CMSC 216
Introduction to Computer Systems
Lecture 12

Administrivia

- Read Reek, Chapter 12
- Start reading Bryant and O'Hallaron Chapter 3 (IA-32 instruction set architecture) and 4.1 (Y86 subset)

Project 3b

- Let’s talk about the project

Chapters 3 and 4.1, Bryant and O'Hallaron

Assembly Language (cont.)

Y86 integer instructions

Instruction	Effect	Description
`addl S,D`	Reg[D] ← Reg[D] + Reg[S]	Addition
`subl S,D`	Reg[D] ← Reg[D] - Reg[S]	Subtract
`andl S,D`	Reg[D] ← Reg[D] & Reg[S]	Bitwise AND
`xorl S,D`	Reg[D] ← Reg[D] ^ Reg[S]	Bitwise XOR
`multl S,D`	Reg[D] ← Reg[D] * Reg[S]	Multiplication*
`divl S,D`	Reg[D] ← Reg[D] / Reg[S]	Integer division*
`modl S,D`	Reg[D] ← Reg[D] % Reg[S]	Remainder*

- All these instructions operate on two integers, and set the condition code flags appropriately
- Notice that only registers are used
- Instructions marked with an asterisk (*) are extensions to Y86 we've added to the ones in the book

Integer instruction example

- Assembler output:

```
0x000: 308003000000 | irmovl $3,%eax   # a = 3
0x006: 308305000000 | irmovl $5,%ebx   # b = 5
0x00c: 6003         | addl   %eax,%ebx # b = a + b
0x00e: f308         | wrint  %eax
0x010: 308620000000 | irmovl $32,%esi  # 32 == ' '
0x016: f168         | wrch   %esi
0x018: f338         | wrint  %ebx
0x01a: 30860a000000 | irmovl $10,%esi  # 10 == '\n'
0x020: f168         | wrch   %esi
0x022: 10           | halt
```

- Simulator run:

```
3 8
...
```

- Notice these instructions are destructive; they overwrite the second operand
  - Need to make copies if you need old values

Condition codes

- Performing integer operations causes various flags to be set, describing the attributes of the result of the operation
- These are used by other, subsequent instructions to perform conditional branching
- The three we are concerned with are:
  - OF: overflow flag; did the operation overflow?
  - SF: sign flag; is the result negative?
  - ZF: zero flag; is the result zero?

Branch instructions

- These are used to perform the effect of if statements, loops, and switches
- When encountered, if a certain condition is true, control flow will then go to the address specified, rather than advancing to the next instruction
  - The address of the next instruction to be executed is held in the program counter; in many architectures, this is held in an accessible register (not so with Y86).
- Labels
  - We use labels (name followed by colon) to represent target addresses
  - When the assembler encounters a label (e.g., `Loop:`), the address of the labeled instruction is used by the assembler to replace all references to that label in the program
  - Labels do not need to be declared before use

Y86 branch instructions

Instruction	Branch if...	Description
`jmp Label`	1	Unconditional jump
`jle Label`	(SF ^ OF) | ZF	Jump if less than or equal to zero
`jl Label`	SF ^ OF	Jump if less than zero
`je Label`	ZF	Jump if equal to zero
`jne Label`	~ZF	Jump if not equal to zero
`jge Label`	~(SF ^ OF)	Jump if greater than or equal to zero
`jg Label`	~(SF ^ OF) & ~ZF	Jump if greater than zero

- Each instruction relies on the condition codes set by the most recent integer instruction

Translating branches

- Since assembly has no else statements, we need to use branching + code reorganization to get if/else
- Consider the following C code:

```
if (i == j)
  printf("=");
else
  printf("X");
printf("\n");
```

```
if (i == j)
   goto Equal;
   printf("X");
   goto EndIf;
Equal:
  printf("=");
EndIf:
  printf("\n");
```

Translating branches, cont.

- We can then take the labeled C code and translate it in a fairly straightforward fashion:

```
        subl %eax,%ebx  # i:%eax,j:%ebx
        je Equal
        irmovl $88,%ecx # else block
        wrch %ecx
        jmp EndIf
Equal:  irmovl $61,%ecx # true block
        wrch %ecx
EndIf:  irmovl $10,%ecx # after if/else
        wrch %ecx
```

Branch Example

- Assembler output:

```
0x000: 308003000000 |        irmovl $3,%eax    # a = 3
0x006: 308304000000 |        irmovl $4,%ebx    # b = 4
0x00c: 2006         |        rrmovl %eax,%esi  # s = a
0x00e: 6136         |        subl   %ebx,%esi  # s = s - b
0x010: 751c000000   |        jge    Else       # if s >= 0 jump
                    |
0x015: f308         |        wrint  %eax       # printf("%d", a)
0x017: 701e000000   |        jmp    Endif      # jump
                    |
0x01c: f338         | Else:  wrint  %ebx       # printf("%d", b)
                    |
0x01e: 30860a000000 | Endif: irmovl $10,%esi
0x024: f168         |        wrch   %esi       # printf("\n")
0x026: 10           |        halt
```

- Simulator output:

```
3
Stopped in 10 steps at PC = 0x27.  Exception 'HLT', CC Z=0 S=1 O=0
...
```