CSE 351 Reference Sheet (Midterm)

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$2^0$</th>
<th>$2^1$</th>
<th>$2^2$</th>
<th>$2^3$</th>
<th>$2^4$</th>
<th>$2^5$</th>
<th>$2^6$</th>
<th>$2^7$</th>
<th>$2^8$</th>
<th>$2^9$</th>
<th>$2^{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
</tr>
</tbody>
</table>

Binary, Decimal, Hexadecimal Equivalents

Assembly Instructions

- **mov a, b**: Copy from a to b.
- **movs a, b**: Copy from a to b with sign extension. Needs two width specifiers.
- **movz a, b**: Copy from a to b with zero extension. Needs two width specifiers.
- **lea a, b**: Compute address and store in b.
  
  *Note: the scaling parameter of memory operands can only be 1, 2, 4, or 8.*
- **push src**: Push src onto the stack and decrement stack pointer.
- **pop dst**: Pop from the stack into dst and increment stack pointer.
- **call <func>**: Push return address onto stack and jump to a procedure.
- **ret**: Pop return address and jump there.
- **add a, b**: Add from a to b and store in b (and sets flags).
- **sub a, b**: Subtract a from b (compute b-a) and store in b (and sets flags).
- **imul a, b**: Multiply a and b and store in b (and sets flags).
- **and a, b**: Bitwise AND of a and b, store in b (and sets flags).
- **sar a, b**: Shift value of b right (arithmetic) by a bits, store in b (and sets flags).
- **shr a, b**: Shift value of b right (logical) by a bits, store in b (and sets flags).
- **shl a, b**: Shift value of b left by a bits, store in b (and sets flags).
- **cmp a, b**: Compare b with a (compute b-a and set condition codes based on result).
- **test a, b**: Bitwise AND of a and b and set condition codes based on result.
- **jmp <label>**: Unconditional jump to address.
- **j<sup>*</sup> <label>**: Conditional jump based on condition codes (*more on next page*).
- **set* a**: Set byte a to 0 or 1 based on condition codes.

**IEEE 754 Floating-Point Standard**

- Value: $\pm 1 \times$ Mantissa $\times 2^{\text{Exponent}}$
- Bit fields: $(-1)^E \times 1.M \times 2^{(E-\text{Bias})}$
  
  *where Single Precision Bias = 127,
  Double Precision Bias = 1023.*

**IEEE 754 Symbols**

<table>
<thead>
<tr>
<th>E</th>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>all zeros</td>
<td>all zeros</td>
<td>± 0</td>
</tr>
<tr>
<td>all zeros</td>
<td>non-zero</td>
<td>± denorm num</td>
</tr>
<tr>
<td>1 to MAX-1</td>
<td>anything</td>
<td>± norm num</td>
</tr>
<tr>
<td>all ones</td>
<td>all zeros</td>
<td>±∞</td>
</tr>
<tr>
<td>all ones</td>
<td>non-zero</td>
<td>NaN</td>
</tr>
</tbody>
</table>

**IEEE Single Precision and Double Precision Formats**

- **Single Precision**: 32 bits (S E M)
  
  - $S$: Sign bit (1 = negative, 0 = positive).
  - $E$: Exponent field (bias 127).
  - $M$: Fractional part.
- **Double Precision**: 64 bits (S E M)
  
  - $S$: Sign bit (1 = negative, 0 = positive).
  - $E$: Exponent field (bias 1023).
  - $M$: Fractional part.
### Conditionals

<table>
<thead>
<tr>
<th>Instruction</th>
<th>(op) s, d</th>
<th>test a, b</th>
<th>cmp a, b</th>
</tr>
</thead>
<tbody>
<tr>
<td>je</td>
<td>d (op) s == 0</td>
<td>b &amp; a == 0</td>
<td>b == a</td>
</tr>
<tr>
<td>jne</td>
<td>d (op) s != 0</td>
<td>b &amp; a != 0</td>
<td>b != a</td>
</tr>
<tr>
<td>js</td>
<td>d (op) s &lt; 0</td>
<td>b &amp; a &lt; 0</td>
<td>b-a &lt; 0</td>
</tr>
<tr>
<td>jns</td>
<td>d (op) s &gt;= 0</td>
<td>b &amp; a &gt;= 0</td>
<td>b-a &gt;= 0</td>
</tr>
<tr>
<td>jg</td>
<td>d (op) s &gt; 0</td>
<td>b &amp; a &gt; 0</td>
<td>b &gt; a</td>
</tr>
<tr>
<td>jge</td>
<td>d (op) s &gt;= 0</td>
<td>b &amp; a &gt;= 0</td>
<td>b &gt;= a</td>
</tr>
<tr>
<td>jl</td>
<td>d (op) s &lt; 0</td>
<td>b &amp; a &lt; 0</td>
<td>b &lt; a</td>
</tr>
<tr>
<td>jle</td>
<td>d (op) s &lt;= 0</td>
<td>b &amp; a &lt;= 0</td>
<td>b &lt;= a</td>
</tr>
<tr>
<td>ja</td>
<td>d (op) s &gt; 0U</td>
<td>b &amp; a &gt; 0U</td>
<td>b &gt;U a</td>
</tr>
<tr>
<td>jb</td>
<td>d (op) s &lt; 0U</td>
<td>b &amp; a &lt; 0U</td>
<td>b &lt;U a</td>
</tr>
</tbody>
</table>

### Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Convention</th>
<th>Name of “virtual” register</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>Return value – <strong>Caller saved</strong></td>
<td>%eax %ax %al</td>
</tr>
<tr>
<td>%rbx</td>
<td><strong>Callee saved</strong></td>
<td>%ebx %bx %bl</td>
</tr>
<tr>
<td>%rcx</td>
<td>Argument #4 – <strong>Caller saved</strong></td>
<td>%ecx %cx %cl</td>
</tr>
<tr>
<td>%rdx</td>
<td>Argument #3 – <strong>Caller saved</strong></td>
<td>%edx %dx %dl</td>
</tr>
<tr>
<td>%rsi</td>
<td>Argument #2 – <strong>Caller saved</strong></td>
<td>%esi %si %sil</td>
</tr>
<tr>
<td>%rdi</td>
<td>Argument #1 – <strong>Caller saved</strong></td>
<td>%edi %di %dil</td>
</tr>
<tr>
<td>%rsp</td>
<td><strong>Stack Pointer</strong></td>
<td>%esp %sp %spl</td>
</tr>
<tr>
<td>%rbp</td>
<td><strong>Callee saved</strong></td>
<td>%ebp %bp %bpl</td>
</tr>
<tr>
<td>%r8</td>
<td>Argument #5 – <strong>Caller saved</strong></td>
<td>%r8d %r8w %r8b</td>
</tr>
<tr>
<td>%r9</td>
<td>Argument #6 – <strong>Caller saved</strong></td>
<td>%r9d %r9w %r9b</td>
</tr>
<tr>
<td>%r10</td>
<td><strong>Caller saved</strong></td>
<td>%r10d %r10w %r10b</td>
</tr>
<tr>
<td>%r11</td>
<td><strong>Caller saved</strong></td>
<td>%r11d %r11w %r11b</td>
</tr>
<tr>
<td>%r12</td>
<td><strong>Callee saved</strong></td>
<td>%r12d %r12w %r12b</td>
</tr>
<tr>
<td>%r13</td>
<td><strong>Callee saved</strong></td>
<td>%r13d %r13w %r13b</td>
</tr>
<tr>
<td>%r14</td>
<td><strong>Callee saved</strong></td>
<td>%r14d %r14w %r14b</td>
</tr>
<tr>
<td>%r15</td>
<td><strong>Callee saved</strong></td>
<td>%r15d %r15w %r15b</td>
</tr>
</tbody>
</table>

### Sizes

<table>
<thead>
<tr>
<th>C type</th>
<th>x86-64 suffix</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>w</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>l</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>q</td>
<td>8</td>
</tr>
</tbody>
</table>