Computer System Architecture
6.5900 Quiz #1
October 13th, 2023

Name: _____

This is a closed book, closed notes exam.
80 Minutes
17 Pages (+2 Scratch)

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Show your work to receive full credit.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not yet taken the quiz.
- Pages 18 and 19 are scratch pages. Use them if you need more space to answer one of the questions, or for rough work.

Part A	_____	40 Points
Part B	_____	25 Points
Part C	_____	35 Points
TOTAL	**_____**	**100 Points**

Name _____

Part A: Caches and Virtual Memory (40 Points)

Question 1 (5 points)

Consider a **2-way set-associative cache** with **128-byte blocks** and **2 sets**. The table below shows a timeline of how the cache metadata (valid bit (V), dirty bit (D), and tags) changes after a series of memory accesses. The leftmost column indicates the address of the memory access and whether they are reads (R) or writes (W), and the rest of the row should indicate the metadata **after** the access is performed.

Fill in the table below by showing how cache metadata changes after each access. If an entry remains unchanged after the memory access, you may leave that entry blank. Assume that the cache uses a least recently used (LRU) replacement policy (the table does not include LRU metadata). As an example, we have filled in the corresponding entries for the first memory access (0xA4C1).

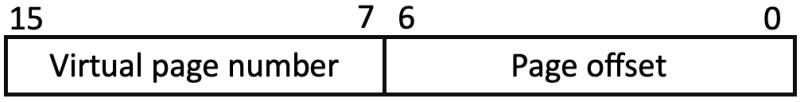
State	Set 0						Set 1					
	Way 0			Way 1			Way 0			Way 1		
	V	D	Tag	V	D	Tag	V	D	Tag	V	D	Tag
Initial state	0	-	-	0	-	-	0	-	-	0	-	-
After R 0xA4C1							1	0	0xA4			
After W 0x2673	1	1	0x26									
After W 0xB51A				1	1	0xB5						
After R 0xA4C0							1	0	0xA4			

Note that the last access is a hit on Set 1, Way 0.

Name _____

Question 2 (3 points)

Ben Bitdiddle recently bought a processor that has **16-bit virtual addresses**. The following figure shows the virtual address format:



What is the size of a page in this system, in bytes?

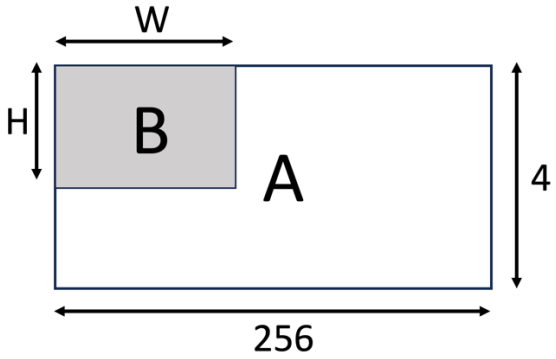
128B

Question 3 (10 points)

Ben's processor has an **8-entry direct-mapped TLB**. The TLB is indexed by the lowest order bits of the virtual page number.

Ben writes the program below, which operates on a matrix A. A has 4 rows and 256 columns. Each element of A is a 32-bit integer, and elements are laid out in row-major order (i.e., consecutive elements of the same row are in contiguous memory locations). The program sums the entries of a top-right submatrix B of matrix A. Assume that A starts at virtual address 0x0000, and `sum` is stored in a register. Ignore instruction fetches.

```
int sum = 0;
for (int i = 0; i < H; i++)
  for (int j = 0; j < W; j++)
    sum += A[i][j];
```



(a) (2 points) How many TLB misses will this program incur for H = 4, W = 256 (i.e., when B = A)?

A consists of 32 contiguous 128B pages. We all elements in A only once in the order they are laid out in memory. So there will be 32 misses.

(b) (2 points) How many pages does a row of A occupy?

8 Pages

(c) (3 points) What is the *minimum* TLB hit rate (i.e., TLB hits divided by number of accesses) that this program can have? What values of H and W cause this minimum? Specify all values of H and W that cause this.

W = 1 achieves the minimum hit rate of 0 for all possible H. Any W > 1 introduces additional accesses to a page already cached by the TLB.

(d) (3 points) What is the *maximum* TLB hit rate this program can achieve? What values of H and W achieve this maximum? Specify all values of H and W that achieve this.

Remember that a page contains 32 elements of matrix A. Max hit rate is having 1 miss to bring the page to the TLB, followed by 31 hits to it i.e., 31/32.

To achieve this, we need W = 32*N where 0 < N <= 8, and H can be any value.

Name _____

Question 4 (6 points)

Alyssa P. Hacker suggests that a larger page size would eliminate most of misses in the direct-mapped TLB for Ben's program.

(a) (3 points) For what values of H and W will the TLB hit rate **not** improve with pages that are double the size? Specify all values of H and W that achieve this.

Now each page contains 64 elements, and a row of A consists of 4 pages.

For W <= 32 and any H, we will have the same number of hits as with smaller pages since having a larger page doesn't reduce compulsory misses.

(b) (3 points) What is the minimum page size that will cause Ben's program to incur at least one TLB hit for all values of H and W such that H*W > 1? Assume page sizes must be a power of 2.

An easy way to solve this is thinking about the scenario where we had the minimum (0) hit rate for Question 4(b). There, accessing only one element per column resulted in no reuse because elements in different columns belonged to different pages.

Thus, the minimum page size to ensure that we have at least one TLB hit is a size greater than 1024B. Since we assume that page sizes grow in powers of two, that is 2048B.

Name _____

Question 5 (6 points)

Alyssa modifies Ben's program as follows, swapping the order of the two nested loops:

```
int sum = 0;
for (int j = 0; j < W; j++)
  for (int i = 0; i < H; i++)
    sum += A[i][j];
```

(a) (3 points) Does Alyssa's program have a better TLB hit rate than Ben's?

No. Alyssa's access pattern will, for the given column, always access four different pages that map to the same entry in the *direct-mapped* TLB (i.e., conflict misses).

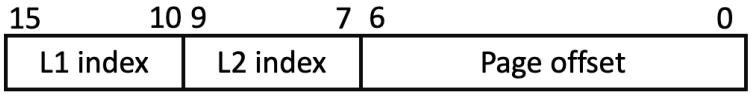
(b) (3 points) Assume W=8. For which values of H will doubling the page size improve the TLB hit rate of Alyssa's program?

H >=2. By doubling the page size TLB entries conflict between the first and third row, and between the second and fourth row.

Name _____

Question 6 (10 points)

Ben's processor uses a two-level hierarchical page table. The virtual address format is as follows:



Assume that all page tables have been swapped out to disk, and do not worry about the pages needed for code.

The L2 index is three bits wide. Thus, one L2 page table contains 8 page table entries.

(a) (2 points) How many L2 page tables does accessing a row of A cover?

1

(b) (2 points) How many unique page table entries are accessed when scanning the entire matrix A? Include both L1 and L2 page table entries.

Recall that A consists of 32 contiguous pages, so that's 32 L2 entries or 4 L2 page tables. To index those tables we need 4 L1 entries, for a total of 36.

(c) (3 points) What is the minimum submatrix size (= H*W) that will cause at least four L2 page table pages to be resident in memory after the program runs? Specify all values of H and W that achieve this.

We need to access elements such that we access four pages with different L1 bits, which means that we access page 0, 8, 16, and 24. The minimum submatrix that achieves this is size 4, where H = 4 and W = 1.

(d) (3 points) What is the maximum submatrix size that will cause exactly one L1 page table to be resident in memory after the program runs? Specify all values of H and W that achieve this.

H = 4 and W = 256

Name _____

Part B: Out-of-Order Processor (25 points)

Question 1 (20 points)

This question uses the out-of-order machine described in the Quiz 1 Handout. We describe events that affect the initial state shown in the handout. Label each event with one of the actions listed in the handout. If you pick a label with a blank (_____), you also have to fill in the blank using the choices (i—vii) listed below. If you pick "R. Illegal action", state why it is an illegal action. If in doubt, state your assumptions.

Example: `I10` finishes execution and writes its result to physical register `FP4`.
Answer: (G): Write a speculative value using greedy data management.
(You can simply write "G".)

(a) Assume physical register `P11` becomes available and holds a value of 1. Instruction `I9` executes and finds that the branch is taken.

(K, iii): Check the correctness of a speculation on branch direction and find a correct speculation

(b) Assume all instructions up to `I5` commit. `I6` commits, clears the valid bit of integer physical register `P7`, and adds `P7` to the free list.

(Q): Commit correctly speculated instruction, and free log associated with greedily updated values

(c) Assume `P6` is written by `I5` and `I5` finishes execution. `I7` is issued, reading physical registers `P4` and `P6`.

(B, i): Satisfy a dependence on register value by bypassing a speculative value

(d) Assume `F10` becomes available, `I10` is issued, and generates an exception due to division by zero. The processor flushes the contents of both reservation stations and starts fetching from the earliest uncommitted instruction. No other action is taken.

(R): Illegal or broken action

(e) Assume `I14` writes to `x5`. `I14` is allocated in the integer reservation station and the commit queue, grabs a new physical register `P8` from the free list, and updates the integer rename table entry of `x5` to `P8`.

(G): Write a speculative value using greedy data management

(f) Assume `FP10` becomes available, and that the FDIV unit is unpipelined. `I10` cannot be issued because the FDIV unit is currently occupied by `I12`.

(A, vii): Satisfy a dependence on functional unit by stalling

(g) Assume instruction `I14` is a conditional branch. The branch predictor is consulted in the decode stage, which predicts taken. The processor starts fetching the next instruction from the target address of the branch.

(E, iii): Satisfy a dependence on branch direction by speculation using a dynamic prediction

(h) Assume all instructions up to `I9` commit. The snapshot of the rename table associated with branch `I9` is freed.

(Q): Commit correctly speculated instruction, and free log associated with greedily updated values
(R): because it is ambiguous whether I9 committed.

(i) Assume instruction `I14` is an `addi x3, x3, 1` instruction, and is allocated in the integer reservation station and the commit queue. The instruction reads physical register `P2` when it is issued.

(B, i): Satisfy a dependence on register value by bypassing a speculative value.
(C, i): Also OK, because we don't specify the state of I7 when I14 is issued.

(j) Instruction `I15` has no entry in the BTB, so the next instruction is fetched from address `0xd8`.

(D, ii): Satisfy a dependence on PC value by speculation using a static prediction

Name _____

Question 2 (5 points)

We profile a program that has 50% integer and 50% floating point instructions. We find that integer and floating-point instructions have the average latencies listed below, when given infinite reservation station and commit queue sizes:

	Integer	**Floating-Point**
Decode to Issue	3 cycles	6 cycles
Decode to Commit	8 cycles	20 cycles
FU latency	2 cycles	12 cycles

If the processor commits 0.5 instruction per cycle on average for the program, how many entries are occupied on average in the reservation stations and the commit queue?

We use little's law for each of the structures: occupancy (N) = throughput (T) * latency (L)

First, T_int = T_fp = 0.25 instructions/cycle.

For integer reservation station, latency is 3 cycles, so total occupancy is 0.75
For fp reservation station, latency is 6 cycles, so total occupancy is 1.5

For commit queue we need to think of occupancy per operation type, then aggregate them.

latency of integer instructions is 8 cycles, so occupancy is 8 * 0.25 = 2.
latency of fp instructions is 20 cycles, so occupancy is 20 * 0.25 = 5.

So total commit queue occupancy is 2 + 5 = 7.

Integer Reservation Station: _____0.75_____

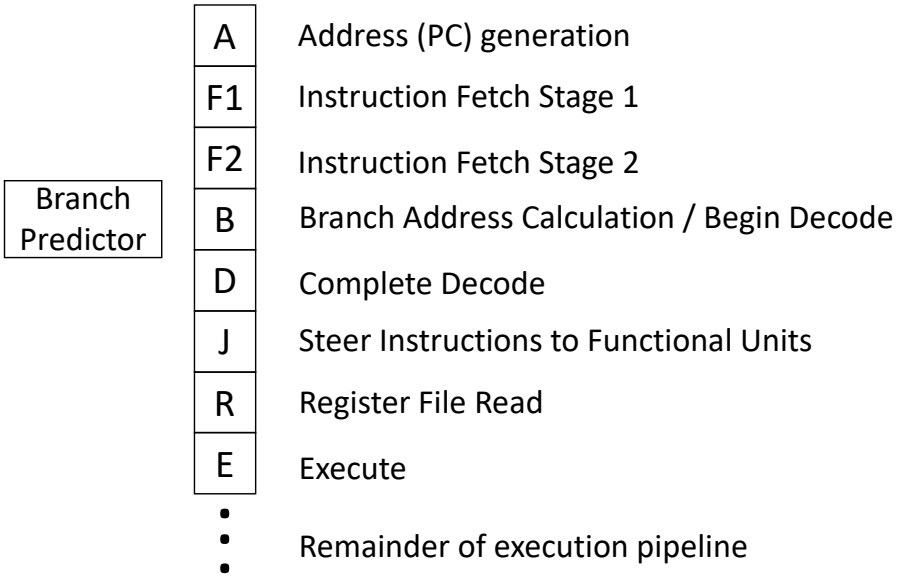
Floating-Point Reservation Station: _____1.5_____

Commit Queue: _____7_____

Name _____

Part C: Branch Prediction and Predication for Complex Pipelines (35 points)

Ben Bitdiddle is designing a branch predictor for a pipelined in-order processor with the following stages.



The processor has the following characteristics:

- Issues at most one instruction per cycle.
- Branch addresses are known at the end of the B stage.
- Branch conditions (taken / not taken) are known at the end of the E stage.

In this pipeline, control flow instructions work as follows:

- The A stage fetches the instruction at PC+4 by default.
- In the B stage (Branch Address Calculation/Begin Decode), conditional branch instructions (e.g., BLE/BNE) look up the predictor. If a branch is predicted to be taken, later instructions are flushed and the PC is redirected to the calculated branch target address.
- JAL instructions simply perform the jump to the target address in the B stage, flushing all later instructions in the pipeline. Do not worry about how JALR is implemented.

Name _____

Question 1 (5 points)

Fill out the following table. List all possible predictions the branch predictor can make for **conditional branches** (one per row). Then, fill the corresponding columns with the number of instructions flushed for each of the branch outcomes. Assume that branches go through the pipeline without any stalls or queueing delays.

	# Instructions Flushed	
Prediction	**Taken**	**Not Taken**
Taken	3	7
Not Taken	7	0

Name _____

For the following questions, consider the following C code and its RISC-V assembly equivalent. **Note that there are no branches other than the ones listed explicitly in the assembly code**:

C code:

```
int X[1000000];
int a, b;
…
for (int i = 1000000; i > 0; i--) {
    a = X[i];
    ...
    // Some work independent of a or b
    ...
    if (a >= 0) {
        b += b + b;
    } else {
        b += b * b;
    }
}
```

RISC-V assembly:

```
# x1, x2, and x3 contain values of &X, a, and b respectively.
_loop:  LW      x2, 0(x1)
        ADDI    x1, x1, -1
        ...
        # Some work independent of a or b
        ...
B1:     BLT     x2, x0, _else
        ADD     x3, x3, x3
        JAL     x0, B2
_else:  MUL     x3, x3, x3
B2:     BNE     x1, x0, _loop
```

Name _____

Question 2 (4 points)

Assume that branch B1 is always Not Taken, and the predictor predicts all branches as Taken. On average, how many instructions will be flushed by the processor per iteration of the loop?

We go through 2 conditional branches, for which the first one we mispredict as taken. In addition, the unconditional JAL is taken, so that's another 3 instructions flushed. So the total # flushed is 7 + 3 + 3 = 13 cycles.

Question 3 (4 points)

Ben's branch predictor consists of a single branch history table (BHT) that is indexed with the bottom 10 bits of the PC. Each entry in the table is a 1-bit counter that predicts taken if 1, and not taken if 0. The actual branch outcome updates the table at the end of the execute stage (1 if Taken, 0 if Not Taken).

Assume that the contents of array `X` are uniformly random between [-101, 100]. Will Ben's predictor have a high or low prediction accuracy for the two branches B1 and B2? Explain your answer.

B2 is pretty obvious (always taken). B1 will only be predicted ~50% correct because the branch direction is entirely random.

Branch B1: High / Low
Branch B2: High / Low

Question 4 (4 points)

Is there a design that can achieve significantly higher accuracy than the BHT for both branches given the assumption about `X` from Question 3? If so, describe the design briefly, stating how prediction works and how the prediction structures are updated after branch resolution. If not, briefly explain why it's not possible.

It's not possible to be meaningfully better. B2 is already predicted perfectly by the simple BHT, and we cannot achieve anything higher than 50% for the completely random branch B1.

Ben now wants to add support for predication (see the predication handout for details). To do so, he adds a predicate register file (PRF). Instructions that set the predicate write to the PRF in the writeback stage. Predicated instructions also read the PRF in the **writeback stage**. If the predicate register is set, the predicated instruction writes back its result to the register file. Otherwise, the instruction doesn't write back its result to the register file (effectively becoming a NOP).

Question 5 (8 points)

Rewrite the previous RISC-V assembly code to use predication instead of the B1 branch. You may change only the code within the red box. You should use the minimum number of instructions possible in your solution.

```
            SETPGE  x2, x0, p0
      (p0)  ADD     x3, x3, x3
      (!p0) MUL     x3, x3, x3
B2:         BNE     x1, x0, _loop
```

Assume that Ben's processor uses the branch predictor design from Question 3 for the following questions.

Question 6 (4 points)

Assume that branch B1 is not taken. How many instructions will be flushed by the processor per iteration of the loop **for your predicated code**?

Now there's a single conditional branch which is always taken (and predicted taken) so 3 instructions will be flushed per iteration.

Question 7 (6 points)

Assume that the contents of array X are uniformly random between [-101, 100], like in Question 3. Do you think replacing the B1 branch with predication will improve overall performance? Justify your reasoning. For full points, explain how many cycles on average using predication would save over keeping the branch (or vice versa).

Yes, it will improve performance, but we will need to calculate exactly how much. We only need to examine B1 branch since the cost for B2 branch will be the same.

In the original code, there are 4 possibilities for B1, all of which have different total # of instructions flushed:

taken & predicted taken: 3 instructions
taken & predicted not taken: 7 instructions
not taken & predicted taken: 10 instructions (+3 due to JAL)
not taken & predicted not taken: 3 instructions (+3 due to JAL)

these are all equally likely, so total average # instructions flushed is 23/4.

In contrast, predication makes the processor execute 1 more instruction in case of branch taken, so there is a 0.5 cycle overhead. So total benefit is 23/4 - 0.5 = 21/4 cycles.

Name _____

Scratch Space

Use these extra pages if you run out of space or for your own personal notes. We will not grade this unless you tell us explicitly in the earlier pages.

Name _____

Scratch Space