COS 126 Lecture 4: TOY machine

An imaginary machine, similar to

* ancient early computers
* today's microprocessers

Box with switches and lights, maybe TTY

Use to introduce

* machine language programming
  (how C program relates to machine)
* computer architecture
  (how machine works)

Inside the box

CPU (central processing unit)
256 16-bit words of memory
8 registers
PC (program counter)

Data Representations

Machine consists of two-state ("ON-OFF")
switches and lights

Use binary encoding to represent values

Ex:

.6375 = 0001100011100111

.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
.	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
.				2^{12}	2^{11}				2^7	2^6	2^5			2^2	2^1	2^0

.6375 = 4096+2048 +128+64+32 +4 +2 +1

Hexadecimal (base-16) notation provides shorthand
binary code four bits at a time

0000	0001	0010	0011	0100	0101	0110	0111
0	1	2	3	4	5	6	7
1000	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
8	9	A	B	C	D	E	F

Ex:

. 6375 = 0001100011100111

. 1 8 E 7

. $6375 = 1*16^3 + 8*16^2 + 14*16^1 + 7*16^0$

. = 4096 + 2048 + 224 + 7

EVERYTHING is encoded in binary (hex)
integers machine instructions, text, reals, ..

TOY machine memory

Contents of machine in hexadecimal ("dump")

```
PC: 0010

     R0:  R1:  R2:  R3:  R4:  R5:  R6:  R7:
     0000 0788 B700 0010 0401 0002 0003 00A0

00: 0000 0000 0000 0000 0000 0000 0000 0000
08: 0000 0000 0000 0000 0000 0000 0000 0000
10: 9222 9120 1121 A120 1121 A121 7211 0000
18: 0000 0001 0002 0003 0004 0005 0006 0007
20: 0008 0009 000A 000B 000C 000D 000E 000F
28: 0000 0000 0000 FE10 FACE CAFE ACED CEDE
  .
  .
  .
E8: 1234 5678 9ABC DEF0 0000 0000 F00D 0000
F0: 0000 0000 EEEE 1111 EEEE 1111 0000 0000
F8: B1B2 F1F5 0000 0000 0000 0000 0000 0000
```

Programmers still look at dumps, even in 1990s

Contents of memory
- record of what program has done
- determines (with PC) what machine will do

Instructions

Program: sequence of instructions

Instruction:

16-bit word (interpreted one way)

Data:

16-bit word (interpreted other ways)

Sixteen different instructions in TOY

0: halt
1: add
2: subtract
3: multiply
4: system call
5: jump
6: jump if greater
7: jump and count
8: jump and link
9: load
A: store
B: load address
C: xor
D: and
E: shift right
F: shift left

Basic Cycle

PC (program counter)
    is memory address of "current" instruction

Instructions change contents of registers and
    memory (and PC) in specific, well-defined
    ways

To run a program
* load the program and data
        (set switches, press LOAD for each word)
* set switches to address of first instruction
* press GO

GO button
    * loads PC from address switches
    * initiates FETCH-INCREMENT-EXECUTE cycle
    * machine runs until halt instruction hit

FETCH (get instruction from memory into CPU)
INCREMENT program counter (PC)
EXECUTE (may require data from or to memory)

Output:
    read contents of memory word in lights
    system call can write output
        to an output device (tty)

Instruction Formats

FORMAT 1: register-register

4 bits	4 bits	4 bits	4 bits
opcode	**dest**	**regA**	**regB**

Ex: 1234 means
add register R3 and R4
put the result in R2
R2 <- R3 + R4

Other instrs: sub, mult, xor, and

FORMAT 2: register-memory, register-immediate

4 bits	4 bits	8 bits
opcode	**dest**	**addr/const**

Ex: 9234 means "load memory loc 34 (hex) into R2"
R2 <- mem[34]
Ex: A234 means "store R2 into memory loc 34"
mem[34] <- R2
Ex: B234 means "load the value 0034 into R2"
R2 <- 0034

Other instrs: shifts, halt, system call, jumps

Sample TOY program 1

PROGRAM: sequence of instructions in memory

Set PC anywhere, press GO:

what happens is determined solely by

- contents of memory
- rules defining instructions

Ex: Suppose memory locations 10-1F contain

```
10: B001 B200 B101 1221 1110 1221 1110 1221
18: 1110 1221 1110 1221 1110 1221 0000 0000
```

Set PC to 10. Press GO. What happens?

Step-by-step trace:

```
10: B001    R0 <- 0001
11: B200    R2 <- 0000            0000
12: B101    R1 <- 0001      0001
13: 1221    R2 <- R2 + R1         0001
14: 1110    R1 <- R1 + R0   0002
15: 1221    R2 <- R2 + R1         0003
16: 1110    R1 <- R1 + R0   0003
17: 1221    R2 <- R2 + R1         0006
18: 1110    R1 <- R1 + R0   0004
19: 1221    R2 <- R2 + R1         000A
1A: 1110    R1 <- R1 + R0   0005
1B: 1221    R2 <- R2 + R1         000F
1C: 1110    R1 <- R1 + R0   0006
1D: 1221    R2 <- R2 + R1         0015
1E: 0000    halt
```

Computes 1 + 2 + 3 + 4 + 5 + 6 = 21

Sample TOY program 2

Suppose memory locations 10-17 contain

10: B006 B201 B101 1221 1110 6113 0000 0000

Set PC to 10. Press GO. What happens?

Step-by-step trace:

```
10: B106    R1 <- 0006        0006
11: B200    R2 <- 0000              0000
12: B001    R0 <- 0001
13: 1221    R2 <- R2 + R1           0006
14: 2110    R1 <- R1 - R0     0005
15: 6113    jump if (R1 > 0)
13: 1221    R2 <- R2 + R1           000B
14: 2110    R1 <- R1 - R0     0004
15: 6113    jump if (R1 > 0)
13: 1221    R2 <- R2 + R1           000F
14: 2110    R1 <- R1 - R0     0003
15: 6113    jump if (R1 > 0)
13: 1221    R2 <- R2 + R1           0012
14: 2110    R1 <- R1 - R0     0002
15: 6113    jump if (R1 > 0)
13: 1221    R2 <- R2 + R1           0014
14: 2110    R1 <- R1 - R0     0001
15: 6113    jump if (R1 > 0)
13: 1221    R2 <- R2 + R1           0015
14: 2110    R1 <- R1 - R0     0000
15: 6113    jump if (R1 > 0)
16: 0000    halt
```

Computes

$$N + (N-1) + \ldots + 3 + 2 + 1 = N(N+1)/2$$

for *any* value N loaded into R1

Horner's method

Problem:

    evaluate 2x^3 + 3x^2 + 9x + 7 at x = 10
    assume "data" stored in locations 30--34

```
        x    a    b    c    d
 30: 000A 0002 0003 0009 0007 0000 0000 0000
```

First try:

    compute x^3, mult. by a; compute x^2, ...
    (cumbersome, inefficient)

Efficient algorithm (Horner's method):

    rewrite ax^3+bx^2+cx+d as ((ax+b)x+c)x+d

```
 10: 9430  R4 <- M[30]    000A x
 11: 9531  R5 <- M[31]    0002 a
 12: 3554  R5 <- R5 * R4  0014 a*x
 13: 9632  R6 <- M[32]    0003 b
 14: 1556  R5 <- R5 + R6  0017 a*x+b
 15: 3554  R5 <- R5 * R4  00DC (a*x+b)*x
 16: 9633  R6 <- M[33]    0009 c
 17: 1556  R5 <- R5 + R6  00E5 (a*x+b)*x + c
 18: 3554  R5 <- R5 * R4  0956 ((a*x+b)*x+c)*x
 19: 9634  R6 <- M[34]    0007 d
 1A: 1556  R5 <- R5 + R6  095D ((a*x+b)*x+c*x)+d
 1B: 4502  write R5 to tty
```

Converts from decimal to hexadecimal

    2397 in base 10 equals 95D in hexadecimal

Does polynomial evaluation for arbitrary x

- many applications
- one raison d'etre for early machines

Basic Characteristics of TOY machine

"von Neumann" machine
- * instructions and data in same memory
- * can change program (control) w/o rewiring
    - * immediate applications
    - * profound implications

sufficient power to perform any computation
    (subject only to memory constraints)

similar to real machines
    helps focus attention on costs
    Ex: Horner's algorithm

simple high-level control

Abstract machine
    implement using any technology
    change implementation *without* having
        to change programs
    simulate on another machine

Real machines

Chip on a board

Input and Output
- to and from chip
- to and from machine
- to and from external devices

Much more memory
- * billions of bits, not 4096
- * 32 (or 64) bits per word, not 16

More kinds of instructions
More kinds of built-in data formats
- * large integers
- * floating point
- * character strings

More ways to address memory

SAME BASIC ABSTRACTIONS