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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Effect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>subl S,D</code></td>
<td><code>Reg[D] ← Reg[D] - Reg[S]</code></td>
<td>Subtract</td>
</tr>
<tr>
<td><code>divl S,D</code></td>
<td><code>Reg[D] ← Reg[D] / Reg[S]</code></td>
<td>Integer division*</td>
</tr>
<tr>
<td><code>modl S,D</code></td>
<td><code>Reg[D] ← Reg[D] % Reg[S]</code></td>
<td>Remainder*</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Branch if...</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp Label</td>
<td>1</td>
<td>Unconditional jump</td>
</tr>
<tr>
<td>jle Label</td>
<td>(SF ^ OF)</td>
<td>Jump if less than or equal to zero</td>
</tr>
<tr>
<td>jl Label</td>
<td>SF ^ OF</td>
<td>Jump if less than zero</td>
</tr>
<tr>
<td>je Label</td>
<td>ZF</td>
<td>Jump if equal to zero</td>
</tr>
<tr>
<td>jne Label</td>
<td>~ZF</td>
<td>Jump if not equal to zero</td>
</tr>
<tr>
<td>jge Label</td>
<td>~(SF ^ OF)</td>
<td>Jump if greater than or equal to zero</td>
</tr>
<tr>
<td>jg Label</td>
<td>~(SF ^ OF) &amp; ~ZF</td>
<td>Jump if greater than zero</td>
</tr>
</tbody>
</table>

• 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:

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

```c
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:

```c
subl %eax,%ebx    # i:%eax,j:%ebx
ej 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:

<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000:</td>
<td>308003000000</td>
<td>irmovl $3,%eax # a = 3</td>
</tr>
<tr>
<td>0x006:</td>
<td>308304000000</td>
<td>irmovl $4,%ebx # b = 4</td>
</tr>
<tr>
<td>0x00c:</td>
<td>2006</td>
<td>rrmovl %eax,%esi # s = a</td>
</tr>
<tr>
<td>0x00e:</td>
<td>6136</td>
<td>subl %ebx,%esi # s = s - b</td>
</tr>
<tr>
<td>0x010:</td>
<td>751c000000</td>
<td>jge Else # if s &gt;= 0 jump</td>
</tr>
<tr>
<td>0x015:</td>
<td>f308</td>
<td>wrint %eax # printf(&quot;%d&quot;, a)</td>
</tr>
<tr>
<td>0x017:</td>
<td>701e000000</td>
<td>jmp Endif # jump</td>
</tr>
<tr>
<td>0x01c:</td>
<td>f338</td>
<td>Else: wrint %ebx # printf(&quot;%d&quot;, b)</td>
</tr>
<tr>
<td>0x01e:</td>
<td>30860a000000</td>
<td>Endif: irmovl $10,%esi</td>
</tr>
<tr>
<td>0x024:</td>
<td>f168</td>
<td>wrch %esi # printf(&quot;\n&quot;)</td>
</tr>
<tr>
<td>0x026:</td>
<td>10</td>
<td>halt</td>
</tr>
</tbody>
</table>

• Simulator output:

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