CMPT 295
Machine-Level Programming
Lecture 10 – Operation leaq and memory addressing modes
Last Lecture

- As x86-64 assembly programmers, we now get to see more of the processor (CPU) state: PC, registers, condition codes

- x86-64 assembly language – Data
  - Integer registers (16) of 1, 2, 4 or 8 bytes + memory address of 8 bytes
  - Floating point registers of 4 or 8 bytes
  - No aggregate types such as arrays or structures

- x86-64 assembly language – Operations
  - \texttt{mov\*} instruction family
  - From register to register
  - From memory to register
  - From register to memory
Today’s Menu

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, move operation
  - Memory addressing modes
- Operation leaq and Arithmetic & logical operations
- Conditional Statement
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations
leaq - Load effective address instruction

- Often used for address computations and general arithmetic computations
- Syntax: leaq Source, Destination
- Example:
  1. Computing addresses
     
     leaq (%rax, %rcx), %rdx

     Once executed, rdx will contain 0x18

     if %rax <- 0x0000000000000008 and %rcx <- 16

     leaq (%rdi, %rdi, 2), %rax

     Once executed, rax will contain 3a

     Operand Destination is a register

     Operand Source is a memory addressing mode expression

C code:

return a*3;
Memory addressing modes

Question: How do we access memory in x86-64 assembly?

Answer: Various “memory addressing modes”

1. Absolute
2. Indirect
3. “Base + displacement”
4. 2 indexed
5. 4 scaled indexed

General Syntax: \( \text{Imm}(r_b, r_i, s) \)

Effect: \( M[\text{Imm} + R[r_b] + R[r_i] \times s] \)

See Table of x86-64 Addressing Modes on Resources web page of our course web site
1. Absolute memory addressing mode

- Use memory address as operand directly in instruction

- Operand syntax: \texttt{Imm}

- Effect: \texttt{M[Imm]}

- Example: \texttt{call 0x400520}
2. Indirect memory addressing mode

- When a register contains an address
  - Similar to a pointer in C
- To access the data at that address, we use parentheses
- General Syntax: \((r_b)\)
- Effect: \(M[R[r_b]]\)

Example:

<table>
<thead>
<tr>
<th>Meaning or effect:</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r_ax) &lt;- (r_ax + r_di)</td>
<td>(r_ax = 15)</td>
<td>(r_ax = 21)</td>
</tr>
<tr>
<td>(r_di) = 6</td>
<td>(r_di = 6)</td>
<td>(r_di = 6)</td>
</tr>
</tbody>
</table>

**Example:**

- \(\text{add} \ %\_rdi, \ %\_rax\) vs \(\text{add} \ %\_rdi, \ (\%\_rax)\)

- Meaning or effect: \(r\_ax\) <- \(r\_ax + r\_di\) vs \(M[r\_ax] <- M[r\_ax] + r\_di\)

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r_ax) = 15</td>
<td>(r_ax) = 15</td>
</tr>
<tr>
<td>(r_di) = 6</td>
<td>(r_di) = 6</td>
</tr>
<tr>
<td>(M[15]) = 11</td>
<td>(M[15] = 11)</td>
</tr>
</tbody>
</table>

**Meaning or effect:**

- \(R[r\_ax]\) <- \(R[r\_ax] + R[r\_di]\)

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M[15]) = 11</td>
<td>(M[15] = 17)</td>
</tr>
</tbody>
</table>

not used
3. “Base + displacement” memory addressing mode

- General Syntax: \textit{Imm}(r_b)
- Effect: \texttt{M[Imm + R[r_b]]}
- Example: \texttt{leaq 7(\%rdi), \%rax}
- Careful here!
  - When dealing with \texttt{leaq}, the effect is \textit{Imm + R[r_b]}
4. Indexed memory addressing mode

1. General Syntax: $(r_b, r_i)$
   - Effect: $M[R[r_b] + R[r_i]]$
   - Example: `movb (%rdi, %rcx), %al`

2. General Syntax: $\text{Imm}(r_b, r_i)$
   - Effect: $M[\text{Imm} + R[r_b] + R[r_i]]$
   - Example: `movw 0xA(%rdi, %rcx), %r11w`
5. Scaled indexed memory addressing mode

1. General Syntax: $(, r_i, s)$
   - Example: $(, \%rdi, 2)$
   - Effect: $M[R[r_i] \times s]$

2. General Syntax: $\text{Imm}(, r_i, s)$
   - Example: $3(, \%rcx, 8)$
   - Effect: $M[\text{Imm} + R[r_i] \times s]$

3. General Syntax: $(r_b, r_i, s)$
   - Example: $(\%rdi, \%rsi, 4)$
   - Effect: $M[R[r_b] + R[r_i] \times s]$

4. General Syntax: $\text{Imm}(r_b, r_i, s)$
   - Example: $8(\%rdi, \%rsi, 4)$
   - Effect: $M[\text{Imm} + R[r_b] + R[r_i] \times s]$
Let’s try it!

<table>
<thead>
<tr>
<th>Expression</th>
<th>Address Computation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>8(%rdx)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(%rdx,%rcx)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(%rdx,%rcx,4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x80(%rdx,2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x % (%rdx, 2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x % (%rdx, 3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>%rdx</th>
<th>0xf000</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rcx</td>
<td>0x0100</td>
</tr>
</tbody>
</table>
Summary

- **leaq** - load effective address instruction
- Using data as operand to an instruction:
  - Immediate (constant)
  - Register (16 registers)
  - Memory (various memory addressing modes)
- General Syntax: \( \text{Imm}(r_b, r_i, s) \)
  1. Absolute
  2. Indirect
  3. “Base + displacement”
  4. 2 indexed
  5. 4 scale indexed
Next lecture

- Introduction
  - C program -> assembly code -> machine level code
  - Assembly language basics: data, move operation
  - Memory addressing modes
- Operation `lea` and Arithmetic & logical operations
- Conditional Statement
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations