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>jr $s0</td>
<td>go to $s0</td>
</tr>
<tr>
<td>jump register</td>
<td>ja 250</td>
<td>go to 1000; $ra=PC+4</td>
</tr>
</tbody>
</table>

### Synchronization in MIPS

- **Load linked:** `ll rt, off(rs)`
  
  Load `rt` with the contents at `Mem[off+rs]` and reserves the memory address `off+rs` by storing it in a special link register (`R_link`).

- **Store conditional:** `sc rt, off(rs)`
  
  Check if the reservation of the memory address is valid in the link register. If so, the contents of `rt` is written to `Mem[off+rs]` and `rt` is set to 1; otherwise no memory store is performed and 0 is written into `rt`.
  
  - Returns 1 (success) if location has not changed since the `ll`
  - Returns 0 (failure) if location has changed

- Note that `sc` clobbers the register value being stored (`rt`)! Need to have a copy elsewhere if you plan on repeating on failure or using value later
MESI Protocol Cheat Sheet

MESI Local Read Hit:
- The line must be in one of MES
- This must be the correct local value (if M it must have been modified locally)
- Simply return value
- No state change

MESI Local Read Miss: A core makes read request to main memory upon a read miss: detailed action depends on copies in other cores
- **Case 1**: One cache has an E copy
  - The snooping cache puts the copy value on the bus
  - The memory access is abandoned
  - The local core caches the value
  - Both lines are set to S
- **Case 2**: No other copy in caches
  - The core waits for a memory response
  - The value is stored in the cache and marked E
- **Case 3**: Several caches have a copy (S)
  - One snooping cache puts the copy value on the bus (arbitrated)
  - The memory access is abandoned
  - The local core caches the value and sets the tag to S
  - Other copies remain S
- **Case 4**: One cache has M (modified) copy
  - The snooping cache puts its copy of the value on the bus
  - The memory access is abandoned
  - The local core caches the value and sets the tag to S
  - The source (M) value is copied back to memory
  - The source value changes its tag from M to S

MESI Local Write Hit: Line must be one of MES
- **M**
  - line is exclusive and already ‘dirty’
  - Update local cache value
  - no state change
- **E**
  - Update local cache value
  - Change E to M
- **S**
  - Core broadcasts an invalidate on bus
  - Snooping cores with an S copy change S to I
  - The local cache value is updated
– The local state changes from S to M

**MESI Local Write Miss:** Detailed action depends on copies in other cores
- **Case 1:** No other copies
  – Local copy state set to M
- **Case 2:** Other copies, either one in state E or more in state S
  – Value read from memory to local cache - bus transaction marked RWITM (read with intent to modify)
  – The snooping cores see this and set their tags to I
  – The local copy is updated and sets the tag to M
- **Case 3:** Another copy in state M
  – Core issues bus transaction marked RWITM
  – The snooping core sees this
    - Blocks the RWITM request
    - Takes control of the bus
    - Writes back its copy to memory
    - Sets its copy state to I
  – The original local core re-issues RWITM request
  – This is now simply a no-copy case
    - Value read from memory to local cache
    - Local copy value updated
    - Local copy state set to M

**Exercise 1.** [40 points] This exercise is about writing synchronization mechanisms in MIPS.

1.1 Write a sequence of MIPS instructions exchanging the contents of a register and a memory word. To be precise, and using the RTL language, write a sequence of MIPS instructions implementing the following: $s4 \leftrightarrow \text{Mem}[s1]$

See Slide 19 of Set 7.2.

1.2 Suppose now that one needs to implement the following the following: \text{Mem}[s4] \leftrightarrow \text{Mem}[s1] in the context of a multi-threaded program where different threads could access Mem[$s4$] and Mem[$s1$]. How would you approach this problem. A sequence of MIPS instructions is not required but a clear explanation is.
The swap $\text{Mem}[s4] \leftrightarrow \text{Mem}[s1]$ should be executed in a critical section. Hence one needs a locking mechanism, similar to the one of Slide 21 of Set 7.2

**Exercise 2.** [60 points] In the following we will assume that we are working with a bus-based multiprocessor machine with four processors. Each processor has its own direct-mapped cache. Each cache is 32 Bytes in size, organized in four sets. The memory in the machine is byte-addressable and the address width of the machine is eight bit.

The machine uses the MESI protocol to maintain cache coherence. In the following $R_P(a)$ means that processor $P$ reads one byte from the address $a$, while $W_P(a) = v$ means processor $P$ writes $v$ to the memory location $a$. Addresses are represented in binary. Initially all cache lines are invalid and all memory locations contain 0. Assume that
- a cache hit takes one cycle (both for read and write operations),
- transferring a cache line (either fetch or write back) takes 8 cycles,
- a request to use a cache line exclusively (transition I to M) takes 2 cycles,
- a request to upgrade to exclusive use (transition S to M) takes 2 cycles.

1.1 Show the transitions of cache lines in the table below. To be precise, at the slot which is at the intersection of Row $R_i(a)$ (respectively $W_i(a) = v$) and Column $P_j$ give the state of the block (= cache line) at $a$ in the cache-memory of Processor $P_j$. **Note:** for that question, you can ignore the column called LATENCY.

<table>
<thead>
<tr>
<th>ACTION</th>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>LATENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_0(00000000)$</td>
<td>E</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>8</td>
</tr>
<tr>
<td>$R_1(00000010)$</td>
<td>S</td>
<td>S</td>
<td>I</td>
<td>I</td>
<td>8</td>
</tr>
<tr>
<td>$R_2(00000011)$</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>I</td>
<td>8</td>
</tr>
<tr>
<td>$W_3(00000100) = 1$</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>M</td>
<td>2 + 8</td>
</tr>
<tr>
<td>$W_3(00000101) = 2$</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>M</td>
<td>1</td>
</tr>
<tr>
<td>$R_0(00000010)$</td>
<td>S</td>
<td>I</td>
<td>I</td>
<td>S</td>
<td>8 + 8, or 8</td>
</tr>
<tr>
<td>$W_0(00000101) = 3$</td>
<td>M</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>2+1</td>
</tr>
</tbody>
</table>

On the second last row, it is reasonable to assume that the two actions are concurrent. So $8 + 8$ is an upper bound and $8$ is a lower bound.

1.2 What is the total number of clock-cycles required to execute the sequence of actions in the above table. **Note:** for that question, you can use the column called LATENCY.

54 or 46.