CS 351 Final Review Quiz

Notes:

- You must explain your answers to receive partial credit.
- You will lose points for incorrect extraneous information, even if the answer is otherwise correct.

Question 1: Short answer [25 points].

a. Define the term *basic block.*

Are long basic blocks or short basic blocks better for optimization? Why?

b. Write a snippet of code (MIPS or high-level pseudocode) exhibiting DLP. What kinds of ISAs allow DLP to be exploited in hardware?

c. Explain the recent emergence of multicore processors. What type of parallelism do they exploit?

d. Many of the fastest supercomputers in the world are clusters rather than special-purpose machines. What are the advantages of using clusters? What are the disadvantages?

e. What is false sharing? Show pseudocode that exhibits false sharing for a dual-core machine with 8-word cache blocks.

Question 2: Compiler optimizations [20 points].

For the following C code:

```
for (int i=0; i<N; i++) {
     a[i] = b[i] + c[i];
     d[i] = e[i] + f[i];
}
```

Assume the following:

- The arrays are integer arrays, where an integer is 1 word long.
- N is very large.
- The loop counter *i*, the array size *N*, and the base addresses of the six arrays are kept in registers at all times.
- The code runs on a machine whose L1 cache is fully associative and consists of four 4-word blocks.

(a) What is the cache hit rate of this code, assuming that the cache is initially empty?

(b) Restructure the code to increase the cache hit rate.

(c) What is this optimization called?

(d) What is the cache hit rate of the modified code?

(e) *This question does not refer to the snippet of code used in Parts (a)-(d).* In MIPS or a higher-level language, show an example of code motion.

Question 3: Exploiting ILP [25 points].

For the following MIPS code:

```
1) LW $t1, 0($t2)
2) ADDI $t1, $t1, 1
3) SW $t1, 0($t2)
4) ADDI $t2, $t2, 4
5) SUB $t4, $t3, $t2
6) ADDI $t4, $t4, 1
```

(a) List the dependencies and antidependencies. For each, give the 2 instructions and the register involved, and state whether it is a true dependency or an antidependency.

(b) Rename the instructions using physical registers $p0 through $p63.

(c) Assume a 5-stage pipeline that works as follows:

- Stage 1: Fetch 2 instructions from memory.
- Stage 2: Decode and rename the instructions; choose up to 2 ready instructions to issue to the execution units.
- Stage 3: (identical to MIPS EX stage, except that 2 instructions can be processed at once)
- Stage 4: (identical to MIPS MEM stage, except that 2 instructions can be processed at once)
- Stage 5: Write back to the physical register file; update the reorder buffer; "officially" complete up to 2 instructions by removing them from the reorder buffer.

Assume that an instruction can start Stage 3 in the cycle immediately after its operands are produced. That is, assume that the issue logic can figure out that an instruction's operands will be ready in time for the next cycle.

List the instructions (by number) that will be fetched in Cycle 1:

List the instructions (by number) that will be fetched in Cycle 2:

List the instructions (by number) that will be decoded and renamed in Cycle 2. Label each instruction either with "READY" or with the physical register(s) it must wait for before it can execute. Mark instructions that will advance to Stage 3 in the next cycle with a *.

List the instructions (by number) that will be fetched in Cycle 3:

List the instructions (by number) that will be decoded and renamed in Cycle 3:

List the instructions that are currently in the instruction window during Cycle 3. Label each instruction either with "READY" or with the physical register(s) it must wait for before it can execute. If an instruction's operands will be ready at the end of Cycle 3, mark it as "READY." Mark instructions that will advance to Stage 3 in the next cycle with a *.

List the instructions (by number) that will be decoded and renamed in Cycle 4:

List the instruction(s) (by number) whose results will be ready at the end of Cycle 4:

List the instructions that are currently in the instruction window during Cycle 4. Label each instruction either with "READY" or with the physical register(s) it must wait for before it can execute. If an instruction's operands will be ready at the end of Cycle 4, mark it as "READY." Mark instructions that will advance to Stage 3 in the next cycle with a *.

List the instructions that are currently in the instruction window during Cycle 5. Label each instruction either with "READY" or with the physical register(s) it must wait for before it can execute. If an instruction's operands will be ready at the end of Cycle 5, mark it as "READY."

List the instruction(s) (by number) that will reach the reorder buffer at the end of Cycle 5. Will these instruction(s) be able to officially commit, or will they remain in the reorder buffer?

Question 4: Cache coherence [20 points].

For this problem, see the state machine on the final page of this test, which is identical to Figure 9.3.4 in your textbook.

(a)Fill out the following chart for a two-processor system executing the following sequence of instructions and adhering to the cache protocol in the diagram. Assume the following:

- The data from addresses 110 and 120 are initially stored in both processors' caches and marked as clean/shared.
- The cache block size is 1 word.

"Before" and "After" refer to the block's state in cache before the instruction is executed and after it is executed, respectively.

Processor: Instruction	**Hit or miss?**	**Block's state in P0 cache (before)**	**Block's state in P0 cache (after)**	**Block's state in P1 cache (before)**	**Block's state in P1 cache (after)**
P0: read 120					
P0: write 120					
P1: write 120					
P1: read 110					
P0: write 110					
P1: read 120					

(b) The MESI protocol is similar to the protocol in the diagram, except that it has an additional state called "Exclusive." If a block is marked "Exclusive," that means that it is present in only one processor's cache and not shared in any other processor's cache, and that it is clean (has not been written). What is the advantage of this additional state?

Question 5: Parallel decomposition [10 points].

(a) What does it mean for a parallel computation to have a *reduction*?

(b) To find the maximum element of an array of size *N* using 4 processors in parallel, where processor P0 initially contains the array elements, you can follow this algorithm:

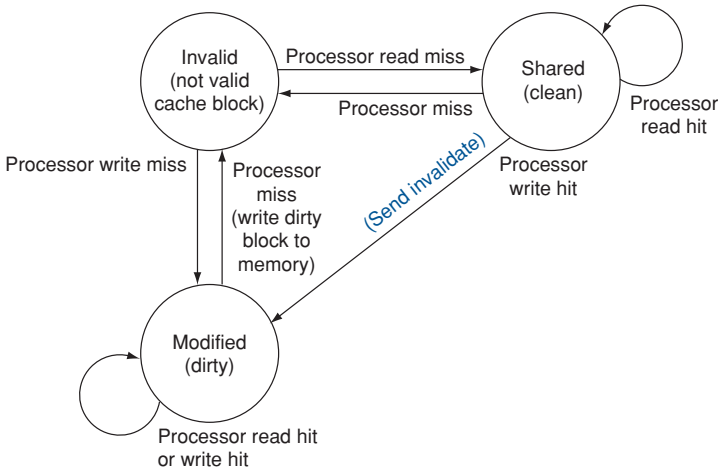
(1) Send N/4 elements to each of the other processors
(2) Have all processors find the maximum of their N/4 elements
(3) Have processors P0 and P1 communicate to find the larger of their individual maximum elements; in parallel, have P2 and P3 do the same thing.
(4) Compare the results from step (3) to find the overall maximum element.

Alternatively, you could just do the entire computation on P0. We call this the sequential algorithm.

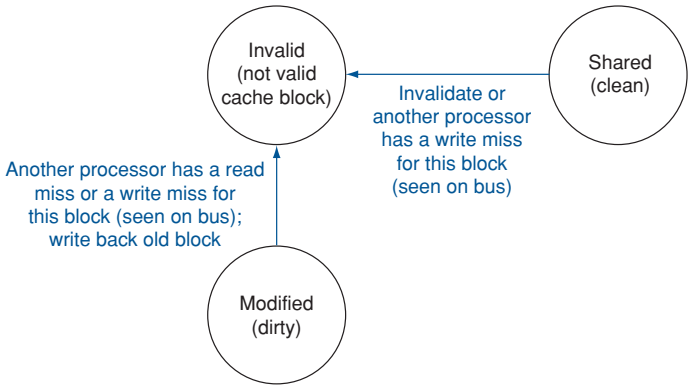
If it takes X cycles to send/receive one element between processors, and Y cycles to compare two elements, how much time does the sequential algorithm take?

How much time does each step of the parallel algorithm take?

Using the above results, what relationship between X and Y must be true for the parallel algorithm to be preferable to the sequential algorithm? (e.g. X < Y)



a. Cache state transitions using signals from the processor



b. Cache state transitions using signals from the bus

FIGURE 9.3.4 A write-invalidate cache coherence protocol. Part a of the diagram shows state transitions based on actions of the processor associated with this cache; part b shows transitions based on actions of other processors seen as operations on the bus. There is really only one state machine in a cache block, although there are two represented here to clarify when a transition occurs. The black arrows and actions specified in black text would be found in caches without coherency; the colored arrows and actions are added to achieve cache coherency.