Indian Institute of Technology (IIT-Bombay)

AUTUMN Semester, 2025
COMPUTER SCIENCE AND ENGINEERING

CS230: Digital Logic Design + Computer Architecture

End Semester Examination

Full Marks: 40

Time allowed: 3.0 hours

INSTRUCTIONS: Do not make any extra assumptions other than those stated in the questions. The marks will only be given if a precise, sound, and clearly written explanation is provided that is understandable by the examiners. Any extra unnecessary writing, verbal arguments based on that, and unnecessary crib will result in a reduction of 5 marks.

Roll Number: _____

Name: _____

QN	1	2	3	4	5	6	7	8	9	10	11	Total
Marks												

1. Choose the correct statement(s). One or multiple statement(s) can be correct. **You will only be given marks if you choose all the correct and no incorrect options.** (2 marks)
   (a) A 1-bit branch predictor has 0% accuracy if the branch is almost always taken.
   (b) Branch prediction happens at the instruction fetch (IF) stage of the pipeline.
   (c) Branch delay slots can be filled by the compiler.
   (d) Pattern history tables (PHT) contain saturating counters.

   Solution: (b)(c)(d)

2. Choose the correct statement(s). One or multiple statement(s) can be correct. **You will only be given marks if you choose all the correct and no incorrect options.** (2 marks)

(a) Multi-level cache mainly improves the hit time.

(b) Multi-level cache mainly improves the miss penalty

(c) Dirty bits in caches are used to indicate that a cache block has been modified or not.

(d) Fully associative cache has the fastest hit time.

Solution: (b)(c)

3. Choose the correct statement(s). One or multiple statement(s) can be correct. **You will only be given marks if you choose all the correct and no incorrect options.** (2 marks)

(a) DRAM has a higher clock frequency than the processor.

(b) Each bank in DRAM has a row buffer.

(c) DRAM data is only refreshed during read.

(d) None of the above is correct.

Solution: (b)

4. Choose the correct statement(s). One or multiple statement(s) can be correct. **You will only be given marks if you choose all the correct and no incorrect options.** (2 marks)

(a) Register file is updated at the writeback stage of the pipeline.

(b) Register file is updated at the memory access stage of the pipeline.

(c) Deeper the pipeline, lesser the impact of a stall.

(d) Exception handling needs to flush the pipeline.

Solution: (a)(d)

5. Choose the correct statement(s). One or multiple statement(s) can be correct. **You will only be given marks if you choose all the correct and no incorrect options.** (2 marks)

(a) 2's complement representation has only one representation for zero.

(b) 1's complement representation has only one representation for zero.

(c) Only positive numbers can be represented with 1's complement representation.

(d) None of the above.

Solution: (a)

6. (**Super Easy Cache**) Suppose you are running a program with the following data access pattern. The pattern is executed only once. The given addresses are byte addresses, and addressing in the memory is consecutive. Assume the cache is initially empty for each of the following questions.

   0x00, 0x08, 0x10, 0x18, 0x20, 0x28

   All counting starts from index 0.

   (a) What would be the miss rate of a direct-mapped cache for this access pattern? Assume the direct-mapped cache is of 1 KB (1024 bytes) size and the cache-block size is 8 bytes. (2 marks)

   (b) For the given memory access pattern, which of the following would decrease the miss rate the most and why? Cache capacity is kept constant (i.e., 1 KB).

      i Increasing the associativity to 2 (i.e., 2-way set associative cache). (1.5 marks)

      ii Increasing the block size to 16 bytes (for the direct-mapped cache). (1.5 marks)

      You have to write what happens to the miss rate (with a concrete miss rate value) for each case and then compare.

Solution: (a) Every accessed byte is 8 bytes apart, so each byte will belong to a separate cache block (since block size is 8 bytes). Therefore, each of the accesses will result in a compulsory miss. The miss-rate will be $\frac{6}{6} = 100\%$.
(b.i) Since each accessed address belongs to a separate cache block, associativity will not change the miss rate here. It will still remain 100%. (b.ii) In this case, the new cache blocks will be $(0x00, 0x08)$, $(0x10, 0x08)$, etc. Therefore, the locality will help, and there will be 3 compulsory misses and 3 hits. The miss rate will be 50%.

Grading: (a) 0 (if unrelated), 1(if at least identifies that the misses are compulsory misses), 2 (if 100% is written).
b.i) If 100% is written give 1.5. If identifies that associativity will not change the miss rate but calculation is wrong; give 0.5. Give 0 in all other cases.
b.ii) If identifies that the locality will improve the miss rate, give 0.5. Give full marks (1.5) if the miss rate calculation is correct. Give 0 in all other cases.

7. (**Combinational Circuits**) This question is on combinational circuits:

   (a) Consider the Boolean function $F(A, B, C) = \sum(1, 3, 5, 6)$. Implement this function with **one** 4×1 MUX, where A **must not** be connected to the select line of the MUX. You can **only** use NOT gates along with the 4×1 MUX. (2 marks)

   (b) you are supplied with only one NOT gate and an unlimited number of AND and OR gates to realize the circuit $T = w'x + x'y + xz'$. Only uncomplemented variables can be used as inputs. Is it possible to realize T with such restrictions. Explain (3 marks)

Solution: (a) Directly from slides.
(b) The Product-of-Sum form of T is: $T = (x+y)(w'+x'+z')$. Applying De-Morgans we can write: $T = (x+y)(wxz)'$. Therefore, the circuit can be represented with a single NOT and unlimited number of AND and OR gates.

(a) Give 2 if the answer is correct (with or without diagram). Even if they connect a different set of input to the select line, give 1 mark. If other gates are used (**except NOT**) give 0.
(b) Give 3 marks while the expression is correct – that is respects the constraints mentioned above. If someone writes "possible" but without any correct answer, give 0.

8. (**AMAT and CPI**) Suppose you have a processor with:
   (a) Clock Rate = 200 MHz (5 ns per cycle)
   (b) Ideal CPI = 1.1 (including control stalls)
   (c) 50% arithmetic/logic, 30% load/store, 20% control 10% of data memory operations incur 50 cycles miss penalty.
   (d) 1% of instruction memory operations also incur 50 cycles miss penalty.

   Compute:

   i CPI with cache miss. (2.5 marks)
   ii (Normalized) AMAT. (2.5 marks)

Solution: See the slides. It has been anounced in the exam hall that hit time should be taken as 1 cycle. So do not give any marks if it has been assumed otherwise. For i) Give 2.5 marks if the steps are correct. Same for ii). Partial marks will only be given if the last step is wrong and the rest of the steps are correct.

9. (**Virtual Memory**) Consider the following two-dimensional array:

```
1 int A[100][100];
```

   Assume that:

   - `A[0][0]` is located at memory address 200 in a paged memory system.
   - The page size is 200 bytes (each integer occupies 1 byte).
   - A small process that manipulates the matrix resides entirely in page 0 (addresses 0–199). Thus, every instruction fetch occurs from page 0.
   - The system has three page frames: one permanently occupied by page 0 (the process), and two available for data pages.
   - The page-replacement policy is **LRU (Least Recently Used)**.
   - Both data page frames are initially empty.

- **Assume:** Arrays are contiguously stored in a row-major order.

Answer the following:

(a) How many page faults are generated by the following array-initialization loop?

```
for (int j = 0; j < 100; j++)
    for (int i = 0; i < 100; i++)
        A[i][j] = 0;
```

(b) How many page faults are generated by the same loop when the order of the indices is reversed?

```
for (int i = 0; i < 100; i++)
    for (int j = 0; j < 100; j++)
        A[i][j] = 0;
```

Solution: (a)

1. The array has total $100 \times 100 = 10000$ elements. Now each page contains 200 elements. So, so total $10000/200 = 50$ pages will be needed to store the array.
2. The first page 0 and the corresponding frame in the memory holds the instructions, and ,therefore, is not important here.
3. Now consider the access pattern of the inner loop. For each column j, all the corresponding rows (containing 100 elements each) will be accessed consecutively. Now, each page can hold 2 rows. So, the access pattern to each page will be $1, 1, 2, 2, 3, 3 \cdots 50, 50$. The first access to each page will be a page fault, and the second access will be a hit. So, there will be totally 50 page faults for the inner loop. Total number of page faults, therefore, will be $50 \times 100 = 5000$.

(b)

1. This time the inner loop traverses column-by-column on a fixed row.
2. Observe that each row is traversed only once.
3. Therefore, the access pattern for pages will be $1, 2, \cdots 50$, but by the outer loop. Also, since each page is used only once, no page reuse would be needed.
4. Overall, there will be 50 page faults needed in this case.

(a) Give 2.5 marks if the count is correct. Otherwise give: 0.5 for counting the pages correctly, 0.5 for identifying that each page holds 2 rows, 0.5 for the access pattern, 1 for correctly counting the page faults.
b) Grading pattern same as (a).

10. (**Pipelining**) The given function determines whether three arrays of `int`s are pointwise different or not. It returns `1` if they are, and `0`, otherwise. Each array has length n. The function runs on a 5-stage, in-order MIPS pipeline with forwarding. Forward branches are always **predicted not taken**, backward branches are always **predicted taken**, and branch targets and decisions are resolved in the (end of) `EX` stage (assume **no branch target buffer**, **no delayed branch**, and the pipeline is **initially empty**). How many cycles does the function take on arrays of length n for the **worst case input**? Show every step of your calculations clearly. The semantics of the instructions used is given in Table 1.

```
diff:# a0 = a, a1 = b, a2 = c, a3 is n
    li   $t3, 0              # t3 is i

loop:
    beq  $t3, $a3, false
    sll  $t4, $t3, 2
    add  $t0, $a0, $t4
    add  $t1, $a1, $t4
    add  $t2, $a2, $t4
    lw   $t0, 0($t0)
    lw   $t1, 0($t1)
    bne  $t0, $t1, true
    lw   $t2, 0($t2)
    bne  $t0, $t2, true
    bne  $t1, $t2, true
    add  $t3, $t3, 1
    j    loop

false:
    li   $v0, 0
    jr   $ra

true:
    li   $v0, 1
    jr   $ra
```

1. **Forward Branch:** Here the branch target address is *higher* than the address of the branch instruction.
2. **Backward Branch:** Here the branch target address is *lower* than the address of the branch instruction.
3. The "worst case input" is the one for which all the iterations of the loop are run completely and there is no intermediate exit from `loop`.

4. Consider that without hazards the CPI is 1, that is (without any data hazard related stall or control hazard related pipeline flush) each instruction is finished per clock cycle.

Instruction	Syntax	Meaning / Definition
`beq`	`beq $rs, $rt, label`	Branch if equal — if `$rs == $rt`, jump to `label`.
`bne`	`bne $rs, $rt, label`	Branch if not equal — if `$rs != $rt`, jump to `label`.
`sll`	`sll $rd, $rt, shamt`	Shift left logical — shift `$rt` left by `shamt` bits, filling with zeros.
`li`	`li $rd, imm`	Load immediate — load a constant value into a register (pseudo-instruction).
`lw`	`lw $rt, offset($rs)`	Load word — load a 32-bit word from memory at address `$rs + offset`.
`j`	`j label`	Jump — unconditionally jump to the given label (address).
`jr`	`jr $ra`	Jump register — jump to the address stored in `$ra` (often used for function return).
`add`	`add $rd, $rs, $rt`	Add — add contents of `$rs` and `$rt` (signed addition).
`sub`	`sub $rd, $rs, $rt`	Subtract — subtract contents of `$rt` from `$rs`.

Table 1: Common MIPS Instructions and Their Definitions

Solution: Worst case execution implies that the entire loop will run for n iterations (i.e., the arrays are indeed equal). Now consider the cases one by one.

1. **How many instructions will be executed?** 1 (initial) + $13n$ (in the loop body) + 1 (extra `beq` in line 5 for loop termination) + 2 (for return from the false branch) = $13n + 4$. Give 0.5
2. **Stalls due to data dependency:** The `bne` at line 12, and `bne` at line 14 will have experiance 1 cycle of stall due to data dependency with `lw` instructions. This dependency will be there throughout the loop iterations. So total $n + n = 2n$ cycle stalls. Give 1
3. **Flushes due to control hazards:** First, consider the fact that jump address calculation and branch condition calculation happens at the EX stage. Therefore, for a taken branch it will always incur 2 flushes (therefore, two cycles wasted), as the address calculation cannot be avoided. For a not taken branch, it will have zero flushes, if the prediction is also "not-taken". In case of misprediction, the conditional branch will also incur 2 flushes, and, ,therefore, 2 cycles of penalty. Now, consider:
   (a) The `j` instruction in line 17 will incur 2 cycles of penalty in each loop iteration as the branch target address will be known only at the end of the EX stage. Note that, although it is always predicted taken, and goes in only one direction for the worst case, there is no saving as the address calculation must happen. So total $2n$ cycles will be needed for flush. (2 marks for this.)

(b) The `bne` branches are all forward branches and (correctly) predicted not taken in all the cases. So they do not incur any flushes.

(c) The `beq` branch in line 5 runs total $n+1$ times. For the first n times the prediction is not taken and correct. So no penalty is incurred. However, the last iteration the branch is taken and ,therefore, incurs two flushes (2 cycles of penalty). 1 marks for this

(d) The `jr` instruction also incurs 2 cycles of flushes as it is also typically a backward branch which is taken. 0.5 for this

Overall, cycles needed for flushes $= 2n + 2 + 2 = 2n + 4$.

4. **First Instruction:** Since it is a 5-stage pipeline started empty, the first instruction will take 5 cycles. But we have already accounted for 1 cycle while counting the instructions. So here we shall account for 4 cycles.

Overall, we have: $(13n + 4) + 2n + (2n + 4) + 4 = 17n + 12$

11. (**Return of Combinational Circuits**) Consider a n-input AND gate (do not make an assumption that it is made from smaller gates; it is a monolithic gate with n input wires and one output wire). During fabrication, defects (also called *faults*) may occur making such a gate unusable. Let us consider that such faults are only limited to the wires, and can be of two types – a) a wire is permanently stuck to logic 0 (called *stuck-at-0* fault) or logic 1 (called *stuck-at-1* fault). We also consider that at a time, only one fault can be there in a circuit (i.e., the n input AND gate).

    Your task is to detect such faults (if any) in the n-input AND gate by providing suitable inputs to it and observing the outputs. Given an (n-bit) input, if the output differs from the desired output, then we conclude that there is a fault and this specific input detects the fault. We call such inputs as *test vectors*. Now answer the following questions:

    (a) How many faults are possible for the n-input AND gate in total (considering there can only be one fault at a time). (1 mark)

    (b) How many test vectors will be needed for the n-input AND gate so that we can declare it fault free after testing? You have to give the minimum count of test vectors with proper justification. Consider that there can only be one fault at a time. (4 marks)

    Hint: Some test vectors may detect multiple faults. You have to count such vectors only once.

Solution: (a) There are total $n + 1$ wires in an n-input AND gate. So total number of possible faults is $2(n + 1)$. It has been anounced multiple times in the exam hall that only one wire is faulted, with one type of fault in one faulty circuit instance. So no partial marks for this.
(b) Observe that, to detect a stuck-at-0 fault in a wire w, you have to give a test vector v which is supposed to set $w = 1$. Since w cannot be set to 1 (stuck-at-0), the computation

becomes faulty. v must also ensure that the effect of the fault reaches the output. Now let us first consider the output wire of the AND gate. If we consider a stuck-at-0 fault, then this fault can only be detected with a test vector $v = \{1\}^n$. Now, consider a stuck-at-0 fault in one of the (n) input wires. Observe that the same test vector $v = \{1\}^n$ detects a stuck-at-0 fault at all of these input wires. Next, consider a stuck-at-1 fault at the output wire. Any vector v which sets the output to 0 can be used to detect this fault, and there are $2^n - 1$ such vectors. However, if we want to detect stuck-at-1 fault for an input wire i, then we need a test vector v with its ith bit set to 0, and the rest of the bits set to 1. For each input wire, detecting stuck-at-1 faults require a separate test vector, and any of these vectors can be used to detect stuck-at-1 fault at the output wire. Therefore, for stuck-at-1 faults, we need total n vectors to declare the circuit fault free. For the stuck-at-0 faults we only need 1 vector. So, in total, we need $n + 1$ test vectors to declare the circuit fault-free. Give 3.5 if someone is able to find the stuck-at-0 fault equivalence condition and corresponding counting correct. Give 1.5 for the stuck-at-1 condition.