COS 126 Lecture 6: TOY simulator

TOY machine now available!

Put a machine language program in a file

4-digit hex numbers to be loaded at 10

```
B102
1111
A120
0000
```

mail it to us

Course staff will

- load your program
- put 10 in the address switches
- press GO
- mail back a dump of memory

Easier way:

```
cp ~cs126/toy/toy.c .
cc toy.c -o TOY
TOY < myprog.toy
```

Example programs also available

```
TOY < ~cs126/toy/horner.toy
```

TOY machine simulator

```
#include <stdio.h>
int R[8], mem[256]; pc = 16;
main()
{
   int i, inst, op, addr, r0, r1, r2;
   for (i = 0; i < 256; i++) mem[i] = 0;
   for (i = 16; i < 256; i++)
      if (scanf("%X", &mem[i]) == EOF) break;
   do
      {
         inst = mem[pc++];
         op = (inst >> 12) & 0XF;
         addr = inst & 0XFF;
         r0 = (inst >> 8) & 0X7;
         r1 = (inst >> 4) & 0X7; r2 = inst & 0X7;
         if (inst & 0X0800) addr = R[r1]+R[r2];
         switch (op)
         {
            case  0: break;
            case  1: R[r0] = R[r1] + R[r2];  break;
            case  2: R[r0] = R[r1] - R[r2];  break;
            case  3: R[r0] = R[r1] * R[r2];  break;
            case  4: printf("%X\n", addr);   break;
            case  5: pc = addr;              break;
            case  6: if (R[r0]>0) pc = addr; break;
            case  7: if (--R[r0]) pc = addr; break;
            case  8: R[r0] = pc; pc = addr;  break;
            case  9: R[r0] = mem[addr];      break;
            case 10: mem[addr] = R[r0];      break;
            case 11: R[r0] = addr;           break;
            case 12: R[r0] = R[r1] ^ R[r2];  break;
            case 13: R[r0] = R[r1] & R[r2];  break;
            case 14: R[r0] = R[r0] >> addr;  break;
            case 15: R[r0] = R[r0] << addr;  break;
         }
      }
   while (op != 0);
}
```

TOY dump

Use C arrays for memory, registers

```
int R[8], mem[256]; pc = 16;
dump()
   {
      int i, j;
      printf("pc: %04X\n", pc);
      printf("regs: ");
      for (i = 0; i < 8; i++)
         printf("%04X ", R[i]);
      printf("\n");
      for (i = 0; i < 32; i++)
         {
            printf("\n%04X: ", 8*i);
            for (j = 0; j < 8; j++)
               printf("%04X ", mem[8*i+j]);
         }
      printf("\n");
   }
```

Dump is in hex, 8 words/line, 4 digits/word

```
0000: 0000 0004 0000 0000 0000 0000 0000 0000
0008: 910A 110A 0002 0000 0000 0000 0000 0000
```

Easy to add other functions to simulator

- trace
- single-step
- breakpoint debugging
- ...

Simulator more powerful than TOY itself!

[TOY probably cheaper]

Want to design a new machine?

use the simulator!

[much cheaper than building a machine]

Bootstrapping

Correspondence between
basic C constructs
and
basic TOY mechanisms

C	TOY
array	indexed addressing
arithmetic expressions	add, mul
logical expressions	xor, and, shifts
switch, if, for, while	conditional jumps

Translate TOY program to C program?
- easy

Translate C program to TOY program?
- straightforward (if tedious)

Translate SIMULATOR to TOY program?
- why not??

BOOTSTRAPPING
- build "first" machine
- implement simulator
- modify simulator to try new designs
- (still going on!)