1. (25%) Assume char is 6-bits and short is 9-bits, convert the following into formats described by each column.

<table>
<thead>
<tr>
<th>unsigned short range:</th>
<th>8051 assembler</th>
<th>big endian two's compl.</th>
<th>big endian one's compl.</th>
<th>big endian signed magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0~511</td>
<td>0~511</td>
<td></td>
</tr>
<tr>
<td>signed char range:</td>
<td></td>
<td>-32~31</td>
<td>-31~31</td>
<td>-31~31</td>
</tr>
<tr>
<td>unsigned char x=042;</td>
<td>.byte 420</td>
<td>100010</td>
<td>100010</td>
<td>out of range</td>
</tr>
<tr>
<td>unsigned char x=42;</td>
<td>.byte 42</td>
<td>101010</td>
<td>101010</td>
<td>out of range</td>
</tr>
<tr>
<td>unsigned char x=0x42;</td>
<td>.byte 42h</td>
<td>Out of range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signed char x=-16;</td>
<td>.word -16</td>
<td>11110000</td>
<td>11110111</td>
<td>10001000</td>
</tr>
</tbody>
</table>

2. (5%) For column 2, disassemble the following 8-bit binary number into assembler and for columns 3 to 6 convert into decimal under the different interpretations. (8051 Hint: MOV A,Rn; ADD A,Rn; ADDC A,Rn or ANL A,Rn).

<table>
<thead>
<tr>
<th>8051 instruction</th>
<th>signed big endian (8_{10})</th>
<th>unsigned big endian (8_{10})</th>
<th>signed big endian (8_{10})</th>
<th>big endian signed magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111101</td>
<td>ADDC A,R5</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
</tbody>
</table>

3. (5%) Using C++ operator precedence, add the correct parenthesis:

\[
w = (a << (b + (c + d))) ;
\]

\[
w += ((a & b) | (c % d)) ;
\]
4. (20%) Using standard C++ data types (i.e. 8-bit char) convert the following using the following values:

- register unsigned char \( u = 0xff, f = 0x96, g = 0x33; \) register signed char \( s = 1, t = 0x46, x = 0xc3; \)

Assign the 8051 registers as follows: \( s = R0, t = R1, x = R2, u = R3, f = R4, g = R5; \) State if overflow or carry has occurred after execution. 8051 MUL, DIV and SUBB instructions are not allowed.

<table>
<thead>
<tr>
<th>two's complement big endian</th>
<th>8051 instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u = f - g; ) ( 1010 \ 0101 )</td>
<td>( \text{MOV A,R4;} \ \text{XRL A,R5;} \ \text{MOV R3,A} )</td>
</tr>
<tr>
<td>( u = g &amp; 0x0f; ) ( 0000 \ 0011 )</td>
<td>( \text{MOV A,R5;} \ \text{ANL A, R0XOF;} \ \text{MOV R5A} )</td>
</tr>
<tr>
<td>( u = g \times 4; ) ( 1100 \ 1100 )</td>
<td>( \text{MOV A,R5;} \ \text{RRC \ RLC; RLC A} )</td>
</tr>
<tr>
<td>( g--; ) ( 0011 \ 0010 )</td>
<td>( \text{Dec R5} )</td>
</tr>
<tr>
<td>( s = t - x; ) ( 1000 \ 0011 )</td>
<td>( \text{MOV A,R2;} \ \text{CPL A;} \ \text{Inc A} )</td>
</tr>
<tr>
<td>( s + x; ) ( 1100 \ 0100 )</td>
<td>( \text{ADD A,R1;} \ \text{Mov R0,A} )</td>
</tr>
<tr>
<td>( s = t \mid x; ) ( 1100 \ 0111 )</td>
<td>( \text{MOV A,R1;} \ \text{OrL A,R2;} \ \text{Mov R0,A} )</td>
</tr>
</tbody>
</table>

5. (15%) Using the 8051 instructions, assemble the instruction into hex, and then execute it showing all changing values:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>1110 0100 E 4</td>
<td>CLR A</td>
<td>0x0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0x42</td>
<td>0x13</td>
</tr>
<tr>
<td>0x1</td>
<td>1011 0011 B 3</td>
<td>CPL C</td>
<td>0x1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0x00</td>
<td>0x13</td>
</tr>
<tr>
<td>0x2</td>
<td>0011 1101 3 D</td>
<td>ADDC A,R5</td>
<td>0x2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0x00</td>
<td>0x13</td>
</tr>
<tr>
<td>0x3</td>
<td>1111 0100 F 4</td>
<td>CPL A</td>
<td>0x3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x13</td>
<td>0x13</td>
</tr>
<tr>
<td>0x4</td>
<td>1111 1010 F 1</td>
<td>MOV R5,A</td>
<td>0x4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0xEC</td>
<td>0x13</td>
</tr>
</tbody>
</table>

Final Values: \( 0x5 \ 0 \ 0 \ 0 \ 0xEC \ 0xEC \)

\( x = 0xC3 = \ 1100 \ 0011 \ b \)
\( -x = 0x3C = \ 0011 \ 1100 \ b \)
\( x + 1 = 0x3D = \ 0011 \ 1110 \ b \)
\( t = 0x46 = \ 0100 \ 0110 \ b \)
\( -x = 0x3D = \ 0011 \ 1110 \ b \)
6. (5%) Assuming standard C++ data types, write a "single" C++ code statement of clearing bit \( d_0 \) to zero, setting bit \( d_3 \) to one, complement bit \( d_1 \) (i.e. invert the bit \( d_1 \)) and leaving all other bits unchanged in the variable \( x \).

\[
\begin{align*}
x & \equiv 10111111_B \\
   & \equiv 00001000_8 \\
x & \equiv 00000010_2 \\
x & = ((x \& 0xBF) | 0x08) ^ 0x02;
\end{align*}
\]

7. (5%) Write the 8051 instructions for problem 6 and assume \( x \) is register R3.

\[
\begin{align*}
\text{MOV} A, R3 \\
\text{ANL} A, #0xBF \\
\text{ORL} A, #0x08 \\
\text{XRL} A, #0x02 \\
\text{MOV} R3, A
\end{align*}
\]

8. (5%) Write the 8051 assembler for char \( w[8] = "-128"; \)

\[\text{byte } "-128", 0, 0, 0, 0 \text{ need 8 characters end of string zero}\]

9. (10%) Rewrite the following C statement without using multiply, divide or modulo: \( u = x \times 13 + y/4 + z/8; \)

\[
\begin{align*}
u & = ((x << 3) + ((x << 2) + x) + y >> 2 + (z \& 0111b)) \\
   & \equiv (y \times 2^{-2}) \\
   & \equiv (y/4 - 2) \\
   & \equiv y << 2
\end{align*}
\]

9. (5%) Why is two’s complement representation preferred over one’s complement?

Two’s complement has only one representation of zero and has one greater decimal value range.

x1 (5% extra credit). What is the advantage of binary representation over ASCII representation of numbers? (i.e. 128 decimal number = 10000000 binary = "128" ASCII);

\[\text{Binary representation uses less memory to store the same numbers.}\]

x2 (5% extra credit). Convert the 24-bit number 0x031879 to mime base64:

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 \\
0 & 4 & 9 & 3 \\
A & x & h & 5
\end{array}
\]