1. The semantics of the instruction in question are given as follows -
\[ \text{Mem[offset + Reg[rs]] = Reg[rt]} \]
Also the offset field is encoded as a separate 32 bit word after the instruction word holding the opcode, rs and rt fields. Hence we need to read the address from the PC.

(i) \( \text{PC} \rightarrow \text{MAR}; \text{PC} \rightarrow \text{AIR} \) // Fetch Phase
(ii) \( \text{Mem[MAR]} \rightarrow \text{MDR}; \text{AIR} + 4 \rightarrow \text{PC} \)
(iii) \( \text{MDR} \rightarrow \text{IR} \)
(iv) \( \text{PC} \rightarrow \text{MAR} \) // Execute Phase
(v) \( \text{Mem[MAR]} \rightarrow \text{MDR}; \text{RF[rs]} \rightarrow \text{AIR} \)
(vi) \( \text{MDR} + \text{AIR} \rightarrow \text{MAR} \)
(vii) \( \text{RF[rt]} \rightarrow \text{MDR}; \text{PC} \rightarrow \text{AIR} \)
(viii) \( \text{MDR} \rightarrow \text{Mem[MAR]}; \text{AIR} + 4 \rightarrow \text{PC} \)

2. The instruction to be considered is ADD (ADDR). The semantics are specified as -
\[ \text{ACC} = \text{ACC} + \text{Mem[ADDR]} \]
The instruction occupies two memory locations, with the ADDR specified as a separate 32 bit word after the instruction word holding the opcode. The MAL routine is specified below -

(i) \( \text{PC} \rightarrow \text{MAR} \) // Fetch Phase
(ii) \( \text{Mem[MAR]} \rightarrow \text{MDR}; \text{PC} + \text{len} \rightarrow \text{PC} \)
(iii) \( \text{MDR} \rightarrow \text{IR} \)
(iv) \( \text{PC} \rightarrow \text{MAR} \) // Execute Phase
(v) \( \text{Mem[MAR]} \rightarrow \text{MDR}; \text{PC} + \text{len} \rightarrow \text{PC} \) // MDR contains ADDR
(vi) \( \text{MDR} \rightarrow \text{MAR} \)
(vii) \( \text{Mem[MAR]} \rightarrow \text{MDR} \) // MDR contains value
(viii) \( \text{MDR} + \text{ACC} \rightarrow \text{ACC} \)

3. First of all, we need to get the semantics right for the instruction in question. The instruction uses memory indirect addressing. The semantics of the instruction are –
\[ \text{Mem[Mem[offset + Reg[rs]]]} = \text{Reg[rt]} \]
The MAL routine is specified as below –

(i) \( \text{PC} \rightarrow \text{MDR}; \text{PC} + 4 \rightarrow \text{PC} \) // Fetch Phase
(ii) \( \text{MDR} \rightarrow \text{MAR} \);
(iii) \( \text{Mem[MAR]} \rightarrow \text{MDR} \);
(iv) \( \text{MDR} \rightarrow \text{MAR} \);
(v) \( \text{MAR} \rightarrow \text{IR} \);
(vi) \( \text{RF[rs]} \rightarrow \text{AIR} \) // Execute Phase
(vii) \( \text{AIR} + \text{SE(offset)} \rightarrow \text{AOR} \)
(viii) \( \text{AOR} \rightarrow \text{MDR} \)
(ix) \( \text{MDR} \rightarrow \text{MAR} \) // MAR = Reg[rs] + offset
(x) \( \text{Mem[MAR]} \rightarrow \text{MDR} \) // Memory Indirect Addressing
(xi) \( \text{MDR} \rightarrow \text{MAR} \) // MAR = Mem[Reg[rs] + offset]
(xii) Reg[rt] ➔ MDR  
(xiii) MDR ➔ Mem[MAR]

4. (i) A direct mapped cache is a 1-way set associative cache in which each block corresponds to one set. So the 16-bit address is split up as follows –

<table>
<thead>
<tr>
<th>2 bits</th>
<th>9 bits</th>
<th>5 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Cache Block/Set No.</td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

The address specified is \(0xFFBC = 0b1111101111111100\), the set number is the 9 bits in the middle, which is \(0b111011111\). This is block number \(479\). The tag is 3.

(ii) 8-way set associative cache corresponds to the fact that 8 blocks are combined into one set. Hence the number of sets in the cache is \(512/8 = 64\) sets. The 16-bit address is split up as follows -

The address specified is \(0xFFBC = 0b1111101111111100\), the set number is the 6 bits in the middle, which is \(0b11111\). This is set number \(31\). The tag is also 31 in this case.

<table>
<thead>
<tr>
<th>5 bits</th>
<th>6 bits</th>
<th>5 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Cache Set No.</td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

(iii) A fully set associative cache has only one set. The 16-bit address is split up as follows -

<table>
<thead>
<tr>
<th>11 bits</th>
<th>5 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

Since there is only one set, the block belongs to set number 0. The tag is the first 11 bits which is \(0b11111011111\) which is 2015.

5. 2-way set associative cache with a total of 2 sets and a block size of 16 bytes. There are 4 blocks in the cache and the 12-bit cache address is split up as follows –

<table>
<thead>
<tr>
<th>7 bits</th>
<th>1 bit</th>
<th>4 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Cache Set No.</td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

The CPU accesses the following addresses –

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Set No.</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>C88H</td>
<td>64H</td>
<td>0</td>
<td>Miss</td>
</tr>
<tr>
<td>Set Number</td>
<td>Block 0 Tag</td>
<td>Block 1 Tag</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3CH</td>
<td>64H</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>39H</td>
<td>3BH</td>
<td></td>
</tr>
</tbody>
</table>

6. In a direct-path based data path instead of using one or more buses, point-to-point connections or direct paths are provided between each pair of components that transfer data. This approach allows many register transfers to take place simultaneously, thereby permitting multiple MAL operations to be executed in parallel. This reduces the execution time at the expense of additional micro-architectural registers and additional wires. Besides, the use of direct paths probably makes the data path’s functioning easier to visualize, because each interconnection is now used for a specific transfer as opposed to a bus based data path where a bus transaction changes from clock cycle to clock cycle. However, the direct-path based data path is even more tailored for a specific ISA.

7. We need a hierarchical memory organization due to three key relationships, which does not permit having a single memory structure, which is fast, inexpensive and implements the entire address space –
   - Larger a memory structure, slower its operation
   - Larger a memory structure, greater its cost
   - Faster a memory structure, greater its cost

When the entire address space is implemented using a simple memory structure, a memory access cannot be completed in 1 clock cycle, as a large memory cannot be fast and inexpensive enough to achieve that. To alleviate this problem, we use the concept of memory hierarchy in which the entire address space is split into multiple levels of memory with different speeds and sizes. The fastest memories are more expensive per bit and are smaller in size than the slower and larger ones. The goal is to achieve performance close to that of the fastest memory and a cost per bit close to that of least...
expensive memory. This succeeds because of the existence of temporal and spatial locality, which indicate that an item accessed now will be accessed again in future (locality in time) and a nearby item will be accessed in future (locality in space).

8 (i) Since the page size is 4 KB, the number of bits for page offset are \( \log_2(4K) = 12 \) bits. 
(ii) The virtual memory address space has \( 2^{22} \) bytes. The number of virtual pages is specified as – Virtual Address Space Size/Page Size = \( 2^{22}/2^{12} = 2^{10} = 1024 \) pages. 
(iii) The number of physical pages = Size of physical memory / Page Size = \( 16K/4K = 4 \) pages. 
(iv) The page table has as many entries as the number of virtual pages. Hence number of page table entries = 1024.