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**

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.

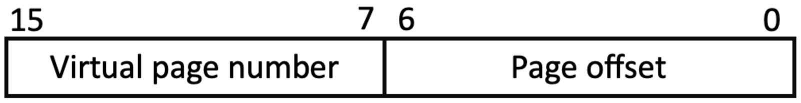
Fill in the table below by showing how cache metadata changes after each access. If an entry remains unchanged after the memory access, you may leave that entry blank. Assume that the cache uses a least recently used (LRU) replacement policy (the table does not include LRU metadata). As an example, we 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`												
`After W 0xB51A`												
`After R 0xA4C0`												

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:



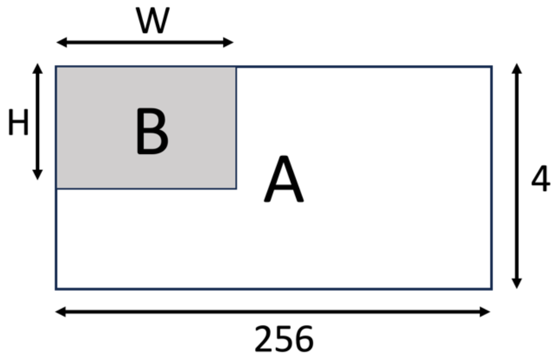
What is the size of a page in this system, in bytes?

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)?

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

(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.

(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.

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.

(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.

Name ____________________________

Question 5 (6 points)

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

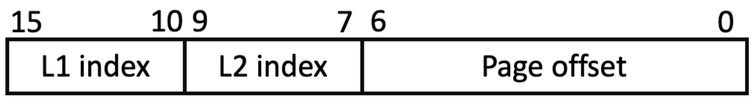
```
int sum = 0;
for (int j = 0; j < W; j++)
  for (int i = 0; i < H; i++)
    sum += A[i][j];
```

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

(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?

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?

(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.

(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.

(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.

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.

(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.

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

(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.

(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`.

(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`.

(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.

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

(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.

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

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?

Integer Reservation Station: _______________

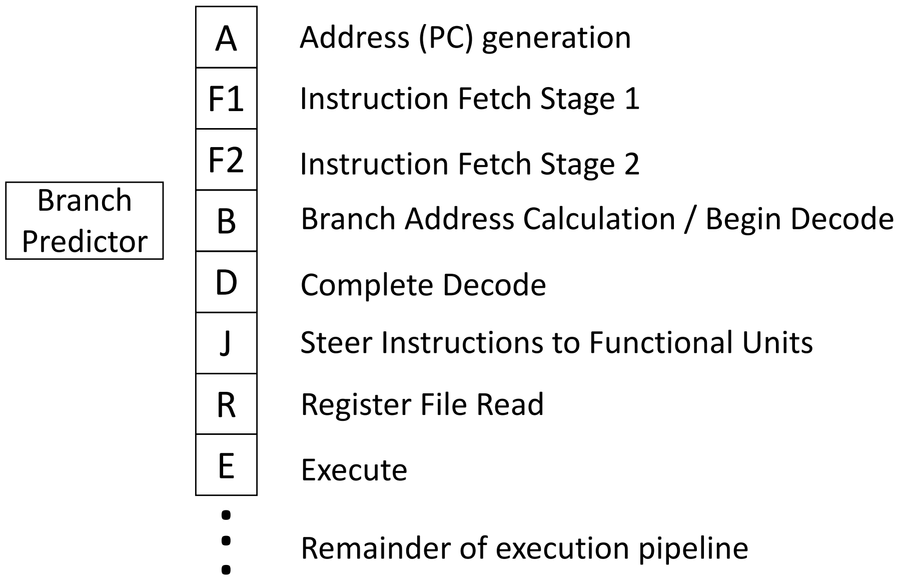
Floating-Point Reservation Station: _______________

Commit Queue: _______________

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.


Branch Predictor
A
Address (PC) generation
F1
Instruction Fetch Stage 1
F2
Instruction Fetch Stage 2
B
Branch Address Calculation / Begin Decode
D
Complete Decode
J
Steer Instructions to Functional Units
R
Register File Read
E
Execute
Remainder of execution pipeline

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**

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
```

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?

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.

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.

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.

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**?

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).

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