University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS152/252A
Spring 2022

J. Wawrzynek
4/7/22

Midterm Exam 2

Name: ______________________________

Student ID number: ______________________________

You have until 11AM to take the exam.

This is a *closed-book, closed-notes* exam, except for one handwritten sheet. Also no calculators, phones, pads, or laptops allowed.

Each question is marked with its number of points (one point per expected minute of time). Start by answering the easier questions then move on to the more difficult ones. You can use the backs of the pages to work out your answers. Neatly copy your answer to the allocated places. **Neatness counts.** We will deduct points if we need to work hard to understand your answer.

Write the student ID numbers of the person to your left and to your right on this page. If you are sitting on an aisle, just indicate "aisle" for left or right.

Left student ID number: ______________________________

Right student ID number: ______________________________

Before you turn in your exam, write your student ID number on all pages.

Friday 31st March, 2023 20:03

1. Comparing Architectures [8 pts]

(a) Below we describe certain types of programs and process/compiler technologies. For each, circle which type of architecture would be *most appropriate*.

i. Programs with many branches, where the branch behavior is not known until runtime.

Out-of-order superscalar VLIW Vector processor

ii. Linear algebra algorithms (such as matrix multiplication).

Out-of-order superscalar VLIW Vector processor

(b) Suppose that you have code that exhibits a high level of thread-level parallelism (TLP). You realize that you could run this code on either a superscalar OoO core with SMT, or on a GPU.

i. Suppose that your code includes deeply nested if-statements. Would you choose the GPU or the superscalar OoO core with SMT? Explain why, in no more than three sentences.

ii. Suppose that your code includes a very large number of scattered memory accesses. Would you choose the GPU or the superscalar OoO core with SMT? Explain why, in no more than three sentences.

iii. Suppose that your code exhibits a very high degree of data-level parallelism (in addition to TLP). Would you choose the GPU or the superscalar OoO core with SMT? Explain why, in no more than three sentences.

2. Out-of-Order Execution and Branch Prediction [21 pts]

Consider the following code, which performs an element-wise "max" operation on two vectors.

```
    // In C
    for (int i = 0; i < N; i++)
      if (a[i] >= b[i])
        c[i] = a[i];
      else
        c[i] = b[i];

    // In assembly
    //   x1 is the pointer to \a"
    //   x2 is the pointer to \b"
    //   x3 is the pointer to \c"
    //   x4 is the length of the vectors (N)

    loop:  ld x5, 0(x1) // a[i]
           ld x6, 0(x2) // b[i]

           blt x5, x6, label1

           sd x5, 0(x3) // c[i] = a[i]
           j label2

  label1: sd x6, 0(x3) // c[i] = b[i]

  label2: add x1, x1, 4
           add x2, x2, 4
           add x3, x3, 4
           add x4, x4, -1

           bnez x4, loop
    end:
```

Suppose that you have an out-of-order, superscalar core with the following characteristics:

- *Up to two* instructions can be fetched, decoded, dispatched, and issued every cycle.
- *Up to two* instructions can be committed every cycle.
- The ROB has *four* entries.
- Adds and branches take 1 cycle to execute.
- Loads and stores take 3 cycles to execute.
- All functional units (including memory units) are fully pipelined.
- All instructions must spend 1 cycle in the write-back stage before their result can be used by a dependent instruction.
- Instructions can commit one cycle after writeback, and ROB entries can be reused one cycle after commit.

- The CPU can execute all instructions speculatively in case of a branch. Speculative stores go into a store buffer to avoid corrupting memory.

Below is the register rename table at the beginning of execution:

Architectural Register	**Physical Register**
x1	P1
x2	P2
x3	P3
x4	P4
x5	P5
x6	P6

The free list is a FIFO with the initial state below. The leftmost element is the head of the free list, and the rightmost element is the tail.

P8	P10	P12	P14	P16	P18	P20

(a) **Register Renaming** [9 pts]

For this section, assume that the CPU correctly predicts all branches (as well as branch targets).

In the table below, fill in the cycle number for when each instruction enters the ROB, issues, writes back, and commits. Also, fill in the new register names for each instruction, where applicable. Use only physical register names for the src entries.

Assume that the ROB is initially empty. Part of the table has been filled in for you.

Time				OP	Dest	Src1	Src2
Enter ROB	Issue	WB	Commit				
0	1	4	5	`ld x5, 0(x1)`			
				`ld x6, 0(x2)`			
				`blt x5, x6, label1`			
				`sd x5, 0(x3)`			
				`j label2`			

Also, fill in the values in the free list when the last instruction in the table above commits:

(b) **Speculation** [5 pts]

i. Suppose that in the instruction stream in ??, the "`blt`" instruction was mispredicted. After the following instructions were flushed, what would be the state of the free list?

ii. Suppose that we removed the speculative store buffer. How could we still maintain precise exceptions in that case? How would this affect program performance? Explain your answer, but in no more than three sentences.

(c) **Branch Prediction** [7 pts]

i. Suppose you have only two entries available in your branch-target buffer (BTB). If you wanted to maximize performance, which of these following three branches would you NOT store in your BTB? Circle only one.

```
blt x5, x6, label1

j label2

bnez x4, loop
```

ii. Now, suppose that the two vectors are actually very small. Vector "a" is {0, 100, 1}. Vector "b" is {9, 0, 9}.
Suppose that our branch predictor is a PC-indexed branch history table (BHT). Each entry in the BHT is a *two-bit saturating counter*. None of the branches in the loop above conflict in the BHT. Assume that all BHT entries begin execution at 10 (weakly taken). What will be the branch prediction accuracy of the "`blt x5, x6, label1`" branch?

3. VLIW [15 pts]

Suppose that we have a VLIW architecture with the following *fully-pipelined* functional units:

- One ALU unit which also performs branches. *1 cycle delay*
- Two FPU units which perform floating-point operations. *3 cycle delay*
- One load/store unit. *2 cycle delay*

Now, consider the code below, which computes the sum and sum-of-squares of a vector. You can assume that *N is very large.*

```
// In C
for (int i = 0; i < N; i++) {
  sum += arr[i];
  sumOfSquares += arr[i]*arr[i];
}

// In assembly
//   f0 is \sum"
//   f1 is \sumOfSquares"
//   x1 points to \arr"
//   x2 points to the end of \arr"

loop: fld f2, 0(x1)
      fadd f0, f0, f2 // sum += arr[i];
      fmul f3, f2, f2
      fadd f1, f1, f3 // sumOfSquares += arr[i]*arr[i];
      addi x1, x1, #4
      bne x1, x2, loop
```

(a) **Scheduling** [3 pts]

Schedule the loop above in the table below. You can just write the opcode and destination registers in the table. You can ignore the fixed integer offsets/constants in the "fld" and "addi" instructions. In other words, when you write the fld and addi instructions in the table, just write "fld f2" and "addi x1". We have filled out one element of the table below to get you started. *Minimize* the number of cycles taken. You can re-order instructions, but do not perform software pipelining or loop unrolling.

label	**ALU**	**FPU**	**FPU**	**MEM**
`loop`				`fld f2`

(b) **Software Pipelining** [10 pts]

Schedule the operations using software pipelining alone (without loop unrolling). Include the prologue and epilogue. Minimize the number of cycles taken by your program.

label	ALU	FPU	FPU	MEM

(c) **Short Answer** [2 pts]

Suppose that you added an infinite number of FPUs and ALUs to this VLIW machine. Would you be able to achieve one cycle per iteration then, without any loop unrolling? (You would still be permitted to re-order and pipeline your code). If so, why? If not, why not? Do not answer with more than three sentences.

4. Multithreading [11 pts]

(a) **Performance** [4 pts]

Consider the code below, which adds a constant value to every element of a vector:

```
// In C
for (int i = 0; i < N; i++)
  arr[i] += x;

// In assembly
//   x1 points to \arr"
//   x2 points to the end of \arr"
//   f0 contains \x"

loop: fld f1, 0(x1)
      addi x1, x1, #4
      fadd f1, f0, f1 // arr[i] += x;
      fsd f1, -4(x1)
      bne x1, x2, loop
```

Suppose that:

- All memory operations take 10 cycles.
- Floating point additions take 3 cycles.
- Integer additititions and branches take 1 cycle. There is perfect branch prediction.

i. How many threads are required to avoid all stalls with fixed round-robin scheduling? Don't reorder the instructions.

ii. Suppose that we instead use data-dependent thread scheduling, which switches threads whenever a stall would occur due to a RAW hazard. (You can assume that WAW and WAR hazards do not cause stalls).
How many threads would be required to avoid all stalls? (Consider only the steady-state). Don't reorder the instructions.

(b) **SMT** [7 pts]

Suppose that you added simultaneous multithreading (SMT) to a superscalar out-of-order core with a unified physical register file.

i. Would adding SMT to this core be expected to increase, decrease, or have no effect on the following values?

- The maximum number of instructions which can be committed every cycle.
- The average number of instructions which would be committed every cycle.
- The average number of instructions which are flushed upon an exception/misprediction. (Assume that the total ROB size remains constant)
- The number of rename tables.
- The number of functional units.

ii. Describe one way in which SMT can reduce the performance of a single particular thread. Your answer should not be more than three sentences.

5. Vector Processors [5 pts]

(a) Suppose you wanted to calculate the sum of all the elements in a vector. (This is a reduction which yields a single scalar result). How would you perform this operation with a vector ISA which supports vector additions, but no reductions? You can answer qualitatively, and you don't need to show any code.

(b) Consider a vector processor with only a single vector lane. Would you expect this vector machine to perform better, worse, or the same as a scalar in-order CPU when running vectorizable code? Explain your answer in no more than three sentences.