Name _____

Computer System Architecture
6.823 Quiz #3
April 25th, 2014
Professors Daniel Sanchez and Joel Emer

This is a closed book, closed notes exam.

80 Minutes
16 Pages

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.

Part A	_____	45 Points
Part B	_____	35 Points
Part C	_____	20 Points
TOTAL	**_____**	**100 Points**

Name _____

Part A: Let's Talk About Loads (45 pts)

Consider the following code sequence:

```
…
I1:  DIV R3, R1, 8
I2:  BNEZ R9, Somewhere
I3:  ST R2, 0(R3)
I4:  LD R1, 8(R4)
I5:  ADD R5, R1, 8
I6:  SUB R10, R6, R7
I7:  MUL R8, R9, R10
I8:  BEQZ R8, Somewhere else
…
```

We will explore how this program behaves on different architectural styles. In all cases, assume the following execution latencies:

- `ADD, SUB`: 2 cycles
- `BNEZ, BEQZ`: 2 cycles
- `LD`: 2 cycles if cache hit, 8 cycles if miss
- `MUL`: 5 cycles
- `DIV`: 10 cycles

Additionally, the `LD (I4)` in this sequence *misses* in the data cache and therefore has a long latency of 8 cycles.

Assume that the branch at `I2` is not taken and fetch and decode never stall (e.g., by missing on the instruction cache or the BTB). Also assume that there are no structural hazards.

Question 1 (5 points):

Loads are often a bottleneck in processor performance, and as such compilers will try to move the loads as early as possible in the program to "hide" their latency. However, in the preceding code sequence, an optimizing compiler *cannot* move the load earlier in the program. Explain why in one or two sentences.

Name _____

Question 2 (6 points):

Show how this program would work on a single-issue in-order pipeline that tracks dependencies with a simple scoreboard. Instructions are issued (i.e., dispatched for execution) in order, but can complete out of order. Assume infinite functional units and full bypassing. Fill in the remainder of the table below.

Instruction	Issue Cycle	Completion Cycle
`I1: DIV R3, R1, 8`	1	11
`I2: BNEZ R9`	2	4
`I3: ST R2, 0(R3)`	11	n/a
`I4: LD R1, 8(R4)`	12	20
`I5: ADD R5, R1, 8`	20	
`I6: SUB R10, R6, R7`		
`I7: MUL R8, R9, R10`		
`I8: BEQZ R8`		

Name _____

Question 3 (6 points):

Assuming a single-issue out-of-order processor, show at which cycles instructions are issued (i.e., dispatched for execution) and complete. Assume that instructions are dispatched in program order if multiple are ready in the same cycle, and *do not speculate on data dependencies.* Again assume infinite functional units and full bypassing.

Instruction	Issue Cycle	Completion Cycle
`I1: DIV R3, R1, 8`	1	11
`I2: BNEZ R9`		
`I3: ST R2, 0(R3)`		
`I4: LD R1, 8(R4)`		
`I5: ADD R5, R1, 8`		
`I6: SUB R10, R6, R7`		
`I7: MUL R8, R9, R10`		
`I8: BEQZ R8`		

In one or two sentences, what is the advantage of an out-of-order architecture vs. the in-order pipeline for this code sequence?

Question 4 (5 points):

Suppose the out-of-order processor chose to execute the load first, *before all other instructions in the code sequence*. What events could cause the load to be aborted, and what mechanisms are required to detect mis-speculation and roll back? Ignore exceptions in your answer.

Name _____

Question 5 (6 points):

Write VLIW code for this instruction sequence, assuming that the VLIW format is:

Memory operation	ALU operation	ALU operation / Branch

Try to make your VLIW code as efficient as possible, including re-ordering any instructions that do not have dependencies. For this VLIW code just use standard MIPS instructions to fill slots without predication or new, VLIW-specific instructions. (That is, simply schedule the instructions already provided.) Assume that the VLIW architecture has a scoreboard that stalls when a result is used before it is ready (e.g., on a cache miss).

In one or two sentences, what is the advantage/disadvantage of a VLIW architecture for this code sequence vs. the out-of-order pipeline?

Josh Fisher points out that if it has a scoreboard, it's not a *true* VLIW. How would the code sequence change if we didn't have a scoreboard?

Name _____

Question 6 (5 points):

VLIW architectures rely heavily on the compiler to expose instruction-level parallelism in the program, so hiding load latency is a major challenge. VLIW compilers developed a technique called *trace scheduling* that merges multiple basic blocks into a single code sequence with software checks to ensure correctness. We profile our program and find that the first branch (`I2`) is almost never taken, so merging both basic blocks is a good idea.

If we use trace scheduling to move the load (`I4`) to be the *first* instruction, what conditions must software check to ensure correctness of the load for this code sequence? Ignore exceptions in your answer.

Name _____

Question 7 (6 points):

To mitigate load latency, you decide to implement a prefetch instruction. `PREFETCH Imm(rs)` takes a single argument, an address, and *hints* to the processor that the given address may be used soon. Crucially, `PREFETCH` is side-effect free—the processor can choose to ignore `PREFETCH`'s without affecting program behavior.

Now consider the following simplified code sequence:

```
DIV R3, R1, 8
ST R2, 0(R3)
LD R1, 8(R4)
ADD R5, R1, 8
```

The diagram below shows how this code executes on an in-order issue processor with scoreboarding. Show how performance can be improved using `PREFETCH`.

Cycle	In-order	In-order w/ Prefetch
1	DIV	
2		
3		
4		
5		
6		
7		
8		
9		
10		
11	ST	
12	LD	
13		
14		
15		
16		
17		
18		
19		
20	ADD	
21		
22	Complete	

Name _____

Question 8 (6 points):

In lecture we discussed an alternative instruction, "load-speculate":

```
LD.S rt, Imm(rs)
```

Load-speculate will fetch the value from memory but if the access faults it instead returns zero and does not cause an exception. Unlike prefetch, it gives not just the address but the source address *and* the destination register, which receives a value from memory. A load-speculate is followed in the program by a "load-check":

```
CHK.S rt, cleanup
```

Load-check checks if the register was written by a `LD.S` that should have caused an exception (e.g., due to a page fault). If it was, then `CHK.S` branches to somewhere else to service the exception and handle any necessary cleanup. `CHK.S` executes in 1 cycle.

Show how to use `LD.S/CHK.S` to speed up the code even further than was possible with `PREFETCH`. Assume scoreboarding and infinite functional units. Assume that in this case the compiler knows that the load (`I4`) can be scheduled before the store (`I3`) safely. Do not show cleanup code.

Cycle	In-order	In-order+`LD.S`+`CHK.S`
1	`DIV`	
2	⇓	
3		
4		
5		
6		
7		
8		
9		
10		
11	`ST`	
12	`LD`	
13	⇓	
14		
15		
16		
17		
18		
19		
20	`ADD`	
21	⇓	
22	Complete	

Name _____

Part B: Exploiting Parallelism (35 points)

Consider the following C code sequence:

```
const int size = 64 * 1024;
int a[SIZE], b[SIZE], c[SIZE];
for (int i = 0; i < SIZE; i++) {
     if (a[i] > b[i]) {
          c[i] = a[i] + b[i];
     }
}
```

This is a repetitive computation with a simple dependency graph. If we look at the MIPS assembly code, we see that a large percentage of the instructions are doing bookkeeping. We'd like to reduce this overhead.

```
          // R1 points to a, R2 points to b, R3 points to c
          // R6 is i
          ADD R6, R0, SIZE
Loop:     LD R4, 0(R1)
          LD R5, 0(R2)
          SUB R8, R4, R5
          BGEZ R8, Skip
          ADD R4, R5, R4
          ST R4, 0(R3)
Skip:     ADD R1, R1, 4
          ADD R2, R2, 4
          ADD R3, R3, 4
          SUB R6, R6, 1
          BNEZ R6, Loop
```

Question 1 (5 points):

Circle the MIPS instructions in the assembly above that perform "useful work" rather than bookkeeping.

Name _____

Question 2 (5 points):

If the loads in the preceding code take four cycles, then this code sequence will stall and performance will suffer. Explain how an in-order, fine-grain multithreaded processor with two threads could mitigate this effect?

How would the program need to change for multhreading? (You do *not* need to write the code.)

Name _____

Question 3 (10 points):

An alternative approach is to hide the load latency within a single thread by using loop unrolling. Loads take four cycles and adds take one cycle. Write a loop unrolled VLIW version of the preceding code using the same VLIW instruction format as in Part A:

Memory operation	ALU operation	ALU operation / Branch

Unroll the fewest number of loop iterations necessary to cover the load's latency. Whatever degree of unrolling you choose, assume it divides the array size. Also assume that predication is allowed:

`(p1) instruction` executes the instruction if predicate register `p1` is set.
`cmp.gt p1, r1, r2` sets predicate register `p1` if `r1` is greater than `r2`.

Finally, `R1` points to `a`, `R2` points to `b`, `R3` points to `c`, and `R6` is `i`.

NOTE: The back of this page has additional space.

Name ______________________________

Additional space:

Name _____

Question 4 (10 points):

Write a vector version using vector instructions and the vector mask register. Assume that the vector machine can do up to 64 operations per instruction, and note that `SIZE` is a multiple of 64.

`VLR` register stores the vector length.
`LV v1, r1, Imm` loads vector register `v1` with memory starting at address `r1` and stride `Imm`. `SV v1, r1, Imm` behaves similarly for stores.
`ADDV v1, v2, v3` adds `v2` and `v3` and puts the result in `v1`.
`SGTVV v1, v2` sets the vector mask register for each vector element in `v1` greater than the corresponding element in `v2` (mask set means the operation is enabled).
`CVM` resets the vector mask register (turns on all elements).

```
          // R1 points to a, R2 points to b, R3 points to c
          // R6 is i
          ADD R6, R0, SIZE
          LI VLR, 64
Loop:
```

```
Skip:     ADD R1, R1, 64*4
          ADD R2, R2, 64*4
          ADD R3, R3, 64*4
          SUB R6, R6, 64
          BNEZ R6, Loop
```

Name _____

Question 5 (5 points):

Is this program easy to map to GPUs? What inefficiencies may arise? Explain your answer in one or two sentences.

Name _____

Part C: TLBs in Outer Space (20 points):

You observe that page table entries exhibit a lot of spatial locality (consecutive pages are often accessed). To increase your processor's TLB hit rate, you increase the block size of your TLB to hold several page table entries:

- Each TLB entry holds 4 page table entries (PTEs).
- Each page table entry is 19 bits containing:
    - A 16-bit physical page number (PPN).
    - Three protection bits:
        - Readable bit
        - Writeable bit
        - Used bit (for page replacement)

NASA has hired you to design a processor for a manned mission to Mars. You need to make sure your TLB works in the hostile environment of outer space.

Question 1 (10 points):

One of NASA's programs executes the following memory operations:

Cycle	Operation	Instruction is ACE?
0	Load from page A (TLB miss)	Yes
15	Load from page A	Yes
20	Load from page A	Yes
30	Load from page A	No
40	Load from page B (TLB miss, evicting block containing A from the TLB)	Yes

For which cycles is the PPN for page A itself ACE?

Cycle	0-15	15-20	20-30	30-40
Is ACE?				

What is the AVF of the PPNs in the TLB block (consider all PPNs in a block, but not tag or protection bits) over cycles 0-40?

Name _____

Question 2 (10 points):

Discuss the AVF of each protection bit (readable, writeable, used) for PTEs in the TLB.

Consider the tag of the TLB. Is the AVF of these tag bits relatively high or relatively low? Explain why in one or two sentences.

You want to protect the data in the TLB. Would you recommend using parity or SECDED (single error correction, double error detection)? Explain your answer in one or two sentences.