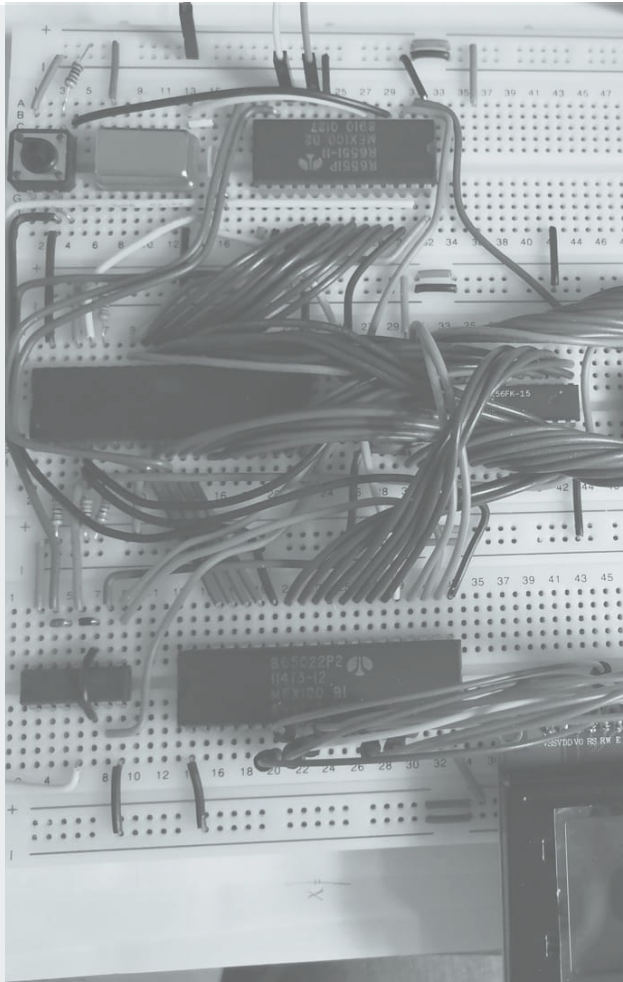
AlexForth for 6502

2022.05 updates

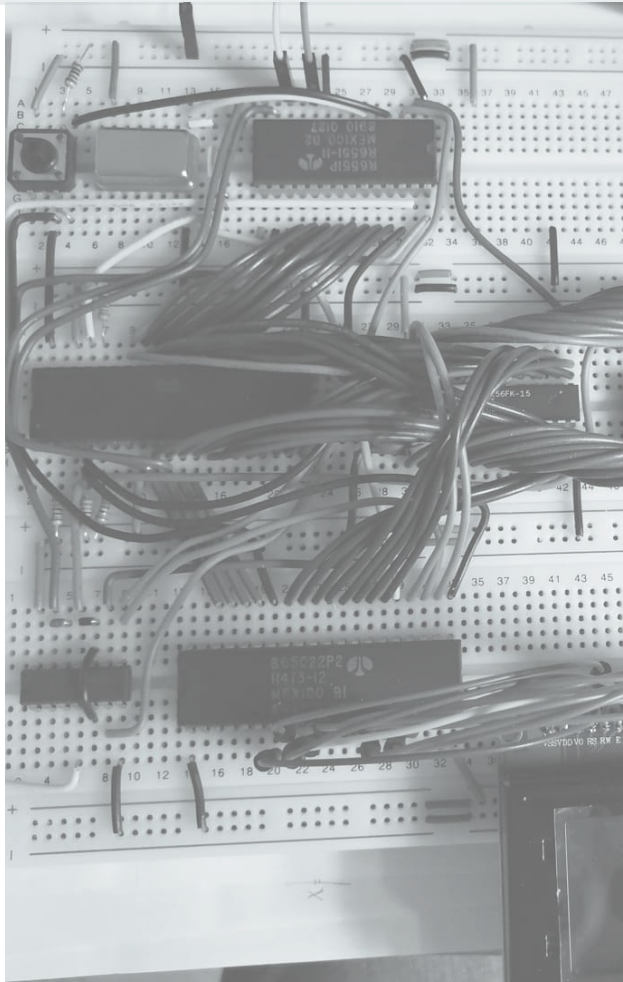
Alexandre Dumont
@adumont

#FORTH2020
May, 14th, 2022



Content

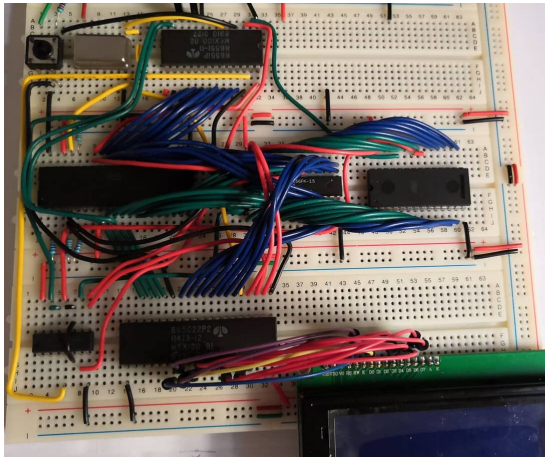
- Introduction
- New features
- Optimizations
  - Quick-win strategies
  - Two stage compilation
- Documentation
- What’s next?

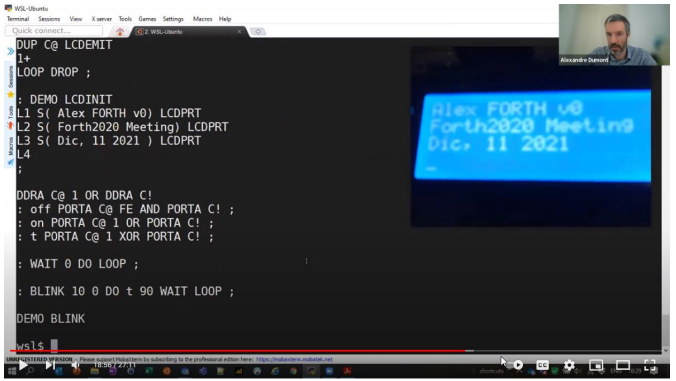


Introduction

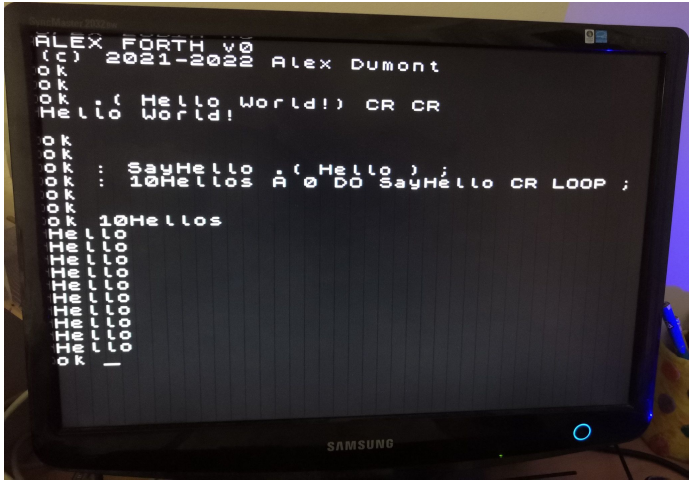
- **AlexForth** is my own implementation of FORTH
- Implemented from scratch:
  - for my 6502 breadboard computer,
  - Direct Threaded Code (DTC) FORTH
  - written in 6502 assembly and Forth
  - assembled with ca65 (cc65 suite)

- I did a presentation and demo to Forth2020 group in Dec. 2021:
  - See https://adumont.github.io/talks/



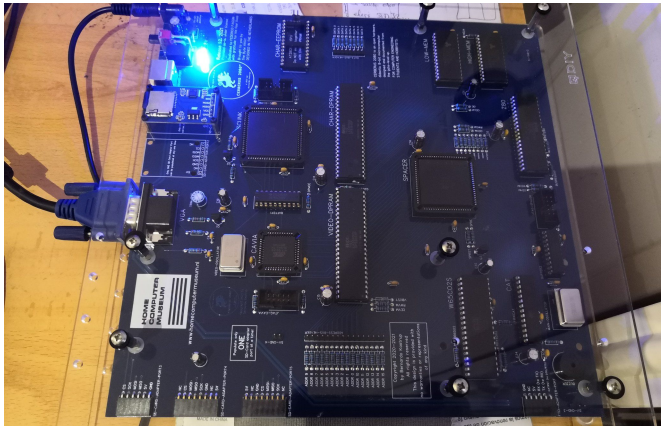


Port to the **Cerberus2080**



- **AlexForth** also runs on the Cerberus2080!

- **Cerberus2080** is a *modern retro computer*
  - Designed by **The Byte Attic**, OSHW
  - Powered by a **6502** ✓ and a **Z80** ✗
  - **PS/2** keyboard input ✓
  - **VGA** output ✓
  - Mass storage (**SD card**) ✗



✓ / ✗ indicate support in AlexForth

Updates

Dictionary updates

- **New words**:
  - `VALUE / TO, CONSTANT`
  - `POSTPONE`
  - `DEFER / IS`
  - `?DO`
  - `< <= > >=` comparison operators
  - Signed output in `.S` and `.`
  - Stack pointer manipulation: `SP@, SP!, RP@, RP!`
    (Can be useful for tracing)

- Rewritten `CREATE/DOES>` to be more compact/efficient

Number base support

- The base (radix) can be changed using **DEC**, **BIN**, **OCT** or **HEX** (default base)
- While in any **BASE**, you can always input numbers using a prefix to force a different base:
  - **#** decimal: `#12`
  - **$** hexadecimal: `$FF`
  - **o** octal: `o1017`
  - **%** binary: `%101100`
- **Notice**:
  - The current base is stored in **BASE** (read/modify using **BASE @** / **BASE !**)
  - Only base 2, 8, 10 and 16 are supported

FORTH decompiler (SEE)*

Example:

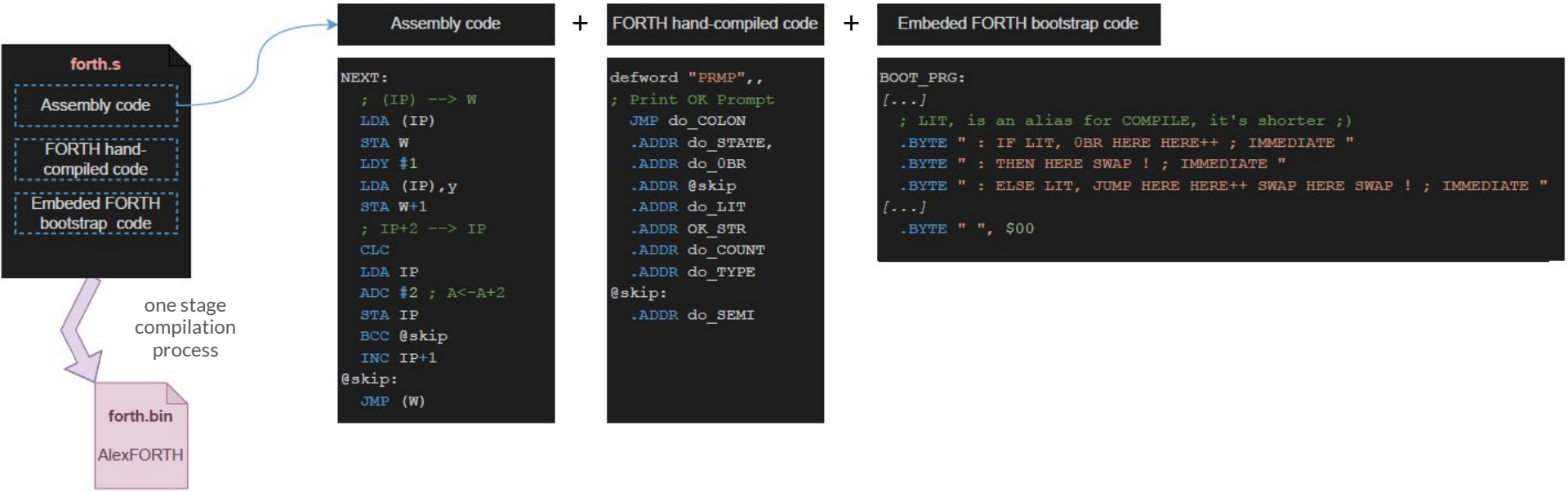
```
ok : T BEFORE DO INSIDE LOOP AFTER ;

ok SEE T
02F3 81FE COLON
02F5 02B7 BEFORE
02F7 8B6B *DO
02F9 02D2 INSIDE
02FB 8B90 *LOOP
02FD 02F9 02F9
02FF 02C4 AFTER
0301 821B EXIT
ok
```

* Atm, **SEE** support is limited, some constructs won't show properly (words not finished with EXIT for example).

Optimization

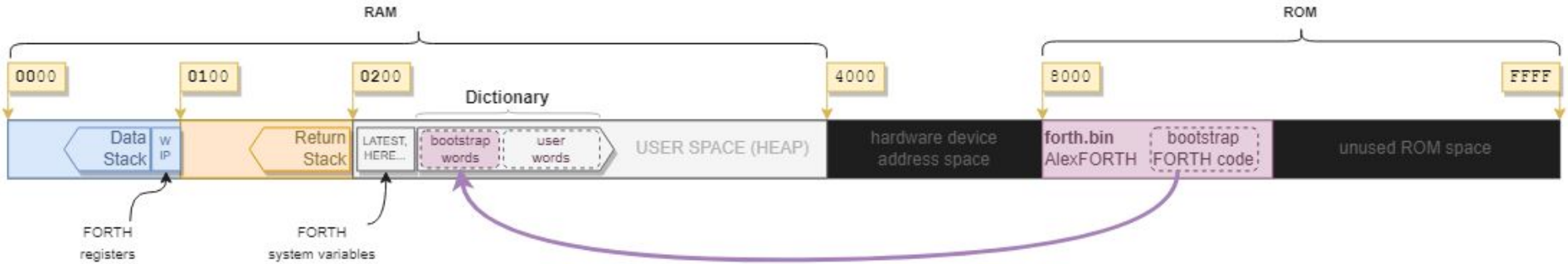
Overview of the source code

AlexFORTH is written in 6502 assembly, but in the `forth.s` file we can see different types of code:



Motivation for optimizations

Until recently AlexFORTH had the *bootstrap FORTH code* embedded as a string into the source file forth.s.
The bootstrap code was interpreted and compiled (into RAM) on every boot of the system:



When bootstrap code is **embeded as string** in the for FORTH rom image, it is **compiled at boot time**

- **RAM space is consumed** (in addition to ROM for the string)
- Bootstrap Compilation is **time consuming**

Measure: Boot time

Boot time definition:

- number of clock cycles
- from the **CPU reset**
- **to the end of bootstrap code compilation**

(= when the user is shown the OK prompt)

Easy to measure with my emulator

Preliminary results (no optimizations):

Empty bootstrap code: < 3000 cycles

Depending of bootstrap code: up to 3.8M cycles

What impacts the boot time?

Heavily related to the interpretation and compilation of the bootstrap code, so:

- Outer interpreter / Compilation time
- Dictionary lookups

Quick-win strategies to optimize boot time

- Convert some words to primitive
  - Can apply to any words, like those that were in bootstrap code
  - Or other words, especially words that are involved in the outer interpreter and compiling words
- Accelerate Dictionary lookup
  - Rearrange words in the dictionary
  - Shortcuts for often used words: : ; @ ! “,” (1 char words, as it’s very easy to compare them)
- Move heavily used FORTH variables to zeropage (slightly faster instructions clock cycles)
- Refactor words (into a new primitive) and rework some algorithms
- Remove words from Bootstrap code and hand-compile them

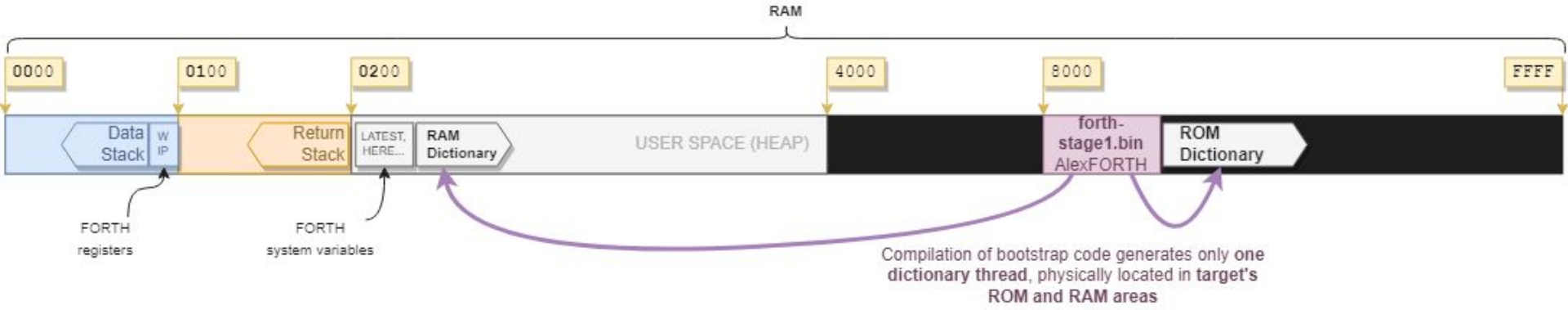
Results: down from 3.8M cycles to 1.8M cycles

Strategy	Effort	Impact
Primitives	High	**40.44%**
ReFactor/Primitive	High	**22.61%**
Dictionary order	Low	19.51%
Shortcuts	Low	9.72%
Hand compiled	High	3.95%
Other	-	3.77%
Grand Total		**100.00%**

Impact on Boot Time Reduction

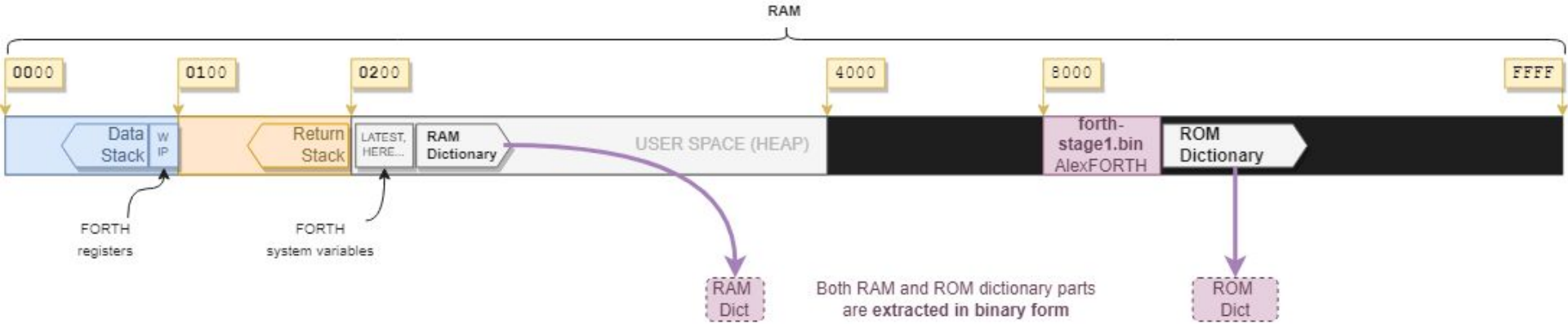
Two-stages compilation: **Stage 1**

Bootstrap.f is now in a separate file. It is feed to the outer interpreter in an **emulator** (xcompiler).

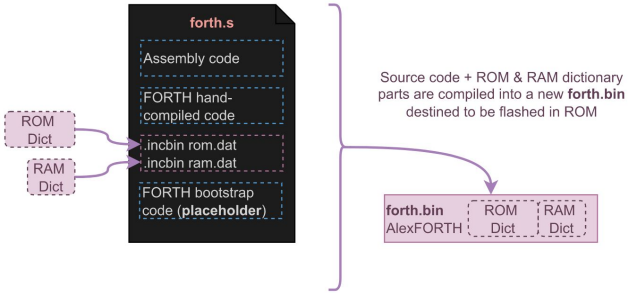


Two-stages compilation: **Stage 1**

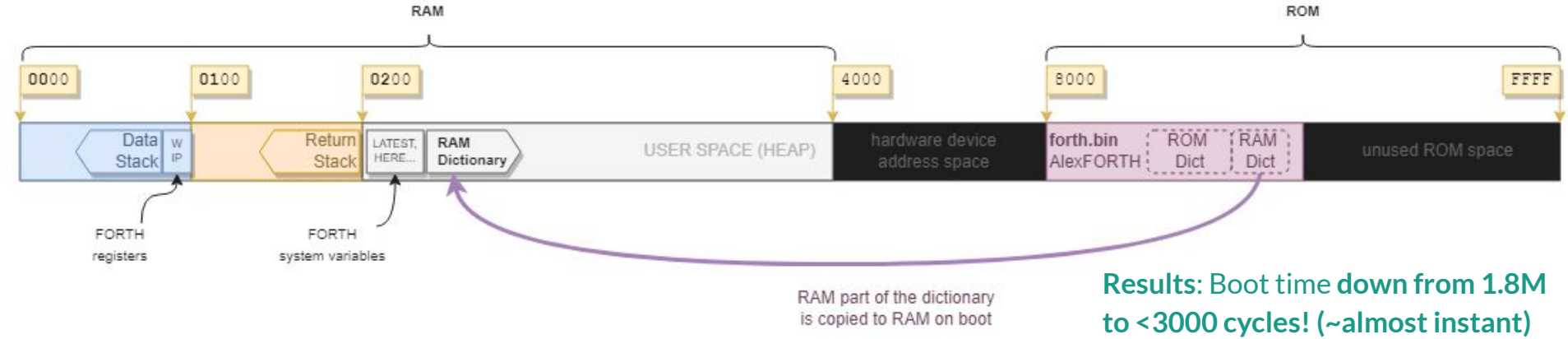
When the (forth) compilation of bootstrap ends, xcompiler will extract ROM and RAM parts of the dictionary into 2 files.



Two-stages compilation: **Stage 2**



The `forth.s` file is compiled again, but this time **embedding the RAM and ROM binary images extracted in Stage 1**, thus generating a single forth-hw.bin binary image, suitable to be flashed in the target 65C02 hardware computer (or run in a target 65C02 emulator)



Results: Boot time **down from 1.8M to <3000 cycles! (~almost instant)**

Documentation

Documentation (1)

Better documentation in the Github repo, in particular:

- **Some usage examples**
  - STRING and CHAR
  - User input (ASK$, ASK#)
  - FORTH Decompiler (SEE)
  - Debugging & Tracing
  - Exceptions (THROW / CATCH)
  - Free Memory

Ask for numbers

```
: ASK# ( -- N ) \ returns the number on the stack
  0A PARSE  ( ADDR LEN )
  NUMBER    ( N )
;

: TEST#
  3 0 DO
    \ Show question
    .( What's your age? )
    \ Ask for the answer and return the number on the stack
    ASK#
    \ Reply using the number
    .( Your age is ) . CR
  LOOP
;
```

Documentation (2)

- **Anatomy of compiled words**

Documents how words are compiled into the dictionary, using SEE and manual annotations:

- **Literals**: String literal, Char literals,...
- **Conditionals**: IF THEN, IF ELSE THEN
- **BEGIN Loops**: BEGIN AGAIN, BEGIN UNTIL,
- BEGIN WHILE REPEAT
- **DO Loops**: DO LOOP, DO +LOOP, ?DO LOOP, ...
- **Defining words**: CREATE and CREATE DOES> examples

```
ok : TEST HEAD IF IF-CLAUSE ELSE ELSE-CLAUSE THEN TAIL ;

ok SEE TEST
0887 8170 COLON
0889 082F HEAD
088B 8B75 0BR           ----+       Jump to ELSE code
088D 0895 0895              |
088F 0840 IF-CLAUSE         |
0891 8BCE JUMP          ----|----+  Jump over ELSE code
0893 0897 0897              |    |  (to the TAIL clause)
0895 087A ELSE-CLAUSE   <---+    |
0897 0851 TAIL          <--------+
0899 818D EXIT
ok
```

What's next?

Limitations (at the moment)

- ~~Only Hexadecimal representation (no BASE/conversions)~~ ✓
- Only Integer numbers (no floating points)
- Only one dictionary (no vocabulary)
- No stack overflow/underflow verification. Easy to crash!
- Case sensitive words, all predefined words are capital case
- No mass storage, no block feature, no save/restore, no editor
- Doesn’t adhere to any standard

Thank you!

Links

AlexForth repository on Github: https://github.com/adumont/hb6502/tree/main/forth

- Source code
- Documentation
- Tools (emulator, 2 stage compiler, debugger)

AlexForth source for the **Cerberus2080**: https://github.com/adumont/CERBERUS2080

My web page with links to all my projects (and these slides): https://adumont.github.io/

Contact me on Twitter: @adumont https://twitter.com/adumont

Alexandre Dumont
@adumont

Backup slides

AlexForth - dev updates 2022.05

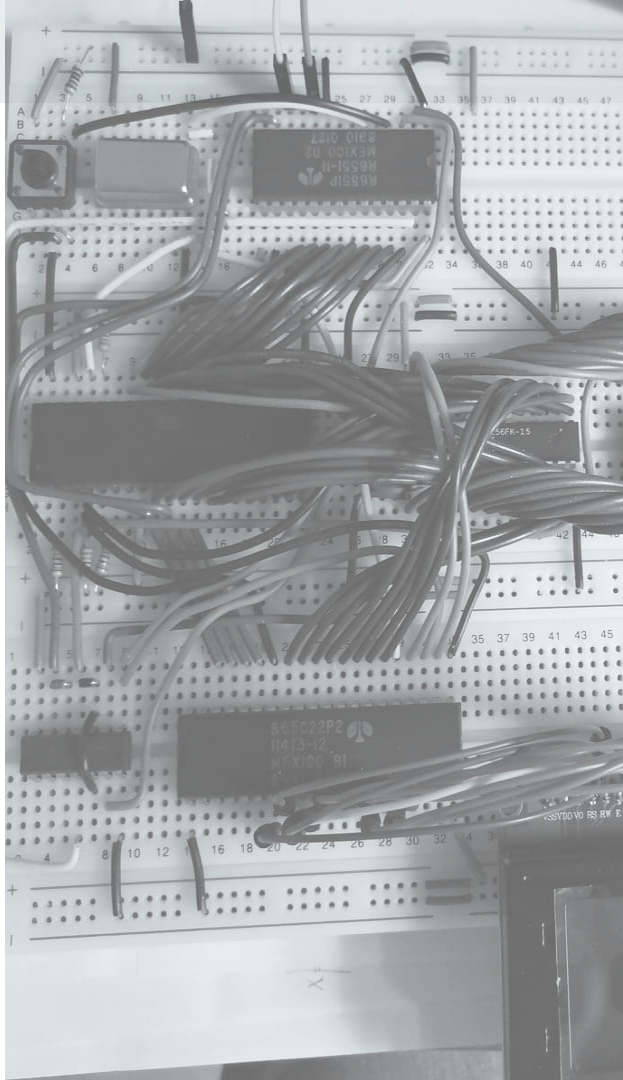
Tentative agenda

- Introduction
- New features
  - New words
  - Tools
- Documentation
- Optimizations
  - Strategies
  - Two stage compilation
- Next steps

Alexandre Dumont
@adumont

#FORTH2020

May, 14th, 2022



Disclaimer

- AlexForth is a personal learning experience
- It doesn't adhere any standards (for now)