IIT JODHPUR

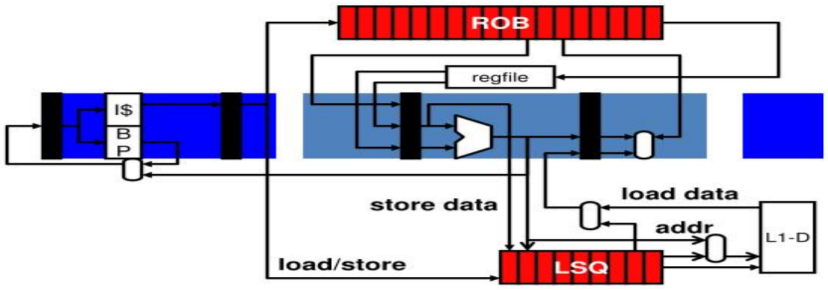
Major Examination: EEL1530 Processor Design-Nov'25 (OPEN-BOOK)

Guidelines (Total time: 180 minutes, Maximum Marks: 45):

- Please read the question paper very carefully. **NO clarification is required in any question. In case of any doubt, assume whatever you wish to and state that in your answer.**
- Usage of internet/any LLM tool is NOT allowed. Answer to-the-point ONLY in legible handwriting.

---

1. You are an intern in a processor design company. You are trying to optimize the out-of-order (OoO) micro-architecture by load-store queue (LSQ) incorporation into it. You come up with a LSQ implementation with the following specifics (a generalized figure of LSQ is shown below):



   - The LSQ can hold up to 6 entries.
   - The LSQ performs store-to-load forwarding if a prior (older) store to the same address exists in the LSQ and has its data available.
   - When a load executes and a matching older store has the same address in the LSQ with data ready, the load forwards from that store rather than reading memory.

   Consider the following instructions in the below order:

```
S1: ST [R1 + 4], R2
L1: LD R5, [R1 + 8]
L2: LD R6, [R1 + 4]
S2: ST [R3 + 200], R4
L3: LD R7, [R3 + 200]
S3: ST [R1 + R2], R3
L4: LD R8, [R1 + 8]
```

   - Show the LSQ contents (entry list with instruction tag, type (L/S), effective address if known, data-ready flag for stores, and outstanding loads waiting status) after all 7 instructions (as shown above) have been issued (before any memory responses). [3]
   - Assume that Registers: R1 = 100, R2 = 8, R3 = 0, R4 = 200 and Mem[100] = 10, Mem[104] = 20, Mem[108] = 30, Mem[200] = 500, give the final values in registers R5, R6, R7, R8 assuming the stores commit to memory in program order and memory is updated accordingly. (If a load forwarded from a store, use forwarded value.) [2]
   - Suppose the LSQ capacity is only 3 entries instead of 6. Explain what could go wrong when issuing this instruction sequence and how the CPU should handle it. [4]

2. Consider a processor micro-architecture with seven pipeline stages: IF – Instruction Fetch, ID – Instruction Decode & register fetch, EX1 – ALU Stage 1, EX2 – ALU Stage 2, MEM – Data Memory Access, WB1 – Write-back Stage 1, WB2 – Write-back Stage 2. You are also provided additional details:

   - The pipeline is fully bypassed except that results written in WB2 are not available to a dependent instruction until the beginning of ID.

- Branches are resolved at the end of EX1. On a branch misprediction, the processor squashes all younger instructions.
- A data hazard stall occurs if an instruction in ID requires the result of an instruction that has not yet reached the end of EX1 (i.e., bypassing cannot supply the value).

```
I1:  R3 ← R1 - R2
I2:  R4 ← R3 + 5
I3:  R6 ← MEM[R4]
I4:  BEQ R6, R0, LABEL
I5:  R7 ← R6 - 1
I6:  LABEL: R8 ← R6 - R9
```

(a) For the below set of instructions, determine the total number of pipeline stall cycles due to: 1) Data hazards, and 2) Branch misprediction
(b) Compute the total number of cycles needed to execute I1–I6 (up to when I6 enters WB1)?
(c) What could be done to solve the problem of stalls as shown in (a)? [3 + 2 + 3]

3. In a deeptech startup, you are tasked with analysis of the processor micro-architecture in out-of-order style with the below details:

   - 2-wide dispatch, 4-wide issue, 3-wide commit
   - 64-entry ROB (Reorder Buffer)
   - Separate Load Queue (LQ) with 32 entries and Store Queue (SQ) with 32 entries
   - 16 reservation stations (RS) shared by all ALU instructions
   - Physical Register File (PRF) has 96 registers
   - Branch predictor accuracy: 90%
   - Load–store forwarding allowed
   - A load may execute speculatively if:
     - No older store with unknown address exists
     - If an older store has a known address and it matches, it must forward from the store
   - Branch misprediction penalty: 12 cycles
   - L1 hit latency = 4 cycles, L2 hit latency = 12 cycles, and DRAM miss penalty = 90 cycles
   - Multiplication takes 3 cycles, ALU instructions take 1 cycle, Loads need 4 cycles if data is in L1.
   - Assume all loads hit in L1 except I3 which misses L1 and L2 and goes to DRAM.

```
I1:  P8  ← P2 + P3
I2:  P9  ← P8 * P4
I3:  P10 ← MEM[P9 + 12]
I4:  MEM[P12] ← P10
I5:  P11 ← P10 + P5
I6:  BEQ  P11, P0, TARGET
I7:  P13 ← P11 * P6
I8:  P14 ← MEM[P7]
I9:  MEM[P7] ← P14
I10: P15 ← P14 + 1
```

(a) Compute the cycle at which each instruction commits. Include delays due to: Operand dependencies, reservation station availability, LSQ stalls, L1/L2/DRAM miss for I3, Branch resolution for I6, 3-cycle multiply latency and 12-cycle misprediction penalty (assume branch mispredicts) [3]

b) Explicitly show the following for above instruction window: Issue cycle, Execute cycles, ROB head [6]

4. For the micro-architecture and instruction window as shown in Question no.-3,

   a) With a proper explanation, determine which of the following resources is the primary bottleneck for this instruction window: Reservation Stations, ROB size, Load Queue / Store Queue, Physical Register File, Functional Unit bandwidth (ALU, MUL, LD/ST)? [3]
   b) If we add a second Load/Store Unit (that will allow us to perform two loads/stores per cycle), how does this change issue, execution, and commit timing? [4]

5. Consider the specifications as listed below for a multi-core CPU design that is also involving physical to virtual memory translation for enhanced performance:

   - Virtual Memory:
     48-bit virtual address, 4 KB page size, 8-way set-associative TLB, 128 TLB entries, TLB hit latency = 1 cycle, TLB miss penalty = 20 cycles (including page table walk)
   - L1 Data Cache:
     Size: 32 KB, Line size: 64 bytes, Associativity: 4-way, Hit latency: 2 cycles, Miss penalty to L2: 12 cycles
   - L2 Cache:
     Size: 256 KB, 8-way associative, Line size: 64 B, Hit latency = 10 cycles, Miss penalty to LLC = 40 cycles

   A junior engineer has written the below code where a loop accesses an array A[N] and a separate array B[N]. You want to check the performance of below code on the above micro-architecture:

```
for (i = 0; i < 8192; i += 64) {
    sum += A[i];      // Load A[i]
    sum += B[i];      // Load B[i]
}
```

   Answer the following questions in the regard to above scenario:

   - For a 48-bit virtual address, show the decomposition of following: Page offset bits, Virtual page number bits, L1 index bits. [3]
   - Assume the loop iterates over all indices i = 0, 64, 128, ... up to 8192 and given that L2 always hits, If both arrays are mapped to the same L1 set(s), compute: Compulsory misses, Conflict misses, Total L1 miss rate? [3]

6. You are trying to implement a dual-core version of the micro-architecture attempted in Ques. no. 1 with a shared memory configuration. Consider the below code fragment that is to be executed on this dual-core version (assume L1 is 32 KB, 64-byte lines, write-back) :

```
Thread T0 (Core 0):                Thread T1 (Core 1):
for (i = 0; i < 1024; i += 16)     {  for (j = 8; j < 1032; j += 16) {
   X[i] = X[i] + 1;                   X[j] = X[j] - 1;
}                                     }
```

   You decide to utilize MESI coherence protocol in order to manage the coherence transitions.
   (a) Identify exactly which indices of T0 and T1 fall in the same cache line. Note that 64-byte cache line can accommodate 16 integers. [2]
   (b) Show the exact MESI state transitions for each core? [4]