1. [8 points] Suppose we have 8-bit signed numbers. Fill-in the following table:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6</td>
<td>1111 1010</td>
<td>0xFA</td>
</tr>
<tr>
<td>+97</td>
<td>01100001</td>
<td>0x61</td>
</tr>
<tr>
<td>-21</td>
<td>1110 1011</td>
<td>0xEB</td>
</tr>
<tr>
<td>+65</td>
<td>0100 0001</td>
<td>0x41</td>
</tr>
</tbody>
</table>

2. [2 points] Suppose you have the decimal number  -1.75
Write this number in IEEE floating point format 754 single precision, showing all the steps to get full credit.

-1.75 = -1.11*2^0
S (1bit) → 1 (negative number)
Exp (8 bits) → x-127 = 0 → x = 127 = 0111 1111
Frac (23 bits) → 110000…000
3. Suppose we have the following logic circuit:

![Logic Circuit Diagram]

a) [8 points] Draw the truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

b) [4 points] Write the Boolean equation of the circuit: twice:

- once from the truth table:
  
  \[ A'BC' \]

- once from the logic circuit (do not simplify using De Morgan’s law)
  
  \[ C'B(AB)' \]
c) [2 points] From the two equations in b) which one is cheaper (i.e. requires less number of gates)? Explain your answer

The first one is cheaper as it needs only one AND gate and two inverters; while the second requires the gates you see in the figure at the beginning of the program (remember that NAND = AND + inverter)

d) [4 points] Re-draw the circuit using only NAND gates. Use as few NAND gate as you can.

There are several ways of doing it. But the general rules are:
- inverter = one-input NAND
- AND = NAND followed by one-input NAND
4. [4 points] Given the following datapath

Suppose we have the instruction: \textbf{\texttt{lwd imm(R5), R6}}. This is not a real MIPS instruction, we invented it for the exam! This instruction does two things: load the memory content from address R5+imm, put it in R6, then increment R5 by imm. That is: \textit{R6 = M[imm + R5] then R5 = R5 + imm} Write the microinstructions needed to execute this instruction. No need to write the fetch phase. Optimize as much as you can (i.e. show minimum microinstructions and put in the same line the microinstructions that will execute in parallel).

\begin{align*}
\text{SE(imm)} & \rightarrow \text{AIR} \\
\text{AIR + RF[R5]} & \rightarrow \text{AOR} \\
\text{AOR} & \rightarrow \text{RF[R5], AOR} \rightarrow \text{MAR} \\
\text{M[MAR]} & \rightarrow \text{MDR} \\
\text{MDR} & \rightarrow \text{MAR} \\
\text{MAR} & \rightarrow \text{RF[R6]}
\end{align*}
5. Suppose \( X = 0xEB \) and \( Y = 0xAC \)

a) [2 points] Translate A and B to binary
\[
X = 1110 1011  \\
Y = 1010 1100
\]

b) [2 points] Assume A and B are signed numbers. Perform \((A+ B)\) using the binary numbers you calculate in a) above.

\[
\begin{array}{ccccccc}
1 & 1 & 1 & 1 & 0 & 1 & 0 \\
+ & & & & & & \\
1 & 0 & 1 & 0 & 1 & 1 & 0 \\
\hline
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1
\end{array}
\]

\(c) \ [2 \text{ points}] \) Did we get an overflow in the operation in number b) above?

No, two negatives added together and resulted in a negative. This extra “1” is considered a sign-extension.

\(d) \ [2 \text{ points}] \) Now, assume X and Y are unsigned numbers. Does X+Y result in overflow? Justify

Yes, because every single “1” counts.