1. Perform the following number conversions:

   A. 0x48EA034D to binary: 100 1000 1110 1010 0000 0011 0100 1101
   B. binary 1 0001 0111 0010 1011 0101 0101 to hexadecimal: 1172B55
   C. 0xD4317 to binary: 110101000011001111
   D. binary 0010110110001110 to hexadecimal: 2D8E
   E. decimal 94 to binary: 1011110
   F. binary 01001101 to decimal: 77

2. Assume we are running code on a 6-bit machine using two’s complement arithmetic for signed integers. Values of type int and unsigned int are represented using 6 bits. TMax is the largest integer represented in this system and Tmin represents the smallest valued integer in the system. Fill in the empty boxes in the table below. The following definitions are used in the table below. Note: You do not need to fill in entries marked with “-”.

   int x = -15;
   unsigned ux = x;

<table>
<thead>
<tr>
<th>Expression</th>
<th>Decimal Representation</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0</td>
<td>00 0000</td>
</tr>
<tr>
<td>-</td>
<td>-6</td>
<td>11 1010</td>
</tr>
<tr>
<td>-</td>
<td>-10</td>
<td>11 1010</td>
</tr>
<tr>
<td>-</td>
<td>20</td>
<td>01 0100</td>
</tr>
<tr>
<td>-</td>
<td>-26</td>
<td>10 0110</td>
</tr>
<tr>
<td>ux</td>
<td>49</td>
<td>11 0001</td>
</tr>
<tr>
<td>x &lt;&lt; 4</td>
<td>16</td>
<td>01 0000</td>
</tr>
<tr>
<td>x &gt;&gt; 1</td>
<td>-8</td>
<td>11 1000</td>
</tr>
<tr>
<td>TMax</td>
<td>31</td>
<td>01 1111</td>
</tr>
<tr>
<td>Tmin</td>
<td>-32</td>
<td>10 0000</td>
</tr>
<tr>
<td>!x</td>
<td>0</td>
<td>00 0000</td>
</tr>
<tr>
<td>~x</td>
<td>14</td>
<td>00 1110</td>
</tr>
</tbody>
</table>
3. Suppose we have the following bytes in memory:

<table>
<thead>
<tr>
<th>Mem addr:</th>
<th>0x200</th>
<th>0x201</th>
<th>0x202</th>
<th>0x203</th>
<th>0x204</th>
</tr>
</thead>
<tbody>
<tr>
<td>values:</td>
<td>0x03</td>
<td>0xF1</td>
<td>0x0A</td>
<td>0x55</td>
<td>0xDA</td>
</tr>
</tbody>
</table>

And the following C code:

```c
char *cp = 0x200;
short *sp = 0x200;
int *ip = 0x200;
```

What are the values of `*cp`, `*sp`, and `*ip` if the system is a little-endian Linux system?

- `*cp = 0x3`
- `*sp = 0xF103`
- `*ip = 0x550AF103`

What would the values be on a big-endian system?

- `*cp = 0x3`
- `*sp = 0x03F1`
- `*ip = 0x03F10A55`

4. A single byte can be represented by 2 hexadecimal digits. Fill in the missing entries in the following table, giving the decimal, binary, and hexadecimal values of different byte patterns.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000 0000</td>
<td>0x00</td>
</tr>
<tr>
<td>167</td>
<td>1010 0111</td>
<td>0xA7</td>
</tr>
<tr>
<td>62</td>
<td>0011 1110</td>
<td>0x3E</td>
</tr>
<tr>
<td>188</td>
<td>1011 1100</td>
<td>0xBC</td>
</tr>
<tr>
<td>55</td>
<td>0011 0111</td>
<td>0x37</td>
</tr>
<tr>
<td>136</td>
<td>1000 1000</td>
<td>0x88</td>
</tr>
<tr>
<td>243</td>
<td>1111 0011</td>
<td>0xF3</td>
</tr>
</tbody>
</table>
5. Suppose that $x$ and $y$ have byte values of 0x66 and 0x39, respectively. Fill in the following table indicating the byte values of the different C expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &amp; y$</td>
<td>0x20</td>
</tr>
<tr>
<td>$x</td>
<td>y$</td>
</tr>
<tr>
<td>$\sim x</td>
<td>\sim y$</td>
</tr>
<tr>
<td>$x &amp; !y$</td>
<td>0x00</td>
</tr>
<tr>
<td>$x &amp;&amp; y$</td>
<td>0x01</td>
</tr>
<tr>
<td>$x</td>
<td></td>
</tr>
<tr>
<td>$!x</td>
<td></td>
</tr>
<tr>
<td>$x &amp;&amp; \sim y$</td>
<td>0x01</td>
</tr>
</tbody>
</table>

6. Using only bit-level and logical operations, write a C expression that is equivalent to $x == y$. In other words, it will return 1 when $x$ and $y$ are equal and 0 otherwise.

$! (x^y)$
7. Fill in the table below showing the effects of the different shift operations on single-byte quantities. The best way to think about shift operations is to work with binary representations. Convert the initial values to binary, perform the shifts, and then convert back to hexadecimal. Each of the answers should be 8 binary digits or two hexadecimal digits.

```
<table>
<thead>
<tr>
<th>X</th>
<th>X &lt;&lt; 3</th>
<th>Logical x &gt;&gt; 2</th>
<th>Arithmetic x &gt;&gt; 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>Binary</td>
<td>Hex</td>
<td>Binary</td>
</tr>
<tr>
<td>0xC3</td>
<td>1100 0011</td>
<td>0001 1000</td>
<td>0x18</td>
</tr>
<tr>
<td>0x75</td>
<td>0111 0101</td>
<td>1010 1000</td>
<td>0xA8</td>
</tr>
<tr>
<td>0x87</td>
<td>1000 0111</td>
<td>0011 1000</td>
<td>0x38</td>
</tr>
<tr>
<td>0x66</td>
<td>0110 0110</td>
<td>0011 0000</td>
<td>0x30</td>
</tr>
</tbody>
</table>

8. Assume we are representing numbers as 4-bit 2's complement numbers. Fill in the table below to determine the inverse of the given number.

```
<table>
<thead>
<tr>
<th>X</th>
<th>-X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>Decimal</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>-8</td>
</tr>
<tr>
<td>D</td>
<td>-3</td>
</tr>
<tr>
<td>F</td>
<td>-1</td>
</tr>
</tbody>
</table>