x86-64 Programming III
CSE 351 Autumn 2020

Instructor:
Justin Hsia

Teaching Assistants:
Aman Mohammed
Ami Oka
Callum Walker
Cosmo Wang
Hang Do
Jim Limprasert
Joy Dang
Julia Wang
Kaelin Laundry
Kyrie Dowling
Mariam Mayanja
Shawn Stanley
Yan Zhe Ong

http://xkcd.com/1652/
Administrivia

- Lab 2 due next Friday (10/30)

- Section tomorrow on Assembly
  - Use the midterm reference sheet!
  - Optional GDB Tutorial slides and Lab 2 phase 1 walkthrough

- Midterm (take home, 10/31–11/2)
  - Find groups of 5 for the group stage
  - Make notes and use the midterm reference sheet
  - Form study groups and look at past exams!
 Aside: movz and movs

- movz\_\_ src, regDest
  - # Move with zero extension
- movs\_\_ src, regDest
  - # Move with sign extension

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

\texttt{movz} SD / \texttt{movs} SD:
- $S$ – size of source ($b = 1$ byte, $w = 2$)
- $D$ – size of dest ($w = 2$ bytes, $l = 4$, $q = 8$)

Example:
- \texttt{movzbq %al, %rbx}

\texttt{0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0xFF \rightarrow %rax}
\texttt{0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF \rightarrow %rbx}
Aside: movz and movs

\[\text{movz}_{\text{SD}} \quad \text{src}, \text{regDest} \quad # \text{Move with zero extension} \]

\[\text{movs}_{\text{SD}} \quad \text{src}, \text{regDest} \quad # \text{Move with sign extension} \]

- 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 (\text{movz}) or sign bit (\text{movs})

\[\text{movz}_{\text{SD}} / \text{movs}_{\text{SD}}: \]
- \(S\) – size of source (\(b = 1\) byte, \(w = 2\))
- \(D\) – size of dest (\(w = 2\) bytes, \(l = 4\), \(q = 8\))

Example:

\[\text{movsbl} \quad (\%rax), \%ebx\]

Copy 1 byte from memory into 8-byte register & sign extend it

Note: In x86-64, any instruction that generates a 32-bit (long word) value for a register also sets the high-order portion of the register to 0. Good example on p. 184 in the textbook.
GDB Demo

- The `movz` and `movs` examples on a real machine!
  - `movzbq %al, %rbx`
  - `movsbl (%rax), %ebx`

- You will need to use GDB to get through Lab 2
  - Useful debugger in this class and beyond!

- Pay attention to:
  - Setting breakpoints (`break`)
  - Stepping through code (`step/next` and `steipi/nexti`)
  - Printing out expressions (`print` – works with regs & vars)
  - Examining memory (`x`)
Reading Review

- Terminology:
  - Label, jump target
  - Program counter
  - Jump table, indirect jump

- Questions from the Reading?
Choosing instructions for conditionals

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

- Come in instruction *pairs*

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Condition Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>addq 5, (p)</td>
<td>“Equal”</td>
<td>(d \ (\text{op}) \ s = 0)</td>
</tr>
<tr>
<td>je:</td>
<td>(*p+5 = 0)</td>
<td></td>
</tr>
<tr>
<td>jne:</td>
<td>(*p+5 \neq 0)</td>
<td></td>
</tr>
<tr>
<td>jg:</td>
<td>(*p+5 &gt; 0)</td>
<td></td>
</tr>
<tr>
<td>jl:</td>
<td>(*p+5 &lt; 0)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Condition Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>orq a, b</td>
<td>“Less”</td>
<td>(d \ (\text{op}) \ s &lt; 0)</td>
</tr>
<tr>
<td>je:</td>
<td>(b</td>
<td>a = 0)</td>
</tr>
<tr>
<td>jne:</td>
<td>(b</td>
<td>a \neq 0)</td>
</tr>
<tr>
<td>jg:</td>
<td>(b</td>
<td>a &gt; 0)</td>
</tr>
<tr>
<td>jl:</td>
<td>(b</td>
<td>a &lt; 0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Condition Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>orq a, b</td>
<td>“Above” (unsigned &gt;)</td>
<td>(d \ (\text{op}) \ s &gt; 0)</td>
</tr>
<tr>
<td>je:</td>
<td>(b</td>
<td>a = 0)</td>
</tr>
<tr>
<td>jne:</td>
<td>(b</td>
<td>a \neq 0)</td>
</tr>
<tr>
<td>jg:</td>
<td>(b</td>
<td>a &gt; 0)</td>
</tr>
<tr>
<td>jl:</td>
<td>(b</td>
<td>a &lt; 0)</td>
</tr>
</tbody>
</table>
Choosing instructions for conditionals

- Reminder: `cmp` is like `sub`, `test` is like `and`
  - Result is not stored anywhere

<table>
<thead>
<tr>
<th>Instruction</th>
<th>cmp a,b</th>
<th>test a,b</th>
</tr>
</thead>
<tbody>
<tr>
<td>je (&quot;Equal&quot;)</td>
<td>b == a</td>
<td>b&amp;a == 0</td>
</tr>
<tr>
<td>jne (&quot;Not equal&quot;)</td>
<td>b != a</td>
<td>b&amp;a != 0</td>
</tr>
<tr>
<td>js (&quot;Sign&quot; (negative))</td>
<td>b-a &lt; 0</td>
<td>b&amp;a &lt; 0</td>
</tr>
<tr>
<td>jns (non-negative)</td>
<td>b-a &gt;= 0</td>
<td>b&amp;a &gt;= 0</td>
</tr>
<tr>
<td>jg (&quot;Greater&quot;)</td>
<td>b &gt; a</td>
<td>b&amp;a &gt; 0</td>
</tr>
<tr>
<td>jge (&quot;Greater or equal&quot;)</td>
<td>b &gt;= a</td>
<td>b&amp;a &gt;= 0</td>
</tr>
<tr>
<td>jl (&quot;Less&quot;)</td>
<td>b &lt; a</td>
<td>b&amp;a &lt; 0</td>
</tr>
<tr>
<td>jle (&quot;Less or equal&quot;)</td>
<td>b &lt;= a</td>
<td>b&amp;a &lt;= 0</td>
</tr>
<tr>
<td>ja (&quot;Above&quot; (unsigned &gt;))</td>
<td>b &gt;U a</td>
<td>b&amp;a &gt; 0U</td>
</tr>
<tr>
<td>jb (&quot;Below&quot; (unsigned &lt;))</td>
<td>b &lt;U a</td>
<td>b&amp;a &lt; 0U</td>
</tr>
</tbody>
</table>

- `cmpq 5, (p)`
  - je: *p == 5
  - jne: *p != 5
  - jg: *p > 5
  - jl: *p < 5

- `testq a, a`
  - je: a == 0
  - jne: a != 0
  - jg: a > 0
  - jl: a < 0

- `testb a, 0x1`
  - je: a_LSB == 0
  - jne: a_LSB == 1
Choosing instructions for conditionals

### Register Use(s)

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>argument x</td>
</tr>
<tr>
<td>%rsi</td>
<td>argument y</td>
</tr>
<tr>
<td>%rax</td>
<td>return value</td>
</tr>
</tbody>
</table>

### Code Example

```c
if (x < 3) {
    return 1;
} else {
    return 2;
}
```

### Instructions

- `cmpq $3, %rdi`
- `jge T2`
- `movq $1, %rax`
- `ret`

- `cmpq $3, %rdi`
- `ja T1:
- `cmp a,b test a,b`
Practice Question 1

long absdiff(long x, long y) {
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}

Register Use(s)
%rdi 1st argument (x)
%rsi 2nd argument (y)
%rax return value

A. cmpq %rsi, %rdi x-y
   jle .L4
B. cmpq %rsi, %rdi x-y
   jg .L4
C. testq %rsi, %rdi x&y
   jle .L4
D. testq %rsi, %rdi x&y
   jg .L4
E. We’re lost…

absdiff:

______________________________

______________________________  # x > y:
movq %rdi, %rax
subq %rsi, %rax
ret

.L4:  # x <= y:
    movq %rsi, %rax  x-y <= 0
    subq %rdi, %rax
    ret

less than or equal to (le)
Choosing instructions for conditionals

<table>
<thead>
<tr>
<th>je</th>
<th>“Equal”</th>
<th>cmp a,b</th>
<th>test a,b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b x == y</td>
<td>2</td>
<td>b&amp;a == 0</td>
</tr>
<tr>
<td>jne</td>
<td>“Not equal”</td>
<td>b != a</td>
<td>b&amp;a != 0</td>
</tr>
<tr>
<td>js</td>
<td>“Sign” (negative)</td>
<td>b-a &lt; 0</td>
<td>b&amp;a &lt; 0</td>
</tr>
<tr>
<td>jns</td>
<td>(non-negative)</td>
<td>b-a &gt;=0</td>
<td>b&amp;a &gt;= 0</td>
</tr>
<tr>
<td>jg</td>
<td>“Greater”</td>
<td>b &gt; a</td>
<td>b&amp;a &gt; 0</td>
</tr>
<tr>
<td>jge</td>
<td>“Greater or equal”</td>
<td>b &gt;= a</td>
<td>b&amp;a &gt;= 0</td>
</tr>
<tr>
<td>jl</td>
<td>“Less”</td>
<td>b &lt; 3</td>
<td>b&amp;a &lt; 0</td>
</tr>
<tr>
<td>jle</td>
<td>“Less or equal”</td>
<td>b &lt;= a</td>
<td>b&amp;a &lt;= 0</td>
</tr>
<tr>
<td>ja</td>
<td>“Above” (unsigned &gt;)</td>
<td>b &gt; a</td>
<td>b&amp;a &gt; 0U</td>
</tr>
<tr>
<td>jb</td>
<td>“Below” (unsigned &lt;)</td>
<td>b &lt; a</td>
<td>b&amp;a &lt; 0U</td>
</tr>
</tbody>
</table>

https://godbolt.org/z/Tfrv33

```c
if (x < 3 && x == y) {
    return 1;
} else {
    return 2;
}
```
Labels

- A jump changes the program counter (%rip)
  - %rip tells the CPU the address of the next instruction to execute

- Labels give us a way to refer to a specific instruction in our assembly/machine code
  - Associated with the next instruction found in the assembly code (ignores whitespace)
  - Each use of the label will eventually be replaced with something that indicates the final address of the instruction that it is associated with
x86 Control Flow

- Condition codes
- Conditional and unconditional branches
- Loops
- Switches
Expressing with Goto Code

C allows \texttt{goto} as means of transferring control (jump)

- Closer to assembly programming style
- Generally considered bad coding style

```c
long absdiff(long x, long y) {
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```c
long absdiff_j(long x, long y) {
    long result;
    int ntest = (x <= y);
    if (ntest) goto Else;
    result = x-y;
    goto Done;

    Else:
    result = y-x;

    Done:
    return result;
}
```
Compiling Loops

- Other loops compiled similarly
  - Will show variations and complications in coming slides, but may skip a few examples in the interest of time

- Most important to consider:
  - When should conditionals be evaluated? (*while* vs. *do-while*)
  - How much jumping is involved?
Compiling Loops

**While Loop:**

C:
```c
while ( sum != 0 ) {
    <loop body>
}
```

x86-64:
```assembly
loopTop:    testq  %rax, %rax
je loopDone
<loop body code>
jmp loopTop
```

**Do-while Loop:**

C:
```c
do {
    <loop body>
} while ( sum != 0 )
```

x86-64:
```assembly
loopTop:    testq  %rax, %rax
jne loopTop
<loop body code>
```

**While Loop (ver. 2):**

C:
```c
while ( sum != 0 ) {
    <loop body>
}
```

x86-64:
```assembly
loopTop:    testq  %rax, %rax
je loopDone
<loop body code>
jne loopTop
```

Note: all jump instructions update the program counter (```rip`)
For-Loop → While-Loop

For-Loop:

```c
for (Init; Test; Update) {
    Body
}
```

While-Loop Version:

```c
Init;
while (Test) {
    Body
    Update;
}
```

Caveat: C and Java have

- `break` and `continue`

- Conversion works fine for `break`
  - Jump to same label as loop exit condition

- But not `continue`: would skip doing `Update`, which it should do with for-loops
  - Introduce new label at `Update`
Practice Question 2

- The following is assembly code for a for-loop; identify the corresponding parts (Init, Test, Update)

- $i \rightarrow %eax, x \rightarrow %rdi, y \rightarrow %esi$

```
.L2:
    movl $0, %eax
    cmpl %esi, %eax
    jge .L4
    movslq %eax, %rdx
    leaq (%rdi,%rdx,4), %rcx
    movl (%rcx), %edx
    addl $1, %edx
    movl %edx, (%rcx)
    addl $1, %eax
    jmp .L2
.L4:
```