ECE411 Computer Organization and Design1
Mid-Term Exam 1
3/5/2024

NetID: _____
First Name:_____
Last Name:_____

Exam Guidelines:

1. This exam has **7 problems**. Make sure you have a complete exam before you begin **[19 pages + the cover page]**.
2. Write your NetID on every page in case pages become separated during grading.
3. You will have **3 hours** to complete this exam.
4. Write all of your answers on the exam itself. If you need more space to answer a given problem, continue on the back of the page, but clearly indicate that you have done so.
5. This exam is closed-book. You may use one sheet of notes. You may use a calculator.
6. **DO NOT** do anything that might be perceived as cheating. The minimum penalty will be a grade of zero.
7. Show all of your work on all problems. Correct answers that do not include work demonstrating how they were generated may not receive full credit, and answers that show no work cannot receive partial credit.
8. **Good luck**!

Question	Part 1	Part 2	Part 3	Part 4	Total
1	/15				/15
2	/5	/5	/4		/14
3	/4	/4			/8
4	/6	/2	/4		/12
5	/4	/3	/3		/10
6	/2	/10			/12
7	/3	/5	/5	/5	/18
Total					/89

NetID: _____

Question 1: Stanley-ISA (15 points)

Stanley owns a tumbler (steel vacuum insulated bottle) factory. Stanley is making tons of money, however his workforce isn't so fortunate. They demand better pay, so they are unionizing and are forming a picket line. Stanley does not want to fairly pay his workers, so he needs your help automating the production process by replacing humans with efficient microprocessors.

Shaping:

This step involves pressing two sheets of metal into cylinders such that they can be inserted inside each other and welded together with space between their walls. The microcontroller is responsible for communicating with the hydraulic press device, detecting faults, and discarding the faulty cylinders. To detect faults, the microcontroller compares the torque of the hydraulic press motor with a golden waveform (since Stanley noticed it changes when there is a defect).

Memory Map:

```
0x200000 - 0x2fffff: Torque samples from the hydraulic press.
0x400000 - 0x4fffff: Golden waveform torque samples
0x800000 - End of memory: Program data
```

Stanley plans on writing multiple methods that he will use in his code. One of these methods will compare the waveforms and decide whether a cylinder should be discarded. This is what his pseudocode looks like:

```
CHECK:
      // checking the torque, making sure it matches with golden waveform
      Iterate through memory at (0x200000 → 0x2fffff) and at (0x400000 → 0x4fffff)
            If (torque[i] != golden_waveform_torque[i])
                  Goto DISCARD, which discards the cylinder and restarts.

      // when the waveforms match up:
      Goto RESTART, which restarts for the next metal sheet.
```

NetID: _____

1) (15 points) You decide to design a Memory-Memory architecture processor. Define all the instructions in your ISA. Write the above program using your newly defined ISA.

- Instructions must have at most two source operands and one destination.
- Source or destination can be the PC register. Assume incrementing the PC by 1 takes you to the next instruction. Unless otherwise noted, PC <= PC + 1 after every instruction.
- Please add comments to your code!
- You do not need to fill all the rows in the table. You only need the instructions necessary to write your assembly code.
- Feel free to deviate from the structure of the pseudocode, as long as the task of CHECK is still fulfilled correctly.

Operation	**Destination**	**Source 1**	**Source 2**	**Description**
addi	memory location	memory location	immediate	M[Dest] <= M[src1] + imm
ldi	Memory location	Memory location	immediate	M[Dest] <= M[M[src1] + imm]
skip_ne		Memory location	Memory location	PC <= PC + (M[src1] != M[src2]) ? 2 : 1
jmp		immediate		PC <= PC + imm (jump to label)

NetID: _____

Example: addi dest, zero, GOLD_WAVE_START // M[dest] <= GOLD_WAVE_START+0

Program:
Macros:
PRESS_WAVE_START: 0x200000
GOLD_WAVE_START: 0x400000
/ / add more macros…

.text
CHECK:
// add code and labels here. Assume DISCARD and RESTART have already been defined.

```
Addi index, index, 1 ;                          // index <= index + 1
Skip_ne index, end ;                            // (index != end) ? continue : RESTART
Jmp RESTART
Ldi gold_val, index, GOLD_WAVE_START    // gold_val <= gold_wave[index]
Ldi press_val, index, PRESS_WAVE_START  // press_val <= press_wave[index]
Skip_ne gold_val, press_val ;                   // (gold_val != press_val) ? DISCARD : CHECK
Jmp CHECK
Jmp DISCARD
```

-4 if indirect memory access is not defined in ISA.
Up to -4 for poorly defined ISA.
Up to -4 for control flow flaws.
Up to -4 for incorrect style (using macros as data, etc).

.data
zero: 0 // this gets initialized to 0 when the program is loaded.
// add more data…
index: -1 // so that first iteration works
gold_val: 0
press_val: 0
end: 0x100000

NetID: _____

Question 2: Promising Pipeline (14 points)

Part 1: Boring Benchmarking

Udit is a hard-working employee at Raw-Cache Inc., a company that is currently designing its newest processor. His current design is a five-stage, RISC-V compliant pipeline with **no forwarding**. Because of this, the assembler will insert nop instructions between dependent instructions. In addition, it currently implements a static not-taken branch predictor. Flushes are handled **immediately**, so mispredicted instructions will be immediately followed by the correct next instruction. Udit has been tasked with benchmarking his interesting design and runs his comprehensive test suite:

```
begin:
xor x5, x5, x5
addi x1, x0, 4
addi x2, x0, 1

loop:
add x5, x5, x1
sub x1, x1, x2
bne x1, x0, loop

done: // filler instructions for branch not-taken
addi x3, x0, 1
addi x4, x0, 2

halt:
```

1) (5 points) After running the benchmark, Udit is surprised by the results and wants to confirm how accurate they are. Help him out by calculating the expected number of cycles that this processor should take to run the benchmark.
   Note the following:
   - Memory responds on the next cycle (like magic memory from the MPs)
   - The register file **is transparent**. Reading from a location that is currently being written to will read the new value, not the old one.
   - Source register values are read during the decode stage (ID)
   - Branches are resolved in the execute (EX) stage. In other words, the flush signal is not raised until a mispredicted branch reaches the execute (EX) stage

Space for part 1's answer

Approach 1: Assuming 2 mispredicted instructions are fetched, the program takes 36 cycles. There are three nops that must be introduced - one between the second addi and loop, and two between the sub and bne instructions.

Approach 2: Assuming 1 mispredicted instruction is fetched, the program takes 33 cycles. The same nops are present in the code.

Part 2: Design Debate

The Raw-Cache design manager, Mr. Wu, is disappointed by both Udit and his processor's performance. As such, Mr. Wu is debating between the following two options to speed things up:

A. Implement a forwarding path from the EX/MEM register to the EX stage
B. Replace the static not-taken predictor with a 2-bit saturating counter

Note: the saturating counter will initialize to 2'b01, corresponding to weakly not-taken. Additionally, implementing the forwarding path will allow the compiler to reduce the number of nop instructions if possible.

2) (5 points) Help Mr. Wu (and Udit) decide on an improvement by determining which option will speed up the benchmark the most, and by how much.

Approach 1 continued: forwarding path removes nops in the loop, keeps the one at the beginning. This reduces it down to 28 cycles. Branch predictor decisions are wrong, right, right, wrong. In total, this saves two cycles bringing it down to 34. Forwarding is better.

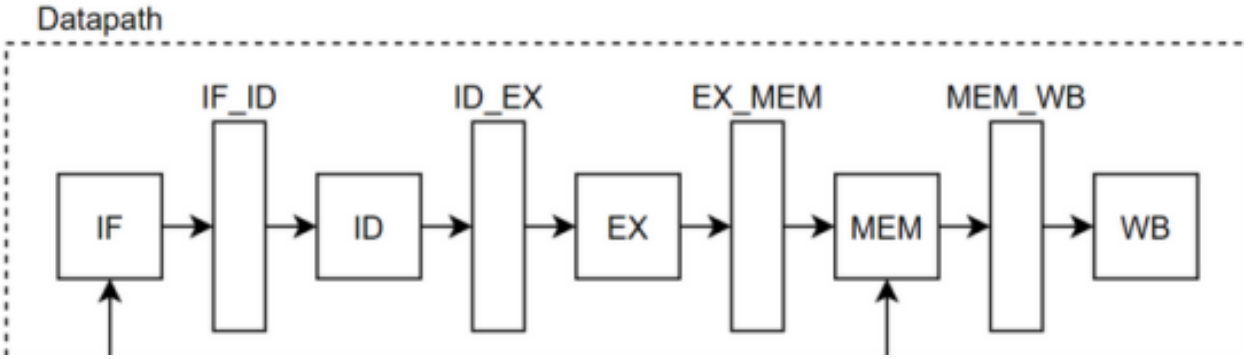
Approach 2 continued: forwarding calculation is similar, reducing the time to 25 cycles. The branch predictor will only save one cycle in total, bringing it down to 32 cycles. Forwarding is better.

NetID: _____

Part 3: Critical Calculations

3) (4 points) Given this diagram of the processor and following timings, find the minimum program execution time for the processor for each design decision.

Implementing the branch predictor will add 2ns delay to the IF stage
Implementing the forwarding path will add 1ns delay to the EX stage



The current stage timings are as follows:

Stage	Delay
IF	3ns
ID	2ns
EX	5ns
MEM	4ns
WB	3ns

Approach 1: Forwarding increases EX time to 6ns, so this is now our critical path. Execution time takes 6ns/cycle * 28 cycles = 128ns. The branch predictor increases IF to 5ns, on par with EX for critical path. Execution time takes 5ns/cycle * 34 cycles = 170ns.

Approach 2: Same calculations with different cycles. Forwarding is 6*25 = 150ns, branch prediction is 5*32 = 160ns.

NetID: _____

Question 3: All Data Goes to Disk (8 points)

You have been hired at Raw-Cache Inc. to design a new memory hierarchy for their latest specialty bitcoin mining processor. In the current design, the processor has a 32KB L1 cache with a 20ns access time and a 90% hit rate. The next level up is 256MB of DRAM, which has a 140ns access time and a hit rate of 65%. Lastly, there is hard-disk based main memory, which has a 512GB capacity, an access time of 1200ns, and always hits.

Ethan, one of the engineers at Raw-Cache Inc., suggests that the processor should use a revolutionary new memory technology called Q-MAR™ instead of DRAM. Q-MAR™ is cheaper so the team can afford to use up to 4GB of it, but it has a 400ns access time. Ethan believes that Q-MAR™ cannot outperform DRAM in terms of access time no matter the hit rate, but you disagree.

1) (4 points) What should the minimum hit rate of Q-MAR™ be for the new memory hierarchy to beat the average memory access time of the old DRAM-based one?

   20 + 0.1(400 + (1 - hit)(1200)) = 76ns
   hit = 0.866 = about 87%

Some CS major decides your bitcoin mining processor would be an excellent database server. The database is read-heavy, with 90% of the memory accesses being reads. The database's working set is also much larger than the L1 cache and DRAM.

2) (4 points) What changes would you make to the memory hierarchy to optimize it for this use case while keeping the average memory access time as low as possible? Assume that no levels of the hierarchy can be made larger, Q-MAR™ has a lower hit rate than DRAM, and hit rates for all levels are fixed.

   Put Q-MAR™ between DRAM and Disk

NetID: _____

Question 4: Gotta Cache ‘em All (12 points)

You are the lead developer of a new online video game called BitBeasts, where players can collect beasts. This game has a trading card system, in which players each have:

- 10 beast cards,
- A unique ID number,
- A *friend*, which is another player that they can trade cards with

You are working on the main database server containing information about each player's cards. A struct defines each player:

```
// assume the following structure is packed
// (i.e. no compiler inserted padding)
typedef struct player{
      uint32_t player_id;         // 32 bits
      uint8_t beast_cards[10];    // 10 * 8 bits
      struct player *friend_ptr;  // 32 bits
} player_t;
```

Several of these structs are located all over the server’s physical memory, and each player_t pointer is aligned to a cache line.

Online players can send information to your server to take action in the game. One of the functions in the player interface allows them to make a trade with their friend (without any confirmation from the friend), defined as the following:

```
void make_trade(player_t *player_ptr, int card_to_trade) {
      // swaps the cards at index card_to_trade in
      // the beast_cards arrays of both the
      // player provided and their friend
}
```

The function operates the following way:

1) Load the player’s card as an offset from player_ptr
2) Load the friend’s memory address
3) Load the friend’s card as an offset from friend_ptr
4) Store the friend’s new card
5) Store the player’s new card.

Note: Any player can have any other player as their friend. For example, if the friend of A is B, the friend of B is not necessarily A.

NetID: _____

Due to constraints set by the server provider, your server can only support at most 60,000 unique players. (Hint: $2^{16} = 65{,}536$)

The server computer system that you are working with currently has:

- 32-bit memory space
- Support for misaligned memory accesses **(which involve 2 cache requests)**
- Little-endian system, with first members of the struct having lower addresses
- 2-way set associative write-allocate, write-back L1 cache with 128-bit (16 byte) cache lines and a total capacity of 8 kilobytes (8,192=2^{13} bytes)
    - The **most** significant bits are used as the cache tags
- LRU cache eviction scheme
- Memory operation time on hit: 1 ns
- Memory operation time on miss: 10 ns (even when writeback is needed)

Suppose the system is currently in a state with the following players all online, and at the following addresses in physical memory:

```
0xeceb0000: player_t A; // has friend B
0x04110010: player_t B; // has friend D
0x03910010: player_t C; // has friend D
0x03850010: player_t D; // has friend A
```

1. (6 points) Fill out the following table after the following consecutive trade calls (assume time is only taken by memory instructions, every other instruction takes no time, and that the cache is initially empty):

Function Call	Number of cache hits (for loads and stores)	Number of cache evictions	Total time for function call
`make_trade(&A, 0)`	3	0	33
`make_trade(&B, 3)`	4	1	24
`make_trade(&C, 6)`	3	2	33
`make_trade(&D, 9)`	5	1	15

2. (2 points) You have been tasked with improving the overall performance of the game.

    a. How can you change the player struct to improve the performance of the game?

```
// You may assume the following structure is packed
typedef struct player{
        _____uint16_t id_____;
        _____uint8_t beast_cards_____;
        _____struct player *friend_ptr_____;
} player_t;
```

    Swapping position of player_id and friend_ptr is also a valid solution.

    b. Why does your change improve the game's performance?

    Only one cacheline is used per player in a make_trade call, so there are less conflicts which leads to better performance.

3. (4 points) You implement your software change. You now find out that only about 100 players are in the game at any given time, and you notice this is because the game is played by different regions of the world at different times of day. After talking to another developer on the team, you realize that all players within a certain region have their structs allocated such that bits [10:4] of their address are all the same.

    a. What impact does the distribution of the addresses at any given time have on the program's performance? What aspect of the cache causes this?

    Performance is impacted negatively because, at any given time, the same cache sets are being used, so there are a lot of conflicts among them. This is due to the cache addressing scheme using set bits that are constantly the same.

    b. Given the issue above you determined, what change could be made to the hardware of the cache itself to improve the performance? (you cannot change the size or associativity of the cache)

    Change bits used for tag/set.

NetID: _____

Question 5: Thrashy Cache (10 points)

Noelle wrote a program which allocates a number of 1 KiB pages incrementally. She parses /proc/<pid>/pagemap to see the following virtual memory to physical memory mappings for the pages of interest:

Virtual Address Start	Physical Address Start
cafebabe000	ecebeceb000
cafebabe400	b00b1e51000
cafebabe800	deadbeef000
cafebabec00	cafebab1000
cafebabf000	12345679000

The program is running on a system with a 8 KiB direct mapped PIPT cache with 64 byte cache lines and 128 sets.

1. (4 points) When running the program, Noelle was surprised by the slow execution time. Determined to fix this, you profile the program execution, and find that most memory accesses are non-compulsory cache misses. Show why this is the case.
   Bottom 13 bits of physical page start is always the same, top 7 of those 13 are used to index set, therefore, there is thrashing b/w each page.

2. (3 points) Propose a modification to the page mappings to improve cache hit rates.

   Purposefully allocate physical pages to be continuous in the PIPT cache if their corresponding virtual pages are continuous (cache coloring)

3. (3 points) Propose a modification to the cache itself to improve cache hit rates without changing the associativity or size.

   VIPT/VIVT. If indexing is virtual, consecutive virtual pages get mapped consecutively.

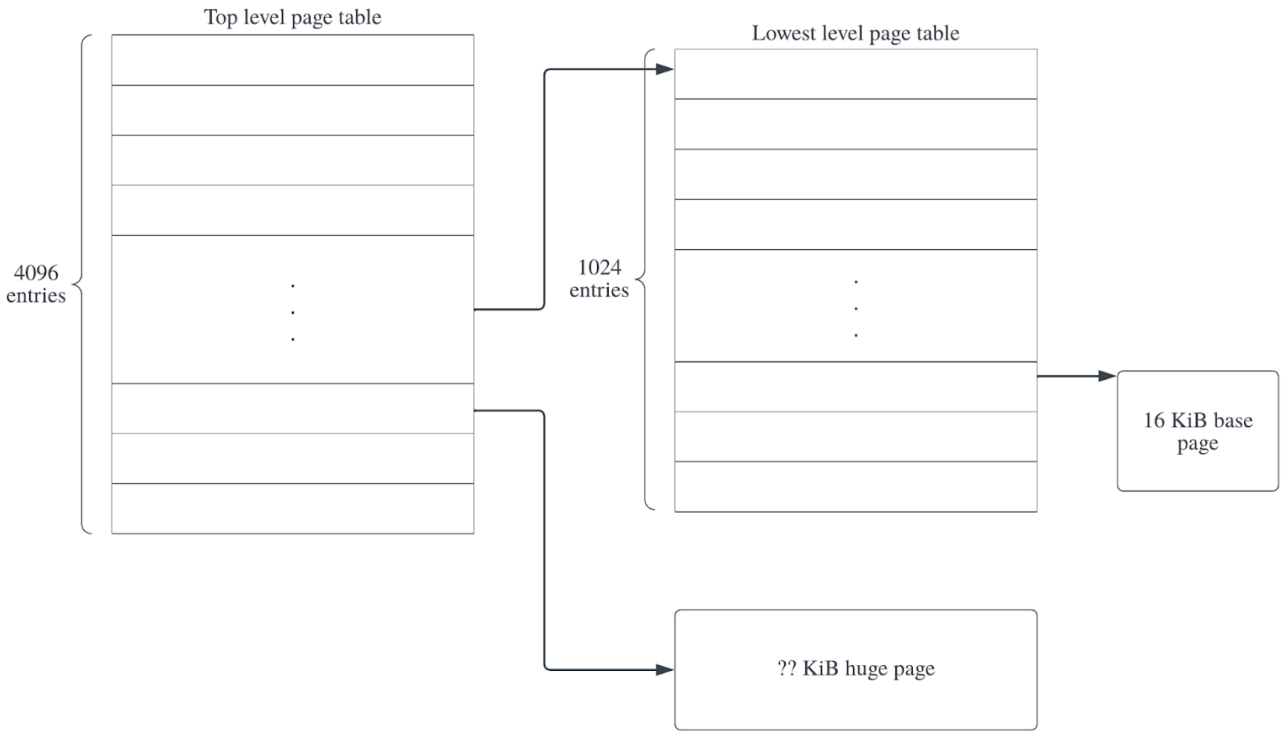
NetID: _____

Question 6: Big Page Energy (12 points)

Consider a virtual memory system with:

- Two levels of page tables
- 36 bit virtual addresses
- 32 bit physical addresses
- 16 KiB base pages

Aditya was tasked with implementing huge pages on this system. A "huge page" is a larger than normal page whose address is stored directly in an entry of the top-level page table, as opposed to the page address being stored in the lower-level page table, and having the top-level contain a pointer to help locate it. Aditya got distracted writing autograder testbenches and disappeared, but left this sketch of the system on the board in ECEB 2022. Using the sketch, answer the following questions:



1) (2 points) What is the size of huge pages in this system (in MiB)?
   16 MiB == 16,384 KiB == 2^24 bytes

2) (10 points) TLBs are used to speed up address translation by caching mappings. Design a TLB to support multiple page sizes. Your design must be detailed: clear drawings/diagrams are preferred over textual explanations. Make sure to include design challenges and issues for partial credit.

   Hint: In general, a set-associative TLB works by using the bottom bits of the virtual page number as an index into a set of physical addresses. The tag bits are then used to pick the correct physical address from the set.

   See Section 2 of https://ieeexplore.ieee.org/document/753337. Partial credit was assigned for identifying core issues/challenges such as not knowing the page size in advance of accessing the TLB, increased conflict misses if using high index bits, etc.

NetID: _____

Question 7: Precisely Out-of-Order Memory (18 points)

Your MP OOO team has finally finished implementing their design. Your TAs required you to implement a design based on Tomasulo's algorithm with precise state. To enforce precise state, you implemented a ROB. One interesting design feature is that your ROB maintains order via a linked list. Each ROB entry points to the next ROB entry that must be committed (based on program order).

Your processor has a cache implemented according to the **write-allocate** and **write-back** policies. Your cache lines are **32 bits**, and reading from memory into the cache overwrites any previous values in that line. Communication between the Cache and DRAM is at the granularity of a cache line; you ***cannot*** mask specific bytes. The processor can read or write to any cache line in a single cycle. If there is a cache miss on cycle X, a read request will be sent to memory on cycle X+1. At the start of cycle X+10, the cache line will have the updated values from memory and can be accessed by the processor (so a ROB entry waiting for that memory value could commit on cycle X+10).

The current state of your ROB is shown below. All ROB entries have received the values they need from the CDB and are waiting to commit. Only 1 ROB entry can be committed per cycle. If committing a store, the cache line is updated in the same cycle the ROB entry is committed. Assume the first instruction we should commit is located at ROB entry 0. Subsequent instructions must be committed in the order indicated by the "Next Entry" field. Here, a word is also 32-bits (so the size of a cache line is equal to one word).

ROB #	Instruction Type	Target Address	Value	Next Entry
0	STORE BYTE	0x6000004	0x8	1
1	STORE HALF WORD	0x6000008	0xBE	3
2	ADD	—	0x5	-
3	STORE BYTE	0x6000005	0x7	5
4	STORE HALF WORD	0x600000A	0xEF	6
5	STORE BYTE	0x6000006	0x6	4
6	STORE BYTE	0x6000007	0x5	2

In all parts, 1 point for correct ordering of the ROB # column. In 7.1, -0.5 per incorrect delta in the Commit Cycle column (max -2). In 7.2 onward, -1 per incorrect delta (max -4). Here, delta is the difference in cycles between an entry in one row and the previous row. For example, if row 1 is off by 1 cycle, and that error is propagated through the problem but later row values are otherwise correct, you would not lose points more than once.

NetID: _____

1) (3 points) In Tomasulo's algorithm, stores can only commit (and write to memory/cache) when they are the oldest instruction in the ROB. Write the cycle number that the instruction commits next to each ROB entry. Assume all reads and writes miss the first time a particular cache line is accessed. However, no cache conflicts exist between the memory addresses shown in the ROB (all cache lines map to unique cache locations). The first entry is done for you. **Assume cycles are counted starting from cycle 0.**

Instruction #	ROB #	Commit Cycle
0	0	10
1	1	21
2	3	22
3	5	23
4	4	24
5	6	25
6	2	26

2) (5 points) Your teammate analyzed some example programs and identified a way to improve the ROB commit logic. He noticed that some test programs involve iterating over a character array. This results in multiple subsequent *Store Byte* instructions to the same 4-byte word within a short time.

Your teammate proposes "store coalescing" to improve performance. His idea is that if you commit a store instruction, you should check if subsequent ROB entries also target the same address. If they do, perform a single commit that combines multiple stores. Subsequent entries should be determined by following the linked list.

Note: You can combine up to four stores if all four stores target different bytes in the same word, and when doing so would not violate precise state.

Fill in the below table with the commit cycles after implementing this optimization. The first instruction is done for you. **Assume cycles are counted starting from cycle 0.**

Instruction #	ROB #	Commit Cycle
0	0	10
1	1	21
2	3	22
3	5	22
4	4	23
5	6	24
6	2	25

3) (5 points) In parallel, your third teammate has also tried improving commit performance. She read some papers and found one that discusses an optimization called a "post-commit store buffer."

A store buffer functions like a FIFO. On committing a store instruction, the ROB will check if there is a free entry in the store buffer. If there is a free entry, the ROB can commit the store instruction. If there is not a free entry, the ROB should stall the store. Entries are removed from the store buffer when the cache can be written, which also takes a single cycle. If an entry is removed from the store buffer on cycle X, the ROB can write a new entry to the store buffer on cycle X+1.

Note: You cannot bypass the store buffer, and writing to it must be done on the same cycle that you commit the entry in the ROB.

Based on this idea, your teammate implements a **4-entry** "post-commit store buffer". An example of the structure is shown below. Her design **does not** include the optimization from part 2. In other words, at most 1 ROB entry can be committed at a time.

Fill in the table again for the instruction commit times achieved by the design with a "post-commit store buffer." The first instruction is done for you.

Assume the buffer starts empty. Assume cycles are counted starting from cycle 0.

Entry	Address	Mask	Value	Valid
0	32'bx	4'b0000	32'bx	1'b0
1	32'bx	4'b0000	32'bx	1'b0
2	32'bx	4'b0000	32'bx	1'b0
3	32'bx	4'b0000	32'bx	1'b0

Table 1: Example empty "post-commit store buffer"

Instruction #	ROB #	Commit Cycle
0	0	0
1	1	1
2	3	2
3	5	3
4	4	12
5	6	23
6	2	24

Extra space for part 3 if needed

4) (5 points) Your teammates present their optimization ideas. You realize that these ideas can be combined for even greater performance. Furthermore, you propose one additional optimization.
Instead of simply checking if there is a free entry in the "post-commit store buffer" described in the previous part, you check the destination addresses in the buffer. If the destination address of an entry in the buffer is the same word as the instruction you are trying to commit, you can update the buffer entry instead of allocating a new entry or stalling. Fill out the table one last time after all of these optimizations.

Instruction #	ROB #	Commit Cycle
0	0	0
1	1	1
2	3	2
3	5	2
4	4	3
5	6	4
6	2	5