Question 1. (20 pts)

Consider the basic pipelined CPU architecture. The CPU does not have any additional circuitry to detect data dependency so it does not stall.

The WB stage updates the destination register during the first half of a cycle and the ID stage fetches its operands during the second half. One instruction is fetched in each CPU cycle and Table 1 shows a segment of a program that is being executed by this pipelined CPU. The first column contains the instructions in the program segment and the remaining columns are labeled by the time (in CPU cycles). The IF, ID, EX, MEM and WB labels in the columns indicate the stages of the pipelined CPU that is processing the instruction at a given time.

The instruction \texttt{add r1, r2, r10} means: \( r_{10} = r_{1} + r_{2} \).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>add r1, r2, r10</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r10, r11, r11</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r10, r11, r12</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r10, r11, r13</td>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r10, r11, r14</td>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Program Segment

At the beginning of the execution, registers \( r_{1} = 1 \), \( r_{2} = 2 \), \( r_{10} = 10 \) and \( r_{11} = 11 \). The instruction “add r1, r2, r10” adds r1 and r2 and stores result in r10. Answer the following questions: One time unit is equal to one CPU clock period.

- What are the instructions fetched at time units 1, 2, 3, 4 and 5 by the IF stage? (5 pts)
  1.
  2.
  3.
  4.
  5.
• What values are fetched into the A-latch and B-latch at time units 2, 3, 4, 5 and 6 by the ID stage? (5 pts)

2.
3.
4.
5.
6.

• Which registers will be updated by the WB stage and what values will be stored at time units 5, 6, 7, 8 and 9? (5 pts)

5.
6.
7.
8.
9.

• As discussed in class, due to pipelining, several instructions are active simultaneously, making some instructions fetch a wrong value(s). We solved the problem with data forwarding (using forwarding registers/buffers).

We can also solve the problem without using forwarding registers by inserting NOP instructions to delay some instruction(s).

Insert the minimum number of NOP instructions in the program segment in Table 1 so that all instructions can fetch their values correctly.

Answer: (Write the program with NOPs below — no table needed) (5 pts)
Question 2. (20 pts)

Consider the pipelined CPU architecture with data forwarding that is also discussed in class.

The WB stage updates the destination register during the first half of a cycle and the ID stage fetches its operands during the second half. One instruction is fetched in each CPU cycle and Table 2 shows a segment of a program that is being executed by this pipelined CPU. The first column contains the instructions in the program segment and the remaining columns are labeled by the time (in CPU cycles). The IF, ID, EX, MEM and WB labels in the columns indicate the stages of the pipelined CPU that is processing the instruction at a given time. The topmost 2 rows contains scratch area for you to fill in the content of the data forwarding buffers produced by the EX stage.

The instruction **add r1, r2, r10** means: \( r10 = r1 + r2 \).

| Buffer2: (tag|value) | | | | | | | | | |
| Buffer1: (tag|value) | | | | | | | | | |
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| add r1, r2, r10 | IF | ID | EX | MEM | WB |
| add r11, r10, r11 | IF | ID | EX | MEM | WB |
| add r10, r11, r12 | IF | ID | EX | MEM | WB |
| add r10, r11, r13 | IF | ID | EX | MEM | WB |
| add r10, r11, r14 | IF | ID | EX | MEM | WB |

Table 2: Program Segment

At the beginning of the execution, registers \( r1 = 1, r2 = 2, r10 = 10 \) and \( r11 = 11 \). Answer the following questions:

1. What values are fetched into the A- and B-latches at time units 2, 3, 4, 5 and 6 by the ID stage? (5 pts)

2. 

3. 

4. 

5. 

6.
2. What are the contents of the tag and value of Buffer1 and Buffer2 at the end of time units 3, 4, 5, 6, 7 and 8? (5 pts)
   If you cannot provide the exact value for some cases, then say that the answer is “unknown”.

3. 
   
   4. 
   
   5. 
   
   6. 
   
   7. 
   
   8. 

3. For each instruction in Table 2: (2 pts per question)
   (a) State if the instruction requires data forwarding to obtain the correct values for the source operands.
   (b) If you answered “yes” for part (a), then which source operands are forwarded?

add r1, r2, r10: Need data forwarding? (Yes/No) If yes, which operand(s):
add r11, r10, r11: Need data forwarding? (Yes/No) If yes, which operand(s):
add r10, r11, r12: Need data forwarding? (Yes/No) If yes, which operand(s):
add r10, r11, r13: Need data forwarding? (Yes/No) If yes, which operand(s):
add r10, r11, r14: Need data forwarding? (Yes/No) If yes, which operand(s):
Question 3. (10 pts)

Consider the pipelined CPU architecture with data forwarding (including the value in the LMDR) that was also discussed in class.

The WB stage updates the destination register during the first half of a cycle and the ID stage fetches its operands during the second half. One instruction is fetched in each CPU cycle and Table 2 shows a segment of a program that is being executed by this pipelined CPU. The first column contains the instructions in the program segment and the remaining columns are labeled by the time (in CPU cycles). The IF, ID, EX, MEM and WB labels in the columns indicate the stages of the pipelined CPU that is processing the instruction at a given time.

The instruction add r1, r2, r10 means: r10 = r1 + r2.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld [r1+r2], r10</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [r10+r3], r11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r10, r11, r12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r10, r11, r13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r11, r12, r14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Program Segment

Answer the following questions:

1. Fill in the above figure with IF, ID, EX, MEM and WB labels to indicate the stage that is processing the instruction. (5 pts)

2. After completing the previous question, circle all EX stages of instructions that require data forwarding. (5 pts)
Question 4. (20 pts)

Consider the following section of an assembly program that contains consecutive branching operations:

```assembly
mov 0, r0
bra LABEL1
bra LABEL2
add r0, 1, r0
LABEL1: bra LABEL3
add r0, 2, r0
LABEL2: add r0, 3, r0
add r0, 4, r0
LABEL3: add r0, 5, r0
add r0, 6, r0
add r0, 7, r0
add r0, 8, r0
add r0, 9, r0
```

- Using the **basic pipelined processor architecture** discussed in class, show the execution of the above program in the table below by showing which pipeline stage contains a particular instruction at the given time unit.

I have filled in the first 2 instructions for you. The table shows that the `mov 0, r0` is being processed by the IF stage at time 1 (i.e., it is being fetched), this instruction is in the ID stage at time 2, and so on. The `bra LABEL1` is being processed by the IF stage at time 2, this instruction is in the ID stage at time 2, and so on.

Complete all rows of the table by finding out which instruction will be fetched and then fill in which pipeline stage the instruction is at certain times.

You only need to fill in the next 7 instructions that are fetched and the stages they are in:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov 0, r0</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bra LABEL1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Fill in instructions) | Fill in IF, ID, EX, MEM, WB
Question 5. (20 pts)

Consider the following section of an assembly program that contains consecutive branching operations:

\[
\begin{align*}
\text{mov} & \quad 0, r0 \\
\text{bra} & \quad \text{LABEL1} \\
\text{bra} & \quad \text{LABEL2} \\
\text{add} & \quad r0, 1, r0 \\
\text{LABEL1}: & \quad \text{bra} \quad \text{LABEL3} \\
\text{add} & \quad r0, 2, r0 \\
\text{LABEL2}: & \quad \text{add} \quad r0, 3, r0 \\
& \quad \text{add} \quad r0, 4, r0 \\
\text{LABEL3}: & \quad \text{add} \quad r0, 5, r0 \\
& \quad \text{add} \quad r0, 6, r0 \\
& \quad \text{add} \quad r0, 7, r0 \\
& \quad \text{add} \quad r0, 8, r0 \\
& \quad \text{add} \quad r0, 9, r0 \\
\end{align*}
\]

X:

- Using the **improved pipelined architecture** discussed in class (with 1 (one) slot branch delay), show the execution of the above program in the table below by showing which pipeline stage contains a particular instruction at the given time unit.

  I have filled in the first 2 instructions for you. The table shows that the `mov 0,r10` is being processed by the IF stage at time 1 (i.e., it is being fetched), this instruction is in the ID stage at time 2, and so on. The `bra LABEL1` is being processed by the IF stage at time 2, this instruction is in the ID stage at time 2, and so on.

  Complete all rows of the table by finding out which instruction will be fetched and then fill in which pipeline stage the instruction is at certain times.

  You only need to fill in the next 7 instructions that are fetched and the stages they are in:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov 0,r0</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bra LABEL1</td>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Fill in instructions)  Fill in IF, ID, EX, MEM, WB
Question 6. (10 pts)

Consider the following program executed by the pipelined CPU with data forwarding hardware. We say that a program (segment) is executed correctly if the effect of the execution in the pipelined CPU is the same as when each instruction is executed completely before the next instruction is fetched (that’s exactly what you would expect...)

- Can the following program be executed correct? (5 pts)

```
add r1, r2, r3  (r3 := r1 + r2)
st  r4, [r3 + 4]
```

If not, add additional forwarding circuitry in the figure on the next page so that the CPU will be able to execute the program correctly.

(Obviously, if your answer is yes to this question, you don’t have to do anything to the figure on the next page....)
• Can the following program be executed correctly? (5 pts)

```
add r1, r2, r3  (r3 := r1 + r2)
st  r3, [r4 + 4]
```

If not, add additional forwarding circuitry in the figure on the next page so that the CPU will be able to execute the program correctly.

(Obviously, if your answer is yes to this question, you don’t have to do anything to the figure....)