Question 1: (7+4+4.5+4.5 = 20 points) For this question, refer to the following pipeline architecture.

![Pipeline Architecture Diagram]

a) Consider the following program segment (4 instructions)

```plaintext
add    r4, r5, r6  
beq    r9, r10, 800  
sw     r7, 100(r8)  
lw      r3, 8(r10)
```

Assume that this program is executing and at the end of some cycle, the “add” instruction is in the MEM/WB buffer. What are the values stored in the following buffers at that time – you should use X if you do not care about the value being 0 or 1.

- **ID/EX.RegWrite:** 0
- **EX/MEM.RegWrite:** 0
- **MEM/WB.RegWrite:** 1
- **ID/EX.MemtoReg:** X
- **EX/MEM.MemtoReg:** X
- **MEM/WB.MemtoReg:** 1
- **ID/EX.MemWrite:** 1
- **EX/MEM.MemWrite:** 0
- **ID/EX.MemRead:** 0(X)
- **EX/MEM.MemRead:** 0(X)
- **ID/EX.Branch:** 0
- **EX/MEM.Branch:** 1
- **ID/EX.RegDst:** X
- **ID/EX.ALUSrc:** 1
b) Now assume that the branch condition of the “beq” instruction, which is in the EX/MEM stage at that time, is true, what actions (in terms of changing the values stored in the inter-stage buffers) should be taken by the architecture during the following cycle?

Note that the branch is resolved while the instruction is in the MEM stage (see slide 45)

EX/MEM.RegWrite = EX/MEM.MemWrite = EX/MEM.MemRead = 0
ID/EX.RegWrite = ID/EX.MemWrite = EX/MEM.MemRead = 0
Flush IF/ID --> IF/ID.opcode = 0

Note that EX/MEM.MemRead is set to zero (rather than x) since we do not want to unnecessarily read from the cache – this consumes energy and may cause a cache miss and unnecessarily evict data from the cache.

c) If 20% of the instructions executing on this architecture are branch instructions, 30% are lw/sw instructions and 50% are ALU instructions, what would be the CPI for the architecture assuming that 70% of the branches are taken. Ignore the effect of other hazards.

Since the branch is resolved in the MEM stage, the penalty for each taken branch is 3. Hence

\[ \text{CPI} = 1 + 0.2 \times 0.7 \times 3 = 1.42 \]

d) Assume that a branch predictor is added to the above architecture so that the branch target is predicted while the instruction is in the IF stage. If a correct prediction can be made 80% of the time about the address of the instruction following a branch instruction, while in the remaining 20% of the time a wrong prediction is made, compute the CPI under the same conditions as part c.

\[ 1 + 0.2 \times 0.2 \times 3 = 1.12 \]
Question 2 (6+6+3=15 points): Assume that the following sequence of instructions execute on the 5-stage pipeline (IF, ID, EX, MEM, WB):

```assembly
add    r5, r2, r1
lw     r3, 4(r5) /* depends on r5 in add */
lw     r2, 0(r2)
sub    r1, r5, r6 /* depends on r5 in add */
or     r3, r6, r3 /* depends on r3 in lw */
sw     r3, 0(r5) /* depends on r3 in or */
```

(a) If there is **no forwarding or hazard detection units** implemented in your pipeline, indicate (on the above code segment) where and how many no-ops would you insert to ensure correct execution without changing the order of the instructions.

(b) If there is **no forwarding or hazard detection units** implemented in your pipeline, rewrite the above code after reordering the instructions (while still guaranteeing correctness) to minimize the number of added no-ops. Include the no-ops that are needed for correctness.

```assembly
add    r5, r2, r1
lw     r2, 0(r2)
sub    r1, r5, r6 /* depends on r5 in add */
lw     r3, 4(r5)
or     r3, r6, r3 /* depends on r3 in lw */
sw     r3, 0(r5) /* depends on r3 in or */
```

By reordering the instructions, we cannot reduce the number of no-op below 4

(c) Now, if your hardware **has forwarding and hazard detection (stalling) units**, how many times will the pipeline stall while executing the original code (before reordering and without no-ops)?

Circle the correct answer:

- 0 times
In this question, you will explore changing the MIPS ISA so that lw/sw instructions do not use an immediate constant. That is, the memory address is found in the register without the capability of adding a constant (this is called register indirect addressing). With this modification, it is possible to have a 4-stage pipeline by merging the EX and MEM stages into one stage (call it EX+M), as shown in the figure below:

(a) Assume that it takes 100ps to read from the register file, 100ps to write into the register file and 220ps to fetch or store into memory (instruction or data). Assume also that the adder and ALU delays are 150ps and 230ps, respectively and ignore all other delays. What is the minimum cycle time for this 4-stage pipeline?

(b) 230

(c) To compare the performance of the 4-stage pipeline with the original 5-stage pipeline, we assume that both have the same cycle time, use forwarding and resolve branches in the third stage (EX or EX+M). The advantage of the 4-stage pipeline is that forwarding completely eliminates stalling due to data hazards. Its disadvantage is that each instruction of the form “lw r1, I(r2)” or “sw rl, I(r2)” has to be replaced by two instructions, “addi r2, r2, I” followed by “lw rl, r2” or “sw rl, r2”. This is found to increase the number of executed instructions by 5%.

a. If the probability of a load-use data hazard in the 5-stage pipeline is 3% (the probability that an instruction is a lw instruction that loads into a register, r, and that this lw instruction is immediately followed by an instruction which reads from r) what would be the CPI of the 4-stage pipeline if the CPI of the 5-stage pipeline is 1.53?

\[
CPI = 1.53 - 0.03 \times 1 = 1.5
\]

b. Assuming the same clock cycle time, which pipeline would be more efficient? – should quantitatively support your answer

Clock cycles to complete for 5 stage = CPI * # of instructions = 1.53 X
Clock cycles to complete for 4 stage = CPI * # of instructions = 1.5 * 1.05X = 1.575X

Hence the 5-stage pipeline is more efficient
**Question 4 (3+6+3+3=15 points):** Consider the following loop which sums up the elements of an array which is stored in memory starting at address 10000. The number of elements of the array is stored in register $s1:

```
add    $t2, $zero, $zero
L: lw     $t1, 10000($s1)
addi  $s1, $s1, -4
add   $t2, $t2, $t1
bne   $s1, $zero, L
```

(a) While the loop is executing, what is the L1 data cache miss rate assuming:

1) A direct mapped cache with block size of 16 Bytes (4 words) ___25%___

2) A 4-way set associative cache with block size of 8 bytes (2 words) ___50%___

3) A 2-way set associative cache with block size of 32 Bytes (8 words) ___12.5%___

(b) Show the loop after unrolling it twice (assuming that it will execute an even number of times)

```
add    $t2, $zero, $zero
L: lw     $t1, 10000($s1)
lw     $t3, 9996($s1)
addi  $s1, $s1, -8
add   $t2, $t2, $t1
add   $t2, $t2, $t3
bne   $s1, $zero, L
```

(c) Assume a 5-stage pipeline with forwarding and hazard detection units and a 1-bit branch prediction. Assume also that the L1 instruction cache miss rate is 0%, the L1 data cache miss rate is 10% and the L1 cache miss penalty is 100 cycles. What is the CPI resulting from executing the original loop a very large number of times?

\[
\text{CPI} = \text{CPI with perfect caches} + 0.25 \times 0.1 \times 100 = \text{CPI with perfect caches} + 2.5
\]

Note that one instruction out of the 4 instructions in the loop (the lw) will access the data cache, and hence only 25% of the instruction will access the cache during execution.

(d) If an L2 with a hit time of 10 cycles is added to the architecture, what is the CPI resulting from executing the original loop a very large number of times assuming that 50% of the requests that are missed in the L1 cache are found in the L2 cache.

\[
\text{CPI} = \text{CPI with perfect caches} + 0.25 \times 0.1 \times (10 + 0.5 \times 100) = \text{CPI with perfect caches} + 1.5
\]
**Question 5 (7.5+7.5=15 points):** For each of the following cache organizations, consider the corresponding sequence of memory references (given as word addresses) and indicate for each reference whether it is a hit or a miss (using the space between the parentheses). Also, show the content of the cache after each memory reference using [tag, M(address), ...] to describe the content of each entry. For example [4, M(46)] indicates that the entry contains tag=4 and the data from memory location 46. Similarly, [4, M(46),M(47)] indicates that the entry contains a block of two words from memory locations 46 and 47. Indicate that an entry, E1, is replaced by another entry, E2, by crossing E1 and writing E2 next to it.

(b) A 16-words direct mapped cache with block size = 2 words

<table>
<thead>
<tr>
<th>Index</th>
<th>Content of cache (ordered from oldest to most recent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[2, M(32),M(33)]</td>
</tr>
<tr>
<td>1</td>
<td>[2, M(34),M(35)] [0, M(2),M(3)]</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[1, M(26),M(27)] [2, M(42),M(43)] [1, M(26),M(27)]</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

The sequence of memory references (given as word addresses):
26 ( m ), 34( m ), 33( m ), 27( h ), 42( m ), 35( h ), 3( m ), 27( m )

(c) A 16-words 2-way set associative cache, with block size = 1 word and LRU replacement.

<table>
<thead>
<tr>
<th>Index</th>
<th>Way 1</th>
<th>Way 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[3,M(24)]</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>[0,M(2)]</td>
</tr>
<tr>
<td>2</td>
<td>[4,M(34)] [1,M(10)]</td>
<td>[3,M(27)]</td>
</tr>
<tr>
<td>3</td>
<td>[0,M(3)]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The sequence of memory references (given as word addresses):
3 ( m ), 24( m ), 34 ( m ), 2( m ), 27( m ), 3( h ), 2( h ), 10( m )
**Question 6: (10 points)**
Compute the number of tag bits required to implement each of the three cache configurations specified below. Assume that each memory word is 32-bit long and the memory is byte addressable with 32-bits addresses.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache size</td>
<td>1 MB</td>
<td>1 MB</td>
<td>1 MB</td>
</tr>
<tr>
<td>Block size</td>
<td>one word (4 Bytes)</td>
<td>8 words (32 Bytes)</td>
<td>2 word (8 Bytes)</td>
</tr>
<tr>
<td>Associativity</td>
<td>direct mapped</td>
<td>direct mapped</td>
<td>4-way</td>
</tr>
<tr>
<td>Number of blocks in cache</td>
<td>$2^{18} = 256 \text{ K}$</td>
<td>$2^{15} = 32 \text{ K}$</td>
<td>$2^{17} = 128 \text{ K}$</td>
</tr>
<tr>
<td>Tag size (in bits)</td>
<td>12</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Number of bits in tag array</td>
<td>$12 \times 256 \text{ K}$</td>
<td>$12 \times 256 \text{ K}$</td>
<td>$14 \times 128 \text{ K}$</td>
</tr>
</tbody>
</table>

**Question 7: (10 points)**
For each of the following statements indicate if it is true or false:

1) Cache block size should be a power of 2 ( x ) true or ( ) false
2) Cache associativity should be an even number ( ) true or ( x ) false
3) In a “write back cache”, a block is written back to memory on every write ( ) true or ( x ) false
4) With write buffers, the process never has to stall on a write ( ) true or ( x ) false
5) DRAM memory cells have to be periodically refreshed ( x ) true or ( ) false
6) More forwarding paths are needed in superscalers than in single pipelines ( x ) true or ( ) false
7) 2-bit branch predictors reduce collisions in the BTB over 1-bit predictors ( ) true or ( x ) false
8) Larger associativity reduces compulsory misses in instruction caches ( ) true or ( x ) false
9) Interleaved memory is slower to access than non-interleaved memory ( ) true or ( x ) false
10) A precise exception terminates all the instructions in the pipeline ( ) true or ( x ) false