Computer System Architecture
6.5900 Quiz #3
December 13th, 2023

Name: _____

This is a closed book, closed notes exam.
80 Minutes
16 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 17 and 18 are scratch pages. Use them if you need more space to answer one of the questions, or for rough work.

Part A	_____	28 Points
Part B	_____	22 Points
Part C	_____	18 Points
Part D	_____	32 Points
TOTAL	**_____**	**100 Points**

Name _____

Part A: VLIW Processors (28 points)

In this question, we will examine the execution of the code below on a single-issue in-order processor and a VLIW processor.

```
      // A, X, and Y hold single-precision (32-bit)
      // floating point values
      float A, X[N], Y[N];
      for (int i = 0; i < N; i++)
          Y[i] = Y[i] + A*X[i];

      // Initial values:
      //  f1 = A
      //  x1 = &X[0]
      //  x2 = &Y[0]
      //  x3 = &X[N] (first address after vector X)
I1:  loop: lw     f0, 0(x1)
I2:        fmul.s f2, f0, f1
I3:        lw     f3, 0(x2)
I4:        fadd.s f4, f2, f3
I5:        sw     f4, 0(x2)
I6:        addi   x1, x1, 4
I7:        addi   x2, x2, 4
I8:        bne    x1, x3, loop
```

Question 1 (5 points)

The code above runs on an in-order, single-issue processor with perfect branch prediction and full bypassing. ALU (integer) operations have a 1-cycle latency (so, thanks to bypassing, consecutive dependent ALU operations execute without stalling), loads and stores have a 2-cycle latency, and floating-point operations have a 3-cycle latency. How many cycles will the processor *stall* per loop iteration?

Name _____

Question 2 (5 points)

If you apply software pipelining to the loop, what is the minimum number of iterations that you would need to overlap to remove all stalls in steady-state operation? You are allowed to reorder instructions. (Hint: You don't need to actually software-pipeline the loop to answer this.)

Question 3 (6 points)

Write the VLIW schedule of the instructions in one iteration of the original loop. You only need to write instructions on the critical. (You may also write instructions off the critical path if you want, which we will not grade). The 3-operation VLIW format is shown below. The VLIW architecture has the same fixed delays as the in-order processor (1/2/3 cycles for ALU/memory/floating-point operations, respectively), and has no stall logic. You may reorder and modify instructions. For full credit, your implementation should use the minimum number of VLIW instructions.

Inst.	ALU/Branch Unit	Memory Unit	Floating Point Unit
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			

Name _____

Question 4 (6 points)

Apply loop unrolling to the VLIW code in Question 3. Unroll the fewest number of iterations required to eliminate all stalls. Whatever degree of unrolling you choose, assume it divides the total number of loop iterations exactly. Again, you only need to write instructions on the critical path.

Inst.	ALU/Branch Unit	Memory Unit	Floating Point Unit
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			

Question 5 (6 points)

What is the maximum throughput, *in floating-point operations per cycle*, that the VLIW processor can achieve by applying software pipelining to the original loop? How much better is this throughput, roughly, than the best throughput of the in-order, single-issue processor? Please explain your answer. (To answer this, it's sufficient to give the nearest integer factor, e.g., about 5x better.)

Name _____

Part B: Transactional Memory & Reliability (22 points)

In this part you will analyze the operation of different hardware TM (HTM) designs, and the concurrency they achieve for different transaction schedules on a 2-core system as described in the handout.

The system runs a program consisting of the following two transactions.

Transaction X
`Begin`
`Read A`
`Write A`
`Read B`
`End`

Transaction Y
`Begin`
`Read A`
`Read B`
`Write B`
`End`

In the following questions, for timing, assume conflict detection and coherence actions all happen in the same cycle when a memory access executes.

Name _____

Question 1 (6 points)

Suppose transaction X starts at cycle 0 and transaction Y starts at cycle 5, and they would produce the following schedule.

Cycle	0	5	10	15	20	25	30	35	40	45
Transaction X	`Begin`		`Rd A`		`Wr A`		`Rd B`		`End`	
Transaction Y		`Begin`		`Rd A`		`Rd B`		`Wr B`		`End`

(a) In the absence of conflict detection (i.e., no HTM), if the memory operations interleaved in the given order, would the transactions be serializable? If so, circle what would be the apparent commit order of the transactions, or circle "Not serializable". (2 points)

X before Y **Y before X** **Not serializable**

(b) Given the two HTM designs described in the handout, indicate in the following table at what cycle a conflict is detected, if any, and which transaction aborts (or neither). (4 points)

	Conflict cycle	**Aborted Transaction (X, Y, or Neither)**
Eager & Pessimistic		
Lazy & Optimistic		

Name _____

Question 2 (10 points)

We now study reliability in the context of HTM, specifically the ACEness of the read and write bits of each cache line.

Note that needless transaction aborts that do not lead to incorrect computation results *are architecturally correct execution*. Thus, if flipping a bit only causes needless transaction aborts, and the final computation outcome is still architecturally correct, this bit is un-ACE.

Consider the lazy & optimistic HTM implementation described the handout, running transactions X and Y shown below (these are a simplified version of the transactions in Question 1, with the same timings and accesses to A, but no accesses to B).

Mark the cells in the following table that correspond to cycles at which the read and write bits of the cache line storing A are ACE, for transactions X and Y. (If you prefer, you can instead list the specific cycles for each of these four bits below the table.)

Cycle		0	5	10	15	20	25	30	35	40	45
Transaction X		Begin		Rd A		Wr A				End	
Transaction Y			Begin		Rd A						End
Txn X	Write bit										
	Read bit										
Txn Y	Write bit										
	Read bit										

Name _____

Question 3 (6 points)

Assume that the HTM implementation includes an error detection mechanism for the read and write bits. Each private cache is extended with an error detection code that is able to reliably inform when a bit flip has happened on at least one of the read/write bits in the cache. However, the mechanism does *not* tell which of the bits has suffered a flip.

Assuming lazy & optimistic HTM implementation from the handout, can you modify the HTM design to eliminate reliability errors **due to bit flips in read/write bits** using this mechanism? Your modifications cannot introduce additional state or error-detection mechanisms. If yes, describe these modifications. If not, explain why not.

Name _____

Part C: Security (18 points)

Consider the C code with labeled line numbers:

```
L0: bool checkPassword(int* password, int* guess, int size) {
L1:   for (int i = 0; i < size; i++) {
L2:    if (password[i] != guess[i]) {
L3:      return false;
L4:    }
L5:  }
L6:  return true;
L7:}
```

The C code roughly produces the following RISC-V 32 assembly:

```
      // Initial register values:
      // a0 = &password[0]
      // a1 = &guess[0]
      // a2 = size (assumed > 0 in the code)
loop:
        lw      t0, 0(a0)
        lw      t1, 0(a1)
        bne     t0, t1, retFalse
        addi    a0, a0, 4
        addi    a1, a1, 4
        addi    a2, a2, -1
        bgt     a2, x0, loop
retTrue:
        li a0, 1   // loads constant 1 into register a0
        ret
retFalse:
        li a0, 0
        ret
```

Name _____

Question 1 (10 points)

(a) How can an attacker learn the password faster than brute force search? (7 points)

(b) Which line(s) is/are the transmitter in the C code? (3 points)

Name _____

Question 2 (8 points)

Rewrite the relevant lines in the assembly code to eliminate micro-architectural side-channels. You can also write C code instead of assembly; if you write C code, we will grade your answer based on the assembly produced by compiling with clang 17.0.0 with no compiler flags.

Name _____

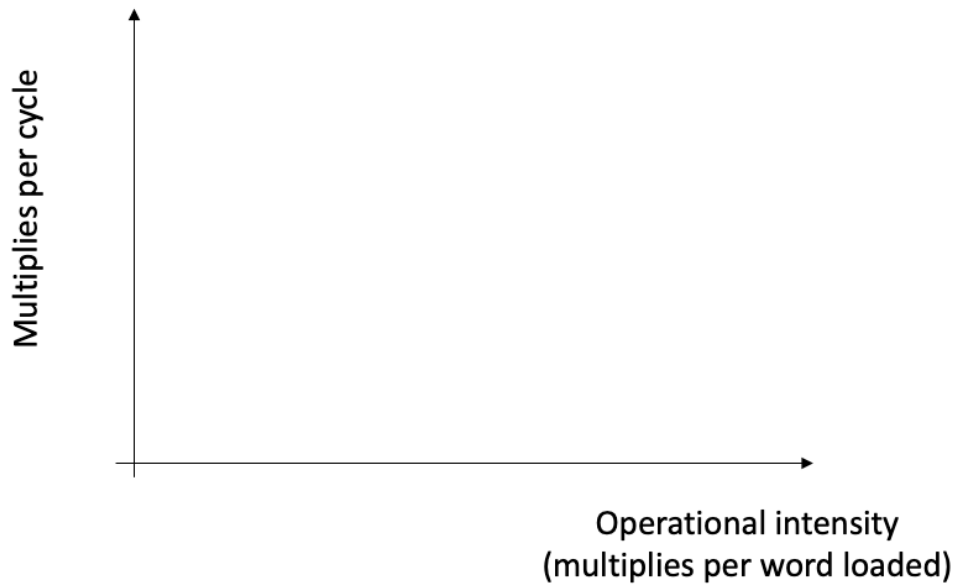
Part D: Accelerators (32 points)

Axel is building a linear algebra accelerator called the Axelerator. It can do 3 multiplies per cycle and has 0.1 words/cycle memory bandwidth. Ryan is a CEO of a laptop manufacturing company; he makes the audacious claim that his second-tier laptop model, the RyanAir, outperforms the Axelerator on some workloads despite having 3x fewer multipliers. The RyanAir has 0.5 words/cycle memory bandwidth. Both machines have a multiplier FU latency of 1 cycle and run at the same clock frequency.

Question 1 (4 points)

In the plot below, draw the rooflines for the Axelerator and the RyanAir. For each of the two rooflines, label the (x, y) coordinates of the *corner* of the roofline, i.e., the point where the system goes from memory bandwidth-bound to compute-bound.

Note: Remember that a multiply has two inputs.



We will analyze the performance of three kernels on the Axelerator and the RyanAir. For all the remaining the questions, assume the following:

- Ignore the cost of all operations but multiplications and loads from main memory. Also ignore dependencies on the loop indices i, j, and k.
- Assume instructions are executed in program order. Assume that at each cycle all multiplications that have their dependencies satisfied are issued (up to the number of multiplier functional units).
- Assume an unbounded on-chip cache.
- Ignore stores (the results of each computation stay on-chip and are not written back to main memory).
- All kernels are assumed to run many times on different inputs. All questions concern only steady-state behavior.
- Assume kernel computation and kernel operand loading are decoupled, i.e., the operands for each run of the kernel are loaded far enough in advance to ensure that they will be available when the kernel computation starts. Therefore, load latency does not matter (but throughput does!).
- On both systems, at any point in time at most one kernel is being computed on.

Name _____

Question 2 (4 points)

The code below performs **matrix-vector multiply** of an NxN matrix and an N-element vector.

```
for (int j = 0; j < N; j++)
  for (int i = 0; i < N; i++)
    result[i] += matrix[i][j] * vector[j];
```

What is the operational intensity of matrix-vector multiply as a function of N? Give your result in multiplies per word loaded from main memory.

Question 3 (4 points)

Which machine performs better on **matrix-vector** multiply as a function of N? Can you use the roofline model to answer this question? If yes, how? If no, why?

Name _____

Question 4 (4 points)

The code below performs **matrix-matrix multiply** of two NxN-element matrices.

```
for (int k = 0; k < N; k++)
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      C[i][j] += A[i][k] * B[k][j];
```

What is the operational intensity of matrix-matrix multiply as a function of N?

Question 5 (4 points)

Which machine performs better on **matrix-matrix** multiply as a function of N? Can you use the roofline model to answer this question? If yes, how? If no, why?

Name _____

Question 6 (4 points)

The code below **computes the N-th factorial** (since the code uses an unsigned 32-bit integer, the result is actually the N-th factorial modulo 2^{32}).

```
unsigned result = 1; // constants don't come from main memory
for (unsigned i = 1; i < N; i++)
   result = result * i;
```

What is the operational intensity of computing the N-th factorial, as a function of N?

Question 7 (4 points)

Which machine performs better on **computing the N-th factorial** as a function of N? Can you use the roofline model to answer this question? If yes, how? If no, why?

Question 8 (4 points)

How can Axel change the Axelerator so that it at least matches the performance of the RyanAir on all the kernels above for all N?

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

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.