Implementing Forth on the RCA 1802

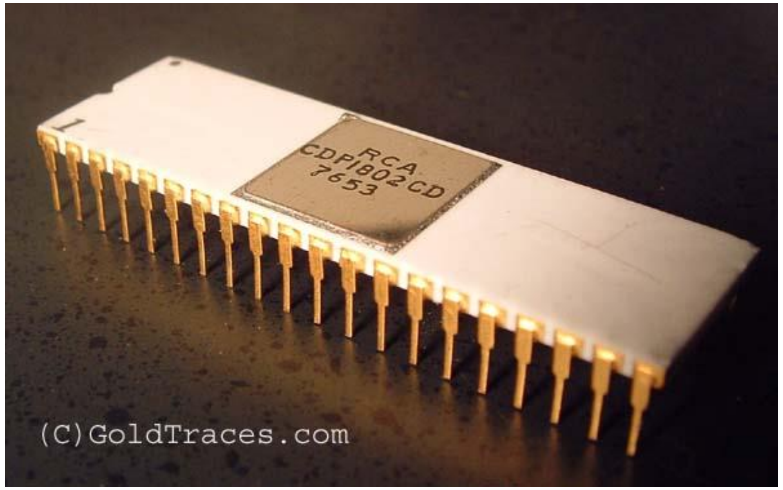
A 40-year-old resource-starved processor architecture

Harold Rabbie

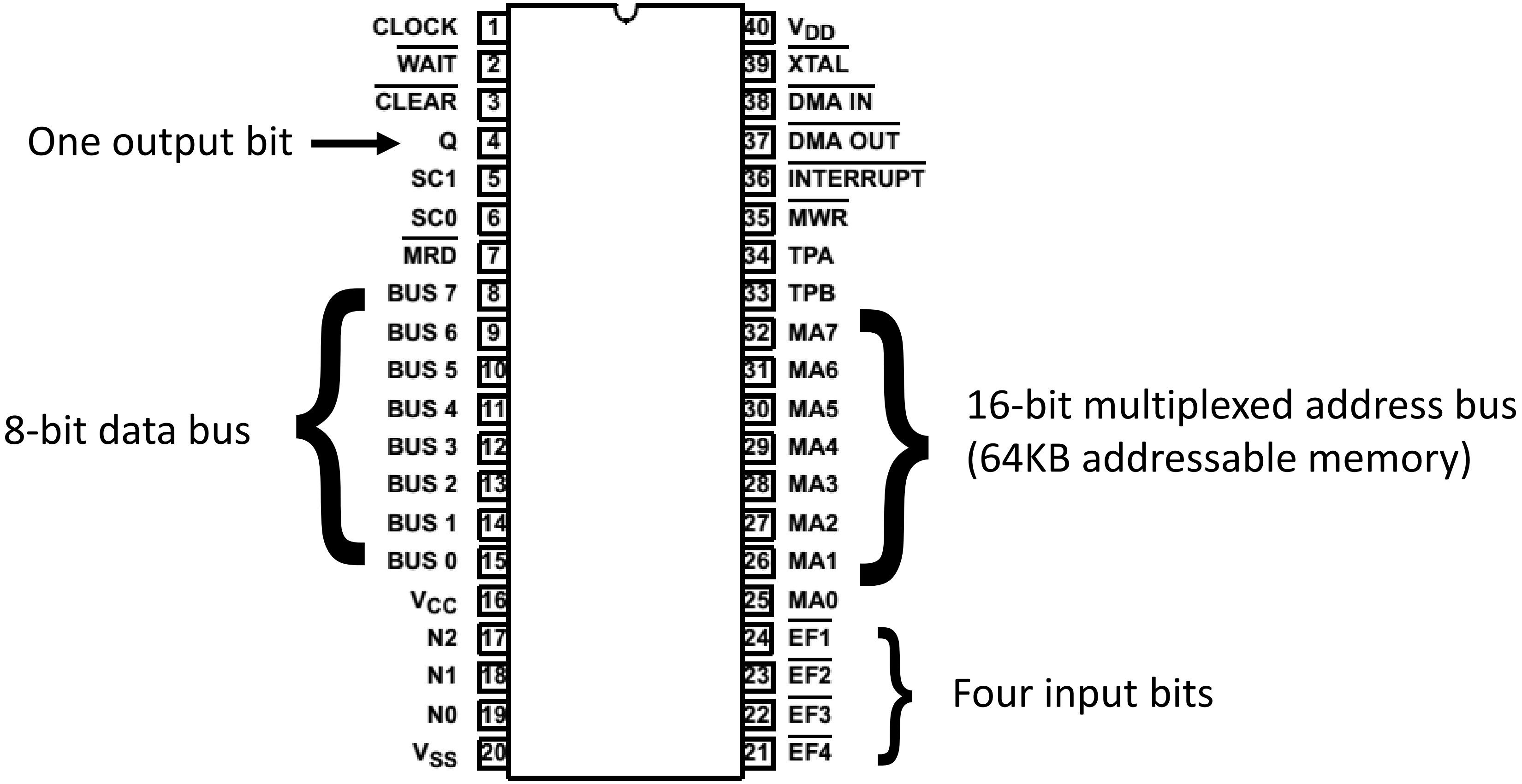
November 2014

RCA 1802 Microcontroller

- First manufactured in 1976
- Static CMOS technology (new at the time)
- Very low power
  - 10 mW at 3.2 MHz
- Radiation hard Silicon-on-Sapphire
  - Used in the Galileo spacecraft mission to Jupiter
- Currently manufactured by Intersil



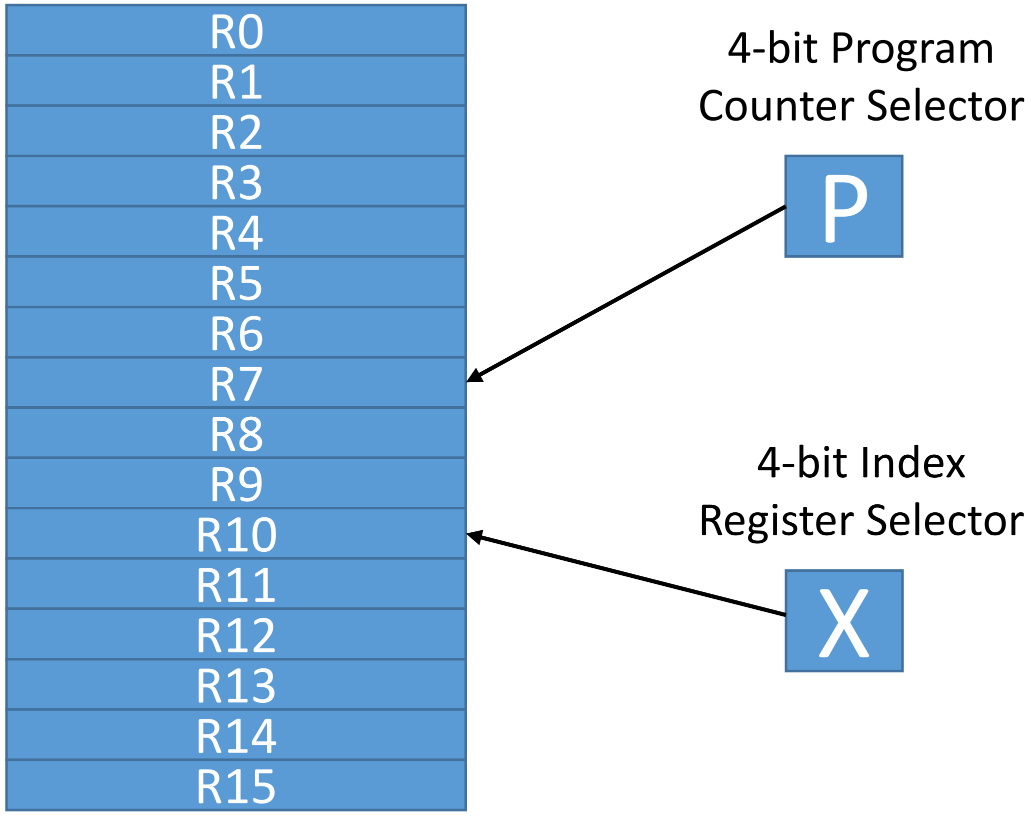
RCA 1802 Hardware Interfaces

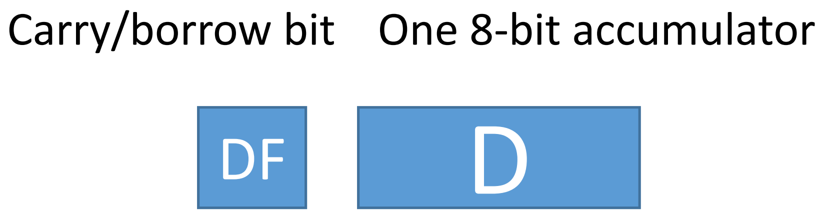


Disk Files
Keyboard
Video
Mouse
Ethernet
Wi-Fi
USB
Serial I/O

RCA 1802 Registers

Sixteen 16-bit pointer registers





Arithmetic is ONLY between the D register and the memory location addressed by the current index register

e.g.
P register contains 7, so R7 is the current program counter

X register contains 10, so R10 is the current index register

Arithmetic instruction at memory location addressed by R7 will operate on D and the value in memory addressed by R10.

RCA 1802 Instruction Set

- Most instructions are 1 byte long
- Most instructions take 16 clock cycles
  - 3.2 MHz clock rate → 200K instr/sec, 5 µsec per instr.
- 8-bit arithmetic instructions
  - D/DF register is always the destination operand
- 11 1-byte instructions that reference a pointer register:
  - GHI, GLO, PHI, PLO, LDN, STR, LDA, INC, DEC, SEP, SEX

4-bit Opcode	4-bit Register

- Short branch 2-byte instructions (within same 256-byte page)
- Long branch 3-byte instructions (anywhere in 64KB address space)

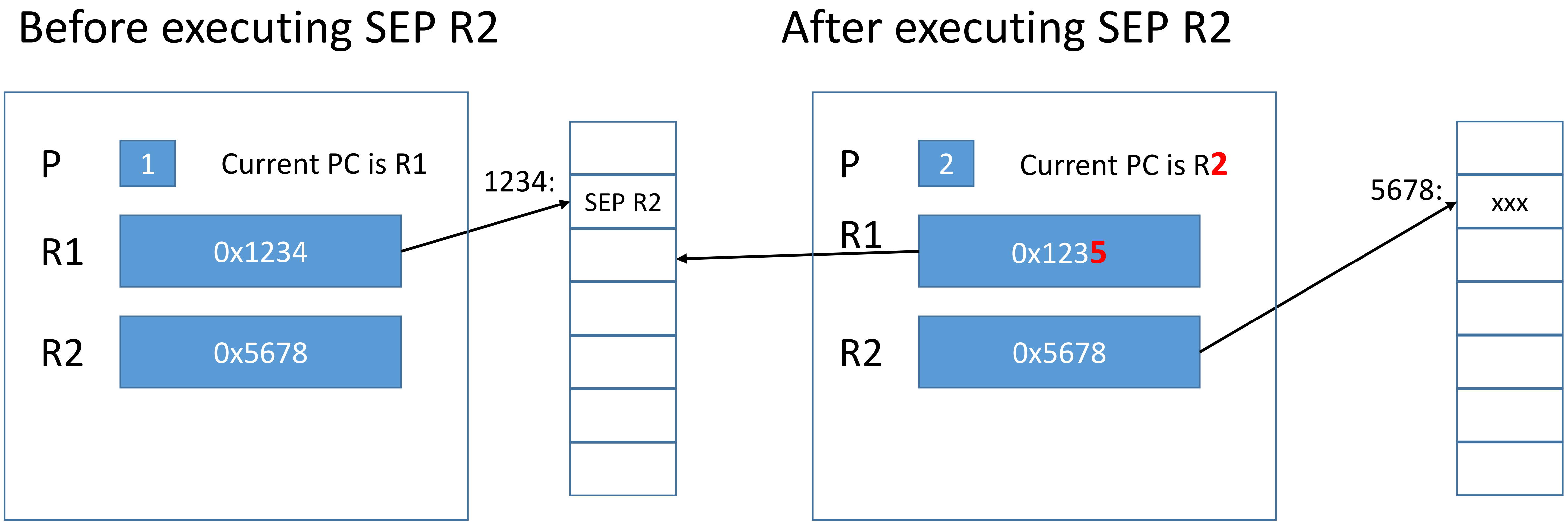
The RCA 1802 Doesn’t Have:

- Conventional call / return instructions
  - The SEP instruction is a possible alternative
- Hardware stacks
  - Need to emulate in software
- Register-to-register arithmetic
  - All arithmetic goes via the D/DF register
- 16/32-bit arithmetic
  - Need to emulate in software with 8-bit operations
- Console I/O
  - Add a UART chip *or*
  - Bit bang using general-purpose I/O bits (EF, Q) *or*
  - Simulate with a host OS

Forth Porting Decisions to Make

- Minimize execution time for most common operations:
  - NEXT, DOCOLON, DOCONST, DOVAR, DOCREATE
  - EXIT, LIT, >R, R>
- How should parameter stack be laid out?
  - Big endian, or little endian?
  - Grow up, or grow down?
- How should return stack be laid out?
  - Big endian, or little endian?
  - Grow up, or grow down?
- Indirect, direct, or subroutine threaded?

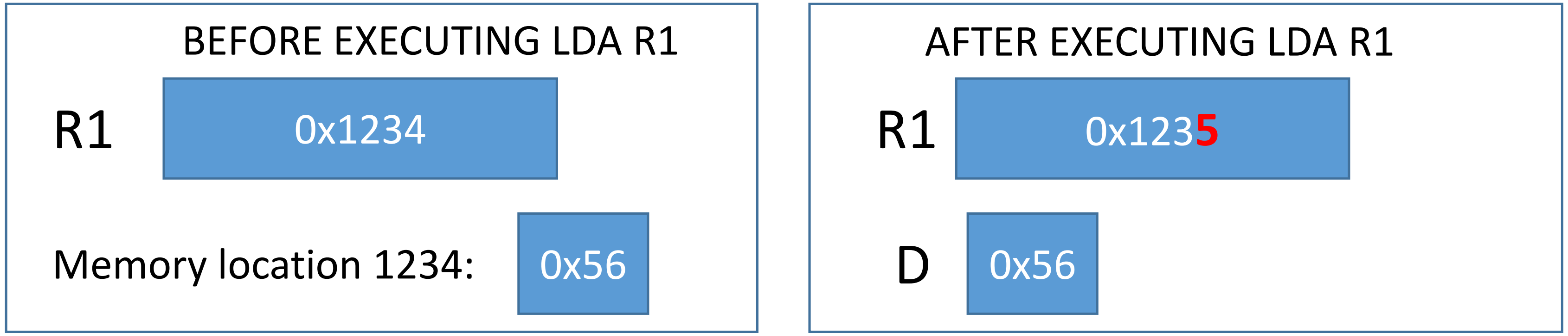
Set Program Counter (SEP) Instruction Example

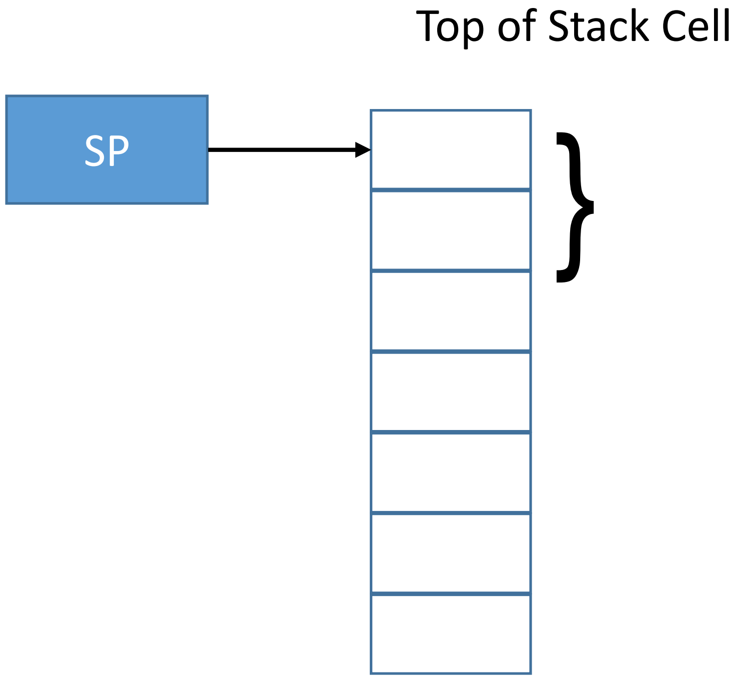


SEP: Only 1 byte (good!) Only 16 different destinations (bad!)

Stack Design – Stacks Grow from High to Low

- RCA 1802 includes the LDA (load and advance) instruction
- e.g. LDA R1 can be used to POP a stack





There's also STXD (store and decrement index register)

Threading Methods `: FOO A B C ;`

- Subroutine Threading

  Header (FOO)

  ```
  subcall A
  subcall B
  subcall C
  jump NEXT
  ```

- Direct Threading

  Header (FOO)

  ```
  subcall docolon
  .DW A
  .DW B
  .DW C
  .DW EXIT
  ```

- Indirect Threading

  Header (FOO)

  ```
  .DW docolon
  .DW A
  .DW B
  .DW C
  .DW EXIT
  ```

Subroutine Threading:

- Body contains machine code
- Not available for RCA 1802, due to lack of general subroutine call instruction

Direct Threading:

- Body starts with machine code
- Needs only a limited number of subroutine call instructions (*)

Indirect Threading:

- Body contains only addresses
- Inner interpreter takes more cycles
- Words are 1 or 2 bytes longer than direct threading

* Except for DOES> case

Direct Threading Example – CONSTANT word

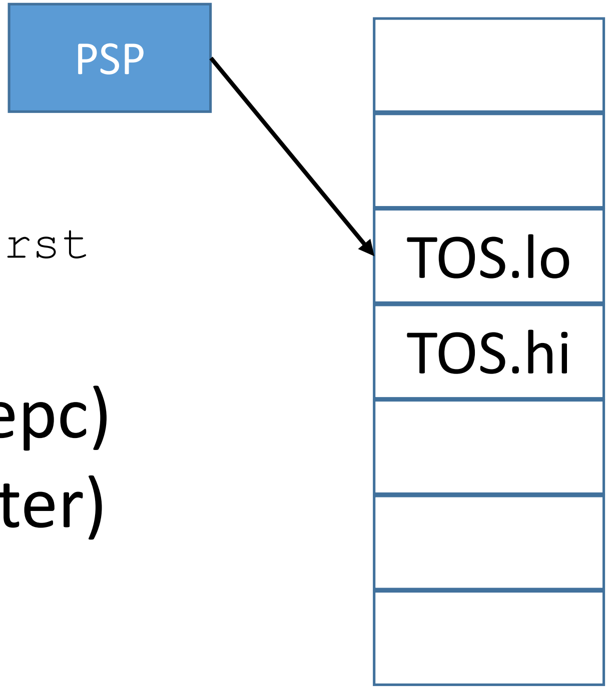
- e.g. `1234 CONSTANT FOO`

Compiles to:

Header (FOO)

```
sep constpc
.DW 1234    ; MSB first
```

Executed with P=0 (codepc)
(R0 is the program counter)

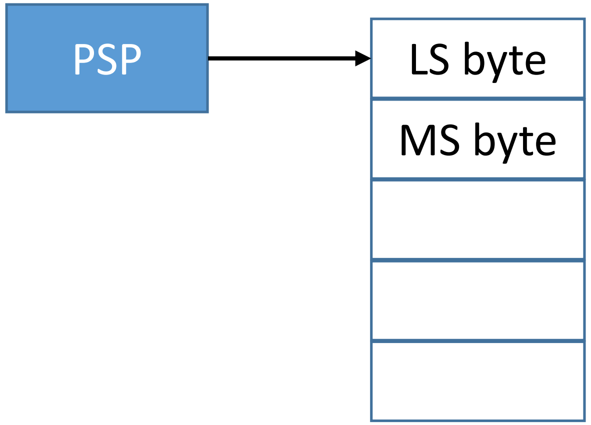



```
; DOCONST, code action of CONSTANT words
        sep nextpc
doconst:
        lda codepc    ; high byte of const
        dec psp       ; param stack ptr
        stxd
        lda codepc    ; low byte of const
        str psp
        br doconst - 1  ; reset constpc
```

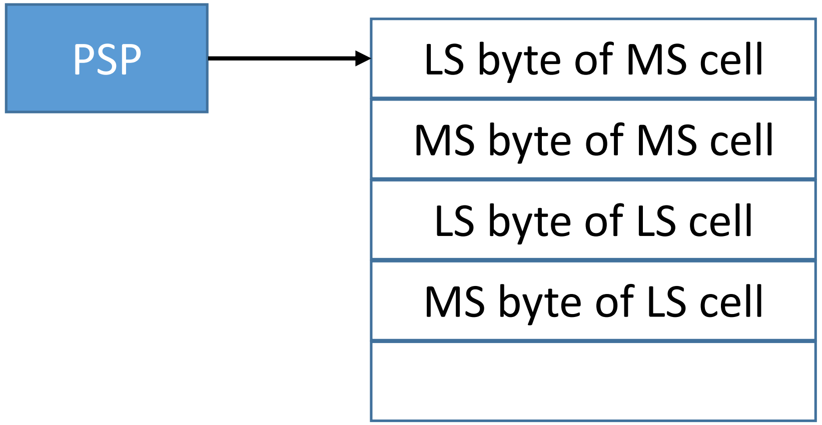
Executed with P=6 (constpc)
(R6 is the program counter)

Stack Endian-ness

- **ANSI 3.1.4.1 Double-cell integers**
  - On the stack, the cell containing the most significant part of a double-cell integer shall be above the cell containing the least significant part.



Single-cell integer on stack stored little-endian



Double-cell integer on stack stored mixed-endian

- Return stack is big-endian to optimize >R and R>

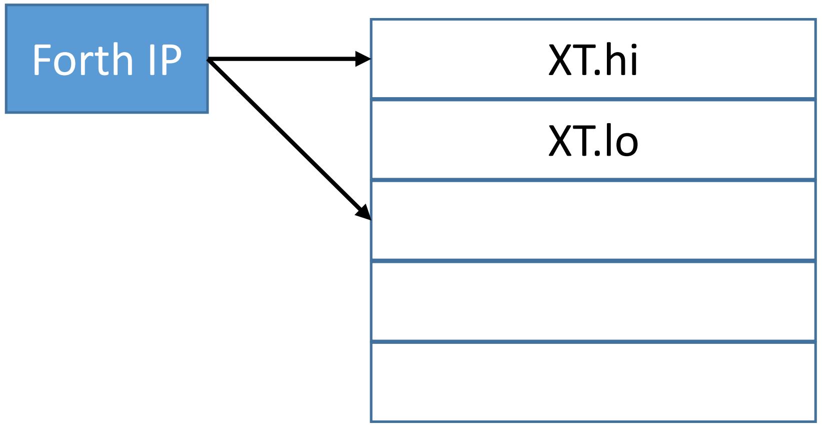
RCA 1802 16-bit Register Usage

- 8 Dedicated Program Counter Registers
  - R0 codepc — machine code words
  - R4 nextpc — inner interpreter — 6 instructions
  - R5 colonpc — words created with : (colon) — 12 instructions
  - R6 constpc — words created with CONSTANT or VALUE — 7 instructions
  - R7 varpc — words created with VARIABLE or CREATE1 — 7 instructions
  - R8 createpc — words created with CREATE — 15 instructions
  - R9 userpc — words created with USER — 8 instructions
  - R10 execpc — code field of EXECUTE — 6 instructions

- 3 Forth Virtual Machine Registers
  - R1 ip — Inner Interpreter Pointer
  - R2 psp — Parameter Stack Pointer - usually set as the index register (SEX 2)
  - R3 rsp — Return Stack Pointer

Inner Interpreter (6 instructions)

```
; NEXT, dispatch next execution token from Forth Instruction Pointer
; entered by sep nextpc
      sep codepc         ; jump to xt
nextd:
      lda ip             ; high byte of xt
      phi codepc
      lda ip             ; low byte of xt
      plo codepc
      br nextd - 1       ; reset nextpc
```



Compiling a VARIABLE word

- **e.g** `VARIABLE FOO`

Compiles to:

Header (FOO)

```
sep varpc
.DW xxxx
```

```
; DOVAR, code action of VARIABLE words
; entered by sep varpc
      sep nextpc
dovar:
      ghi codepc ; high byte of addr
      dec psp
      stxd
      glo codepc ; low byte of addr
      str psp
      br dovar - 1     ; reset varpc
```

7 Instructions

Executed with P=0
`codepc` is the program counter

Executed with P=7
`varpc` is the program counter

DOES> Overrides default runtime semantics for CREATE’d word

<u>Other language</u>	<u>FORTH</u>
	`: char-array CREATE ALLOT DOES> + ;`
`char a[10];`	`10 char-array a`
`a[5] = 42;`	`42     5 a     C!`

Defining word defines a class with a single method
Default runtime semantics push address of body

Using CREATE to define a word

- **e.g** `CREATE FOO`

Compiles to:

Header (FOO)

```
sep createpc
.DW noop
; may be overridden by DOES>
; followed by BODY

noop:    sep nextpc
```

```
; DOCREATE, code action of CREATE'd words
; entered by sep createpc – 15 instructions!

        sep codepc
docreate:
        lda codepc              ; high byte of DOES> part
        phi temp1
        lda codepc              ; low byte of DOES>
        plo temp1
        ghi codepc              ; push PFA to param stack
        dec psp
        stxd
        glo codepc
        str psp
        ghi temp1               ; need to enter DOES> part
        phi codepc              ; with codepc
        glo temp1
        plo codepc
        br docreate - 1         ; reset createpc
```

Why did <BUILDS go away?

There is a need to distinguish between cases where DOES> may or may not be used

Fig-Forth `: char-array <BUILDS ALLOT DOES> + ;`
ANS Forth `: char-array CREATE ALLOT DOES> + ;`

Creating Word	FIG-Forth	ANS-Forth	Camel Forth 1802
<BUILDS	DOES> *is* used		
CREATE	DOES> *is not* used	DOES> *may be* used	DOES> *may be* used
CREATE1			DOES> *may not* be used

Example usage `: VARIABLE CREATE1 CELL ALLOT ;`

CamelForth ANSI-compliant FORTH compiler

- Brad Rodriguez, McMaster University, Ontario, Canada
- Designer of “Pathetic Instruction Set Computer”
- CamelForth project started 1994
- Ports available for
  - Intel 8051, 8086
  - Zilog Z80, Z180
  - Motorola 6809
  - TI MSP430
  - RCA 1802

Word Header in CamelForth 1802

LFA	Link	Pointer to NFA of previous word in the dictionary
	Flag	0 = Normal, 1 = Immediate
NFA	S | Count	Smudge bit + Length of word name
	name[0]	Name of word (up to 127 characters)
	name[1]	
	……	
CFA/XT	Code field	

- Code words – assembler instructions
- Colon words – **`SEP colonpc`** + list of XT’s
- CONSTANT, VALUE – **`SEP constpc`** + 2 bytes of data
- VARIABLE, CREATE1 – **`SEP varpc`** + body data
- CREATE – **`SEP createpc +`** XT + body data

ANSI X3.215-1994 compliance of CF1802

Word Set	Standard Words	CamelForth 1802	Notes
6.1 Core Words	133	133	
6.2 Core Extension Words	46	43	3 obsolescent
8.6.1 Double-Number Words	20	3	M+, DNEGATE, DABS
15.6.1 Programming-Tools Words	5	4	SEE not implemented
15.6.2 Programming-Tools Extension Words	13	8	ASSEMBLER, EDITOR not implemented
17.6.1 String Words	8	8	

NOT IMPLEMENTED

Double Extension, Floating, Search, Search Extension, Block, Block Extension
Exception, Facility, Local, Local Extension, File, File Extension, Memory

Passes John Hayes & Gerry Jackson's ANSTESTS version 0.7

Some statistics for CamelForth 1802 v1.1

Constant words	12
Code words	91
Colon words	163
User words	9
Total words	275
Dictionary size	6,657 bytes

- Minimal ROM footprint < 4KB
  - Sufficient functionality to compile rest of words from FORTH source

Performance - Loop Counting to 64K

- FORTH code

0 BEGIN 1+ DUP 0= UNTIL DROP

1+	8 inst
DUP	9 instr
0=	6 instr
?BRANCH	11 instr
NEXT	6 * 4 instr.

- Total 58 instructions per loop
- 64K loops -> 19 seconds

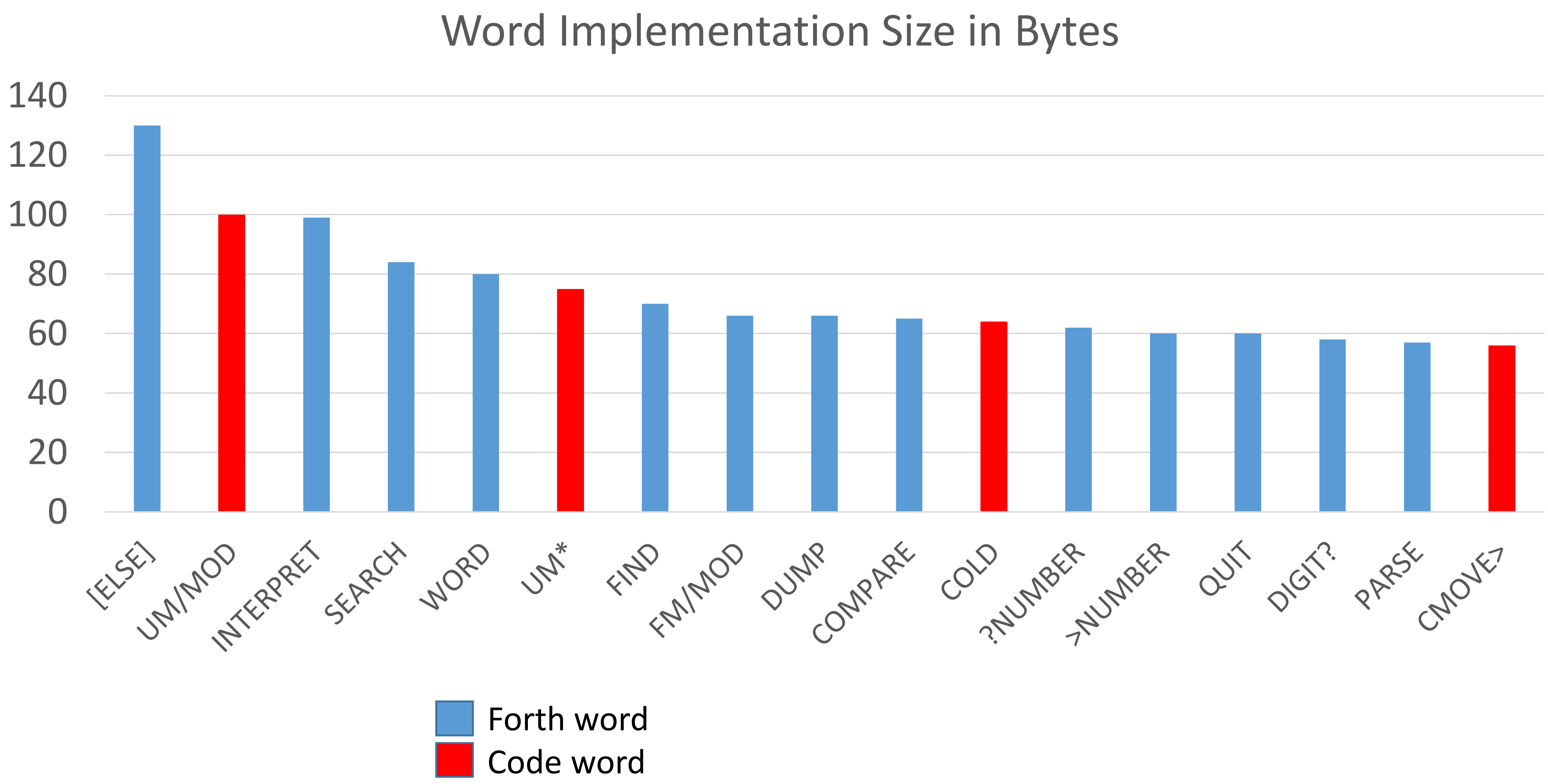
- Assembly code

```
1$:  INC Rn
     GLO Rn
     BNZ 1$
     GHI Rn
     BNZ 1$
```

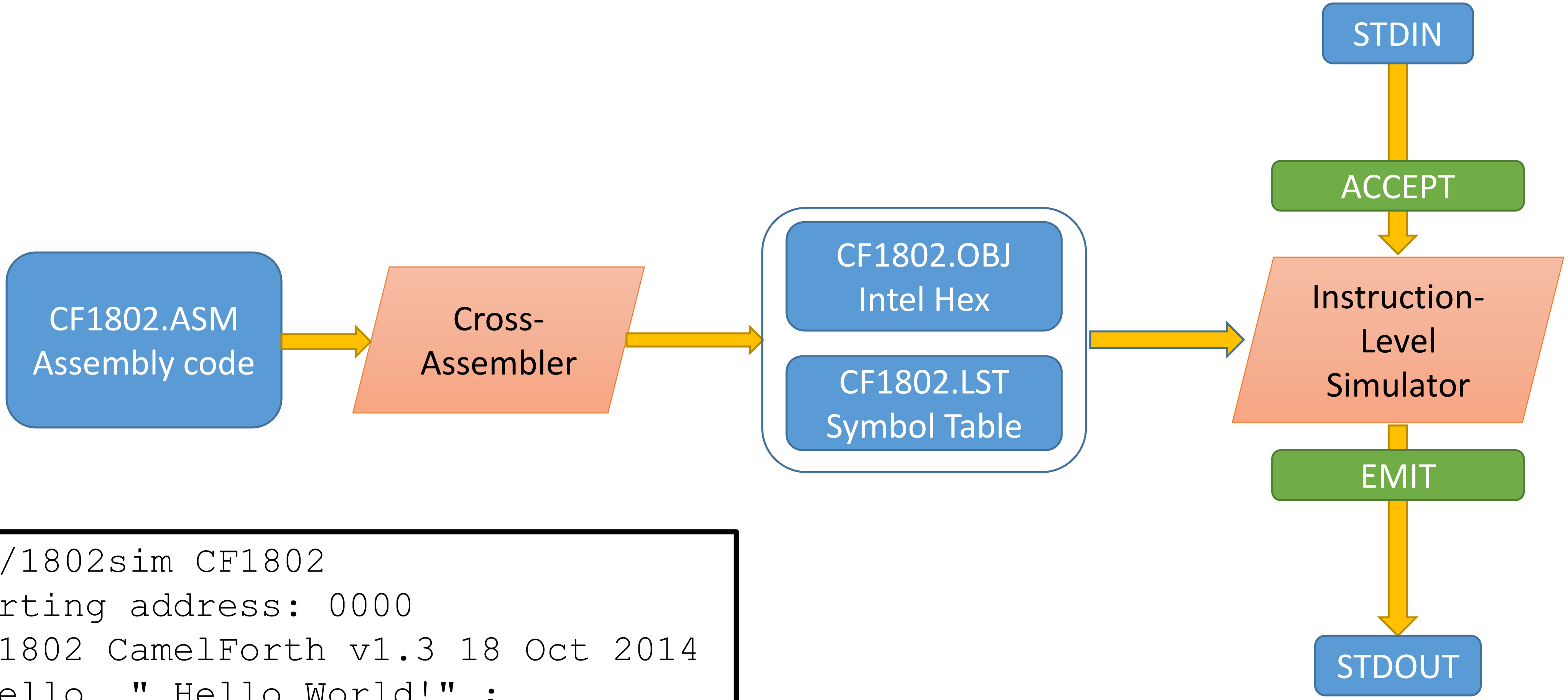
- Total: 3.008 instructions per loop
- 64K loops -> 0.98 seconds

FORTH : assembler ~ 19 : 1

Implementation Complexity



CamelForth 1802 Demo Setup




```
# ./1802sim CF1802
Starting address: 0000
RCA1802 CamelForth v1.3 18 Oct 2014
: hello ." Hello World!" ;
ok
```

Advantages of Simulation over Real Hardware

- Run-time error checking with no performance penalty
  - Stack underflows
  - Write to pre-defined dictionary area
  - Execution of undefined opcodes
- Symbolic execution tracing
  - FORTH word level with stack contents
  - Machine code level
- Cycle-accurate timing measurements
- ~600 times faster than RCA 1802 hardware