The quiz consists of two exercises. The expected duration is 30 minutes. All answers should be written in the answer boxes.

**MIPS Cheat Sheet**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Translation in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>load word</td>
<td>lw $s0, 4($s1)</td>
<td>$s0 = Memory($s1+4)</td>
</tr>
<tr>
<td>store word</td>
<td>sw $s0, 4($s1)</td>
<td>Memory($s1+4) = $s0</td>
</tr>
<tr>
<td>add</td>
<td>add $s0, $s1, $s2</td>
<td>$s0 = $s1 + $s2</td>
</tr>
<tr>
<td>subtract</td>
<td>sub $s0, $s1, $s2</td>
<td>$s0 = $s1 - $s2</td>
</tr>
<tr>
<td>add immediate</td>
<td>addi $s0, $s1, 2</td>
<td>$s0 = $s1 + 2</td>
</tr>
<tr>
<td>subtract immediate</td>
<td>subi $s0, $s1, 2</td>
<td>$s0 = $s1 - 2</td>
</tr>
<tr>
<td>shift left</td>
<td>sll $s0, $s1, 2</td>
<td>$s0 = $s1 ≪ 2 bits</td>
</tr>
<tr>
<td>shift right</td>
<td>srl $t2, $t2, 1</td>
<td>$t2 = $t2 / 2</td>
</tr>
<tr>
<td>branch on equal</td>
<td>beq $s0, $s1, L</td>
<td>if ($s0 == $s1) go to L</td>
</tr>
<tr>
<td>branch on not equal</td>
<td>bne $s0, $s1, L</td>
<td>if ($s0 != $s1) go to L</td>
</tr>
<tr>
<td>set on less than</td>
<td>slt $s0, $s1, $s2</td>
<td>if ($s1 &lt; $s2) $s0 = 1 else $s0 = 0</td>
</tr>
<tr>
<td>jump</td>
<td>j L</td>
<td>go to L</td>
</tr>
<tr>
<td>jump register</td>
<td>jr $s0</td>
<td>go to $s0</td>
</tr>
<tr>
<td>jump and link</td>
<td>ja 250</td>
<td>go to 1000; $ra=PC+4</td>
</tr>
</tbody>
</table>

**Table 1: Control signals for some core MIPS instructions**

There are 3 types of pipeline hazards:
• Structural hazards: Attempt to use the same resource by two different instructions at the same time

• Data hazards: Attempt to use data before it is ready in instructions involving arithmetic and data transfers. An instruction’s source operand(s) are produced by a prior instruction still in the pipeline

• Control hazards: Attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated; branch instructions

A first example of data hazard:

```plaintext
add $t0,$t1,$t2
sub $t4,$t0,$t3
and $t5,$t0,$t6
or $t7,$t0,$t8
xor $t9,$t0,$t10
```

Note: ALU-ALU forwarding is used.

A second example of data hazard:

```plaintext
lw $t0, 0($t1)
nop
sub $t3,$t0,$t2
and $t5,$t0,$t4
or $t7,$t0,$t6
```

Note: MEM-ALU forwarding is used.

**Exercise 1.** [40 points] Consider the datapath shown in Figure 1. In this question, we study the following MIPS instruction:

```plaintext
slti rt, rs, 17  # if (Reg[rs] < 17) Reg[rt] = 1
                 # else Reg[rt] = 0
```
1.1 Describe the five stages of the datapath for this instruction \texttt{slti}.

Stage 1 Instruction fetch: fetch \texttt{slti} instruction, increment PC by 4

Stage 2 Decode: decode to determine it is the shift left operation, then read register $t2$ to $rs$ and set $imm16$ to 1 (or 10 1 0)

Stage 3 Execution: shift the value of $t2$ by 1 bit in the ALU unit

Stage 4 Memory: idle

Stage 5 Register write: write the result of Stage 3 into register $t2$
1.2 Which blocks (PC, PC Ext, Adder by 4, Adder by PC Ext, Inst Memory, RegFile, Extender, ALU, Data Memory) in Figure 1 perform a useful function for this instruction `slti`?

PC, Adder by 4, Inst Memory, RegFile, Extender, ALU

1.3 What are the values of the control signals generated by the controller in Figure 1 for this instruction `slti`?

- \( \text{nPC\_sel} = 0 \) or `+4`
- \( \text{RegDst} = 1 \) or `Rd`
- \( \text{RegWr} = 1 \)
- \( \text{ExtOp} = 0 \) or `unsigned` (or unknown)
- \( \text{ALU\_src} = 1 \) or `imm`
- \( \text{ALU\_ctr} = \text{`shift left`} \)
- \( \text{Mem\_Wr} = 0 \)
- \( \text{Mem\_to\_Reg} = 0 \) or `ALU`

**Exercise 2.** [60 points] We would like to execute the following MIPS code on a 5-stage pipelined processor. The successive five stages of this pipeline are denoted by IF, DEC, EXE, MEM, WB.

```
loop: lw $t0, 0($s1)  # $t0=array element
       addu $t0, $t0, $s2  # add scalar in $s2
       sw $t0, 0($s1)   # store result
       addi $s1, $s1, -4 # decrement pointer
       bne $s1, $0, loop # branch if $s1 != 0
```

where \( \$s1 \) stores a base memory address of a 32-bit integer array with size of 9, and \( \$s2 \) stores a number added to each element of the array.

Note that forwarding and instruction re-ordering can be used to resolve data hazards and control hazards.

2.1 For the first iteration on a single pipelined processor, draw the pipeline execution diagram, such that the minimum amount of stalls is inserted in the pipelined execution. Also indicates what type of forwarding (ALU-ALU, ALU-MEM, MEM-ALU, MEM-MEM) it is if forwarding is used.
The sequence is right [4] and 9 clock cycles in total are used [2].

1. MEM-ALU forwarding [2]
2. ALU-ALU forwarding (It is not necessary)
3. ALU-MEM forwarding [2]

2.2 Apply loop unrolling for 3 times and write down your MIPS code.
2.3 Now we would like to schedule your loop unrolled MIPS code on a two-issue, 5-stage pipelined processor with the following constraints:

- the first issue deals only with ALU or branch instructions,
- the second issue deals only with data transfer (loads and stores),
- we wish to avoid pipeline stalls.

Propose a schedule for the first iteration so as to achieve a high IPC (instructions per cycles).
<table>
<thead>
<tr>
<th>ALU or branch</th>
<th>Data transfer</th>
<th>Clock Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addu [1]</td>
<td>lw</td>
<td>3</td>
</tr>
<tr>
<td>addu</td>
<td>sw [1]</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>