PROBLEM 1. [20 points] Consider the following truth table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

1.1 Convert the above truth table into two reduced sums (one for \(a\) and one for \(b\)) of products of the Boolean variables \(x, y, z\).

\[
a = \bar{x}y\bar{z} + \bar{x}yz + \bar{x}y\bar{z} + xy\bar{z} \\
≡ \bar{x}y + \bar{x}y\bar{z} + \bar{x}y\bar{z} \\
≡ \bar{x}z + \bar{x}yz + \bar{x}y\bar{z} \\
≡ \bar{y}\bar{z} + \bar{x}yz + \bar{x}y\bar{z} \\
≡ \bar{x}y + \bar{x}z + \bar{y}\bar{z}
\]

\[
b = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xy\bar{z}
\]

1.2 Draw, with AND, OR, NOT gates (with not more than 2 bits of input), a circuit that implements the above truth table.

\(a\) is a Minority circuit, \(b\) is NOT of a 3-way XOR.

PROBLEM 2. [30 points] Consider a pipe-lined process (like the laundry example given in class) with \(s\) stages. Assuming \(n\) tasks are being processed by this pipe-line. Both \(s\) and \(n\) are positive integers. In each of the scenarios listed below in (2.1), (2.2) and (2.3), compute formulas giving the following quantities:

- \(C(n, s)\): the total number of clock-cycles during which the pipe-lined process runs,
- \(R(n, s)\): the speedup ratio with respect a non-pipe-lined serial execution,
- \(F(n, s)\): the fill time, that is, the total number of clock-cycles before the pipe-line runs at full occupancy (that is, before all pipe-line stages are used),
- \(D(n, s)\): the drain time, that is, the total number of clock-cycles remaining after the pipe-line no longer runs at full occupancy,
- \(O(n, s)\): the percentage of time during which the pipe-line runs at full occupancy (that is, during which all pipe-line stages are used).
The three considered scenarios are the following ones.

2.1 Each stage runs within the same amount of time.

\[
\begin{align*}
C(n, s) &= n + s - 1 \\
R(n, s) &= \frac{ns}{n+s-1} \\
F(n, s) &= s - 1 \\
D(n, s) &= s - 1 \\
O(n, s) &= \frac{n-s+1}{n+s-1}
\end{align*}
\]

2.2 Each stage, but the first one, runs within \( t \) units of time (say pico-seconds) meanwhile the first stage runs within \( rt \) units of time where \( r \) is a constant greater than one, thus, the first stage is slower than the other ones. You can assume \( t = 1 \) in order to simplify calculations. Keeping \( t \) non-specialized (so as a symbol instead of letting be 1) yields a bonus of 5 marks.

One can assume that the clock-cycle has length of longest stage, \( rt \). Moreover, note there is a duality of sorts for solutions here. They can be given in terms of seconds (w.r.t. \( t \)) or by number of clock cycles.

\[
\begin{align*}
C(n, s) &= n + s - 1 \text{ or } rt(n + s - 1) \\
R(n, s) &= \frac{rt(s-1)}{rt(n+s-1)} \\
F(n, s) &= s - 1 \text{ or } rt(s - 1) \\
D(n, s) &= s - 1 \text{ or } rt(s - 1) \\
O(n, s) &= \frac{rt(n-s+1)}{rt(n+s-1)} = \frac{n-s+1}{n+s-1}
\end{align*}
\]

2.3 Each stage, but the last one, runs within \( t \) units of time (say pico-seconds) meanwhile the last stage runs within \( rt \) units of time where \( r \) is a constant greater than one, thus, the last stage is slower than the other ones. You can assume \( t = 1 \) in order to simplify calculations. Keeping \( t \) non-specialized (so as a symbol instead of letting be 1) yields a bonus of 5 marks.

In each of the scenarios (2.1), (2.2) and (2.3), you can assume that every stage runs within one clock-cycle. Moreover, you can assume that this clock-cycle is equal in time to the longest stage of the pipeline.

One should reason as in 2.2. The values are exactly the same as 2.2.

**PROBLEM 3.** [20 points] In this exercise, we consider a new MIPS instruction which is specified in the RTL language:

\[
\begin{align*}
(1) \quad PC &\leftarrow PC + 4; \\
(2) \quad Reg[Rt] &\leftarrow Mem[Reg[Rs]]; \\
(3) \quad Reg[Rs] &\leftarrow Reg[Rs] + 1;
\end{align*}
\]

3.1 Which existing blocks in Figure[1] (if any) can be used for this instruction?

All of them.

3.2 Which new functional blocks (if any) do we need for this instruction?

3.3 What new signals do we need (if any) from the control unit to support this instruction?

3.4 Repeat questions 3.1, 3.2, 3.3 with this other new MIPS which is specified in the RTL language:

\[
\begin{align*}
(1) \quad PC &\leftarrow PC + 4; \\
(2) \quad Mem[Reg[Rt]] &\leftarrow Reg[Rs]; \\
(3) \quad Reg[Rs] &\leftarrow Reg[Rs] + 1;
\end{align*}
\]
Figure 1: The basic implementation of the data-path and control for a MIPS instruction.

PROBLEM 4. [20 points] Consider the following Boolean expression:

\[ \overline{x}y + \overline{x}\overline{y}z + x\overline{y}z + \overline{x}y\overline{z} + x\overline{y}z + \overline{x}\overline{y}z. \]

4.1 Using the simplification rules in Slide 18 on the PDF version of the set of slides 5.3, show that the above expression is logically equivalent to

\[ x\overline{y} + yz + \overline{x}. \]

\[
\begin{align*}
\overline{x}y + \overline{x}\overline{y}z + x\overline{y}z + \overline{x}y\overline{z} + x\overline{y}z + \overline{x}\overline{y}z \\
= (\overline{x}y + \overline{x}\overline{y}z) + (x\overline{y}z + \overline{x}y\overline{z}) + (x\overline{y}z + \overline{x}y\overline{z}) \\
= \overline{x}z(y + \overline{y}) + x\overline{y}(z + \overline{z}) + y\overline{z}(x + \overline{x}) \\
= \overline{x}z(1) + x\overline{y}(1) + y\overline{z}(1) \\
= x\overline{y} + y\overline{z} + \overline{x}.
\end{align*}
\]

4.2 Give a Boolean circuit (using the logic gates AND and NOT implementing the latter expression.

PROBLEM 5. [10 points] Suppose we want to add a new instruction, `beqr`, which will be used like this: `beqr $x, $y, $z` will branch to the address in $z$ if $x$ and $y$ are equal, otherwise continue to the next instruction. Show any changes to the data-path in Figure 2 that would need to be made in order for this instruction to work. If any control signals are added, indicate their values for the two possible outcomes of the `beqr` instruction. That is, for when $x == y$ and when $x != y$.

Regarding to the pink region in Figure 3, the mux is added [1], the RegtoPC is added from Controller [1], and one of the input to mux is from registers [1].

If $x == y$, $nPC\_sel = 1$, equals = 1 and RegtoPC = 1 and $ALU\_ctr = 'Equal'$. [2]

Otherwise, $nPC\_sel = 1$, equals = 0, RegtoPC = 1 and $ALU\_ctr = 'Equal'$. [2]
Figure 2: Basic data-path in a simplified format

Figure 3: Solution to problem 5: the pink region [3], nPC_sel & AND gate [2] and ALUctr [1]