x86-64 Programming II
CSE 351 Winter 2018

Instructor:
Mark Wyse

Teaching Assistants:
Kevin Bi
Parker DeWilde
Emily Furst
Sarah House
Waylon Huang
Vinny Palaniappan

http://xkcd.com/409/

Homework 2 (Ints and Floats) due Today!
- By 11:59 pm, no late submissions!
- x86-64 part due 1/29

GDB Tutorial Session
- Tomorrow, Thursday 1/25, 5 – 6 pm
- Room: SIG 134

Conditional codes
Conditional and unconditional branches
Loops
Switches

Control Flow

Conditionals and Control Flow

Conditional branch/jump
- Jump to somewhere else if some condition is true, otherwise execute next instruction

Unconditional branch/jump
- Always jump when you get to this instruction

Together, they can implement most control flow constructs in high-level languages:
- if (condition) then (...) else (...) while (condition) (...) do (...) while (condition)
- for (initialization; condition; iterative) (...) switch (...)
Condition Codes (Implicit Setting)

- *Implicitly set by arithmetic* operations
  - (think of it as side effects)
  - Example: `addq src, dst ↔ r = d+s`
  - `CF=1` if carry out from MSB (unsigned overflow)
  - `ZF=1` if `r==0`
  - `SF=1` if `r<0` (assuming signed, actually just if MSB is 1)
  - `OF=1` if two’s complement (signed) overflow
    
    $$(a\geq 0 \&\& d>0 \&\& r<0) || (a<0 \&\& d<0 \&\& r\geq 0)$$

- *Not set by* `lea` *instruction* (beware!)

Condition Codes (Explicit Setting: Compare)

- *Explicitly set by* Compare *instruction*
  - `cmpq src1, src2`
  - `cmpq a, b` *sets flags* based on `b-a`, but doesn’t store
  - `CF=1` if carry out from MSB (used for unsigned comparison)
  - `ZF=1` if `a==b`
  - `SF=1` if `(b-a)<0` (signed)
  - `OF=1` if two’s complement (signed) overflow
    
    $$(a>0 \&\& b<0 \&\& (b-a)>0) \mid (a<0 \&\& b>0 \&\& (b-a)<0)$$

Condition Codes (Explicit Setting: Test)

- *Explicitly set by* Test *instruction*
  - `testq src2, src1`
  - `testq a, b` *sets flags* based on `a&b`, but doesn’t store
  - Useful to have one of the operands be a *mask*
  - `ZF=1` if `a&b==0`
  - `SF=1` if `a&b<0` (signed)
  - `CF` and `OF` set to 0

Using Condition Codes: Jump

- *j* *Instructions*
  - Jumps to *target* (an address) based on condition codes

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jmp target</code></td>
<td>1</td>
<td>Unconditional</td>
</tr>
<tr>
<td><code>je target</code></td>
<td>ZF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td><code>jne target</code></td>
<td>~ZF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td><code>jz target</code></td>
<td>SF</td>
<td>Zero</td>
</tr>
<tr>
<td><code>jnz target</code></td>
<td>~SF</td>
<td>Nonzero</td>
</tr>
<tr>
<td><code>jg target</code></td>
<td>~ (SF &amp; OF)</td>
<td>Greater Signed</td>
</tr>
<tr>
<td><code>jge target</code></td>
<td>~ (SF &amp; OF)</td>
<td>Greater or Equal Signed</td>
</tr>
<tr>
<td><code>jl target</code></td>
<td>(SF &amp; OF)</td>
<td>Less Signed</td>
</tr>
<tr>
<td><code>jle target</code></td>
<td>~ (SF &amp; OF)</td>
<td>Less or Equal Signed</td>
</tr>
<tr>
<td><code>ja target</code></td>
<td>~ (CF &amp; SF)</td>
<td>Above Unsigned</td>
</tr>
<tr>
<td><code>jbe target</code></td>
<td>CF</td>
<td>Below Unsigned</td>
</tr>
</tbody>
</table>

Using Condition Codes: Set

- *set* *Instructions*
  - Set low-order byte of *dst* to 0 or 1 based on condition codes
  - Does not alter remaining 7 bytes

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>seta dst</code></td>
<td>ZF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td><code>setb dst</code></td>
<td>SF</td>
<td>Zero</td>
</tr>
<tr>
<td><code>setne dst</code></td>
<td>SF</td>
<td>Not Zero</td>
</tr>
<tr>
<td><code>setc dst</code></td>
<td>~SF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td><code>setz dst</code></td>
<td>~ZF</td>
<td>Not Signed</td>
</tr>
<tr>
<td><code>setns dst</code></td>
<td>~ZF</td>
<td>Not Signed</td>
</tr>
<tr>
<td><code>setg dst</code></td>
<td>~ (SF &amp; OF)</td>
<td>Greater Signed</td>
</tr>
<tr>
<td><code>setle dst</code></td>
<td>~ (SF &amp; OF)</td>
<td>Greater or Equal Signed</td>
</tr>
<tr>
<td><code>setl dst</code></td>
<td>(SF &amp; OF)</td>
<td>Less Signed</td>
</tr>
<tr>
<td><code>setle dst</code></td>
<td>~ (SF &amp; OF)</td>
<td>Less or Equal Signed</td>
</tr>
<tr>
<td><code>seta dst</code></td>
<td>~ (CF &amp; SF)</td>
<td>Above Unsigned</td>
</tr>
<tr>
<td><code>setb dst</code></td>
<td>CF</td>
<td>Below Unsigned</td>
</tr>
</tbody>
</table>
Reminder: x86-64 Integer Registers

- Accessing the low-order byte:

<table>
<thead>
<tr>
<th>OP</th>
<th>RAX</th>
<th>RBX</th>
<th>RCX</th>
<th>RDX</th>
<th>RSI</th>
<th>RDI</th>
<th>RSP</th>
<th>RBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>movs</td>
<td>%r8</td>
<td>%r9</td>
<td>%r10</td>
<td>%r11</td>
<td>%r12</td>
<td>%r13</td>
<td>%r14</td>
<td>%r15</td>
</tr>
</tbody>
</table>

Reading Condition Codes

- set* Instructions
  - Set a low-order byte to 0 or 1 based on condition codes
  - Operand is byte register (e.g. %al, %dil) or a byte in memory
  - Do not alter remaining bytes in register
    - Typically use movzbl (zero-extended mov) to finish job

Example:

```c
int gt(long x, long y) {
    return x > y;
}
```

Aside: movz and movs

- movz src, regDest: Move with zero extension
- movs src, regDest: Move with sign extension

- Copy from a smaller source value to a larger destination
- Source can be memory or register; Destination must be a register
- Fill remaining bits of dest with zero (movz) or sign bit (movs)

Example:

```c
movzbq %al, %rbx
```

Using Condition Codes: Jump

- j* Instructions
  - Jumps to target (an address) based on condition codes

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>je</td>
<td>SF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td>js</td>
<td>SF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td>je</td>
<td>SF</td>
<td>Negative</td>
</tr>
<tr>
<td>js</td>
<td>SF</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>jg</td>
<td>-SF</td>
<td>Greater (Signed)</td>
</tr>
<tr>
<td>jge</td>
<td>-SF</td>
<td>Greater or Equal (Signed)</td>
</tr>
<tr>
<td>jl</td>
<td>(SF)</td>
<td>Less (Signed)</td>
</tr>
<tr>
<td>jle</td>
<td>(SF)</td>
<td>Less or Equal (Signed)</td>
</tr>
<tr>
<td>jae</td>
<td>CF</td>
<td>Above (unsigned &quot;&lt;&quot;)</td>
</tr>
<tr>
<td>jbe</td>
<td>CF</td>
<td>Below (unsigned &quot;&lt;&quot;)</td>
</tr>
</tbody>
</table>
Choosing instructions for conditionals

- All arithmetic instructions set condition flags based on result of operation (op)
  - Conditionals are comparisons against 0

```
addb $5, (%p)  ; p = p + 5
je  "Equal"    ; d < op1 == op2
jne "Not equal"   ; d < op1 != op2
jl "Less than"  ; d < op1 < 0
jle "Less than or equal"  ; d < op1 <= 0
jg "Greater than"  ; d < op1 > 0
jge "Greater than or equal"  ; d < op1 >= 0
jnb "Less than or equal"  ; d < op1 & 0 < 0
jnb "Less than or equal"  ; d < op1 & 0 <= 0
jb "Greater than or equal"  ; d < op1 & 0 > 0
```

Choosing instructions for conditionals

- Reminder: `cmpq` is like `subq`, `testq` is like and
  - Result is not stored anywhere

```
jne a, (%rdi)   ; a != 0
jl (%rdi) < 0U   ; x < 0
jge (%rdi) > 0U   ; x >= 0
jg (%rdi) > 0U   ; x > 0
jnb (%rdi) <= 0U  ; x <= 0
jnb (%rdi) < 0U   ; x < 0
jb (%rdi) > 0U    ; x > 0
```

Question

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
<th>2nd argument [y]</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>1st argument [x]</td>
<td></td>
</tr>
<tr>
<td>%rax</td>
<td>2nd argument [y]</td>
<td></td>
</tr>
<tr>
<td>%trax</td>
<td>return value</td>
<td></td>
</tr>
</tbody>
</table>

A. `cmpq %rax, %rdi`  ; `.L4`
B. `cmpq %rax, %rdi`  ; `.L4`
C. `testq %rax, %rdi`  ; `.L4`
D. `testq %rax, %rdi`  ; `.L4`
E. `We're lost...`

Summary

- Control flow in x86 determined by status of Condition Codes
  - Showed Carry, Zero, Sign, and Overflow, though others exist
  - Set flags with arithmetic instructions (implicit) or Compare and Test (explicit)
  - Set instructions read out flag values
  - Jump instructions use flag values to determine next instruction to execute

- [https://godbolt.org/g/KntpyvG](https://godbolt.org/g/KntpyvG)