN-TO-FILE filename.f
```

For your and the whole system safety, a “NextZXOS Write error” is issued if the specified file already exists: if you really need to, you have to `UNLINK` it in advance.

3.2 Graphics mode and Layer facility

The following definitions are available after you type **NEEDS GRAPHICS**.
To forget this library from dictionary you can type **NO-GRAPHICS** or **FORGET GRAPHICS**.
This library is still work-in-progress because I'm improving it every so often.

The ZX Spectrum Next's machine can handle several Graphic-Modes and vForth is able to use almos all of them.

In all the following definitions, the x-coordinate is the vertical distance from the top-left corner of the grid, the y-coordinate is the horizontal distance from the top-left corner of the grid

LAYER! n ---

This definition changes Graphic-Mode. The parameter `n` can be one of the following values:

- **00** to switch to **Layer 0** - Standard Spectrum (ULA) mode, 256 w x 192 h pixels, 8 colors total (2 intensities), 32 x 24 cells, 2 colors per cell. Equivalent to Basic's LAYER 0.
- **10** to switch to **Layer 1,0** - LoRes (Enhanced ULA) mode, 128 w x 96 h pixels, 256 colors total, 1 color per pixel. Equivalent to Basic's LAYER 1,0.
- **11** to switch to **Layer 1,1** – Standard Res (Enhanced ULA) mode, 256 w x 192 h pixels, 256 colors total, 32 x 24 cells, 2 colors per cell. Equivalent to Basic's LAYER 1,1.
- **12** to switch to **Layer 1,2** – Timex HiRes (Enhanced ULA) mode, 512 w x 192 h pixels, 256 colors total, only 2 colors on whole screen. Equivalent to Basic's LAYER 1,2.
- **13** to switch to **Layer 1,3** – Timex HiColour (Enhanced ULA) mode, 256 w x 192 h pixels, 256 colors total, 32 x 192 cells, 2 colors per cell. Equivalent to Basic's LAYER 1,3.
- **20** to switch to **Layer 2** – 256 w x 192 h pixels, 256 colors total, one color per pixel. Equivalent to Basic's LAYER 2,1.

To ease of use, this definition accepts `n` which can be expressed both in **DECIMAL** or in **HEX**, without confusion, and since there is no ambiguity, the following two lines gives the same result

```
HEX     12 LAYER!
DECIMAL 12 LAYER!
```

This primitive definition *just* changes Graphics-Mode without any other side effect. Instead, the hereby definitions **LAYER0**, **LAYER10**, **LAYER11**, **LAYER12**, **LAYER13** and **LAYER20** also modify the overall behavior of some other graphics definitions.

The following table shows the peculiarity of each of six Graphics-Mode when applicable

	L0	L11	L12	L13	L10	L2
Char-Size	8	4	8	4	4	4
V-RANGE	0C0	0C0	0C0	0C0	060	0C0
H-RANGE	100	100	200	100	080	100
PIXELADD	L0	L0	L12	L0	L10	L2
POINT	L0	L0	L0	L0	L1	L1
PLOT	L0	L0	L0	L0	L1	L2
XPLOT	L0	L0	L0	L0	L1	L2
PIXELATT	L0	L0	na	L13	na	L2
XY-RATIO	1	1	2/	1	1	1
EDGE	=	=	=	=	L1	L1

LAYER0 ---

Set screen mode to Standard ULA, legacy ZX Spectrum mode, also set the characters to 4 pixel wide, and modify the overall behavior of all graphics definitions to work with this specific pixel resolution.

LAYER10 ---

Set screen mode to LoRes mode, set the characters to 4 pixel wide, and modify the overall behavior of all graphics definitions to work with this specific pixel resolution.

LAYER11 ---

Set screen mode to Standard Res (Enhanced ULA) mode, set the characters to 4 pixel wide, and modify the overall behavior of all graphics definitions to work with this specific pixel resolution.

LAYER12 ---

Set screen mode to Timex HiRes (Enhanced ULA) mode, set the characters to 8 pixel wide, and modify the overall behavior of all graphics definitions to work with this specific pixel resolution, for example, the correct aspect-ratio for **CIRCLE** is enforced.

LAYER13 ---

Set screen mode to Timex HiColour (Enhanced ULA) mode, set the characters to 4 pixel wide, and modify the overall behavior of all graphics definitions to work with this specific pixel resolution.

LAYER2 ---

Set screen mode to Layer 2, set the characters to 4 pixel wide, and modify the overall behavior of all graphics definitions to work with this specific pixel resolution.

IDE_MODE! **n** ---

This is a primitive definition to switch Graphic-Mode via NextZXOS using **M_P3DOS** call $01D5
It's called by **LAYER!** that prepares n in a suitable way. This definition is also available via NEEDS **IDE_MODE!**

LAYER0	:	$0000
LAYER10	:	$0100
LAYER11	:	$0101
LAYER12	:	$0102
LAYER13	:	$0103
LAYER2	:	$0200

IDE_MODE@ --- hl de bc a

This is a primitive definition to query current Graphic-Mode via NextZXOS using `M_P3DOS` call $01D5.
Usual result are:

LAYER	:	hl	de	bc	a	(a in binary)
LAYER0	:	$1620	$000F	$0800	$00	(00 00 00 00)
LAYER10	:	$0C20	$FF00	$0400	$01	(00 00 00 01)
LAYER11	:	$1840	$000F	$0400	$05	(00 00 01 01)
LAYER12	:	$1840	$0000	$0800	$09	(00 00 10 01)
LAYER13	:	$1840	$0038	$0400	$0D	(00 00 11 01)
LAYER2	:	$1840	$0038	$0400	$02	(00 00 00 10)

3.2.1 Low-level definitions

The following definitions **PIXELADD**, **PIXELATT**, **PLOT** and **POINT** are coded in assembler for maximum performance.

ATTRIB --- n

Value that specifies the byte used as color-attribute in all subsequent graphics command. To modify it you have to use the **VALUE - TO** semantics

```
DECIMAL  216  TO  ATTRIB
```

The value of **ATTRIB** is saved across any Layer switch so that each Graphic-Mode keeps its own value.

PIXELADD x y --- n

Depenging on current Graphic-Mode, determine address of a pixel and fit MMU7 8K page if needed.

PIXELATT x y --- n

Depenging on current Graphic-Mode, determine address of a pixel and fit MMU7 8K page if needed.

PLOT x y ---

Depenging on current Graphic-Mode, plot a pixel using current **ATTRIB**.

POINT x y --- c

Depenging on current Graphic-Mode, it returns the attribute value of a pixel.
For Layer 1,0 and Layer 2 modes this is simply the sequence **PIXELADD C@**.
For Layer 0 Layer 1,1 Layer 1,2 and Layer 1,3 modes, this definition returns a true-flag if the pixel is set or a false-flag if the pixel is unset.

3.2.2 High-level definitions

CIRCLE **x y r** **---**

Draw a circle with center at `x y` and radius `r` using the current **ATTRIB** color and Graphic-Mode. As stated above, x-coordinate is the vertical and y-coordinate is horizontal. This definition does not use

DRAW-LINE **x0 y0 x1 y1** **---**

Draw a line from `x1 y1` to `x0 y0` using the current **ATTRIB** color and Graphic-Mode. As stated above, x-coordinate is the vertical and y-coordinate is horizontal.

PLOT **x y** **---**

Draw a pixel at `x y` using the current **ATTRIB** color and Graphic-Mode. As stated above, x-coordinate is the vertical and y-coordinate is horizontal.

PAINT **x y** **---**

Experimental: Try to paint a well-shaped convex area, provided that `x y` is some "center" to start. As stated above, x-coordinate is the vertical and y-coordinate is horizontal.

3.2.3 Colors & Attributes

Here is a set of definition that invoke the Standard-ROM routines to change the screen colors. All these definitions end with a dot (.) to spacify that it works via `EMIT` and to avoid confusion with other definitions.

.BORDER **b** **---**

Immediately set the current BORDER color. It uses ROM routine $2297 via **CALL#**.

.BRIGHT **b** **---**

Depending on on current Graphics Mode, set the current BRIGHT attribute for any subsequent output operations.

.FLASH **b** **---**

Depending on on current Graphics Mode, set the current BRIGHT attribute for any subsequent output operations.

.INK **b** **---**

Depending on on current Graphics Mode, set the current INK color for any subsequent output operations.

.INVERSE **b** **---**

Depending on on current Graphics Mode, set the current INVERSE attribute for any subsequent output operations.

.OVER **b** **---**

Depending on on current Graphics Mode, set the current OVER attribute for any subsequent output operations.

.PAPER **b** **---**

Depending on on current Graphics Mode, set the current PAPER color for any subsequent output operations.

3.3 DIR and LED – the Large file Editor and

Source text-files can be edited directly within vForth environment using **LED** – the Large file EDitor – that handles text files up to 17.568 rows, 85 characters each. The **LED** definition is available after you type: **NEEDS LED** .

LED asks NextZXOS for as many 8K-pages are needed to keep the text file in RAM, and an "Out-of-memory" error is issued if a page is not available, for instance because you already used some BANK command from Basic.

Along with **LED** you often need **DIR**.

After you type **LED** you enter a simple full-screen editor that can modify current file one screen at a time. While within **LED**, you are allowed move to next page or to previous page using the command explained below.

Remember: to ***quit*** **LED** editor, you have to use [**Edit**] key followed by [**Q**] key.

Remember: to ***save*** the file, you have to use [**Edit**] key followed by [**W**] key.

This editor works only while the display-mode is **LAYER 1,2**.

CD ---

Available after **NEEDS CD**. Change current directory. Used in the form

CD xxx

Warning : changing to a different directory impede any subsequent use of **NEEDS** until current directory is restored to its default.

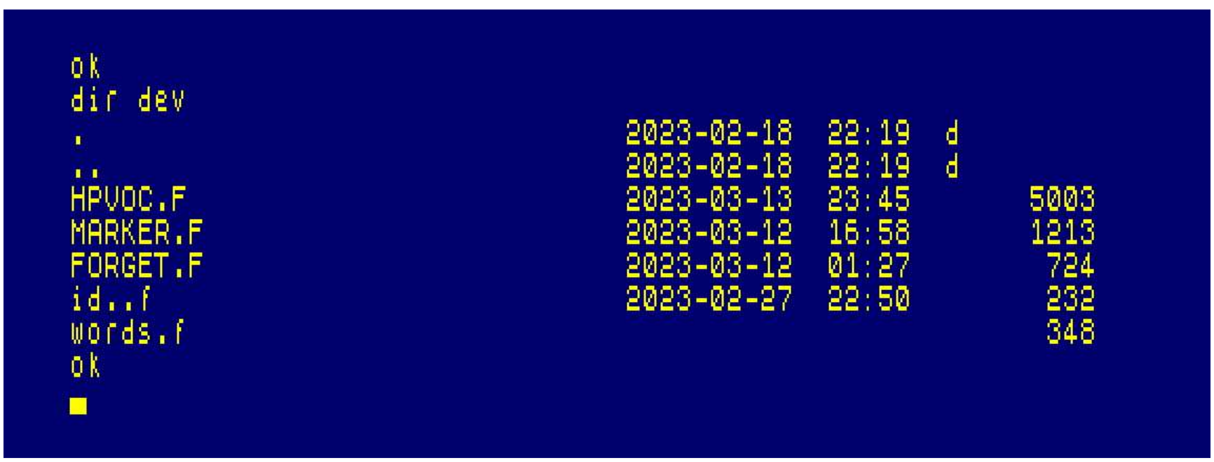
DIR ---

Available after **NEEDS DIR**. Used in the form

DIR xxx

it displays the content of directory **xxx** for example: **DIR dev**

During display, [EDIT] key suspends output until the key is released.



LED --- cccc

Available after **NEEDS LED**. Used in the form

LED cccc

opens specified file cccc and enters **LED** editor. For example **LED lib/dir.f**
This editor inherits most of its commands and behavior from the previously described **EDIT** editor except that is has 85 characters per line instead of 64. See previous paragraph (§3.1) for details..

Along **LED** command, some more sub-commands are available to better handle a text-file.

LED-EDIT ---

Used in the form

LED-EDIT

re-enters the **LED** editor after you quit it to continue editing the same file you previously opened that should still be in upper 8k RAM pages, provided you haven't corrupted its content in some way.

LED-SAVE ---

Used in the form

LED-SAVE

saves back the file you previously open in **LED** editor, using the current filename you already specified using **LED** or **LED-FILE**. The same function is invoked during **LED** editing via [EDIT]-W.

LED-FILE --- cccc

Used in the form

LED-FILE cccc

modify the filename that **LED-SAVE** will write to. This allow to save to a different filename.

PWD ---

Available after NEEDS PWD. Prints current directory.

TOUCH --- cccc

Used in the form

TOUCH cccc

Accept the followin string cccc as a filename and update file-timestamp. If the file doesn't exist, it's created at zero length. Available after NEEDS TOUCH.

UNLINK --- cccc

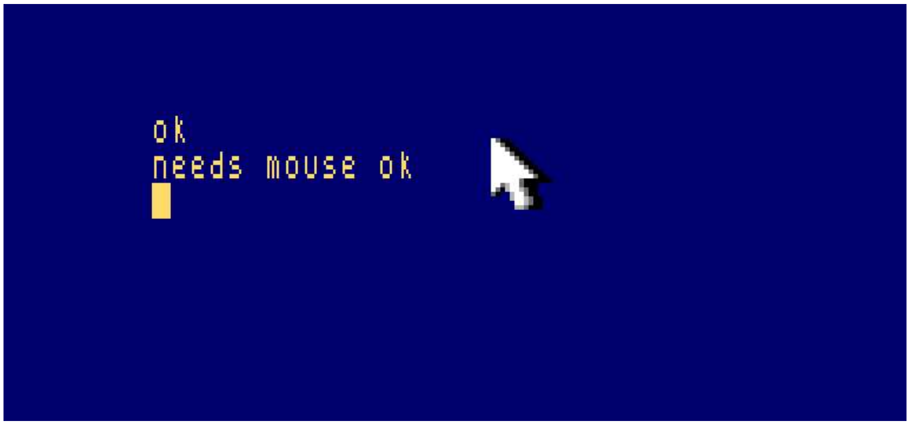
Used in the form

UNLINK cccc

Accept the following string as a filename and remove it from disk, Beware, there is no way to recovery.
At this moment, you must specify the drive e.g. **unlink c:dummy.f** and despite it could be deemed a bug, I keep this behavior to improve security and avoid unwanted destructive operations. Available after NEEDS UNLINK.

3.4 Mouse

An *interrupt-driven mouse facility* is made available via **`NEEDS MOUSE`** (which requires **`INTERRUPTS`** described in Chapter 3.5).



Sprite #0 image is made a white arrow shape with shadow; its definition is given directly in Forth code, as follows (this, by the way, shows how easily Forth language can perform such a task):

```
HEX  14 REG@ CONSTANT E3 \ Global Transparency Colour
:  "  00  C,  ;  \ Black
:  |  6D  C,  ;  \ Dark-Grey
:  v  B6  C,  ;  \ Light-Gray
:  M  FF  C,  ;  \ White
:  _  E3  C,  ;  \ Transparency

\ Semi-graphical mouse-face definition 10x8-pixels arrow
CREATE MOUSE-FACE
\ 0 1 2 3 4 5 6 7 8 9 A B C D E F \
\ ------------------------------- \
  M | " " _ _ _ _ _ _ _ _ _ _ _ _ \  0
  M M | " " _ _ _ _ _ _ _ _ _ _ _ \  1
  M M M | " " _ _ _ _ _ _ _ _ _ _ \  2
  M M M M | " " _ _ _ _ _ _ _ _ _ \  3
  M M M M M | " " _ _ _ _ _ _ _ _ \  4
  M M M M M M | " " _ _ _ _ _ _ _ \  5
  M M M M M M M | " " _ _ _ _ _ _ \  6
  M M M M M M M M | " " _ _ _ _ _ \  7
  M M M M M M M M M | " " _ _ _ _ \  8
  M M M M M M | " " " " " " _ _ _ \  9
  M M | v M M | " " " _ _ _ _ _ _ \  A
  M | " v M M v | " " _ _ _ _ _ _ \  B
  | " _ _ v M M | " " _ _ _ _ _ _ \  C
  _ _ _ _ v M M v | " " _ _ _ _ _ \  D
  _ _ _ _ _ M M v | " " _ _ _ _ _ \  E
  _ _ _ _ _ v v | " " " _ _ _ _ _ \  F
```

The hardware is polled every 20 ms (during interrupt) and data is processed as follows:

1. Three bytes are collected as "raw-data" from three hardware ports:
   - **$FFDF** (Kempston mouse Y vertical) stored in **MOUSE-RX** variable (result is multiplied by 256).
   - **$FBDF** (Kempston mouse X horizontal) stored in **MOUSE-RY** variable (result multiplied by 256).
   - **$FADF** (Kempston mouse Wheel andButtons) stored in **MOUSE-RS**
     - three lower bits are decoded as follow:
       bit 0: right button, zero when pressed, one when released.
       bit 1: left button, zero when pressed, one when released.
       bit 2: wheel click-down event, zero when pressed, one when released.
     - the four higher bits contain the current wheel position among sixteen different cyclical position coded between 0 and 15.

2. Current raw-data (at time t) is compared against the previous ones (at time t-1) revealing signed deltas:
   - $\text{MOUSE-DX} \leftarrow \text{MOUSE-RX}_t - \text{MOUSE-RX}_{t-1}$
   - $\text{MOUSE-DY} \leftarrow \text{MOUSE-RY}_t - \text{MOUSE-RY}_{t-1}$
   - $\text{MOUSE-DS} \leftarrow \text{MOUSE-RS}_t - \text{MOUSE-RS}_{t-1}$

3. If delta x or y (`MOUSE-DX` , `MOUSE-DY`) are non-zero, compute new mouse position and move the corresponding sprite, clipping within screen range:
   - **MOUSE-X** : vertical distance from top-left corner
   - **MOUSE-Y** : horizontal distance from top-left corner

4. If delta-buttons (`MOUSE-DS`) is non-zero, the following event-flags are provided in **MOUSE-S**:
   $0000 : no event
   $0001 : right button click-down
   $0002 : left button click-down
   $0004 : wheel button click-down
   $0010 : wheel forward direction
   $0100 : right button click-up
   $0200 : left button click-up
   $0400 : wheel button click-up
   $1000 : wheel backward direction
   Events are OR-ed together and kept persistent in **MOUSE-S** until they're consumed via **MOUSE** that reset it to zero.

The mouse position and status can be inspected using the following definitions:

MOUSE-XY --- n1 n2

Returns the current mouse position in pixel

`n1` : vertical distance from top-left corner in pixel. Range is 0 – 319.
`n2` : horizontal distance from top-left corner in pixel. . Range is 0 – 255.

?MOUSE --- f

Returns a true-flag if there is a mouse-click event.

MOUSE --- n

Collects and consumes the latest "persistent" click-events values and reset **MOUSE-S** to zero. The event(s) is (are) reported as shown in point 4 above.

3.5 Interrupt Service Routine

After you type **NEEDS INTERRUPTS** a few new definitions will be loaded in memory along with some low-level definitions that allow setting-up an Interrupt-Driven definition: The ISR must be a single definition suitably defined. This is a standard IM 2 interrupt routine implementation. In the future, I hope to be able to exploit the new Next's IM 2 interrupt vector mode. Programming an Interrupt Service Routine using Forth itself is tricky and if not correctly coded, it can impair the system or cause a system-crash. As said, this library still does not exploit the new ZX Spectrum Next interrupt vector, this will be soon implemented: this means that the all these definitions listed here below will go under a deep overhaul when I'll code it, but my aim is to keep backward compatibility as much as possible.

A Z80's maskable interrupt occurs every 20 ms (50 times per second), and when it occurs, a CALL to a specific routine is performed. In vForth we use IM 2 interrupt mode by preparing a 257 bytes vector-table at $6200, filled with $63, so that the interrupt service routine is located at $6363 to jump to the suitable code – i.e. the definition **ISR-SUB** – that makes possible to a Forth definition to be executed as interrupt.

First, **ISR-SUB** performs a RST $38 to fulfill the legacy ISR, then it must save the whole *Forth machine status* by pushing to stack the value of CPU registers and then saving Forth's Return-Stack-Pointer and Calculator-Stack-Pointer.
Second, it prepares Forth virtual registers (Calculator-Stack Pointer, Return-Stack-Pointer and Instruction-Pointer) to execute the xt contained in **ISR-XT** variable, then a jump to the Inner-interpreter via JP (IX) is performed. Interrupts stay disabled during the execution of such a xt.

After the xt contained in **ISR-XT** is executed, the **ISR-RET** definition is executed restoring back the machine status by retrieving Calculator-Stack Pointer, Return-Stack-Pointer and Instruction-Pointer and then popping all CPU registers before returning from the interrupt routine and re-enabling interrupts.

It's worth to be noticed that, since an interrupt may occur in the middle of the execution of any part of Forth system, not everything can be performed within an interrupt service routine, and care must be put to avoid critical interference with the main program, such as trying to write system variables or invoking peculiar definitions that are known to modify the code they're going to execute, such as **CASEON**, **CASEOFF**, or memory areas such as most Floating-Point operations.

ISR-OFF ---

Disable Interrupt Utility by restoring IM 1 and I register to its default $3F value.

ISR-XT xt ---

Variable that contains the xt of the definition that will be executed in background at each Interrupt. It is always followed by the execution of **ISR-RET** so that **ISR-XT** can be viewed as the pointer to an anonymous definition that contains two definitions: the *interrupt-service-routine* definition and the *return-from-interrupt* definition.

ISR-ON ---

Enable *Interrupt Service Utility*: This definition prepares "IM 2 Vector Table" at address $6200-$6300 filling it with all $63 and set Interrupt Mode 2, so that when an Interrupt is issued a CALL to address $6363 is performed.

At address $6363 is a jump to address of **ISR-SUB** body i.e. **[' ISR-SUB >BODY]** and it's used in the form

```
ISR-OFF
' <your-isr-word>  ISR-XT  !
ISR-ON
```

Then, **<your-isr-word>** is executed in background at each Interrupt.

During an Interrupt, Forth uses a separate Calculator Stack (4 bytes below current SP) and a separate Return Stack located at $6330. Care must be paid to avoid any critical interference with the normal *foreground* Forth execution.

Typical usage is to control some Sprite movement or poll *mouse* and *joystick*, some demos are available.

The following example keeps the display filled with evenly spaced dots in Layer 1,1 or Layer 1,2 modes.

```
: ISR-WORD

      $80 $57FF C!
      $5701 $5700 $0FF CMOVE>
      $5700 $4700 $100 CMOVE
      $4700 $4F00 $100 CMOVE
;
ISR-OFF
' ISR-WORD ISR-XT !
ISR-ON
```

ISR-EI ---

Low-level “enable interrupt”. It actually executes an EI opcode.

ISR-DI ---

Low-level “disable interrupt”. It actually executes a DI opcode.

ISR-IM1 ---

Low-level “interrupt mode 1”. It actually executes an IM 1 opcode. This is the default *mode* for any ZX Spectrum.

INT-IM2 ---

Low-level “interrupt mode 2”. It actually executes an IM 2 opcode. It relies on a “vector table” located

ISR-SYNC ---

Low-level “sync to video”. It actually executes an HALT opcode to force the machine to wait until the next interrupt.

SETIREG b ---

Low level Z80 register I setting. It actually executes an LD I, A opcode.

ISR-RET ---

Low-level “return from interrupt” definition. It restores all registers and returns control to Forth foreground execution.

ISR-SUB ---

Low-level “interrupt service routine” definition. It saves all registers and gives control to `INT-XT` background definition execution. Interrupt SP is initialized at 4 bytes below current SP. Interrupt RP is initialized at $6330 and allows room for 14 cells.

3.6 Block Search and Locate Utility

This group of definitions allow you to look for text within the Screens / Blocks and are available after you type alternatively:

NEEDS LOCATE or
NEEDS GREP or
NEEDS BSEARCH or
NEEDS COMPARE

LOCATE ---

Used in the form

LOCATE cccc

this definition examines all Screens between 1 and 1000 looking for the definition of **cccc** showing the Screen where the first occurrence is found, then it makes it the "current screen", just like **LIST** for example:

LOCATE EDIT

takes a few seconds to search in which Screen **COMPARE** is defined, and if found it shows the Screen using **LIST**.

GREP ---

Used in the form

GREP cccc

this definition examines all Screens between 1 and 2000 looking for any occurrence of string **cccc** showing them in a table form, for example

GREP EDIT

will take some more time to complete and gives something like the following

BSEARCH   **n1 n2**   **---**

Used in the form

n1 n2 BSEARCH cccc

this definition examines all Screens between n1 and n2 looking for any occurrence of definition **cccc** showing them in a table form. This definition is used by **GREP** that in fact is defined as **1 2000 BSEARCH** .

COMPARE   **a1 b1 a2 b2**   **---**

Given two string descriptors, that is address and length, (a1, b1) and (a2, b2), this definition compares the two strings and returns:

0	if they're equal
1	if String1 > String2
-1	if String1 < String2

For example:

```
CREATE S1 ," Hello world!"
CREATE S2 ," Hello world?"
S1 COUNT S2 COUNT COMPARE .
```

will print -1 since the two strings differs only for the last character and the ASCII code of `!` comes before the code of `?` , so the string comparison S1 < S2 is true. Compare the result of the following two rows:

```
S2 COUNT S1 COUNT COMPARE .
S1 COUNT S1 COUNT COMPARE .
```

CASEON and **CASEOFF** modify the behavior of **COMPARE** definition being case-sensitive or case-insensitive.

3.7 Debugger Utility

The following definitions are available after you type **NEEDS SEE** or usually after a regular **AUTOEXEC**.

Also, this section exposes in detail how definitions are stored in dictionary memory.

In the **Indirect-Threaded** version (version 1.6), low-level definitions CFA contains the address of PFA that in turn contains the actual definition machine code; in a colon definition the CFA points to the address of the routine that handles that kind of definition.

In **Direct-Threaded** version (version 1.7), a Low-Level definition takes two bytes less, since CFA directly contains the actual definition machine code; then a Colon-definition needs one additional byte in CFA to allow room for a “CALL” op-code to call the address that handles that kind of definition. This allows some 25% of more speed at the cost of using a little more memory used.

SEE ---

Used in the form

SEE cccc

it prints how definition cccc is defined along with its NFA, LFA, CFA, PFA information.

If cccc is a regular colon-definition, the result will show something very close to the original source code the definition was coded from.

For example, the definition **TYPE** is a colon-definition that emits to video an **n** bytes-long string stored at address **a**, and it's defined as follow:

```
: TYPE    ( a n -- )
    BOUNDS
    ?DO
        I C@ EMIT
    LOOP ;
```

If you type

```
SEE TYPE
```

the system will emit something like this:

```
Nfa: E7CD 84
Lfa: E7D2 7C3 LEAVE
Cfa: 6F5D 6A21
BOUNDS (?DO) 12 I C@ EMIT (LOOP) -8 EXIT ok
```

The first line shows **TYPE** **N**ame **F**ield **A**ddress (**$E7CD**) followed by **$84** that is the counter byte of a 4-bytes length name. The counter byte always has the most significant bit set, that is **$80** added to **$04** giving **$84**.

The second line is the **L**ink **F**ield **A**ddress (**$E7D2**) which holds a *heap-pointer* **$07C3** to **LEAVE**’s NFA that in this case happens to be the previous definition in the dictionary.

You will notice these two fields (NFA and LFA) are stored above address $E000 (i.e. MMU7) and they are effectively stored in one of the extended 8K-RAM pages.

The third line is the **C**ode **F**ield **A**ddress (**$6F5D**) that contains the actual machine code to be run which in this case is a "CALL" to the ENTER routine of every colon-definition, located at **$6A21**.

The fourth line represents the **P**arameter **F**ield **A**ddress and, in this case, is in some way a definition *decompilation* but literals and offsets are shown in "inverse video". For example the number **-8** after **(LOOP)** is the "displacement" to where the Instruction Pointer has to jump to go back to next iteration, that is 4 cells backward. In this example **(?DO)** and **(LOOP)** are the *compiled counterpart* of **?DO** and **LOOP** that in fact normally won't be *compiled by themselves,* instead they control the compilation of some other definitions.

Another example, the definition **NIP** that removes the second element from Stack, isn't a colon-definition, but a low-level definition coded in machine-code as follow:

```
CODE NIP  ( n1 n2 -- n2 )
        POP       HL|     \  pop  hl
        EX(SP)HL          \  ex   (sp), hl
        Next              \  jp   (ix)
        C;
```

and if you type

```
SEE NIP
```

you'll see

```
Nfa: E2F1 83
Lfa: E2F5 2E8 DROP
Cfa: 6900 DDE3
6900 E1 E3 DD E9  00 03 E1 F1  ac]i  aq
```

In this case, since **NIP** is a low-level definition, the PFA part is shown as a hexadecimal **DUMP** that is it has no PFA part, but it's a real machine-code routine. This is an example of use of the **ASSEMBLER** built-in vocabulary available via **NEEDS ASSEMBLER**.

Again, the first line shows **NIP**'s NFA (**$E2F1** in this case) and **$83**, the count-byte, that indicates a 3-bytes length definition name.
The second line is **NIP**'s LFA (**$E2F5**) that contains a *heap-pointer* **02E8** to **DROP**'s NFA, that is the previous definition in dictionary.
The third line is **NIP**'s CFA (**$6900**) which content contains the machine-code routine itself.
Examining the subsequent **DUMP** you should be able to locate **E1** for POP HL, **E3** for EX(SP),HL and **DD E9** for JP(IX) to the inner interpreter address that is compiled by **NEXT** Assembler definition (§3.10).
The bytes that follows – **00 03 E1 F1** – are the beginning of the subsequent definition compiled in dictionary (**TUCK** in this case). Try **$0300 FAR 8 DUMP** to inspect in HEAP its NFA.

Last example is the definition **IF** a colon-definition that compiles a conditional branching in the program flow, defined as follows:

```
: IF    ( -- a 2 ) \ compile-time
    COMPILE 0BRANCH
    HERE 0 , 2 ; IMMEDIATE
```

with the following output:

```
Nfa: EC66 C2
Lfa: EC69 C77 BACK
Cfa: 7FFE 6A21
COMPILE  0BRANCH  HERE 0 , 2 EXIT ok
```

In this case, since **IF** is an **IMMEDIATE** definition, the NFA length-byte is **$C2** instead **$82**.

3.7.1 The Inner-interpreter

At the very core level of any Forth system lies the *Inner Interpreter*, that needs a few information to keep the system itself alive, namely the *Instruction Pointer*, the *Calculator Stack Pointer* and the *Return Stack Pointer*.

The latest version of vForth keeps such fundamental status information in some permanently dedicated Z80 registers:

- **BC** The *Instruction Pointer* that points to the current "xt" being executed
- **SP** the *Data Stack Pointer* that points to the "top-of-stack" element
- **DE** the *Return Stack Pointer* used to track sub-routines calls and keep some value of IP.
- **IX** the address of the *Inner Interpreter* routine, aka "NEXT".

Any other register, such as A, F, H, L and all alternate registers A', F', B', C', D', E', H', L', are free to be used by the low-level definitions.

```
NEXT:
    ld      a, (bc) ; fetch the xt pointed by Instruction Pointer (bc).
    inc     bc
    ld      l, a
    ld      a, (bc)
    inc     bc      ; IP is incremented to point the following xt.
    ld      h, a
    jp      (hl)    ; jump to xt address also known as "CFA"
```

```
CFA:                ; "CFA" contains the actual machine-code to be executed
    call    ENTER   ; and it can be simply a CALL to the piece of code peculiar
                    ; to that kind of definition.
                    ; For example a "colon-definition" has a CALL to ENTER
PFA: ...
```

```
ENTER:
    ex      de, hl  ; put Return Stack Pointer (de) to hl register
    ld      (hl), b ; and save the current value of Instruction Pointer (bc)
    inc     hl
    ld      (hl), c
    inc     hl
    ex      de, hl  ;
    pop     bc      ; Instruction Pointer now contains PFA
    jp      (ix)    ; jump back to NEXT
```

DUMP **a u ---**

Performs a "dump" of a memory area from address `a` for `u` bytes or until **[Break]** is pressed. The value of `u` is always rounded to the nearest greater multiple of 8.

Visualization is always in hexadecimal, current base is maintained. For example:

```
DECIMAL 448 60 DUMP
```

will print the Standard ROM content starting from address 448 ($01C0) for 64 bytes, i.e. the nearest greater multiple of 8 and keeps **DECIMAL** as the current **BASE**.

```
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
```

.WORD a ---

Given a CFA, this definition prints the **ID.** It is used by **SEE** to perform some “decompilation”.

.S ---

Prints the current content of Calculator Stack without destroying its content.
For example, supposing to start with an empty stack,

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

will print

```
0 1 2 3 ok
```

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
```

3.8 Floating-Point Option

This is an simple Floating-Point Option Library that exploits the native standard ZX Spectrum Floating-Point capabilities, with some limitations and differences. This library is not yet compatible with DOT-command version.

To load this Floating-point Option Library you have to use **NEEDS FLOATING**.

To perform any floating-point operations you first need to push one or two numbers onto *Spectrum's calculator stack* using **>W** definition. then you need to call the floating-point calculator using **FOP** definition (that calls RST $28 service routine). Finally, you have to pop the result from Spectrum's calculator stack using **W>** definition.

For example, to define a definition that returns the value of *pi* you can code something like this:

```
: PI
  [  1 0 >W 36 FOP  \ atan(1)
     4 0 >W 04 FOP  \ *4
     W>  ]  DLITERAL
;
```

A floating point in Spectrum's calculator stack takes 5 bytes, instead in Forth Calculator Stack it takes 4 bytes only i.e. the same as a Double-Precision Integer". This means there is some **precision loss**: Maybe in the future I'll be able to fix this fact.

Thinking the floating-number stored in CPU registers HLDE, the sign is the msb of H, so you can check for sign in the integer-way. The exponent+128 is stored in the following 8 bits of HL and the significand/mantissa is stored the remaining bits of HL and 16 bits of DE. The fifth byte of a standard floating-point number is then defaulted to a fixed value.

If the floating-number is an integer between 0 and 65535, then it is kept on stack the same as a double-precision integer. To verify this fact you can type.

```
FLOATING  DECIMAL  65535.0  65537.0  .S
```

that displays

```
65535  0   128  18560
```

where the two single precision integer 65535 and 0 are the representation of 65535.0 while the two integers 128 and 18560 are the internal bit-representation of 65537.0

The integer on TOS always keeps the sign information of the floating-number.

Most of the definitions described below are created using **<BUILDS DOES>** technique.

3.8.1 Floating-point option activation and deactivation

To import the floating point library option you must type **NEEDS FLOATING** and then, you can use **FLOATING** to enable the floating-number interpretation and **INTEGER** to disable it and remain within the integers.

FLOATING ---

Activate floating-point numbers mode. **NMODE** user variable is set to 1.

INTEGER ---

Deactivate floating-point numbers mode. **NMODE** user variable is set to 0.

NO-FLOATING ---

Discard from dictionary the Floating-Point option by executing a specific **MARKER** to restore memory and pointers to the state before **NEEDS FLOATING** were invoked.

3.8.2 Floating-point number conversion

To import the floating point library option you must type **NEEDS FLOATING** and then, you can use **FLOATING** to enable the floating-numbe

D>F d --- fp

Convert a double-precision integer into a floating-number. See **F>D**.

F>D fp --- d

Convert a floating-number into a double-precision integer truncating to the lower integer. It's the opposite of **D>F**.

FLOAT n --- fp

Convert a single-precision-integer into a floating-number. See **FIX.**

FIX fp --- n

Convert a floating-number a into single-precision-integer. It's the opposite of **FLOAT.**

3.8.3 Representation and constants

F>PAD fp --- u

The representation of floating-number fp is stored in **PAD**. The number u is the length of the string.

F.R fp u ---

Prints fp on a field of u characters to video or current stream (See **SELECT**).

F. fp ---

Prints fp to video or current stream (See **SELECT**).

1/2 --- fp

Put on TOS the value 0.5.

PI --- fp

Put on TOS the value of pi.

3.8.4 Arithmetics

F- **fp1 fp2 --- fp3**
Floating point difference: fp3 := fp1 – fp2

F+ **fp1 fp2 --- fp3**
Floating point addition: fp3 := fp1 + fp2

F* **fp1 fp2 --- fp3**
Floating point product: fp3 := fp1 * fp2

F/ **fp1 fp2 --- fp3**
Floating point division: fp3 := fp1 / fp2

FNEGATE **fp1 --- fp2**
Floating point negate, i.e. : fp1 := - fp2

FSGN **fp1 --- fp2**
Floating point sign. Fp2 is the sign of fp1.

FABS **fp1 --- fp2**
Floating point absolute value

F/MOD **fp1 fp2 --- fp3 fp4**
Floating point division and reminder: fp4 is the quotient of fp1 / fp2 and fp3 is the reminder.

F** **fp1 fp2 --- fp3**
Floating point power: fp3 := fp1 ^ fp2

FMOD **fp1 fp2 --- fp3**
Floating point module: fp3 := fp1 mod fp2

F*/ **fp1 fp2 fp3 --- fp4**
Floating point scale operation: fp4 := fp1 * fp2 / fp3 using an intermediate precision of native 5 bytes instead of 4.

F< **fp1 fp2 --- f**
Floating point comparison: f is TRUE if fp1 < fp2, FALSE otherwise.

F> **fp1 fp2 --- fp3**
Floating point comparison: f is TRUE if fp1 > fp2, FALSE otherwise.

F0< **fp1 --- f**

Floating point comparison: f is TRUE if fp1 < 0, FALSE otherwise.

F0> **fp1 --- f**

Floating point comparison: f is TRUE if fp1 > 0, FALSE otherwise.

3.8.5 Log, Exp, Trig

FLN **fp1 --- fp2**

Floating point Natural Logarithm. fp2 := ln(fp1)

FEXP **fp1 --- fp2**

Floating point Exponentation: fp2 := exp(fp1)

FINT **fp1 --- fp2**

Integer truncation. If the floating-number is an integer between 0 and 65535, then it is kept on stack the same as a double-precision integer. **`1.4 FINT`** gives 1.0 but **`-1.4 FINT`** gives -2.0

FSQRT **fp1 --- fp2**

Square root.

FSIN **fp1 --- fp2**

Sine in radians.

FCOS **fp1 --- fp2**

Cosine in radians.

FTAN **fp1 --- fp2**

Tangent in radians

FASIN **fp1 --- fp2**

Arc-sine in radians

FACOS **fp1 --- fp2**

Arc-cosine in radians

FATAN **fp1 --- fp2**

Arc-tangent in radians.

RAD>DEG **fp1 --- fp2**

Convert radians to degrees.

DEG>RAD **fp1 --- fp2**

Convert degrees to radians.

3.8.6 Low-level definitions.

FOP **n ---**

Low-level definition that invokes Floating-Point-Operation `n` .

>W **fp ---**

Takes a floating-point number `d` from Calculator Stack and put to Floating-Pointer Stack.

W> **fp ---**

Takes a floating-point number `d` from Calculator Stack and put to Floating-Pointer Stack.

3.9 Line Editor

The following definitions are available after you include the **EDITOR** vocabulary via **NEEDS EDITOR** or **90 LOAD** using the old fashion way. Most of the logic shown in this section is used by **LED** "The Large file Editor".

The Line Editor has a dozen definitions 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 definition already available in the "core" dictionary. To clear a Screen I foreseen a **BCLEAR** definition, but I left it commented somewehere for now, deeming it too dangerous for my tastes; instead I usually use **BCOPY** from an actually empty Screen. You may type **NEEDS BCOPY.**

The definition **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 7 **BUFFER**s**.**

EMPTY-BUFFERS is another vital definition: 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 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# 1,000 and put a text "\ One thousand screens" on the first line of it. The definition `L` repeats 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 definitions, 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**, **BUFFER**, **INDEX**, **L/SCR**, **LIST**, **LOAD**, **MESSAGE**, **PAD**, **SCR**, **STRM**, **TIB.**

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

Text Input Buffer (keyboard) **TIB**	Parsing Operation	**PAD**	Edit Operations	One **BLOCK BUFFER**	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 assuming it contains a counted-string.

B **---**

"Back" one Screen. This definition 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`. This definition's standard name is **COPY**, but I deemed too close to the Basic keyword **COPY**.

E **n** **---**

"Erase" a row. This definition fills line `n` with spaces. It does **UPDATE**.

H **n** **---**

"Hold" a row in **PAD**. This definition 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 definition inserts line `n` using text in **PAD**. The original line `n` and the following ones are moved down and the last is lost.

L **---**

"List" current Screen. This definition does **SCR @ LIST**.

LINE **n** **--- a**

Leaves 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 definition sets to next Screen by increasing **SCR** and prints it using **LIST**.

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

"Put" a line. This definition 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 definition takes text currently in **PAD** and put it to line n.

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

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

SAVE **---**

Does **UPDATE** and **FLUSH** saving this Screen and all previously modified Screens back to disk.

ROOM **---**

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

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

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

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** phase. Maybe, this definition should be included in the "core" dictionary. n1 is the value of **>IN** and n2 the value of **BLK** as were left by **ERROR**. This way, **WHERE** shows on screen the block number, the line number, the very same line highlighting in "inverse video" the definition that caused the error. When invoked after an error during the loading via **NEEDS** or **INCLUDE**, then the result is poor, because it always reports the row #0 of block #1 due to the way these two definitions are coded.

WIPE **---**

Set content of current Screen to blanks.

3.10 ASSEMBLER vocabulary

The definitions provided by **ASSEMBLER** vocabulary are available after you type **100 LOAD** or after you include it via **NEEDS ASSEMBLER**. Then, you can list this vocabulary using **ASSEMBLER WORDS**.

It's a Zilog Z80 adaptation of the Intel 8080 assembler written by Albert van der Horst available at https://github.com/albertvanderhorst/ciasdis.

To create a new definition using this Assembler you have to use **CODE**. Compilation **STATE** is not modified, so usually you assemble things while in interpret **STATE**. This means you have to pay a little more attention when you need to dereference a definition CFA using **'** (tick). Usually, a **CODE** definition should end with **NEXT** that compiles a **JP (IX)** op-code, and finally **C;** which makes the new definition available for subsequent dictionary search.

Between the starting **CODE** and the ending **C;** instructions are given using their specific op-codes followed by as many parameters as needed. The following table describes all type of arguments used by the op-code list below:

rr|	:	BC|	DE|	HL|	SP|	and when the case		IX|	IY|	and	AF|
r|	:	B|	C|	D|	E|	H|	L|	A|	and	(HL)|	**{ source registers }**
r'|	:	B'|	C'|	D'|	E'|	H'|	L'|	A'|	and	(HL)'|	**{ destination registers }**
f|	:	NZ|	Z|	NC|	CY|	PO|	PE|	P|	M|	**{ flags used by JP, CALL and RET }**	
f'|	:	NZ'|	Z'|	NC'|	CY'|	**{ same as above f| but used by JR }**					
b|	:	0|	1|	2|	3|	4|	5|	6|	7|	**{ bit number operand }**	
a|	:	00|	08|	10|	18|	20|	28|	30|	38|		
d	:	byte displacement **{ used by JR }**									
n	:	byte value (8 bits)									
nn	:	integer value (16 bits)									
aa	:	address									
r	:	Next hardware-register number									

COMMAER's are definitions that enforce a syntax checking while assembling op-codes and parameters.

- **N,** immediate single byte value.
- **NN,** immediate 16 bits value.
- **AA,** memory address value.
- **P,** port address value (16 bits) and, in **NEXTREG** op-code, Next's hardware-register number (8 bits).
- **D,** displacement in relative jump JR.
- **LH,** used by Next's **PUSHN** to compile big-endian 16-bits argument.

Other peculiar definitions are

- **CY|** and **CY'|** to specify “carry-flag” to be different from **C|** or **C'|** “register”.
- **(IX+** and **(IY+** to begin a “index-register” argument and)| to close it.
- **(IX'+** and **(IY'+** same as above, but for destination argument.

You can use **(IY+** operand wherever you can use **(IX+** operand.

Some single byte op-code was renamed to have a better near-Z80 notation. To avoid some Forth-Assembler name clash, it is preferred using some peculiar notation, for example **EXAFAF EX(SP)HL EXDEHL** instead of EX AF, AF' or EX (SP),HL or EX DE,HL. Also, we explicitly say A for all arithmetic/logic opcodes, e.g. **ANDA r|** instead of AND r and so on. IX and IY index-register cause most trouble because they add both a prefix and a displacement and because they can be used in conjunction with CB prefix. In this case we use some custom late-compilation definitions to fix things but relaxing some of the syntax check that the Albert's core provided. Z80N extensions are all ED-prefixed, so the follow the same way introducing a new **LH,** commaer to enforce a better syntax check.

3.10.1 Complete list of Assembler Op-Code

Here's the correspondence between Forth and original Z80 mnemonic, Z80N in **red**.

```
    FORTH ASSEMBLER                         Z80 MNEMONIC
---------------------------------       ---------------------------------
    ADCA      (HL)|                         ADC A, (HL)
    ADCA  (IY+ d )|                         ADC A, (IY+d)
    ADCN         n     N,                   ADC A, n
    ADCA         r|                         ADC A, r
    ADCHL       rr|                         ADC HL, BC/DE/HL/SP
    ADDA      (HL)|                         ADD A, (HL)
    ADDA  (IY+ d )|                         ADD A, (IY+d)
    ADDN         n     N,                   ADD A, n
    ADDA         r|                         ADD A, r
    ADDHL       rr|                         ADD HL, BC/DE/HL/SP
    ADDHL,A                                 ADD HL, A
    ADDDE,A                                 ADD DE, A
    ADDBC,A                                 ADD BC, A
    ADDHL,      nn    NN,                   ADD HL, nn
    ADDDE,      nn    NN,                   ADD DE, nn
    ADDBC,      nn    NN,                   ADD BC, nn
    ADDIY       rr|                         ADD IY, BC/DE/IY/SP
    ANDA      (HL)|                         AND (HL)
    ANDA  (IY+ d )|                         AND (IY+d)
    ANDN         n     N,                   AND n
    ANDA         r|                         AND r
    BIT          b|      (HL)|              BIT b, (HL)
    BIT          b|  (IY+ d )|              BIT b, (IY+d)
    BIT          b|         r|              BIT b, r
    BRLCDE,B                                BRLC DE, B
    BSLADE,B                                BSLA DE, B
    BSRADE,B                                BSRA DE, B
    BSRFDE,B                                BSRF DE, B
    BSRLDE,B                                BSRL DE, B
    CALLF        f|  aa    AA,              CALL Z/NZ/C/NC/PO/PE/P/M, aa
    CALL             aa    AA,              CALL aa
    CCF                                     CCF
    CPA       (HL)|                         CP (HL)
    CPA   (IY+ d )|                         CP (IY+d)
    CPN          n     N,                   CP n
    CPA          r|                         CP r
    CPD                                     CPD
    CPDR                                    CPDR
    CPI                                     CPI
    CPIR                                    CPIR
    CPL                                     CPL
    DAA                                     DAA
    DEC      (HL)'|                         DEC (HL)
    DEC  (IY'+ d )|                         DEC (IY+d)
    DECX        rr|                         DEC BC/DE/HL/SP
    DECX        IX|                         DEC IX
    DECX        IY|                         DEC IY
    DEC         r'|                         DEC r
    DI                                      DI
    DJNZ         d     D,                   DJNZ d
    EI                                      EI
    EX(SP)HL                                EX (SP), HL
```

```
    EX(SP)IY                                EX (SP), IY
    EXAFAF                                  EX AF, A'F'
    EXDEHL                                  EX DE, HL
    EXX                                     EXX
    HALT                                    HALT
    IM0                                     IM 0
    IM1                                     IM 1
    IM2                                     IM 2
    IN(C)   (HL)'|                          IN (c)
    INA         n    P,                     IN A, (n)
    IN(C)      r'|                          IN r, (c)
    INC     (HL)'|                          INC (HL)
    INC (IY'+ d )|                          INC (IY+d)
    INCX       rr|                          INC BC/DE/HL/SP
    INCX       IX|                          INC IX
    INCX       IY|                          INC IY
    INC        r'|                          INC r
    IND                                     IND
    INDR                                    INDR
    INI                                     INI
    INIR                                    INIR
    JP(C)                                   **JP (C)**
    JPHL                                    JP (HL)
    JPIX                                    JP (IX)
    JPIY                                    JP (IY)
    JPF         f|  aa   AA,                JP Z/NZ/NC/C/PO/PE/P/M, aa
    JP              aa   AA,                JP aa
    JRF        f'|   d    D,                JR C/NC/Z/NZ, d
    JR               d    D,                JR d
    LD(X)A     rr|                          LD (BC/DE), A
    LD      (HL)'|          r|              LD (HL), n
    LDN     (HL)'|    n     N,              LD (HL), r
    LDN (IY'+ d )|    n     N,              LD (IY+d), n
    LD(IY+    d )|          r|              LD (IY+d), r
    LD()A            aa    AA,              LD (nn), A
    LD()X      rr|   nn    AA,              LD (nn), BC/DE/SP
    LD()IY           aa    AA,              LD (nn), IY
    LD()HL           aa    AA,              LD (nn), HL
    LDA(X)     rr|                          LD A, (BC/DE)
    LDA()            aa    AA,              LD A, (aa)
    LDAI                                    LD A, I
    LDAR                                    LD A, R
    LDX        rr|   nn    NN,              LD BC/DE/HL/SP, nn
    LDX()      rr|   nn    AA,              LD BC/DE/SP/IY, (aa)
    LDHL()           aa    AA,              LD HL, (aa)
    LDIA                                    LD I, A
    LDX        IY|   nn    NN,              LD IY, nn
    LDRA                                    LD R, A
    LDSPHL                                  LD SP, HL
    LDSPIX                                  LD SP, IX
    LDSPIY                                  LD SP, IY
    LD         r'|     (HL)|                LD r, (HL)
    LD         r'|  (IY+ d )|               LD r, (IY+d)
    LD         r'|          r|              LD r, r
    LDN        r'|    n     N,              LD r, n
    LDD                                     LDD
    LDDR                                    LDDR
    LDDRX                                   **LDDRX**
    LDDX                                    **LDDX**
    LDI                                     LDI
    LDIR                                    LDIR
```

LDIRX	**LDIRX**
LDIX	**LDIX**
LDPIRX	**LDPIRX**
LDWS	**LDWS**
MIRRORA	**MIRROR A**
MUL	**MUL**
NEG	NEG
NEXTREGA r P,	**NEXTREG r, A**
NEXTREG r P, n N,	**NEXTREG r, n**
NOP	NOP
ORA (HL)|	OR (HL)
ORA (IY+ d)|	OR (IY+d)
ORN n N,	OR n
ORA r|	OR r
OTDR	OTDR
OTIR	OTIR
OUT(C) (HL)'|	OUT (c), 0
OUT(C) r'|	OUT (c), r
OUTA n P,	OUT (n), A
OUTD	OUTD
OUTI	OUTI
OUTINB	**OUTINB**
PIXELAD	**PIXELAD**
PIXELDN	**PIXELDN**
POP AF|	POP AF
POP rr|	POP BC/DE/HL
POP IX|	POP IX
POP IY|	POP IY
PUSH rr|	PUSH BC/DE/HL/AF
PUSH IX|	PUSH IX
PUSH IY|	PUSH IY
PUSHN nn LH,	**PUSH nn**
RES b| (HL)|	RES b, (HL)
RES b| (IY+ d)|	RES b, (IY+d)
RES b| r|	RES b, r
RES b| r| (IY+ d)|	RES r, b, (IY+d)
RET	RET
RETF f|	RET Z/NZ/C/NC/PO/PE/P/M
RETI	RETI
RETN	RETN
RL (HL)|	RL (HL)
RL (IY+ d)|	RL (IY+d)
RL r|	RL r
RL r| (IY+ d)|	RL r, (IY+d)
RLA	RLA
RLC (HL)|	RLC (HL)
RLC (IY+ d)|	RLC (IY+d)
RLC r|	RLC r
RLC r| (IY+ d)|	RLC r, (IY+d)
RLCA	RLCA
RLD	RLD
RR (HL)|	RR (HL)
RR (IY+ d)|	RR (IY+d)
RR r|	RR r
RR r| (IY+ d)|	RR r, (IY+d)
RRA	RRA
RRC (HL)|	RRC (HL)
RRC (IY+ d)|	RRC (IY+d)
RRC r|	RRC r
RRC r| (IY+ d)|	RRC r,(IY+d)
RRCA	RRCA

RRD					RRD
RST	a|				RST n
SBCA	(HL)|				SBC A, (HL)
SBCA	(IY+ d)|				SBC A, (IY+d)
SBCN	n	N,			SBC A, n
SBCA	r|				SBC A, r
SBCHL	rr|				SBC HL, BC/DE/HL/SP
SCF					SCF
SET	b|	(HL)|			SET b, (HL)
SET	b|	(IY+ d)|			SET b, (IY+d)
SET	b|	r|			SET b, r
SET	b|	r|	(IY+ d)|		SET r, b, (IX+d)
SETAE					**SETAE**
SL1	(HL)|				SL1 (HL)
SL1	(IY+ d)|				SL1 (IY+d)
SL1	r|				SL1 r
SL1	r|	(IY+ d)|			SL1 r, (IY+d)
SLA	(HL)|				SLA (HL)
SLA	(IY+ d)|				SLA (IY+d)
SLA	r|				SLA r
SLA	r|	(IY+ d)|			SLA r, (IY+d)
SRA	(HL)|				SRA (HL)
SRA	(IY+ d)|				SRA (IY+d)
SRA	r|				SRA r
SRA	r|	(IY+ d)|			SRA r, (IY+d)
SRL	(HL)|				SRL (HL)
SRL	(IY+ d)|				SRL (IY+d)
SRL	r|				SRL r
SRL	r|	(IY+ d)|			SRL r, (IY+d)
SUBA	(HL)|				SUB (HL)
SUBA	(IY+ d)|				SUB (IY+d)
SUBN	n	N,			SUB n
SUBA	r|				SUB r
SWAPNIB					**SWAPNIB**
TESTN	n	N,			**TEST n**
XORA	(HL)|				XOR (HL)
XORA	(IY+ d)|				XOR (IY+d)
XORN	n	N,			XOR n
XORA	r|				XOR r

3.10.2 Assembler Example - Checksum

This definition calculates the checksum of addresses between `a` and `a+u` inclusive. The algorithm simply adds each byte (mod 256).

```
NEEDS ASSEMBLER
CODE CHECKSUM ( a u -- b )
      exx                  \ needs this to preserve BC and DE
      pop      bc|         \ counter
      pop      hl|         \ start address
      xora      a|         \ zero to accumulator
HERE  adda   (hl)|         \ add mod 256
      cpi                  \ increment HL, decrement BC
      jpf      pe|  AA,    \ loop back if BC is not zero
      ld       c'|    a|
      push     bc|         \ return result
      exx                  \ restore register BC and DE
      NEXT
C;                         \ this performs SMUDGE
```

Up to now, there is no labels, since the same functionality is given by a clever use of `HERE` and stack manipulation, and because routines are kept short.

For example:

```
DECIMAL 448 60 CHECKSUM .
67 ok
```

3.10.3 Assembler Example - WHITE-NOISE from Egghead 3

This is an example taken from “Egghead 3” that uses the first bytes of ROM as a pseudo-random number list used to produce a white noise. The source is available in Screen# 166 of standard BLOCK file.

Labels seem a little more complicated, but jus remember each **HERE** is resolved using **BACK,** without confusion.

Also, this is an example of *forward reference* resolved directly patching the code just compiled: the address to the subroutine is patched by saving the instruction address (the first `HERE`) and then using the sequence `HERE SWAP 1+ !` to patch the previous CALL address.

```
NEEDS ASSEMBLER
DECIMAL
CODE WHITE-NOISE ( -- )
      di
      exx
HERE  call  0000  AA,         \ 0000 is patched here below (*)
      exx
      ei
      next
\
\ subroutine
HERE SWAP 1+ !                \ patch the previous CALL  (*)
      ldn   e'|   250   NN,
      ldx   hl|   300   NN,
HERE  push  de|               \ ←——————... (†) main loop
      ldn   b'|    32   N,
HERE    push      bc|         \ ←
        ld  a'|   (hl)|       \
        incx      hl|         \
        andn      248   N,    \
        outa      254   P,    \
        ld  a'|   e|          \
        cpl                   \
HERE        dec    a'|        \ ←
            jrf   nz'| BACK,  \
        pop   bc|             \
      djnz  BACK,             \
      pop   de|
      ld    a'|    e|
      subn  24     N,
      cpn   30     N,
      retf   z|               \ return from routine
      retf  cy|               \ return from routine
      ld    e'|    a|
      cpl
HERE  ldn   b'|    40   N,    \ ←
HERE                          \ ←
        djnz  BACK,           \
      dec   a'|               \
      jrf  nz'|   BACK,       \
      jr    BACK,             \ ———→ ... (†)
```

To test it you need to set CPU's speed at 3.5 MHz via **SPEED!**.

```
NEEDS SPEED!
0 SPEED! WHITE-NOISE
```

3.10.4 Assembler Example "ms" millisecond delay

This definition waits at least `u` milliseconds, with `u` < 8192, indentation helps to glimpse the various nested loops. The outer loop lasts exactly 3,500 T-states so that it can be used to produce milliseconds delays on a standard 3.5 MHz CPU. Some CPU time is spent to compute the correct number of iterations.
Also, this is an example of *forward relative-jump* **HOLDPLACE** resolved via **DISP,**

```
DECIMAL
CODE ms ( u -- )
    exx                                     \ preserve BC and DE
    pop     hl|
    ld      a'|    l|
    ora      h|
    jrf     z'| HOLDPLACE                   \ skip to end when u is zero (*)

        ldx     bc|  $243B  NN,             \ get current CPU speed
        ldn     a'|     07   N,             \ to calculate the correct
        out(c)  a'|                         \ delay
        inc     b'|
        in(c)   a'|
        andn    3   N,
                                            \ this is a begin-while-repeat loop.
        HERE                                \ keep this address for later (♠)
        jrf     z'| HOLDPLACE               \ condidional forward to (♦)
            addhl   hl|                     \ double HL based on CPU speed
            dec     a'|
            jr  SWAP BACK,                  \ uncontidional loop back to (♠)
        HERE DISP,                          \ resolve forward jrf (♦)

        HERE                                \ begin of outer loop (♣)
            nop                             \    4 T
            ldn   b'| 204 N,                \    7 T
            HERE                            \        inner loop (♥)
                nop                         \ 3463 T = (4+13)*203 + (4+8)
            djnz  BACK,                     \        back to inner loop (♥)
            decx  hl|                       \    6 T
            ld    a'|   l|                  \    4 T
            ora    h|                       \    4 T
        jrf  nz'| BACK,                     \   12 T    ( -5 T on final loop )
                                            \ 3500 T total
                                            \ repeat outer loop (♣)

    HERE DISP,                              \ resolve skip-to-end when zero (*)

    exx
    NEXT
C;
```

HOLDPLACE --- a1

ALLOT the next byte as placeholder of a relative-jump displacement. The address `a1` points to this placeholder and should be resolved by a subsequent **DISP,** or a derived definition such as **BACK,**

DISP, a1 a2 ---

Compute the displacement from `a2` to `a1` and compile it to address `a2` as displacement of a relative-jump op-code. The following snippet implements an IF-THEN-ELSE phrase in Assembler:

```
cpn    $60 N,
jrf  cy'|  HOLDPLACE              \ if lowercase
     subn  $20  N,                \ quick'n'dirty uppercase
HERE DISP,                        \ aka THEN,
```

The following example implements a complete IF-THEN-ELSE phrase that check “carry-flag”:

```
jrf  cy'|  HOLDPLACE              \ IF,
     ldx   hl|    1    NN,
jr   HOLDPLACE  SWAP HERE DISP,   \ ELSE,
     ldx   hl|   -1    NN,
HERE DISP,                        \ THEN,
```

BACK, a1 ---

Compute the displacement from `HERE` to `a1` and compile it to address `HERE` as displacement of a relative-jump op-code. The following example implements a BEGIN-UNTIL loop in Assembler:

```
\ Wait for a standard key-press.
HERE                              \ BEGIN,
    bit      5| (iy+ 1 )|         \ FLAGS (5C3A+1)
jrf     z'| BACK,                 \ UNTIL,
```

3.10.5 Syntax check error messages

During assembly, the following error messages may be reported

#26 Error at postit time.
The previous instruction is incomplete.
For example:

```
ldn     a'|    0   N,
ldn     b'|    0      \  missing N,
nop
```

reports: **nop? Error at postit time.** meaning the previous instruction is incomplete.

Another example:

```
call    $0DAF       \  missing AA,
nop
```

reports: **nop? Error at postit time.** meaning the previous instruction is incomplete.

#28 Unexpected fixup/commaer.
There is an unexpected argument.
For example:

```
ldn   a'| $00 n,
ldn    b| $00 n,    \  should be  b'|  instead of  b|
```

reports: **b|? Error at postit time.**

Another example:

```
inc   a'|
inc    b|           \  should be  b'|  instead of  b|
```

reports: **b|? Error at postit time.**

#29 Commaer data error.
This is the most common error issued whenever the wrong kind of operand is used
For example:

```
push   nc|     \  cannot be used with nc|
```

Other messages are used internally by the vocabulary itself
#27 Inconsistent fixup.
#30 Commaer wrong order.
#31 Programming error.
#33 Programming error.

3.11 Raspberry Pi Zero accelerator

If you have the Accelerated version of the ZX Spectrum Next, or a Raspberry Pi Zero installed, then you can experiment with it via vForth system. Some useful definitions are available after **`NEEDS RPI0`** and you'll be able to start a two-way communication stream using **`RPI0`** via simple UART I/O.

The rationale is as follows. The main loop continuously polls both the RPi0 UART for any incoming data, and the keyboard for user input, any key pressed is immediately transmitted to RPi0, and any byte received from RPi0 is immediately sent to screen.

[ENTER] key has a special behavior such that, once "0x0D" is transmitted to RPi0, up to 8192 bytes are fast-read from UART to a RAM buffer and – only then – slowly sent to screen allowing long output without data-loss.

Other keys works as follows:

[BREAK]	quit to Forth prompt
[CAPS LOCK]	toggles caps lock and sends nothing.
[>=]	transmits 0x03, ETX or ^C that helps to emulate CTRL-C key-press.
[TRUE VIDEO]	transmits 0x04, EOT or ^D that normally produces a "normal exit" from whatever you where in.
[INV VIDEO]	transmits 0x05, ENQ or ^E
[EDIT]	transmits 0x07, BEL or ^G
[DELETE]	transmits 0x08, BS or ^H and it is the normal back-space key.
[←]	transmits 0x08, BS or ^H same as [DELETE]
[→]	transmits 0x09, LF or ^I
[↓]	transmits 0x0A, BS or ^J
[↑]	transmits 0x0B, VT or ^K
[EXTENDED]	transmits 0x0E, S0 or ^N
[GRAPH]	transmits 0x0F, SI or ^O
[<>]	transmits 0x18, CAN or ^X that helps to emulate CTRL-X key-press.
[<=]	transmits 0x1A, SUB or ^Z that helps to emulate CTRL-Z key-press.
[AT]	transmits 0x1B ESC or ^[that helps to emulate escape sequences

Remaining ASCII characters **`~ | \ [ ] { }`** are produced via SYMBOL-SHIFT as usual.

Typing **`RPI0`** RPi0's UART first is correctly configured and when you get the prompt "**SUP>**" you know you're in.

If you hit [BREAK] you immediately quit back to Forth prompt leaving RPi0 in whatever state it is, which usually you can re-enter again using just **`TERM`** instead of the whole route of **`RPI0`**.

Once you're "connected" to RPi0, as an experimental feature, you can send a line of text back to Forth to be interpreted via **`INTERPRET`**: this behavior is achieved via **#** character that it's usually used as a comment within Linux shell and even Python or Perl. This feature is disabled or enabled setting **`UART-META`** variable to zero or non-zero.

3.11.1 RPi0 high-level communication

Here is how to interact with RPi0 command prompt once it is initialized via **RPI0-INIT**. The following high-level definitions must be imported using **NEEDS**.

ASK --- **n**

Accept the following text from current input stream (keyboard or load from file or Screen), up to the next CR or LF, and send it to RPi0 terminal followed by a $0D character to start whatever command is sent, then collecting to **PAD** up to 1024 bytes reply provided by RPi0 and returning **n** as the length of such a reply. For example, to ask which Bash-shell version RPi0 is currently running, you can type from keyboard:

```
ASK  echo $BASH_VERSION  ↵
DISPLAY

4.4.12(1)-release
```

Notice you have to hit [ENTER] after the first line to let know ASK definition where the command ends, and in another source line examine PAD and display.

DISPLAY **n** ---

Counterpart of **ASK**, this definition uses emits n characters-long string currently stored at **PAD**.

SEND **a1 n1 a2 n2** --- **n3**

Send **n2** bytes-long string stored at address **a2** to RPi0, collecting up to **n2** replied bytes to address **a1**, returning **n3** as the length of such a reply. Used by **ASK**. For example, to calculate 2^400 using Python we can do

```
NEEDS S"
: TEST
  HERE 200
  S" python -c 'print 2**400' " SEND
  HERE SWAP TYPE
;

CR TEST CR
2582249878086908589655919172003011874329705792829223512830659356540647622016841194
6
29645353280137831435903171972747493376
ok
```

In this case we temporarily use 200 bytes at **HERE** to receive the answer that is immediately displayed.

THIS --- **a n**

Accept text from current input stream preparing the string to be sent to RPi0. Used by **ASK**.
This definition is much like **WORD** with the difference that no delimiter is given and the rest of the line is assumed as the string which address **a** and length **n** is given as result.
The example given in **ASK** definition can be moved into a file or a Screen, and later used via **INCLUDE** or **LOAD**.

```
Scr# 899
  0 ( Example for THIS )
  1  ASK  echo $BASH_VERSION
  2  DISPLAY
```

Then you can type

```
899 LOAD
4.4.12(1)-release
```

3.11.2 RPi0 UART low-level communication

The following constants are defined for ease of use:

`$5C08 CONSTANT UART-LASTK`	system variables alias which stores newly pressed key.
`$5C78 CONSTANT UART-FRAMES`	system variables alias frame counter incremented every 20ms.
`$5C6A CONSTANT UART-FLAGS2`	system variables alias (Bit 3 set when CAPS SHIFT or CAPS LOCK is on)
`$143B CONSTANT UART-RX-PORT`	hardware receive-data I/O port
`$133B CONSTANT UART-TX-PORT`	hardware transmit-data I/O port
`$153B CONSTANT UART-CT-PORT`	hardware control-data I/O port

Here is a list of low-level definitions.

?UART-BYTE-READY --- f

Check if UART has received a byte: true flag when byte ready, false elsewhere

?UART-BUSY-TX --- f

Check if UART is busy sending a byte, non-zero when transmitter is busy sending a byte

RPI0 ---

Main definition to connect to Raspberry Pi Zero. It performs **RPI0-INIT** to setup I/O and Baud-Rate, **TERM-INIT** to reduce font-size to let 85 columns display. A this point, the main loop is controlled by **TERM**.

RPI0-INIT ---

This definition should be invoked once to set-up the correct baud-rate so that the following setup is performed:

1. Run at maximum speed: `3 7 REG!`
2. Select RPi0 UART sending a suitable value to the control port: `$40 $153B P!`
3. Ask the hardware NextREG 17 ($11) “Video Timing” to determine the actual CPU speed, then compute a 14 bits *prescalar* dividing the actual MHz clock speed by the 115,200 baud rate obtaining a 14 bits *prescalar* to be sent to UART receive port split in two parts:

```
DUP      $7F AND $143B P!     \ send low 7 bits of 14 bits
7 RSHIFT $80 OR  $143B P!     \ send high 7 bits of 14 bits
```

4. Enable peripheal: `$30 $A0 REG!          \ PI Peripheal enable`

RPI0-SELECT ---

Select Raspberry PI Zero UART and set Baudrate as shown in §Errore: sorgente del riferimento non trovata above.

UART-1ST-TIMEOUT --- a

Variable used by `UART-RX-BURST` to timeout the first byte. At startup it's defaulted at 50.000 that is 200 ms.

UART-2ND-TIMEOUT --- a

Variable used by `UART-RX-BURST` to timeout any subsequent byte after the first. At startup it's defaulted at 5.000 that is 20 ms.

UART-RX-BYTE --- b | 0

Accept a byte if available, 0x00 if no byte is available.

UART-RX-TIMEOUT n --- c | 0

Wait for a byte within timeout expressed in ms. If no byte is available, 0 is returned.

UART-RX-BURST a n1 --- n2

Accept from UART up to `n1` bytes and store them at address `a` allowing for a fixed timeout: Timeout to receive the first byte is set at 200 ms, timeouts for the next bytes is set at 20 ms, this means we assume RPi0 to reply within 200 ms to any command we issue and any subsequent delay greater than 20 ms ends data collection.

At 115.200 Baud, one bit duration is 8.68 microseconds a 512 bytes-buffer is filled in about 4.4 ms but a 512 bytes burst-read takes 200 T per byte (102400) plus enter-exit time that at 28 MHz is about 3.7 ms so we should keep draining the buffer faster than it fills up.

UART-SET-BAUDRATE d ---

Set baud rate to `d`.

UART-SET-PRESCALAR n ---

Send 14 bits *prescalar* `n` to UART.

UART-TX-BYTE b ---

There is no transmit buffer so program must make sure the last transmission is complete before sending another byte. Wait until transmission is possible or Break is pressed. There are a few short-cut to easily send some control characters, such as `UART-SEND-EOT`, `UART-SEND-ETX`, `UART-SEND-CR`, `UART-SEND-LF`.

UART-SEND-TEXT a n ---

Send a string of text to RPi0 by repeatedly invoking `UART-TX-BYTE`.

UART-WAIT-B b ---

wait for a specific byte (or [BREAK] key to bailout).

UART-WAIT-CR-LF ---

Wait for a CR-LF sequence.

UART-WAIT-FOR a n ---

Wait for a specific **n** bytes-long string stored at address **a**. It uses `UART-WAIT-B`.

UART-WAIT-PROMPT ---

Wait for a specific string "SUP> " to be received. There is no timeout: the only way to stop the wait is pressing [BREAK]

TERM ---

Wait for a specific string "SUP> " to be received. There is no timeout: the only way to stop the wait is pressing [BREAK]

4 Technical specifications

4.1 Z80 CPU Registers

CPU Registers are used in the in the following way:

AF – Available for normal operations.
BC – *Forth Instruction Pointer:* should be preserved on Enter-and-Exit a definition and during ROM/OS calls.
DE – *Forth Return Stack Pointer*: should be preserved on Enter-and-Exit a definition and during ROM/OS calls.
HL – Available as *Work Register*

SP – *Forth Data Stack Pointer*

AF'– Available, sometime used for backup purpose
BC'– Available, used in I/O operations or tricky definitions
DE'– Available, often the Low part when used for 32-bit manipulations
HL'– Available, often the High part when used for 32-bit manipulations

IX – Used to point to the Forth "inner-interpreter": **i**t should be preserved during ROM/OS calls.
IY – Used by ZX System, must be preserved to let the standard keyboard to be served during Interrupts.

4.2 Single Cell 16 bits Integer Number Encoding

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

```
                         H            L
16 bit:      HL:    | sbbb 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 the "little-endian" way, that is the lower address has the least significant part, in the L register. The byte at higher address has the most significant part, in the H register, as usual for the Zilog Z80 processor.

4.3 Two cells 32 bits Integer Number Encoding

The second integer format requires two *integers* to form a 32 bits number, referred to as *double* or *long*, that allows integers between –2,147,483,648 and +2,147,483,647 where the sign is kept on the most significant bit of the first *integer*.

Imagine a *double precision integer* stored in CPU register in this way:

```
                 H'          L'          D'          E'
32 bits:  | sbbb 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 the Calculator Stack the double-integer integer requires four contiguous bytes split into two integers 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, in the higher address, that is the second element from the 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)

To adhere to the Standard, in RAM it is kept as L' H' E' D'. See how `2@` and `2!` are defined to understand this fact.

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

4.4 Double Cell Floating-Point Number Encoding

There is another optional format that use 32 bits as a *double-precision integer*, but all bits are used in a different way to allows representation a *floating point number* approximately between ±0.3E-38 and ±1.7E+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 or significand. The sign in this position allows you reusing most of the semantics of *double-precision integers* to deal with the sign.

```
                 H'          L'          D'          E'
32 bits f.p.:  | sxxx xxxx | xbbb bbbb | bbbb bbbb | bbbb bbbb |
```

See Floating-Point Option section for more details.

4.5 Single Cell 16 bits Heap Pointer Address Encoding

This is Spectrum Next's peculiar 16 bits *Heap Pointer* Address Encoding that leverages on MMU7 i.e. Z80 memory space addresses between $E000 and $FFFF. The three most significant bits represent an 8kibyte-page between 32 and 39 ($20 and $27), lower bits are taken as offset from $E000. A specific definition `>FAR` takes care of converting an heap-pointer address to an $E000 offset and paging to MMU7 the correct 8kibyte of physical RAM. Any NextZXOS call and most of I/O operations restore page 1 at MMU7 , so in most cases data stored in Heap has to be moved to PAD before being used elsewhere.

```
                                  H           L
16 bit:    HL:                | pppb bbbb | bbbb bbbb |

                  |    Page     |        Offset         |
                  | 0010 0ppp   | 111b bbbb | bbbb bbbb |
```

4.6 Dictionary memory structure

From version 1.7, a dictionary definition storage is split into two parts, the name-space in `HEAP` (see §**6.3)**, and the code-space in main memory. The name-space of a new definition begins at *heap-pointer address* (§**4.5**) given by **`HP`** (the current Heap Pointer), and the code-space part of it begins at **`HERE`** as usual, but all data previously contained in NFA, LFA and CFA are replaced by a single "backward" heap-pointer to point back to name-space location, as a "mirror".
The linked-list formed by each LFA uses *heap-pointers* to be decoded to actual RAM pointer via `>FAR`.

For example the two contiguos definitions **`SWAP`** and **`DUP`** appears in memory as follow (as per build 20231014, since it's perfectly possible that a different build shows different addresses):

```
Heap memory:
NFA    0302   | 84 | 53 57 41 D0 |       | len |  'S'  'W'  'A'  'P'  |
LFA    0307   | F9 02 |                 | 02F9 |   heap-pointer to previous NFA definition
CFA    0309   | 0F 69 |                 | 690F |   xt memory-address of this definition

Main memory:
Mirror 690D   | 09 03 |                 | 0309 |   backward-heap-pointer to  SWAP's  CFA
  xt   690F   |   E1   |      E3       |    E5    |  DD  E9  |
              | pop hl | ex (sp),hl    | push hl  |  jp (ix) |

Heap memory:
NFA    030B   | 83 | 44 55 D0 |          | len |  'D'  'U'  'P'  |
LFA    030F   | 02 03 |                 | 0302 |   heap-pointer to previous NFA definition SWAP
CFA    0311   | 16 69 |                 | 6916 |   xt memory-address of this definition

Main memory:
Mirror 6914   | 11 03 |                 | 0311 |   backward-heap-pointer to  DUP's  CFA
  xt   6916   |   E1   |    E5   |    E5   |  DD E9  |
              | pop hl | push hl | push hl | jp (ix) |
```

You can verify yourself all of that by typing

```
SEE SWAP
Nfa: E302 84
Lfa: E307 2F8 TUCK
Cfa: 690F E5E3
690F E1 E3 E5 DD   E9 11 03 E1   ace]i  a

$E302 9 DUMP
E302 84 53 57 41   D0 F9 02 0F    SWAPy
E30A 69 83 44 55   D0 02 03 16   i DUP

$690D 2 DUMP
690F 09 03 E1 E3   E5 DD E9 11     ace]i

SEE DUP
Nfa: E30B 83
Lfa: E30F 302 SWAP
Cfa: 6916 E5E5
6916 E1 E5 E5 DD   E9 19 03 F1   aee]i  q

$E30B 8 DUMP
E30B 83 44 55 D0   01 03 25 69    DUP  %i

$6914 2 DUMP
6914 11 03 E1 E5   E5 DD E9 19     aee]i
```

5 Error messages

Error messages strings are stored at Screens from # 4 to # 7 that are therefore reserved. See **MESSAGE** (§ 6.1).

Code	Message	Meaning
#0	is undefined.	Definition not found or number conversion invalid with this **BASE**
#1	Stack is empty.	Returning to “ok” prompt, the interpreter detects this situation
#2	Dictionary full.	
#3	No such a line.	Line number should be between 0 and 15
#4	has already been defined.	Creating a definition whose name already exists.
#5	Invalid stream.	Used by Microdrive version
#6	No such a block.	Block number should be between 1 and **#SEC**
#7	Stack full.	The interpreter detects this situation
#8	Old dictionary is full.	
#9	Tape error.	Not used anymore
#10	Wrong array index.	
#11	Invalid floating point.	Issued during floating invalid operation
#12	Heap full.	
#13	Division by zero.	Issued during floating invalid operation
#14	Patching the wrong word.	
#17	Can't be executed.	
#18	Can't be compiled.	
#19	Syntax error.	Mismatch in a structure.
#20	Bad definition end.	Mismatch in **DO-LOOP** structure
#21	is a protected word.	Trying to forget to an address below **FENCE**
#22	Aren't loading now.	Issued by a misplaced **-->**.
#23	Forget across vocabularies.	You have to use **DEFINITIONS** beforehand
#24	RS loading error.	
#25	Cannot open stream.	
#26	Error at postit time.	Issued by **ASSEMBLER** dictionary operation
#27	Inconsistent fixup.	Issued by **ASSEMBLER** dictionary operation
#28	Unexpected fixup/commaer.	Issued by **ASSEMBLER** dictionary operation
#29	Commaer data error.	Issued by **ASSEMBLER** dictionary operation
#30	Commaer wrong order.	Issued by **ASSEMBLER** dictionary operation
#31	Programming error.	Issued by **ASSEMBLER** dictionary operation
#33	Programming error.	Issued by **ASSEMBLER** dictionary operation
#34	Checksum error.	Issued by **ASSEMBLER** dictionary operation
#38	Not a BMP file.	
#39	NextZXOS Opendir error.	
#40	NextZXOS Out of memory.	
#41	NextZXOS Open error.	Issued to signal **F_OPEN** fails to open a file
#42	NextZXOS Close error.	Issued to signal **F_CLOSE** fails to close a file
#43	File not found.	Issued by **INCLUDE** or **NEEDS**
#44	NexZXOS doscall error.	Issued to signal a generic NextZXOS doscall error
#45	NextZXOS pos error.	Issued to signal **F_SEEK** fails to change current file position
#46	NextZXOS read error.	Issued to signal **F_READ** fails to read from file
#47	NextZXOS write error.	Issued to signal **F_WRITE** fails to write to file
#50	Incorrect result.	Used by **TESTING** suite
#51	Wrong number of result.	Used by **TESTING** suite
#52	Cell number before '->' does not match ...}T spec.	Used by **TESTING** suite
#53	Cell number before and after '->' does not match.	Used by **TESTING** suite
#54	Cell number after '->' below ...}T does not match.	Used by **TESTING** suite

6 The Dictionary

6.1 The "core" dictionary

'0x00' --- (immediate)

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

! n a ---

stores an integer n in the memory cell at address a and a + 1. Pronounced "store".
Zilog Z80 microprocessor is a little-endian CPU that holds lower byte at lower address and higher byte in the higher address.

!CSP ---

saves the value of SP register in CSP user variable. It is used by : and ; for syntax checking. ~~Also, CASE use it for the same purpose.~~

 # d1 --- d2

From a double-precision integer d1 it produces the next ASCII character to be put in an output string using HOLD. The number d2 is d1 divided by the current BASE and is kept for subsequent elaborations. This definition is used between <# and #> .
See also #S.

#> d --- a u

terminates a numeric conversion started by <# . This definition removes **d** and leaves the values suitable for TYPE.
See also # and #S.

#BUFF --- n

Constant, the number of available buffers. This build has 7 buffers located at address between FIRST @ and LIMIT @.

#S d1 --- d2

This definition 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 maximum number of a BLOCK, so that half #SEC is the maximum number of a Screen. This system comes with a file "!Blocks-64.bin" that keeps persistency of all BLOCKs. Try #SEC LIST.

' --- **cfa**

Pronounced "tick". Used in the form

' cccc

this definition leaves the **cfa** of definition `cccc`, that is its **xt** or the value to be compiled or passed to `EXECUTE`. If the definition `cccc` is not found after the `CURRENT` and `CONTEXT` search phases, then an error **#0** is raised, that is the message "`cccc` is undefined". In a previous version of this Forth, this definition returned **pfa**: we changed this previous standard to return **cfa**.

(--- **(immediate)**

Enclose a comment. Used in the form

(cccc)

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 the end of line.

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

This is the primitive low-level definition compiled by `+LOOP`.

(.") ---

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

(;CODE) ---

This is the primitive definition compiled by **;**`CODE`. It over-writes the **cfa** of `LATEST` definition to make it point to the machine code starting from the following address.

(?DO) ---

This is the primitive definition compiled by `?DO`.

At compile-time it compiles the **cfa** of `(?DO)` followed by an offset as for `BRANCH` used to jump after the whole `?DO ... LOOP` structure in case the limit equals the initial index, otherwise it is equivalent to `(DO)`.

(?EMIT) **c1** --- **c2**

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

- $06 → print-comma
- $07 → bell rings
- $08 → back-space
- $09 → tabulator
- $0D → carriage return
- $0A → new line (emitted as a $0D on the fly)

For not listed character, `c2` is equal to `c1`.

(ABORT) ---

Definition executed in case of error issued by `ERROR` when `WARNING` contains a negative number. This definition usually executes `ABORT` but can be patched with some user defined one at the PFA of `(ABORT)`.

(COMPARE)   a1 a2 n   --   b

This definition performs a lexicographic compare of `n` bytes of text at address `a1` with `n` bytes of text address `a2`. The compare is case-sensitive or case-insensitive based on the last execution of `CASEON` and `CASEOFF`.
When executed, this definition returns a numeric 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
See also `CASEON` and `CASEOFF`.

(DO)   ---

This is the primitive compiled by `DO`.

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

Searches in the dictionary starting from address `a2` a definition which text name is kept at address `a1`; returns a `cfa`, the first byte `b` of nfa and a `tf` on a successful search; elsewhere a `ff` only.
The search is case-sensitive or case-insensitive based on the last execution of `CASEON` and `CASEOFF`.
Address `a2` must be the `nfa` of the first definition involved in the search in the vocabulary.
In previous versions of this Forth, it returned a `pfa`, we changed our mind, better a `cfa`.
Byte `b` keeps the length of the found definition 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`.
See also `CASEON` and `CASEOFF`.

(LEAVE)   ---

Direct procedure compiled by `LEAVE` that discards the current DO-LOOP frame and executes an unconditional jump. The memory cell following `(LEAVE)` contains the offset to be relatively added to the Instruction Pointer to jump after the corresponding `(LOOP)` or `(+LOOP)`.

(LINE)   n1 n2   --- a b

Retrieves Line `n1` of Screen `n2` and send it to buffer. It returns the address `a` within the buffer and a counter `b` that is `C/L` (=64) meaning a whole line.

(LOOP)   ---

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

(MAP)   a2 a1 n c1 --- c2

Translate character `c1` using mapping string `a1` and `a2`. If `c1` is present within string `a1` then the corresponding position within string `a2` is taken as translation. If `c1` is not present within string `a1`, then it is not translated, and `c2` remains equal to `c1`. `n` is the length of strings `a1` and `a2`.

For example, the following code is used to translate the characters **: ? / * | \ < > "** – that are illegal when used in a filename – to the corresponding legal characters chosen among **% ^ % & $ _ { } ~** .

```
here to ndom^     (   : ? / * | \ < > "   )
  char :  c,   char ?  c,   char /  c,   char *  c,
  char |  c,   char \  c,   char <  c,   char >  c,   char "  c,
```

```
here to ncdm^     (    % ^ % & $ _ { } ~    )
  char _  c,    char ^  c,    char %  c,    char &  c,
  char $  c,    char _  c,    char {  c,    char }  c,    char ~  c,

: needs-ch ( a -- ) \ Replace illegal characters in filename counted-zstring
  count bounds
  Do
      [ ncdm^ ] Literal
      [ ndom^ ] Literal
      9 i c@ (map) i c!
  Loop
;
```

(NEXT) --- a

Constant. It is the address of "next" entry point for the **Inner Interpreter**. When creating a new definition using machine code, the last op-code should be an unconditional jump to this address. If the newly created definition wants to return an *integer* value on TOS, it has to do it beforehand ~~it should jump to the previous address; and if it wants to return a *double-precision integer* value, it should jump to the next previous one~~.

This Forth implementation *always* keeps `(NEXT)` value in **IX register.** For example, to create two definitions that disables and enables interrupts, without an `ASSEMBLER`, you could use the following snippet:

```
CODE   ISR-DI   HEX
  F3 C,          \  di
  DD C, E9 C,    \  jp (ix)
  SMUDGE         \  now a dictionary search will find this definition

CODE   ISR-EI   HEX
  FB C,          \  ei
  DD C, E9 C,    \  jp (ix)
  SMUDGE         \  now a dictionary search will find this definition
```

(NUMBER) d a --- d2 a2

Converts the ASCII text at address `a+1` in a double-precision integer using the current `BASE`. Number `d2` is left on top of stack for any subsequent elaborations, `a2` is the address of the first non-converted character.
Used by `NUMBER` and `(EXP)` in the Floating-Point Option.

(PREFIX) a --- a2

During a `NUMBER` interpretation, if the character at address `a+1` is a "$" or a "%" modifies `BASE` accordingly and increments `a`: "$" for hexadecimal (base 16), "%" for binary (base 2).

(SGN) a --- a2 f

Determines if the character at address `a+1` is a sign (+ o −) and if found increments `a`. The flag `f` indicates the sign: `ff` for a positive sign `+` or no sign at all, `tf` for a negative sign − . If a is incremented then variable `DPL` is incremented as well. Used by da `NUMBER` and `(EXP)` in the Floating-Point Option.

***** n1 n2 --- n3

Computes the product of two integers.

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

Compute (n1 · n2) / n3 using a double-precision integer for the intermediate value to avoid precision loss.

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

Leaves the quotient n5 and the reminder n4 of the operation (n1· n2) / n3 using a double-precision integer for the intermediate to avoid precision loss.

+ n1 n2 --- n3

Leaves the sum of two integer.

+! n a ---

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

+- n1 n2 --- n3

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

+BUF a1 --- a2 f

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 if the index did not cross the boundary between the loop limit minus one and the loop limit, the execution jumps back to the beginning of the loop. Otherwise the execution leaves the loop. On leaving the loop, the parameters are discarded and the execution continues with the following definition.
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

Returns 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.

+TO n --- cccc

Used in the form

```
n +TO cccc
```

If not compiling, add the value n to cccc . At compile-time it compiles a literal pointer to cccc's PFA followed by a plus-store-command (+!) so that later, at run-time the literal is used by the ! definition to alter cccc value.
Definition cccc was created via VALUE. See also TO. This definition is available after NEEDS +TO.

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

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

," **---**

Compile a "Counted-ZString". It calls `WORD` to read characters from the current input stream up to a delimiter `"` and stores such a string at `HERE`. In a "Counted-ZString" the length of the string is stored as the first byte and the string itself ends with a NUL character (0x00). For example

```
,"  Hello"
```

compiles: **05 48 65 6C 6C 6F 00**
where 05 is the length of "Hello" string which is followed by a 00 'nul' character.

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

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

--> **---**

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.

-?EXECUTE **---**

Simple conditional execution semantics, used in the form

-?EXECUTE cccc

it invokes `-FIND` to search for **cccc** definition. If found **cccc** is executed. If not found **cccc** is ignored.

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

Duplicates `n` if it is non zero. This definition is available only for backward compatibility. See also `?DUP`.

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

Used in the form `-FIND cccc`.
It accepts a word (a non-space character sequence delimited by spaces) from the current input stream, storing it at address `HERE` + 1 and its length at `HERE`. Then, it run a search for it in the `CONTEXT` vocabulary first, then in the `CURRENT` vocabulary. If such a definition is found, it leaves the `cfa` of the definition, its length-byte `b` and a `tf`. Otherwise only a `ff`.

-TRAILING **a n1** **---** **a n2**

It can be used to trim trailing spaces from a counted string described by `a`, `n1`. This definition assumes that a string `n1` characters long is already stored at address `a` containing a word i.e. a characters sequence delimited by spaces. It determines `n2` as the position of the first delimiter after the word.

`.` **`n`** **`---`**

Emits the integer `n` followed by a space.

`."` **`---`** **`(immediate)`**

Used in the form

```
." cccc"
```

At compile-time, within a colon-definition, it accepts text from the input sources until a quote character (`"`) is encountered, then it compiles the primitive to output the text followed by the string `cccc`. 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 emits the text to output.

`.(` **`---`** **`(immediate)`**

Used in the form

```
.( cccc)
```

acting as `." cccc "` but the string is delimited by a closed-parenthesis.

`.C` **`c`** **`---`** **`(immediate)`**

Used in the form

```
c .C xxxx C
```

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

`.LINE` **`n1 n2`** **`---`**

Sends line `n1` of **BLOCK** `n2` to the current peripheral ignoring the trailing spaces.

`.R` **`n1 n2`** **`---`**

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`**

Computes `n3` = `n1`/ `n2`, the quotient of the integer division. This system uses floored division via `M/MOD` and implements `UM/MOD` in machine-code, `FM/REM` and `SM/MOD` as derived definitions. See `M/MOD` for details.

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

Computes the quotient `n4` and the reminder `n3` of the integer division `n1`/ `n2`. The reminder has the sign of `n1`. This system uses floored division via `M/MOD` and implements `UM/MOD` in machine-code, `FM/REM` and `SM/MOD` as derived definitions. See `M/MOD` for details.

0 --- **n**

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

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

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

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

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

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

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**

Increments by one the number on TOS.

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

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 ---

Stores the double-precision integer held on TOS to address a.

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

Doubles the number on TOS.

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

Increments by two the number on TOS.

2- n1 --- n2

Decrements by two the number on TOS.

2/ n1 --- n2

Halves the number on TOS.

2@ a --- d

a --- n-lo n-hi

Fetches the double-precision integer at address `a` to TOS.

2CONSTANT d --- (immediate) (compile time)

--- d (run time)

This is a defining definition that creates a double-precision constant. Used in the form

```
d 2CONSTANT cccc
```

it creates the new definition `cccc` which `pfa` holds the number **d**. When `cccc` is later executed it put **d** on TOS. This definition is not available at startup, it has to be loaded via `NEEDS 2CONSTANT`.

2DROP d ---

n1 n2 ---

Discards a double-precision integer from the TOS, i.e. discards the top two integer.

2DUP d --- d d

n1 n2 --- n1 n2 n1 n2

Duplicates the double-precision 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

Copies to TOS the second double-precision integer from top.
This definition isn't available at startup and must be included via `NEEDS 2OVER`.

2ROT d1 d2 d3 --- d2 d3 d1

n1 n2 n3 n4 n5 n6 --- n3 n4 n5 n6 n1 n2

Rotates the three top double-precision integers, taking the third and putting it on top. The other two double-precision integers are pushed down from top by one place. This definition isn't available at startup and must be included via `NEEDS 2ROT`.

2SWAP d1 d2 --- d2 d1

Swaps the two double-precision integers on TOS.

2VARIABLE --- (immediate) (compile time)
--- a (run time)

This is a defining definition used in the form:

```
2VARIABLE cccc
```

creates the new definition `cccc` with the pfa capable to hold a double-integer. When `cccc` is executed, it puts on TOS the pfa of `cccc` that is the address that holds the double-integer value.
When used in the form

```
cccc 2@
```

the content of the double-precision variable `cccc` is left on TOS.
When used in the form

```
d cccc 2!
```

the double-precision value on TOS is stored to the double-precision variable `cccc`.
This definition is not available at startup, it must be loaded via `NEEDS 2CONSTANT`.

3 --- n

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

3DUP n1 n2 n3 --- n1 n2 n3 n1 n2 n3

Duplicates the three top integer on Stack. This definition is not available at startup, it must be loaded via `NEEDS 3DUP`.

: ---

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

```
: cccc ... ;
```

creates in the dictionary a new definition `cccc` so that, when later invoked, it executes the sequence of already existing definitions `'...'`. The `CONTEXT` vocabulary is set to be the `CURRENT` and compilation continues while the value of `STATE` is non-zero. Definitions having the bit 6 of its length-byte set are immediately executed instead of being compiled.

:NONAME --- xt

This is a defining definition that creates and begins a name-less colon-definition. It returns the `xt` of the word being defined. Such `xt` should be kept in some way, for example as a `CONSTANT`. Used in the form, for example

```
:NONAME ... ;
CONSTANT cccc
```

This definition is not available at startup, it must be loaded via `NEEDS :NONAME` but the source-file that contains this definition is named %noname.f

; --- (immediate)

Ends a colon-definition and stops compilation.. It compiles `EXIT` and executes `SMUDGE` to make the definition accessible to subsequent search.

;CODE --- (immediate)

Used in the form

: cccc ... ;CODE

terminates a colon definition stopping compilation of definition `cccc` and compiling `(;CODE)`.
Usually `;CODE` is followed by a suitable machine-code sequence. It is used to define a creator of new kinds of definitions.

;S ---

This definition is obsolete and kept for backward compatibility reasons only. It is usually the last `xt` compiled in a colon definition by `;` and it does the action of returning to the calling definition. It is used to force the immediate end of a loading session started by `LOAD`.
Obsolete, prefer `EXIT`. This definition isn't available at startup and must be included via `NEEDS ;S` sequence.

< n1 n2 --- f

Leaves a `tf` if `n1` is less than `n2`, `ff` otherwise.

<# ---

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

<> n1 n2 --- f

Leaves a `tf` if `n1` isn't equal to `n2`, `ff` otherwise. This definition isn't available at startup and must be included via `NEEDS <>` sequence and the file loaded is {}.f

<BUILDS ---

Used in a colon definition in the form

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

Subsequent execution of `cccc` in the form

cccc nnnn

creates a new definition `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.

The “Floating Point Option Library” available via `NEEDS FLOATING` provides a good example of use of this structure.

`<BUILD` differs from `CREATE` because it allocate the pointer to the `DOES>` part of the defining word.

<NAME cfa --- nfa

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

= **n1 n2 --- f**

Leaves a tf if n1 equals to n2, ff otherwise.

> **n1 n2 --- f**

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.

>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.

>NUMBER **d a u --- d2 a2 u2**

This is the standard numeric conversion routine available for completeness only after NEEDS >NUMBER. This definition converts digits from the string a, u accumulating digits in number d. Conversion stops when any character that is not a legal digit is encountered returning the result d2 and the string parameters a2 and n2 for the remaining characters in the string. For historical reasons, this system doesn't use >NUMBER, instead it uses a non-standard version definition (NUMBER).

>R **n ---**

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> within the same colon-definition.

? **a ---**

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

?COMP **---**

Raises an error message #17 if the current STATE is not compiling state.

?CSP **---**

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 definitions 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 did not cross the boundary between the loop limit minus one and the loop limit, then the executions returns to the corresponding `?DO`, otherwise the two parameters are discarded and the execution continues after the `LOOP`.
The limit `n1` and the initial value `n2` are determined during the execution and can be the result of other previous operations. Inside a loop the definition `I` copies to TOS the current value of the index.
Se also: `I`, `DO`, `LOOP`, `+LOOP`, `LEAVE`. In particular `LEAVE` allows leaving the loop immediately.
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

?DO- [a1 n1] a n ---

This is a peculiar definition equivalent to `BACK` but fitted for `?DO`. It computes and compiles a relative offset from `a` to `HERE` and, in the case, it completes the `BRANCH` part previously compiled by `?DO` that left `a1` and `n1`. It is used by `LOOP`, `+LOOP`. If the loop begins with `DO` then `a1` and `n1` won't be there and no `BRANCH` will be compiled.

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

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

?ERROR f n ---

Raises an error message # `n` if `f` is true.

?EXEC ---

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

?LOADING ---

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

?PAIRS n1 n2 ---

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

?STACK ---

Raises an error message #1 if the stack is empty and we tried to consume an element from the calculator stack.
On the other hand, an error message #7 if the stack is full.

?TERMINAL --- f

Tests the keyboard for a [BREAK] key-press. Leaves a `tf` if the [BREAK] key is pressed, `ff` otherwise. Useful to stop an indefinite loop, for example:

```
BEGIN ... ?TERMINAL UNTIL
```

@ a --- n

Reads cell at address `a` and put an integer on TOS.

ABORT ---

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

ABORT" f --- (run time)

Used in a colon definition in the form

```
... ABORT" cccc" ...
```

At run-time if TOS is non-zero, prints the message `cccc` and performs **ABORT** to return to command-prompt. If TOS is zero does nothing and passes to the instruction after the message.

ABS n --- u

Leaves the absolute value of `n`.

ACCEPT a n1 --- n2

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 `QUERY`.

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 ---

This definition is used to reserve some space in the dictionary or to free memory. It adds the signed integer `n` to `DP` (Dictionary Pointer) user variable.

ALIGN ---

force `HERE` to an even address. This definition is available after `NEEDS ALIGN`.

ALIGNED a1 --- a2

force `a1` to an even address. This definition is available after `NEEDS ALIGNED`.

AND n1 n2 --- n3

It executes an bitwise AND operation between the two integers. The operation is performed bit by bit.

AUTOEXEC ---

This definition is executed the first time the Forth system boots and **loads Screen# 11**. Once called, it patches ABORT definition to prevent any further executions at startup. Anyway, you can still invoke it directly.
This allows you to perform some automatic action at startup.

B/BUF --- n

Constant. Number of bytes per buffer. In this implementation is 512.

B/SCR --- n

Constant that indicates the number of Blocks per Screen. In Next version is 2, that means a Screen is 1024 byte long. In Microdrive version it was 1…

BACK a ---

Computes and compiles a relative offset from a to HERE. Used by AGAIN, UNTIL, LOOP, +LOOP.

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 ---

Quits Forth and returns to Basic returning to the caller USR the unsigned integer on TOS.

BETWEEN n1 n2 n3 --- f

Available after NEEDS BETWEEN returns a true flag if n2 <= n1 <=n3 else a false flag.

BEGIN --- (immediate) (run time)
--- a n (compile time)

Used in colon definition in one of the following 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 definitions 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.

BINARY ---

Sets BASE to 2, that is the binary base. This definition is available after NEEDS BINARY.

BL --- **c**

Constant for “Blank”. This implementation uses ASCII and `BL` is 32.

BLANK **a n** ---

Fills with “Blank” `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`.

BLK-FH --- **a**

Variable containing file-handle to block's file !Blocks-64.bin.

BLK-FNAME --- **a**

Variable containing the counted-zstring “`!Block-64.bin`” . as produced by **,”** definition.
See also **,”** definition.

BLK-INIT ---

Initialize BLOCK system. It opens for update (read/write) file “`!Block-64.bin`” .

BLK-READ **a n** ---

Read block n to address a. `See also F_READ.`

BLK-SEEK **n** ---

Seek block n within blocks!.bin file. `See also F_SEEK.`

BLK-WRITE **a n** ---

Take text content at address a to disk block `n`. `See also F_WRITE.`

BLOCK **n** --- **a**

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`.

BOUNDS **a n** --- **a+n a**

Given an address and a length (a n) calculate the bound addresses suitable for `DO ... LOOP`.
It is used by `TYPE`.

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**

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 **---**

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** **---**

Stores a byte `b` to address `a`.

C, **b** **---**

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**

Puts on TOS the byte at address `a`.

CALL# **n1 a** **--- n2**

Performs a CALL to the routine at address `a`. First argument `n1` is passed via bc register *and* a register. The routine can return bc register which is pushed on TOS. This definition is useful to call normal ZX Spectrum ROM routines.
This definition is available after `NEEDS CALL#`.

CASEOFF **---**

Sets case-sensitive search OFF. changes the system behavior so that `(FIND)` can search the dictionary ignoring case, and `(COMPARE)` compares two strings ignoring case.

CASEON **---**

Sets case-sensitive search ON. It changes the system behavior so that (FIND) will search the dictionary case sensitive, and (COMPARE) will compare the two strings case sensitive.

CATCH **xt** **--- 0 | n**

In conjunction with **THROW**, this definitions put in place the error-handling facility. Used in colon-definition in the form

```
... ['] wwww CATCH …
```

this definition saves the current status of data-stack and return-stack, and performs **xt** via `EXECUTE`. If the execution of **xt** completes normally, i.e. with a `0 THROW` execution or with no `THROW` execution at all, it restores such saved status and leaves `0` on top of stack to signal no error occurred. Otherwise, `THROW` restores the most recent data-stack and return-stack status – saved by **CATCH** – so that the execution continues. See also **THROW**. This definition is available after `NEEDS CATCH`.

CELL --- **2**

In this implementation a cell is two bytes. This definition is available after `NEEDS CELL`.

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

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

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

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

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

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**

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 definition in the input stream.

CLS ---

Clears the screen using the ZX Spectrum ROM routine 0DAFh.

CMOVE **a1 a2 n** ---

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`.

CODE ---

Defining definition used in the form

```
CODE cccc
```

it creates a new dictionary entry for the definition `cccc` with the cfa ~~of such a definition pointing to its pfa~~ that is empty for the moment, `HERE` points that location; then some machine-code instruction should be added using `C,` that will be compiled from `HERE` onwards. The new definition is created in the `CURRENT` vocabulary but won’t be found by `(FIND)` because it has the `SMUDGE` bit set. Once the definition construction is complete, it is programmer's responsibility to execute `SMUDGE` to make visible.
This definition is redefined / overridden by `ASSEMBLER` vocabulary available after `LOAD`ing Screens 100-165, this allows the programmer to use a pseudo-standard Z80 notation to create a new low-level definition using assembler directly.

Here is an example that creates a definition `SYNC-FRAME` to wait for the next maskable interrupt:

```
CODE SYNC-FRAME HEX
      76 C,        \ halt       ;  wait for interrupt or reset
      DD C, E9 C, \ jp (ix)     ;  jump to the inner interpreter
      SMUDGE
```

COLD ---

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

COMPILE ---

Used in the form

`COMPILE cccc`

At compile-time, it determines the xt of the definition that follows `COMPILE` and compile it in the next dictionary cell.

COMPILE, xt ---

Used within a colon-definition to put `xt` in the following cell of the dictionary. The dictionary pointer (**DP**) is incremented by two locations.

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

Defining definition that creates a constant. Used in the form

n CONSTANT cccc

it creates the definition `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 definition search begins.

COUNT a1 --- a2 b

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 ---

Transmits a 0x0D to the current output peripheral.

CREATE	---		**(compile time)**
	---	**a**	**(run time)**

Defining definition used in the form

CREATE cccc

it creates a new dictionary entry for the definition `cccc` with the **pfa** still empty.
When `cccc` is executed, it puts on TOS the **pfa** of `cccc`
Often used with `ALLOT` to reserve space in the dictionary to be later used, for instance as an array.
See also `VARIABLE`.

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`.

CURS ---

Shows a (flashing) cursor on current video position and wait for a keypress.
Depending on CAPS-LOCK state, the faces of flashing cursor are different depending on the content of a few bytes in ORIGIN area:

`HEX`		
`026 +ORIGIN C@ .` → `8F`	█	Full square graphic character
`027 +ORIGIN C@ .` → `8C`	▄	Lower-half square graphic character
`028 +ORIGIN C@ .` → `5F`	_	Underscore character

When CAPS-LOCK is On the cursor switches between █ (8F) and _ (5F)
When CAPS-LOCK is Off the cursor switches between █ (8F) and ▄ (8C)
You can modify this behavior putting some suitable values on these three bytes. For example you can make disappear the flashing cursor using the following patch:

```
HEX
BL 026 +ORIGIN C!
BL 027 +ORIGIN C!
BL 028 +ORIGIN C!
```

D+ d1 d2 --- d3

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

D+- d1 n --- d2

Forces the double-precision integer `d1` to have the the sign of `n`.
It is used by `DABS` .

D- d1 d2 --- d3

Subtract d2 from d1. This is a 32 bits subtraction. Available after `NEEDS D-`.

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

Emits a double-precision integer followed by a space. The double-precision 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 ---

Emits a double-precision 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.

D0= d --- f

True if `d1` = 0. This is a 32 bits comaprison. Available after `NEEDS D0=`.

D< d1 d2 --- f

True if `d1` < `d2`. This is a 32 bits comaprison. Available after `NEEDS D<`.

D= d1 d2 --- f

True if `d1` equals `d2`. This is a 32 bits comaprison. Available after `NEEDS D=`.

DABS d --- ud

Leaves the absolute value of a double-precision integer.

DECIMAL ---

Sets `BASE` to 10, that is the decimal base.

DEFINITIONS ---

To be used in the form

```
cccc DEFINITIONS
```

it 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 a Forth oriented `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, and any number between 4 and 15 to refer to a Basic OPEN# channel.

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

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.

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

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

DMAX d1 d2 --- d3

Leaves the maximum between d1 and d2. Available after NEEDS DMAX.

DMIN d1 d2 --- d3

Leaves the minimum between d1 and d2. Available after NEEDS DMIN.

DNEGATE d1 --- d2

Computes the opposite double-precision 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 definitions 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 did not cross the boundary between the loop limit minus one and the loop limit, then the executions returns to the corresponding DO, otherwise the two parameters are discarded and the execution continues after the LOOP.
The limit n1 and the initial value n2 are determined during the execution and can be the result of other previous operations. Inside a loop the definition I copies to TOS the current value of the index.
See also: I, DO, LOOP, +LOOP, LEAVE. In particular LEAVE allows leaving the loop immediately.
At compile-time DO compiles (DO) and leaves the address of the following location and the number n to syntax-check.

DOES> ---

Definition that defines the execution action of a high-level creative definition. DOES> changes the **pfa** of the definition being defined to point the definitions 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 definition, and this allows the interpreter to use this area. Obvious uses are new vocabularies (Assembler), multidimensional array, closures, and other compiling operations.

The "Floating Point Option Library" available via NEEDS FLOATING provides a good example of use of this structure.

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-precision 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** **---**

Drops the value on TOS. See also `OVER`, `NIP`, `TUCK`, `SWAP`, `DUP`, `ROT`.

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

Duplicates the value on TOS. See also `OVER`, `DROP`, `NIP`, `TUCK`, `SWAP`, `ROT`.

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

Copy TOS to the Return Stack. See also `DUP`, `>R`, `R@`.

DU< **ud1 ud2 --- f**

True if `ud1` < `ud2`. This is a 32 bits comaprison. Available after `NEEDS D<`.

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** **---**

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** **---**

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

EMPTY-BUFFERS **---**

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 definition 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.

ENUMERATED n --- (immediate) (compile time)

Simple enumeration utility available after NEEDS ENUMERATED. Used in the form
Used in the form

```
n ENUMERATED name_0 name_1 ... name_n
```

For example:

```
8 ENUMERATED _black _blue _red _magenta _green _cyan _yellow _white
```

This definition has a limitation due to the way **LOAD** is implemented in that it cannot cope with **BLOCK**s boundaries or – for source-file – must be written in a single row that cannot be longer than 511 characters.

ERASE a n ---

Erases n memory location starting from a, filling them with 'null' characters (0x00).

ERROR b --- n1 n2
--- ff

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 definition is printed followed 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 is displayed. Such a number can be positive or negative and lay beyond **BLOCK** 4. Then, 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. Otherwise, if **BLK** is zero, then only a ff is left on TOS. In both cases, the final action is **QUIT** .
If **WARNING** is –1 then **ABORT** is executed, which resets the system to command prompt. The user can (with care) modify this behavior patching **(ABORT)** definition by putting a specific **xt** at its **>BODY**.

EVALUATE a u ---

This definition is available after **NEEDS EVALUATE**. Save the current input source specification. Store minus-one (-1) in **SOURCE-ID**. Make the string described by a and u the input source, set **>IN** to zero, and invoke **INTERPRET**. When the parse area is empty, restore the prior input source specification. Other stack effects are due to the words **EVALUATE**d.
This definition assumes the string **a** has been created in **HEAP** via **S"**, and MMU7 is correctly in place, as usually is.

EXEC: n ---

Vectorised fast case structure. Used in colon-definition in the form

```
: MY_ACTION_LIST ( n -- )
  EXEC:
      word0  \ executed when n = 0
      word1  \ executed when n = 1
      word2  \ executed when n = 2
    ...
;
```

to execute the definition indexed by **n** .

Warning: there is no run-time checking on **n** and if **n** is out of range, then a crash is likely to happen.

This definition is not available at startup, it must be loaded via **NEEDS EXEC:** . The source file is EXEC_.f

EXECUTE cfa ---

Executes the definition which cfa is held on TOS.

EXIT ---

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

EXP --- a

User variable that holds the exponent in a floating-point conversion. It is not used until the *Floating Point Option* is enabled via NEEDS FLOATING.

EXPECT a n ---

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. This definition isn't available at startup, it must be loaded using NEEDS EXPECT.

FENCE --- a

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

FILL a n b ---

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.

FLIP n1 --- n2

Exchange high and lower byte of n1. Available after NEEDS FLIP.

FLUSH ---

Executes SAVE-BUFFERS. It saves to disk the buffers marked "modified" by UPDATE.

FM/MOD d n1 --- n2 n3

Floored Division. Leaves the quotient n3 and the reminder n2 of the integer division of d /n1.
This system has only UM/MOD coded in machine-code.

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	4	-2
10	-7	-4	-2
-10	-7	-3	1

FORGET ---

Used in the form

```
FORGET cccc
```

removes from the dictionary the definition cccc and all the preceding definitions. Care must be put when more than one VOCABULARY is involved. Use MARKER instead.
See also DP.

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).

F_CLOSE fh --- f

NextZXOS primitive: it closes a file handle fh previously opened with F_OPEN or F_OPENDIR. Flag f is 0 for OK.
This is just a wrapper around NextZXOS API via RST 8 call followed by $9B service number.

F_FGETPOS fh --- d f

NextZXOS primitive: given an open file handle fh returns the position d. Flag f is 0 for OK.

F_GETLINE a n1 fh --- n2

Given a filehandle **fh** read at most n1 characters as the next line (terminated with $0D or $0A) and stores it at address a and returns n2 as the number of bytes read, i.e. the length of line just read.

F_INCLUDE fh ---

Given an open file-handle **fh**, this definition includes the source from file. This definition is used by INCLUDE and NEEDS.

F_OPEN a1 a2 b --- fh f

NextZXOS primitive: it opens a file using a filespec given at address a1 and returns filehandle number n, n1 is "mode" as specified in "NextZXOS and esxDOS APIs" standard documentation. Integer fh is the filehandle. Flag f is 0 for OK
It uses an RST 8 call followed by $9A service number. See also F_CLOSE.
Parameters in details as per standard documentation *NextZXOS and esxDOS APIs (Updated 24 May 2023)*:

a1 (filespec) is a null-terminated string, such as produced by **,"** or **S"** definitions.
a2 is address to an 8-byte header data used in some cases.
b is access mode-byte, that is a combination of any/all of:

esx_mode_read	$01 request read access
esx_mode_write	$02 request write access
esx_mode_use_header	$40 read/write +3DOS header

plus one of:

esx_mode_open_exist	$00 only open existing file
esx_mode_open_creat	$08 open existing or create file
esx_mode_creat_noexist	$04 create new file, error if exists
esx_mode_creat_trunc	$0C create new file, delete existing

F_OPENDIR a --- fh f

NextZXOS primitive: given a z-string address, open a file-handle to the directory. Integer fh is the filehandle. Flag f is 0 for OK.

F_READ a u1 fh --- u2 f

NextZXOS primitive: it reads at most u1 bytes from file handle fh and stores them at address a. Returns u2 as the actual bytes read. Flag f is 0 for OK. It uses RST 8 call followed by $9D service number.

F_READDIR a1 a2 fh --- u f

NextZXOS primitive: Given a pad address a1, a filter z-string a2 and a file-handle fh obtained via F_OPENDIR fetch the next available entry of the directory. Return 1 as ok or 0 to signal end of data then return 0 on success, True flag on error

F_SEEK d fh --- f

NextZXOS primitive: it seeks position d at open file given by filehandle fh. It uses an RST 8 call followed by $9F service number. Flag f is 0 for OK.

F_SYNC fh --- f

NextZXOS primitive: it syncs to disk open file given by filehandle fh. It uses an RST 8 call followed by $9C service number. Flag f is 0 for OK.

F_WRITE a u1 fh --- u2 f

NextZXOS primitive: it takes u1 bytes at address a and writes them to filehandle fh. It uses an RST 8 call followed by $9F service number. Returns u2 as the actual bytes written. Flag f is 0 for OK.

HANDLER --- a

User variable that holds the current error-handler. See CATCH and THROW.

HERE --- **a**

Leaves the address of next location available on the dictionary.

HEX ---

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.

HP --- **a**

User variable that holds the heap-address of the first free byte on Heap. See `HEAP` section.

HP, **n** ---

Puts `n` in the next location available in the heap and increments `HP` (heap pointer) by 2.

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.

I' --- **n**

Used between `DO` and `LOOP` (or `DO` and `+LOOP`, `?DO` and `LOOP`, `?DO` and `+LOOP`). It puts on TOS the *limit* of the loop. This definition isn't available at startup and must be included via `NEEDS I'`.

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 definitions 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 ---

Marks the latest definition 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 a definition 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.

INCLUDE ---

It is used in in the form:

```
INCLUDE cccc
```

starts interpretation of text read from file cccc.
This definition has a known bug, the `INCLUDE`d source text file must end with an empty line.
See also `NEEDS` and `LOAD`

INDEX **n1 n2** ---

Prints the first line of all screens between `n1` and `n2`. Useful to quick check the content of a series of screens.

INKEY --- **b**

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 they 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-precision integer instead of a simple integer.
After execution of the found definition, the control is given back to the caller procedure.

INVERT **n1** --- **n2**

Inverts all bits. This definition is available after `NEEDS INVERT`.

INVV ---

"Inverse video". It enables Inverse-Video attribute mode. See also `TRUV`.
This definition isn't available at startup and must be included via `NEEDS INVV`.

J --- **n**

Used inside a DO-LOOP gives the index of the *first* outer loop. See also `I`.
This definition isn't available at startup and must be included via `NEEDS J`.
E.g.

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

In this case **J** is used to get the index of the outer DO-LOOP while **I** gives the index of the inner DO-LOOP.

K --- **n**

Used inside a DO-LOOP gives the index of the *second* outer loop. See also **I**.
This definition isn't available at startup and must be included via `NEEDS K`.
E.g.

DO .. DO .. DO .. K .. LOOP .. LOOP .. LOOP

Anyway, in Forth, it isn't a good programming technique nesting loop, better split the definition.

KEY --- **b**

Waits for a key-press, without showing a flashing cursor. To show a cursor you have to use `CURS` just before `KEY`. 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	same as SHIFT-1 [EDIT]
C9	<>	→	06	same as CAPS-SHIFT + 2 and toggles CAPS-LOCK On and Off
C8	>=	→	20	same as SHIFT-0 [BACKSPACE]

L/SCR --- **n**

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

LATEST --- **nfa**

Leaves the `nfa` of the latest definition in CURRENT vocabulary.

LEAVE ---

Forces the conclusion of a `DO ... LOOP` by compiling `(LEAVE)` followed by an offset to the first instruction after the corresponding `LOOP` or `+LOOP`.

LFA **pfa** --- **lfa**

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 set at $E000 to allow MMU7 as a general purpose 8K RAM bank. See also: `FIRST`.

LIST **n** ---

Prints screen number n and sets `SCR` to `n`.

LIT --- **n**

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)

Compile-time, `LITERAL` compiles `LIT` followed by the value `n` in the following cell. This is an immediate definition and, even within 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 ---

Start interpretation of Screen `n`. The loading phase ends at the end of the screen or at the first occurrence of `EXIT`.
~~If n is negative, instead of loading from Screen# n, it loads text directly from stream n as previously OPEN# from Basic.~~
See also `-->`

LOAD-BYTES a n ---

Available after `NEEDS LOAD-BYTES`. Load `n` bytes to address `a` reading from filename which spec is held in `PAD`. This is just a wrapper around low-level definitions and errors are issued in case of I/O problems. `PAD` content is volatile. Typical usage is:

```
PAD" test.bin"      ( the space is needed but is not part of filename )
<address> <size> LOAD-BYTES
```

See also: `SAVE-BYTES`

LOAD2BLOCK n ---

Available after `NEEDS LOAD2BLOCK`. Load up to 448 bytes to specified `BLOCK` number `n` reading from filename which spec is held in `PAD`. This is just a wrapper around low-level definitions and errors are issued in case of I/O problems. `PAD` content is volatile. WARNING: In this implementation a Screen occupies two `BLOCK`s so for example **Screen# 440** is `BLOCK` **880**. Typical usage is:

```
PAD" test.bin"      ( the space is needed but is not part of filename )
<blocknumber> LOAD2BLOCK
```

See also: `LOAD-BYTES`

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

Used in colon definition 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 if the index did not cross the boundary between the loop limit minus one and the loop limit. Otherwise the execution leaves the loop. On loop leaving, the parameters are discarded and the execution continues with the following definition. 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 purpose. In this Forth implementation it is used during compilation phase by `CASE`.

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-precision integer.

M*/ **d1 n1 n2** --- **d2**

Mixed operation. Compute (`d` · `n1`) / `n2` using a "triple precision integer" as the intermediate value to avoid precision loss. This definition isn't available at startup and must be included via `NEEDS M*/`. The source file is M&%.f

M+ **d u** --- **d2**

Mixed operation. It leaves the sum of `d` and unsigned `u` as a double-precision integer `d2`.
This definition is available after `NEEDS M+`

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

Mixed operation. It leaves the quotient `n2` of the integer division of a double-precision integer `d` by the divisor `n1`.

M/MOD **d1 n1** --- **n2 n3**

Mixed operation. It leaves the remainder `n2` and the quotient `n3` of the integer division of a double-precision integer `d` by the divisor `n1`. The sign of the reminder is the same as `d`. This system uses floored division via `M/MOD` and implements `UM/MOD` in machine-code, `FM/REM` and `SM/MOD` as derived definitions.

MARK **a n** ---

`TYPE` in inverse video. This definition is not available at startup, it has to be loaded via `NEEDS MARK`.

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

Used outside 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` was created. This removes `cccc` and all subsequent definitions. This definition allows forgetting across vocabularies since it keeps track of `VOC-LINK`, `CURRENT`, `CONTEXT` values.

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

Leaves the maximum between `n1` and `n2`.

MESSAGE n ---
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
Leaves the minimum between n1 and n2.

MMU7! n ---
This definition 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 definition 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
Divides n1 by n2 and leaves the remainder n3. The sign is the same as n1.

MS u ---
Waits at least u milliseconds.
At 28 MHz, u must be < 8192.
At 14 MHz, u must be < 16384.
At 7 MHz, u must be < 32768.
At 3.5 MHz, u must be < 65536.
This definition isn't available at startup and must be included via **NEEDS MS**.
N.B. Interrupts aren't disabled during execution.

M_P3DOS n1 n2 n3 n4 a --- n4 n5 n6 n7 f
This is the NEXTZXOS 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
f = 0 for OK, non zero for KO.
This definition calls uses RST 08 followed by $94 to call the specified routine.
Value returned on register IX is also stored at HEX 2A +ORIGIN before IX is restored to its fixed value.

Some NEXTZXOS primitives are coded by their own definition (e.g F_OPEN, F_OPENDIR, etc), but most of them aren't. For example all LAYER definitions use IDE_MODE! which in turn uses M_P3DOS.

NEEDS ---

Used in in the form:

```
NEEDS cccc
```

if the definition cccc is not present in dictionary, then it starts interpretation of text read from file **inc/*cccc*.f** and, if this is not found, gives a second chance from file **lib/*cccc*.f**

Some characters are illegal for filename: noticeably the “double-quote” character (") is among them. In such case, these characters are converted into “tilde” (~) and *that* file is then searched for.
For example:

`NEEDS S"` searches the file `S~.f` instead of an illegal filename `S".f`

Here is the complete map:

```
: ? / * | \ < > "
_ ^ % & $ _ { } ~
```

At the moment we are writing, this `NEEDS` definition has a flaw: in case of interpretation/compilation error, the file/handle remains open and you have to close it manually using something like `2 F_CLOSE DROP`,

This definition differs from **INCLUDE** because **NEEDS cccc** refers to a dictionary entry whilst `INCLUDE cccc` refers to a full-path filename with explicit extension.

This definition is defined as follow: Since any given Screen # n occupies **BLOCK**s n and n+1, `NEEDS` exploits BLOCK number 1 – which normally isn’t reachable – and uses it as a temporary buffer for each line read from file, for this reason a text source line cannot exceed 511 bytes, but for ZX Spectrum's standards is a lot more than anybody would need.

NEGATE **n** --- **-n**

Changes the sing of `n1`

NFA **pfa** --- **nfa**

Converts a definition’s **pfa** into its **nfa**. See also `CFA`, `LFA`, `PFA`, `>BODY`, `<NAME`.

NIP **n1 n2** --- **n2**

Removes the second element from TOS. See also: `OVER`, `DROP`, `TUCK`, `SWAP`, `DUP`, `ROT`.

NMODE --- **a**

User variable that indicates how double-precision numbers are interpreted. During the input, numbers can be read as double-precision integer numbers or floating-point numbers. This variable is modified by the optional definition `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`.

NOT ---

Equivalent to `0=`

NUMBER **a --- d**
a --- fp (floating-point)

Converts a counted string at address a with a in a double-precision number. If NMODE is 0, the string is converted to double-precision integer. Position of the last decimal point encountered is kept in DPL.
If **NMODE** is 1, a floating-point number conversion is tried instead of an simple double-precision integer conversion. If no conversion can be done, and error #0 is raised. See § 2.8 for more details.

OCTAL **---**

Changes the base to octal, setting BASE to 8. To use this definition you have to type NEEDS OCTAL.

OFFSET **--- a**

Current edit position within current screen. Used by Line-Editor.

OPEN< **--- fh**

Used in the form

```
OPEN< cccc
```

this definition invokes **F_OPEN** NextZXOS and opens a file cccc. It returns file-handle number fh. This definition is used by INCLUDE. At the moment, this definition cannot be compiled and should be used only in interpretation phase.

OR **n1 n2 --- n3**

Executes a bitwise OR 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**

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

P! **b u ---**

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@ **u --- b**

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 **---**

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

PFA **nfa --- pfa**

Converts a definition's nfa to its pfa. See also CFA, LFA, NFA, >BODY, <NAME.

PICK n --- pfa

Picks the **n-th** element from TOS. This means:

`0 PICK` is the same as `DUP`
`1 PICK` is the same as `OVER`

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`.

POSTPONE ---

Available after **NEEDS POSTOPNE** and used in colon defintion in the form:

```
: cccc ... POSTPONE wwww ... ;
```

It blends the functionality of `[COMPILE]` and `COMPILE` appending to the current definition the correct `wwww` *compilation behavior*. This is part of the modern standard, but the Author prefers the old-fashion `[COMPILE]` and `COMPILE`.

PREV --- a

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

QUERY ---

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 ---

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

R@ --- n

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

Removes the top value from Return Stack and put it on TOS. See also `>R`, `R@` and `RP!`.

R>DROP ---

Removes the top value from Return Stack. See also `>R`, `R@` and `DROP`. Available after `NEEDS R>DROP`.

RECURSE ---

Used only at compile-time inside a colon-definition, it compiles the definition being created to put in place a recursion call. This definition is available after a `NEEDS RECURSE`.

REG! b n ---

Writes value `b` to Next REGister `n`.

REG@ n --- b

Reads Next REGister `n` giving byte `b`.

REMOUNT ---

This definition is available only after `NEEDS REMOUNT`.
Enter the unmount/mount routine. Interactively the user is asked for a Y key-stroke, and the system waits for that key-stroke allowing the manipulation of the SD.

RENAME ---

Used in the form:

```
RENAME cccc xxxx
```

it searches the definition `cccc` in the `CONTEXT` vocabulary and changes its name to `xxxx`. The two definition names `cccc` and `xxxx` **must have the same length**. This definition is available after `NEEDS RENAME`.

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

Rotates the three top elements, taking the third and putting it on top. The other two elements are pushed down from top by one place. See also `OVER`, `DROP`, `NIP`, `TUCK`, `SWAP`, `DUP`.

ROLL n1 ... k --- n2 ... n1

Rotates the k top elements, taking the k-th and putting it on top. The other k – 1 elements are pushed down from top by one place. The index k is zero based, so that 0 ROLL does nothing, 1 ROLL is SWAP and 2 ROLL is ROT.
See also ROT. This definition isn't available at startup, it needs to be imported via NEEDS ROLL .

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

Leaves the current value of Return Stack Pointer.

RSHIFT n1 u --- n2

Shifts right an integer n1 by u bit.

S0 --- a

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

S>D n --- d

Converts a 16 bit integer into a 32 bit double-precision integer, sign is preserved. An obsolete version S->D is still available via NEEDS.

SAVE-BYTES a n ---

Available after NEEDS SAVE-BYTES. Saves n bytes from address a writing filename which spec is held in PAD. It creates new a new file, error if it already exists. PAD content is volatile.
Typical usage is:

```
PAD" test.bin"       ( the space is needed but is not part of filename )
<address> <size> SAVE-BYTES
```

See also: LOAD-BYTES

SCR --- a

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

SELECT n ---

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 ---

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 unchanged. Used between <# and #>.

SM/REM d n1 --- n2 n3

Symmetric Division. Leaves the quotient `n3` and the reminder `n2` of the integer division of `d` / `n1`.
This system has only UM/MOD coded in machine-code.

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	-3	-1
10	-7	3	-1
-10	-7	-3	1

SMUDGE ---

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

```
SMUDGE FORGET cccc
```

SOURCE-ID --- a

User variable that keeps the file-handle used during `INCLUDE` or `NEEDS`.

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

Returns the content of SP register before `SP@` was executed.

SPACE ---

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

SPACES n ---

Emits `n` spaces.

SPAN --- a

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

SPLASH ---

Shows splash screen build date-number.

SPLIT n1 --- n2 n3

Split the two bytes of n1 into two separate numbers, n2 low byte, n3 high byte. Available after `NEEDS SPLIT`.

STATE --- **a**

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

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

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`.

THROW **... n** --- **... | ... n**

In conjunction with **CATCH**, this definitions put in place the error-handling facility. If n is zero, just discard n. If n is non-zero, it restores the most recent data-stack and return-stack status – saved by the corresponding **CATCH** – so that the execution continues just after that point. See also **CATCH**. This definition is available after `NEEDS THROW`.

TIB --- **a**

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

TO **n** ---

Used in the form:

TO cccc

It assigns the value `n` to the variable `cccc` previously defined via `VALUE`. This definition available after `NEEDS TO`.

TOGGLE **a b** ---

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

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

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`.

TRUV ---

“True video”. It disables Inverse-Video attribute mode. See also `INVV`.

This definition isn't available at startup and must be included via `NEEDS INVV`.

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

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

TYPE		**a n ---**

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

U.		**u ---**

Emits an unsigned integer followed by a space.

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

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-precision integer.

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

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

UNLOOP		**---**

This definition is specified by the "standard". It iscard the loop-control parameters for the current nesting level. It is neveer used because `LOOP` and `+LOOP` provide the same functionality.

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		**---**

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 definition converts given character `c1` to upper-case. If `c1` is not between "a" and "z", then `c1` is left unchanged.

USE		**---**

Used outside colon-definitions in the forms

```
USE filename
```

It tries to open `filename`, and if it succeeds, it closes the current `BLOCK`s file and uses the one just opened instead. To restore the standard/default file, you must use `BLK-INIT` which uses the filename given by `BLK-FNAME`.

USED --- 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 definition 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, SOURCE-ID, SPAN, HANDLER, HP.

VALUE n ---

Defining definition used in the form:

n VALUE cccc

Creates the definition cccc that acts as a variable but which syntax behaves as a constant . 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 @. This definition is available after **NEEDS VALUE**.

VARIABLE ---

Defining definition used in the form:

VARIABLE cccc

creates the definition cccc with the pfa containing the initial value 0. When cccc is executed, it puts on TOS the pfa of cccc that is the address that holds the value.
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 ---

Sets DEVICE 2 to select the video as current output peripheral. See SELECT and DEVICE.

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 uses that field, in each vocabulary definition, as pointer-chain.

VOCABULARY ---

Defining definition used in the form

VOCABULARY cccc

creates the definition `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 definitions 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 ---

Executes a warm system restart. It closes and reopens Block/Screen file then does `ABORT`.
It does not `EMPTY-BUFFERS` and you should be able to recover any transient work.

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 a definition during compilation. It must be between 1 and 31.

WITHIN n1 n2 n3 --- f

Perform a comparison of a test value n1 against a lower-limit n2 and a upper-limit n3.
When n2 < n3, return a true-flag if n2 <= n1 < n3, a false-flag otherwise.
When n2 > n3, return a true-flag if n2 <= n1 OR n1 < n3, a false-flag otherwise.
This definition is available only after `NEEDS WITHIN`.

WORD **c --- a**

Reads one or more characters from the current input stream up to a delimiter `c` and stores such string at `HERE` that is left on TOS. `WORD` leaves at `HERE` the length of the string as the first byte 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` returns.

WORDS **---**

Shows a list of definitions of `CONTEXT` vocabulary. [Break] stops.

XOR **n1 n2 --- n3**

Executes a bitwise XOR operation between the two integers. The operation is performed bit by bit.

ZAP **---**

Used in the form

```
ZAP cccc
```

it searches for the definition `cccc` to produce a standalone executable of it by the creation of the following binary files in the current directory:

```
cccc-core.bin
cccc-user.bin
cccc-heap.bin
```

These three binary files contain the current status of the whole vForth system that can be resumed later.
You have to manually modify the basic loader "Standard-Loader.bas" in order to load these three binary files and invoke a vForth cold-start that runs `cccc` instead.
And in fact, **ZAP** modifies the definition of COLD by patching the first xt to be `cccc` and the second xt to be **BYE**.

[**--- (immediate)**

Used in colon defintion in the form:

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

it suspends compilation. The definitions that follow [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.

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

It is the same as the sequence `[ ' wwww ] LITERAL`.
It is used in colon-defintion in the form:

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

At compile time, `[']` compiles `LIT` and **xt** of `wwww` definition in the following cell.

[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 an immediate definition isn't compiled but executed and to compile an immediate definition it isn't possible to use the sequence `COMPILE wwww` but it is necessary using the sequence `[COMPILE] wwww`.
For example, to create an alias `ENDIF` for `THEN` you can type:

```
: ENDIF [COMPILE] THEN ;
```

**** ---

Used in the from:

```
\ ...
```

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

] ---

Resumes the compilation suspended by `[` so it is possible to continue the definition.

6.2 Case -Of structure

The following definitions aren't available at startup, it must be loaded via `NEEDS CASE`

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 `CASE` definition 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, if any.
At compile time `CASE` leaves the 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 definition is used in colon-definition as parto of a Case-Of structure.
At run-time it compares the matching value `nk` with the matching value `n0` that was on TOS before the beginning of the Case-Of structure.
At compile-time, it compiles `(OF)` that does a `0BRANCH` . The numbers `n1` and `n2` are used for syntax checking and an address `a` is left and used by `ENDCASE` to resolve the branch.
See also `CASE`.

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

This definition ends an “Of-EndOf” clause started with `OF`.
At compile-time it acts like a `THEN`, first compiling a BRANCH to be later resolved by `ENDCASE` to skip any subsequent “Of-End-Of” clauses and resolving here 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 definition 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 definition is the run-time semantic compiled by `OF` definition. 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 def after `ENDOF` is performed.

6.3 Heap Memory Facility

Among ZX Spectrum Next new features is the huge amount of RAM. Strings are dictionary expensive, so it would be useful storing them in heap as constant-strings and fetch them at need. The question is how to leverage all that memory in Forth. More, 8K of room is a good place to store an array of strings, or even numeric array and implement some matrices algebra. The definitions that handle The Heap are available after loading via **`NEEDS HEAP`**, but all low-level definitions are available in the core dictionary.

6.3.1 Heap-Pointer encoding and decoding

The Heap Memory Facility introduces a new kind of Pointer, the Heap-Pointer that represents the offset inside a virtual memory area called "heap". A Heap-Pointer is not an usual address pointer and we need a way to encode both **page number** and **address offset** in a usual Z80 16-bits integer.

Two definitions are available to perform these coding and decoding operations: **`>FAR`** and **`<FAR`** called "to-far" and "from-far" respectively.

Given a page number **`n`** and an address **`a`** (to be intended as an offset of addresses between $E000 and $FFFF, but only the lower part bits are used) **`<FAR`** definition decodes the page number in the most significant bits of **`ha`** and the offset in the remaining least significant bits. The opposite function is performed by **`>FAR`** that splits a 16-bit Heap-Pointer into two numbers again, the page part **`n`** and the offset part **`ha`** between $0000 and $1FFF.

In the following paragraphs a couple of possible implementations are described in detail.

6.3.2 Heap structure

The Heap can be seen as a "double linked-list" starting at 8K page $20 offset $0002. The User variable `HP` keeps the "heap-pointer" to the next available location on Heap. So, at startup, **`HP`** is $0002 that correspond to page $20 offset $0002. Heap space is shared with name-space dictionary.

A Heap memory allocation reserves the requested number of bytes and advances **`HP`** to point to the next available location on Heap. The previous value of **`HP`** is also stored at the location that was available *before* the memory allocation was requested to put in place a "linked-list".

In other words:
1. **`HP`** is advanced of one cell (2 bytes) to make room for the forward linked-list pointer.
2. Current **`HP`** value returned by the memory allocation (memory is not initialized and its content is undefined)
3. **`HP`** is advanced to the number of bytes requested (plus 2 to ensure room for 2 trailing 0x00 character).
4. **`HP`** is advanced of one cell (2 bytes) to make room for the backward linked-list pointer.

Here is a real case example:

Let **`HP`** contains **$0F80** and the Heap memory looks as follows (Location is expressed in the form "$page.$offset")

```
Page.Address   Content
 20:0F80       ....  free memory pointed by HP so that  HEX HP ?  gives  F80
```

Then, we want to reserve a 7-bytes chunk of memory and we can type **7 HEAP** that returns $0F82 as "Heap-Pointer" to that new area of memory and **HP** will be advanced to $0F8D.

After the execution the memory will look like this:

```
Page.Address  Content

 20:0F80      0F8D   ( forward pointer )
 20:0F82      00 00 00 00 00 00 00     ( 7 bytes just allocated )
 20:0F89      00 00                    ( plus trailing 0x00 )
 20:0F8B      0F80   ( backward pointer )
 20:0F8D      ....   free heap memory pointed by HP
```

Then we can create a new string with Heap-Storage typing **S" 123"** which returns on TOS the actual real address $EF90 and the length $0003 that can directly used with **DUP TYPE** this way the Heap-Pointer is not lost, usually you want to keep it in a safe place such as a **CONSTANT,** or it should be saved beforehand via **HP@ CELL+**.

This allocation requires ten bytes on Heap: the forward pointer, the byte of length, the string itself 3+1 byte long, a two trailing 0x00, the backward pointer.

After the execution, the memory will look like this:

```
Page.Address  Content

 20:0F80      0F8D   ( forward pointer )
 20:0F82      00 00 00 00 00 00 00     ( 7 bytes just allocated )
 20:0F89      00 00                    ( plus trailing 0x00 )
 20:0F8B      0F80   ( backward pointer )
 20:0F8D      0019   ( forward pointer )
 20:0F8F      03     ( string-length byte )
 20:0F90      31 32 33  ( the string "123" )
 20:0F93      00 00     ( plus trailing 0x00 )
 20:0F95      0F8D   ( backward pointer )
 20:0F97      ....   free memory pointed by HP
```

Then, we want to reserve another 5-bytes chunk and we can type **5 HEAP** that returns $0F99 as "Heap-Pointer" to that new area of memory and HP will be advanced to $0FA2.

After the execution the memory will look like this:

```
Location      Content

 20:0F80      0F8D   ( forward pointer )
 20:0F82      00 00 00 00 00 00 00     ( 7 bytes just allocated )
 20:0F89      00 00                    ( plus trailing 0x00 )
 20:0F8B      0F80   ( backward pointer )
 20:0F8D      0019   ( forward pointer )
 20:0F8F      03     ( length byte )
 20:0F90      31 32 33  ( the string "123" )
 20:0F93      00 00     ( plus trailing 0x00 )
 20:0F95      0F8D   ( backward pointer )
 20:0F97      0FA2   ( forward pointer )
 20:0F99      00 00 00 00 00       ( 5 bytes )
 20:0F9E      00 00     ( plus trailing 0x00 )
 20:0FA0      0F97   ( backward pointer )
 20:0FA2      ....  free memory pointed by HP
```

Now, you should be able to see the Linked-List starting at $**0F80** that points to $**0F8D** that points to $**0F97** and then $**0FA2** is the current value of `HP`.

You can follow all these Forward-Pointers using the following procedure:

```
HEX
0F80 .S \ Stack is 0F80 as Heap-Pointer, that is 20:0F80, the beginning of Heap Memory.
FAR  .S \ Stack is EF80 as real Address (and page $20 is fitted in MMU7)
@    .S \ Stack is 0F8D as Heap-Pointer
FAR  .S \ Stack is EF8D as real Address (and page $20 is fitted in MMU7)
@    .S \ Stack is 0F97 as Heap Pointer
FAR  .S \ Stack is EF97 as real Address (and page $20 is fitted in MMU7)
@    .S \ Stack is 0FA2 as Heap Pointer
```

Likewise, the Backward-Pointers sequence would be:

```
HP@ .S \ Stack is 0FA2 as Heap-Pointer
CELL-  \ Stack is 0FA0 as Heap-Pointer
FAR .S \ Stack is EFA0 as real Address (and page $20 is fitted in MMU7)
@   .S \ Stack is 0F97 as Heap-Pointer
CELL-  \ Stack is 0F95 as Heap-Pointer
FAR .S \ Stack is EF95as real Address (and page $20 is fitted in MMU7)
@   .S \ Stack is 0F8D as Heap Pointer
CELL-  \ Stack is 0F8B as Heap-Pointer
FAR .S \ Stack is EF8B as real Address (and page $20 is fitted in MMU7)
@   .S \ Stack is 0F80 as Heap Pointer
```

Some low-level definitions are available to allow store and retrieve "to and from" Heap memory and how to avoid that a string isn't "paged away" in the middle of processing i.e. how to guarantee a page to stay in place across Standard-ROM calls or I/O disk operations that use page-bank C000-FFFF for their purposes:

`MMU7!` is used to fit a given 8K page number at E000h (i.e. MMU7).
`>FAR` is used to decode a "16 bit pointer" splitting it into "page & offset" as shown above.

The User Variable `HP` has been introduced to keep track of room in Heap: it's “the pointer” to the next available space on Heap.

Most of the following definitions are available after loading via `NEEDS HEAP`

+" **ha --- ha**

Assuming `ha` is a Heap-Address Pointer to a “counted string” and this is the last chunk of memory of Heap, this definition accepts some text from the current input-source, parse it looking for a quote **"** that is the common “string terminator”, and appends to the previous string on Heap. It returns the same Heap-Address Pointer to a “counted string” but the “count-byte” is incremented correctly. No page boundary check is performed.

+C **ha c --- ha**

Consume a character `c` from the current input source and append the string being created in Heap at `ha`. The heap pointer `ha` is returned unchanged.

>FAR **ha --- a p**

Given a Heap-encoded Pointer `ha` this definition decodes the top bits as one of the 8K-page available page p and the lower bits as the offset from $E000 `a` . It does not modify what MMU7 page is.
Since version 1.7, this definition is always available. Previous versions needs `NEEDS >FAR`.
See `<FAR`, `MMU7!`

<FAR **a p --- ha**

Given an offset-address **a** (to be intended as a physical address between $E000 and $FFFF) and a page number **p** for an 8K-page this definition encodes the page number in the most significant bits of **ha** and an offset in the remaining bits. It does not modify MMU7 page.
Since version 1.7, this definition is always available. Previous versions needs `NEEDS <FAR`.
See `>FAR`, `MMU7!`

FAR **ha --- a**

This definition converts a heap-pointer `ha` into an offset `a` (at $E000) and perform the correct 8K paging on MMU7. It simply calls `>FAR` and `MMU7!`

H" **--- ha**

Accepts a text from the current input-source and stores it to Heap. It returns a “heap-address-pointer” to a counted string.

HEAP **n --- ha**

This definition reserves `n` bytes on Heap and returns the “heap-address-pointer”. This `ha` can be turned into a constant name using `POINTER`.

POINTER **ha --- a**

It works like `CONSTANT` but it returns a "FAR-resolved" offset-pointer from E000h .
A possible use is : `H" ccc" POINTER P1`

`SKIP-HP-PAGE`   `n ---`

Check if `n` bytes are available at the top of Heap on current 8K-page, otherwise advance `HP` to skip to the beginning of next 8K-page. It raises an "Heap Full" error if there is no more room in Heap.

`S"`   `--- a n`

Accept text from the current input-source and store it to Heap as a "Counted-ZString".
At compile time it compiles **`(H")`** followed by an Heap-pointer just after it, which at run-time returns a real-address (at MMU7) and a counter representing the "counted-string" that can be used
If `STATE` is 0, i.e. we aren't compiling, the c

`(H")`   `--- a n`

This is the run-time counterpart of **`S"`** that uses the Heap-Pointer in the following cell to fit the right 8K-Page in MMU7 using `FAR` definition and leave the real-address `a` and the length of the string `n`.

6.3.3 Heap Pointer description for 64 kiBytes space

The following solution allows 64K of physical RAM Heap: Since an 8K offset requires 13 bits, the remaining 3 bits can be used to encode, say, from page 32 ($20) to page 39 ($27). For instance:

```
                         16 bit pointer
                |  pppb bbbb | bbbb bbbb |
                |            |           |
  |    Page     |         Offset         |
  |  0010 0ppp  |  111b bbbb | bbbb bbbb |
```

The encoding/decoding definitions would be something like the following:

```
\ >far routine
\ input:  hl : heap-pointer
\ output:  a : page
\        : hl : address starting from $E000
      ld      a,h
      ex      af,af'  ; save high part
      ld      a,h
      or      $E0
      ld      h,a     ; hl = offset at $E000
      ex      af,af'
      rlca
      rlca
      rlca
      and     $07     ; so there are eight pages
      add     $20     ; this is peculiar to this example
      ret
```

```
\ <far routine
\ input :  a : page number between 32 and 39
\        : hl : address starting from $E000
\ output: hl : heap-pointer
      and     $07     ; ? questionable: it could be SUB $20
      rrca
      rrca
      rrca
      ex      af,af'  ; keep bits 765 in alternate A
      ld      a,h
      and     $1F
      ld      h,a
      ex      af,af'   ; retrieve bits 765
      or       h
      ld      h,a
      ret
```

6.4 Testing Suite

This is an adaptation of the ANS test harness based on the work originally developed by John Hayes, see https://forth-standard.org/standard/testsuite for details.

The suite is loaded using `NEEDS TESTING` and "Core test-set" can be execute by typing

```
INCLUDE  ./test/core-tests.f
```

In general, a test is given in the form

```
T{  ...  ->  ...  }T
```

for example:

```
T{ 1 DUP -> 1 1 }T
```

TESTING **---**

This definition is much like a comment, it displays the whole source line where it is.

T{ **---**

Begin a test phrase that ends with `}T`. It records pre-test stack depth to be compared later.

-> **---**

Record depth and contents of stack to be copared after `}T`.

}T **---**

End a test phrase begun with `T{`. It compares two stack images. Any discrepancies is shown by repeating the current test SOURCE line involved followed by one of the error

6.5 Other Utilities

SHOW-PROGRESS **n** **---**

Useful within long-lasting definitions to display a "rolling-bar" that show that your ZX Spectrum hasn't hanged or crashed. This definition isn't available at startup and must be included via `NEEDS SHOW-PROGRESS`.

?VOCAB **---**

Useful to see which `VOCABULARY` is `CURRENT`, `CONTEXT` and the linked-list described by `VOC-LINK`.

VIEW **---**

Used in the form `VIEW cccc` used to display a file. Pressing [EDIT] during ouput allows you to temporarily pause the stream of output. Pressing [BREAK] will stop the output and return to prompt. Available after `NEEDS VIEW`.

7 The Memory Map

The memory is divided differently depending on which way vForth is started:

1. As dot-command, the core is loaded in DivMMC RAM at address $2000-$3FFF, and the dictionary continues at $8000. To preserve previous BASIC memory state, MMU4, MMU5 and MMU6 are fitted with 8k pages #40, #41 and #42 ($28-$2A); heap pages from #32 to #39 ($20-$27) are fitted in MMU7 ($E000-$FFFF) when needed.
I've found experimentally that, since MMU2 and MMU3 are extensively swapped using Layer 1,2, the original content of MMU3 must be kept saved in page #43 ($2B) as backup and restored on return to BASIC.
This means twelve 8K-Pages are requested (via NextZXOS call), from **#32** to **#43 ($20-$2B)**.

2. As standard program, the RAMTOP is lowered, the core is loaded in memory at $6366 and the dictionary continues upward in a linear way. Standard BANKS 5-2-0 configuration is used, i.e. pages $0A, $0B, $04, $05, $00, $01, with the exception that MMU7 is also used to access any 8K RAM Page other than $01. Eight 8K-Pages are requested via NextZXOS call, from **#32** to **#39 ($20-$27)** and fitted in MMU7 ($E000-$FFFF) when needed.

In both ways, the Name-Space part of dictionary is loaded in **BANK 16**, or to be more specific, **8K-Pages #32 ($20)** and **#33 ($21)**, then the next six Pages from **#34 to #39 ($22-$27)** are reserved as HEAP memory (§6.3) and used to keep the Name-Space along with any other HEAP objects such as long strings: this way both `FORGET` and `MARKER` free memory from HEAP and Dictionary in the same call. Also, vForth should refuse to start if these page aren't available.

```
Address     Name        Description
---------   ---------   ------------------------------
0000-3FFF               ROM of Spectrum
2000-3FFF               Dot-command memory area.
4000-57FF               Display file
5800-5AFF               Attribute file.
5B00-5BFF               System variables 128K RAM (former Printer buffer)
5C00-5CEF               System variables
            *CHANS      Stream map
            *PROG       Basic program
            *VARS       Basic variables
            *E_LINE     Line in editing
            *WORKSP     Workspace
            *STKBOT     Floating point Stack Bottom
            *STKEND     Floating point end
            *SP         Z80 Stack Pointer register in Basic
61FF        *RAMTOP     Logical RAM top (RAMTOP var is 23730)
6200-6300               IM2 ISR vector table (non dot-command version)
6301-6330               Return Stack during ISR (20 entries)
6331-6362               Stack area during OS operations
6363                    ISR entry point (JP address)
6366         ORIGIN     Forth Origin (non dot-command version)
                        FENCE @
             LATEST     CURRENT @ @
             HERE       DP @
             PAD        HERE 68 +  (+44h)
             ...        Dictionary grows upward
             ...        Dictionary free memory
             SP@        Calculator Stack grows downward
D0E8                    S0 @
D0E8         TIB        TIB @
             RP@        Return Stack grows downward: it can hold 80 entries
D188                    R0 @
D188-D1D8               User variables area (about 50 entries)
D1E4         FIRST      First buffer: There are 7 buffers (516 * 7 = 3612 bytes)
E000         LIMIT      First byte outside Forth.
E000-FFFF    MMU7       8K Page that can page any of the 224 banks of RAM
FFFF         P_RAMT     Phisical ram-top
```

Contents