Assembly code

```
NEXT:
  ; (IP) --> W
  LDA (IP)
  STA W
  LDY #1
  LDA (IP),y
  STA W+1
  ; IP+2 --> IP
  CLC
  LDA IP
  ADC #2 ; A<-A+2
  STA IP
  BCC @skip
  INC IP+1
@skip:
  JMP (W)
```

FORTH hand-compiled code

```
defword "PRMP",,
  JMP do_COLON
  ; Print OK Prompt
  .ADDR do_STATE,
  .ADDR do_0BR
  .ADDR @skip
  .ADDR do_LIT
  .ADDR OK_STR
  .ADDR do_COUNT
  .ADDR do_TYPE
@skip:
  .ADDR do_SEMI
```

Embedded FORTH bootstrap code

```
BOOT_PRG:
  [...]
  ; LIT, is an alias for COMPILE, it's shorter ;)
  .BYTE " : IF LIT, 0BR HERE HERE++ ; IMMEDIATE "
  .BYTE " : THEN HERE SWAP ! ; IMMEDIATE "
  .BYTE " : ELSE LIT, JUMP HERE HERE++ SWAP HERE SWAP ! ; IMMEDIATE "
  [...]
  .BYTE " ", $00
```

forth.s
- Assembly code
- FORTH hand-compiled code
- Embedded FORTH bootstrap code

Memory map (RAM / ROM): 0000, 0100, 0200, 4000, 8000, FFFF

Data Stack | W IP | Return Stack | LATEST. HERE... | Dictionary | USER SPACE (HEAP) | hardware device address space | **forth.bin** AlexFORTH | unused ROM space

FORTH registers — FORTH system variables

---

forth.s → **forth.bin** AlexFORTH

Memory map (RAM / ROM): 0000, 0100, 0200, 4000, 8000, FFFF

Data Stack | W IP | Return Stack | LATEST. HERE... | **Dictionary** (bootstrap words / user words) | USER SPACE (HEAP) | hardware device address space | **forth.bin** AlexFORTH / bootstrap FORTH code | unused ROM space

FORTH registers — FORTH system variables

When bootstrap code is **embedded 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**

Stage 1 Compilation

Source code is now split in two files:
- forth.s
- bootstrap.f

Stage 1:
- build forth.s (without bootstrap code)
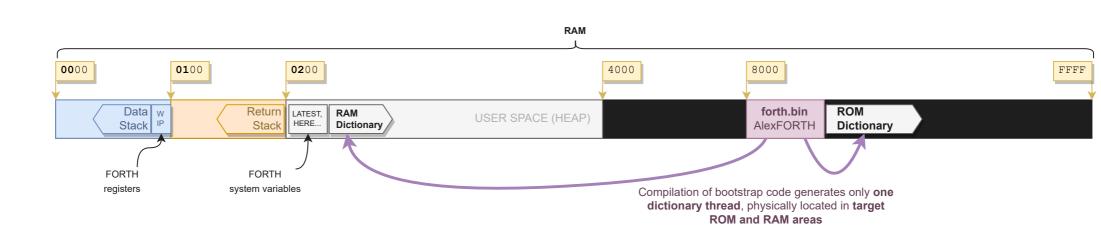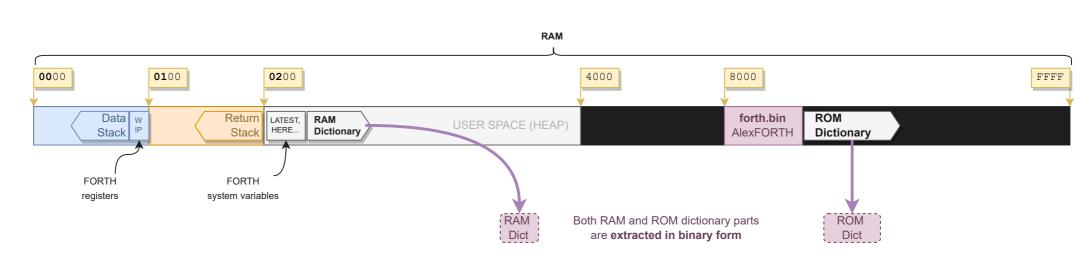- run xcompiler.py

xcompiler will:
- run a 6502 emulated computer that will run the forth.bin image
- feed bootstrap code as input
- FORTH will interpret and compile it
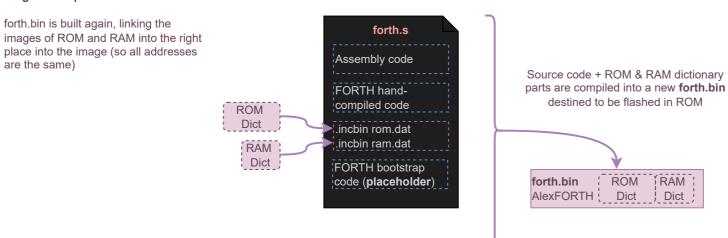- At the end of the code, it will terminate FORTH
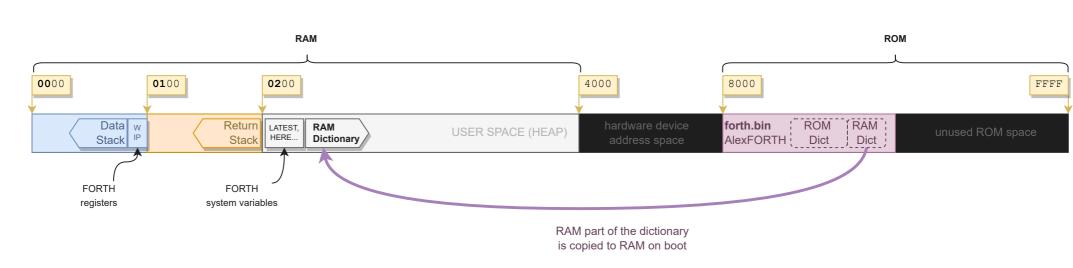- Finally it will extract the generated binary image of the dictionary

forth.s
- Assembly code
- FORTH hand-compiled code
- FORTH bootstrap code (placeholder)

bootstrap.f
- FORTH bootstrap code

Memory map (RAM): 0000, 0100, 0200, 4000, 8000, FFFF

Data Stack | W IP | Return Stack | LATEST. HERE... | **RAM Dictionary** | USER SPACE (HEAP) | **forth.bin** AlexFORTH | **ROM Dictionary**

FORTH registers — FORTH system variables

Compilation of bootstrap code generates only **one dictionary thread**, physically located in **target ROM and RAM areas**

Memory map (RAM): 0000, 0100, 0200, 4000, 8000, FFFF

Data Stack | W IP | Return Stack | LATEST. HERE... | **RAM Dictionary** | USER SPACE (HEAP) | **forth.bin** AlexFORTH | **ROM Dictionary**

FORTH registers — FORTH system variables

RAM Dict → / ROM Dict → Both RAM and ROM dictionary parts are **extracted in binary form**

Stage 2 Compilation

forth.bin is built again, linking the images of ROM and RAM into the right place into the image (so all addresses are the same)

forth.s
- Assembly code
- FORTH hand-compiled code
- .incbin rom.dat
- .incbin ram.dat
- FORTH bootstrap code (placeholder)

ROM Dict → RAM Dict →

Source code + ROM & RAM dictionary parts are compiled into a new **forth.bin** destined to be flashed in ROM

forth.bin AlexFORTH | ROM Dict | RAM Dict

Memory map (RAM / ROM): 0000, 0100, 0200, 4000, 8000, FFFF

Data Stack | W IP | Return Stack | LATEST. HERE... | **RAM Dictionary** | USER SPACE (HEAP) | hardware device address space | **forth.bin** AlexFORTH / ROM Dict / RAM Dict | unused ROM space

FORTH registers — FORTH system variables

RAM part of the dictionary is copied to RAM on boot

Result: is a **fast boot** with a single-threaded dictionary split across ROM and RAM.
- Pointers are restored (LATEST, HERE...)
- No bootstrap code compilation on boot.
- More free RAM
- ROM file is smaller