PACE figFORTH Implementation

Adapting the figFORTH Glossary to a word-addressing computer

David Kilbridge

FORTH Day 2009

November 21, 2009

Review of implementation project

- Nine reference implementations
- All written to a common API specification, the figFORTH glossary
  - which evolved during the project
- Published and released into the public domain at 1979 WCCF

Design issues

- How are data to be represented in memory?
- What is the connection between the Forth virtual machine and the actual hardware?

Data representation: obvious choices

- One stack cell per 16-bit word
  - Stack pointers are native addresses
  - Change by 1 to push or pop
- One dictionary field per 16-bit word
  - Compiler stores one address per word
  - Inner interpreter deals with native addresses

How to store text data?

- Two choices:
  - One character per word
  - Two characters packed into 1 word

One character per word

- Probably the best choice today
  - Storage is cheap
  - Allows i18n via UTF-16 encoding
  - Each character gets a native address
  - Possible issues with data exchange
    - May have to unpack characters on input, pack on output

Two characters per word

- My choice at the time
  - Most efficient use of expensive storage
  - Unicode, ISO-8859 didn’t exist
  - Allowed (FIND) to compare 2 bytes at a time
    - WORD aligns strings consistently with count in the MSB
  - Used existing I/O subsystem transferring packed characters

Implications of packed characters

- Need a way to address each byte of a word
- Forced to add abstract addresses for bytes
- How does byte addressing work?

New conversion words

- BYTE addr --- baddr (= addr * 2)
  - Returns address of most-significant byte of word (big-endian)
  - Used ~ 15x in figFORTH nucleus
- CELL baddr --- addr (= baddr / 2)
  - Returns address of word containing byte
  - Used ~ 7x in figFORTH nucleus

Words that use byte addresses

- C@
- C!
- CMOVE
- COUNT
- ENCLOSE
- EXPECT
- HLD @
- (LINE)
- (NUMBER)
- TOGGLE
- TRAVERSE
- -TRAILING
- TYPE
- #>

Elaborations to Glossary, part 1

C@	baddr --- b
C!	b baddr ---
CMOVE	baddr1 baddr2 count ---
COUNT	addr1 --- baddr2 n
ENCLOSE	baddr1 c --- baddr1 n1 n2 n3
EXPECT	baddr count ---
HLD	(contains a byte address)

Elaborations to Glossary, part 2

(LINE)	n1 n2 --- baddr count
(NUMBER)	d1 baddr1 --- d2 baddr2
TOGGLE	baddr b ---
TRAVERSE	baddr1 n --- baddr2
-TRAILING	baddr n1 --- baddr n2
TYPE	baddr count ---
#>	d --- baddr count

Word addresses used for:

- SP, RP, IP, W, UP, @, !
- DP and all dictionary fields
- TIB, PAD, disk buffers
- Most words in the figFORTH glossary

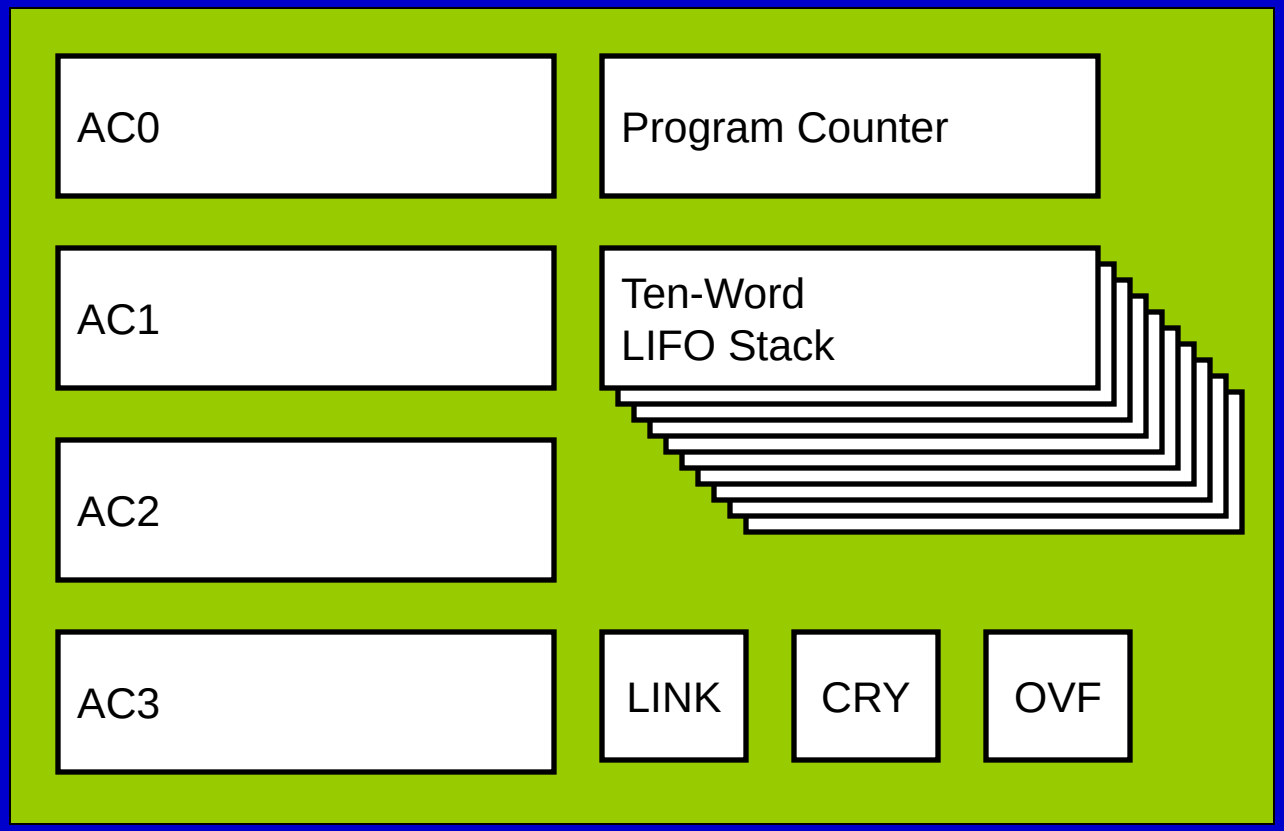
Objections to byte addressing

- Creates a 32K-word barrier in memory
  - A problem we wished we had!
  - No worse off than with a byte-addressed machine with 64K bytes of memory
- Schizophrenic API
  - Have two distinct pointer types without strong-typing support from the language
  - Forces programmer to remember which type of address to use where

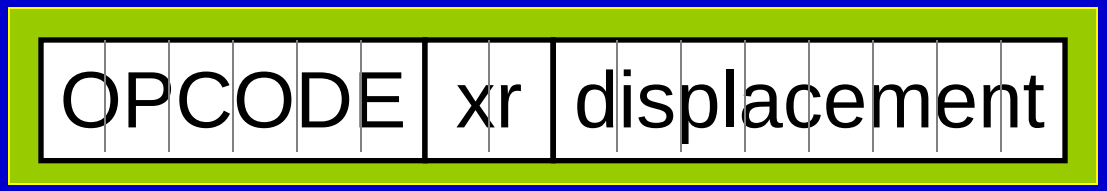
PACE architecture

- Unit of addressing = 16-bit word
- All machine instructions are 1 word long
  - No inline addresses or extension words
  - All EAs are short offsets from a base…
    …or indirectly from there (LD, ST, JMP, JSR)
- Memory-mapped (and bit-banging) I/O
- 6-level priority interrupts

Programmer's model



Typical memory reference instruction:



- Four pages directly addressable
  - xr = 0: 0000 to 00FF (or FF80 to 007F)
  - xr = 1: (PC) - 80 to (PC) + 7F
  - xr = 2: (AC2) - 80 to (AC2) + 7F
  - xr = 3: (AC3) - 80 to (AC3) + 7F

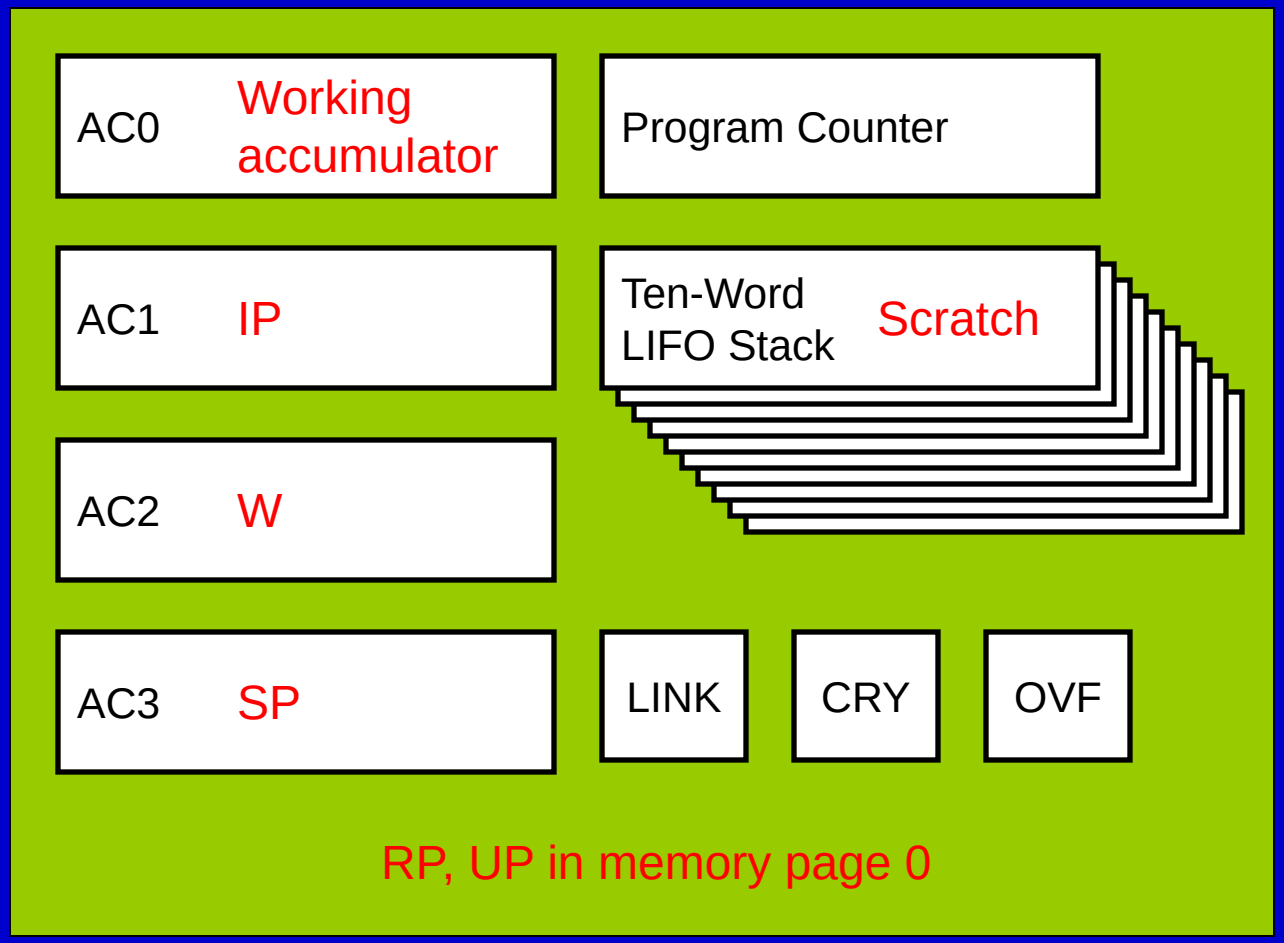
Capabilities of all registers

- $\leftarrow\rightarrow$ flags, stack, other registers, memory
- Equality test with memory
- Add from memory
- Load immed., Complement, Add immed.
- Add w/wo carry, And, Xor other registers
- Shift and rotate

Special register capabilities

- AC0:
  - Test =0, ≠0, ≥0, <0, and bits 0, 1, 2
  - Additional operations with memory operands:
    - And, Or, Subtract w/borrow, Decimal add, Greater-than test, Mask test, Load byte
  - Load and store Indirect
- AC2 and AC3:
  - Serve as base or index registers

Forth register assignment



Use internal stack as parameter or return stack?

- Only 10 words deep
- Only top word is accessible
- Not easily extended into memory
  - Would need to enable and service stack full/empty interrupts
- Just use for register saving, JSRs and interrupts (if required by the installation)

SP, IP, W usage

```
LIT:    .WORD   .+1
        RCPY    IP,X        ; PICK UP
        LD      0,(X)       ;  VALUE
        AISZ    IP,1        ; STEP IP OVER
PUSH:   AISZ    SP,-1       ; EXTEND STACK
PUT:    ST      0,(SP)      ; STORE VALUE
NEXT:   RCPY    IP,X
        AISZ    IP,1        ; INCREMENT IP
        LD      W,(X)       ; ADDR OF NEXT WORD
        JMP     @(W)        ; JUMP THRU CODE ADDR
```

RP usage

```
TOR:    .WORD   .+1
        DSZ     RP          ; EXTEND RETURN STACK
        LD      0,(SP)      ; GET DATA ITEM
        ST      0,@RP       ; PUT ON RETURN STACK
        JMP     POP         ; POP FROM DATA STACK

FROMR:  .WORD   .+1
        LD      0,@RP       ; GET FROM RETURN STK
        ISZ     RP          ; POP RETURN STACK
        JMP     PUSH        ; PUSH ON DATA STACK

R:      .WORD   .+1
        LD      0,@RP       ; GET NONDESTRUCTIVE
        JMP     PUSH        ; PUSH ON DATA STACK
```

Summary

- The PACE implementation split the figFORTH Glossary's concept of a single type of address into two:
  - Most addresses in the Forth virtual machine are native word addresses, allowing efficient execution
  - Byte addresses are used for packed text handling and compatibility with 8-bit algorithms