Computer Architecture and Engineering

CS152 Quiz #4

April 13, 2010

Professor Krste Asanovic

Name:____ANSWER KEY_____

This is a closed book, closed notes exam.

80 Minutes

8 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.**
- **Please write your name on every page in the quiz.**
- **You must not discuss a quiz's contents with students who have not yet taken the quiz. If you have inadvertently been exposed to the quiz prior to taking it, you must tell the instructor or TA.**
- **You will receive no credit for selecting multiple-choice answers without giving explanations if the instructions ask you to explain your choice.**

Writing name on each sheet	**_____**	**1 Point**
Question 1	**_____**	**24 Points**
Question 2	**_____**	**19 Points**
Question 3	**_____**	**18 Points**
Question 4	**_____**	**18 Points**
TOTAL	**_____**	**80 Points**

NAME:_____

Problem Q4.1: Static Scheduling 24 points

In this problem, we consider the execution of a code segment on a single-issue, in-order processor and a VLIW processor. The code we consider is the SAXPY kernel, which scales a vector X by a constant A, adding this quantity to a vector Y.

```
I1    loop: ld   f0, 0(r1)     # for(i = 0; i < N; i++)
I2          fmul f2, f0, f1    #   Y[i] = Y[i] + A*X[i];
I3          ld   f3, 0(r2)
I4          fadd f4, f2, f3
I5          st   f4, 0(r2)
I6          addi r1, r1, 4
I7          addi r2, r2, 4
I8          bne  r1, r3, loop
```

Problem Q4.1.A 4 points

Assume that we execute this code segment on a single-issue, in-order processor with perfect branch target prediction and full bypassing. ALU operations have a one-cycle latency, loads have a three-cycle latency, and floating-point operations have a four-cycle latency (even when bypassed to store instructions). How many cycles will the processor stall per loop iteration?

There's a 2-cycle stall due to the RAW hazard between I1 and I2; a 2-cycle stall between I3 and I4 (which overlaps a 2-cycle stall between I2 and I4; don't double-count!); and a 3-cycle stall between I4 and I5.

_____7_____ Stall Cycles per Iteration

Problem Q4.1.B 6 points

Reschedule the code to minimize the number of stall cycles, but do not software pipeline or unroll the loop. Now, how many cycles will the processor stall per loop iteration?

```
I1    loop: ld   f0, 0(r1)
I2          fmul f2, f0, f1
I3          ld   f3, 0(r2)
I4          fadd f4, f2, f3
I6          addi r1, r1, 4
I7          addi r2, r2, 4
I5          st   f4, -4(r2)
I8          bne  r1, r3, loop
```

One of many possible solution is to move the addis between the fadd and the st, but don't forget to change the store's displacement to account for the new value of r2!

_____5_____ Stall Cycles per Iteration

NAME:_____

Problem Q4.1.C **8 points**

Software pipeline the loop to eliminate all stalls for the processor in Q4.1.A. You may omit the prolog and epilog code.

A simple algorithm is to reverse the data flow of the loop, turning RAW hazards into WAR hazards. This loop has a 4-stage software pipeline, with the load of X being from iteration i, the load of Y and the fadd being from iteration i-1, the fmul being from iteration i-2, and the store being from iteration i-3.

```
I1    loop: st   f4, -12(r2)
I2          fadd f4, f2, f3
I3          ld   f3, -4(r2)
I4          fmul f2, f0, f1
I5          ld   f0, 0(r1)
I6          addi r1, r1, 4
I7          addi r2, r2, 4
I8          bne  r1, r3, loop
```

Problem Q4.1.D **6 points**

Now, software pipeline the loop for a VLIW machine with the same functional unit latencies as the processor in Q4.1.A. Omit prolog and epilog code. The VLIW processor has one ALU/branch unit, one memory unit, and one floating-point unit. What speedup does the VLIW machine offer compared to the single-issue processor when both run software-pipelined code?

You can just schedule the SW pipelined code from above onto this VLIW machine. Unfortunately, the fadd->fmul latency still isn't covered, so we need NOPs; to fix this, we could have unrolled the loop once, but that wasn't required.

Cycle	ALU/Branch Unit	Memory Unit	Floating-Point Unit
1	addi r1,r1,4	st f4,-12(r2)	fadd f4,f2,f3
2	addi r2,r2,4	ld f3,-4(r2)	fmul f2,f0,f1
3	bne r1,r3,loop	ld f0,-4(r1)	*
4	*	*	*
5	*	*	*
6			
7			
8			
9			
10			

_____8/5_____ Speedup

Problem Q4.2: Vectors 19 points

In this problem, we analyze the performance of the SAXPY kernel from Q4.1 on a vector machine. The baseline vector processor we consider has the following features:

- 32 elements per vector register
- 32 vector registers
- 4 lanes
- One integer ALU per lane (one cycle latency)
- One FPU per lane (four cycle latency, pipelined)
- One memory unit per lane (four cycle latency, pipelined)
- No chaining
- A separate five-stage pipeline for scalar instructions. (The scalar unit does not interlock if the vector unit is stalled.)

Problem Q4.2.A 9 points

Vectorize the original code from Q4.1, assuming the X and Y arrays do not overlap. You may assume that N is a multiple of the vector length, and that the vector length register has already been set accordingly.

The code is nearly identical to the original code of Q4.1, except we use vector registers and increment the addresses by the vector length in bytes (32x4).

```
I1    loop: LV     V0, 0(R1)
I2          FMULVS V2, V0, F1
I3          LV     V3, 0(R2)
I4          FADDV  V4, V2, V3
I5          SV     V4, 0(R2)
I6          ADDI   R1, R1, 128
I7          ADDI   R2, R2, 128
I8          BNE    R1, R3, loop
```

NAME:_____

Problem Q4.2.B **5** points

When executing your vectorized code on the baseline vector processor, how many floating-point operations complete per cycle on average? (Note that without chaining, a dependent vector instruction cannot begin execution until its source instruction has completed writeback, so the functional unit latencies are effectively one cycle longer than stated.)

Each loop iteration performs 64 flops. The easiest way to determine how many cycles it takes is to schedule the code. There's a 13-cycle latency between dependent instruction issue (32/4 cycles to compute, 4 cycles for FU latency, 1 cycle for writeback). The scalar instructions are overlapped easily with the SV, so they aren't shown. Don't forget that two memory ops can't be executing at once, so there's a structural hazard between SV and the next LV!

Instruction	Iter 1 begins	Iter 2 begins
LV	0	48
FMULVS	13	…
LV	14	
ADDV	27	
SV	40	

So, we have 64 flops / 48 cycles (4/3 flops/cycle).

Problem Q4.2.C **5** points

Suppose we add chaining support to the vector processor. Now, how many floating-point operations complete per cycle on average?

Now, there's only a 4 cycle FU latency between dependent instructions. Be careful not to schedule two memory operations to be running concurrently!

Instruction	Iter 1 begins	Iter 2 begins
LV	0	24 (structural)
FMULVS	4 (RAW)	…
LV	8 (structural hazard)	
ADDV	12 (RAW)	
SV	16 (RAW)	

So, we have 64 flops / 24 cycles (8/3 flops/cycle).

NAME:_____

Problem Q4.3: Multithreading 18 points

In this problem, we once again consider the SAXPY kernel from Q4.1, analyzing its performance on a fine-grained multithreaded in-order processor.

Aside from threading, the processor we consider is identical to the pipeline from Q4.1.A. It has perfect branch target prediction and full bypassing. ALU operations have a one-cycle latency, loads have a three-cycle latency, and floating-point operations have a four-cycle latency.

Problem Q4.3.A 6 points

The first implementation of multithreading we consider employs fixed round-robin scheduling: every cycle, the processor fetches an instruction from a different thread. What is the minimum number of SAXPY threads needed to eliminate all stalls? Justify your answer.

We need to be able to cover the 3 stall cycles (with no intervening instructions), so we need 3 additional threads, for a total of 4.

_____4_____ Threads

Problem Q4.3.B 6 points

Now, assume that the thread scheduler is aware of data hazards and can switch threads when an interlock would have occurred due to a RAW hazard. What is the minimum number of SAXPY threads needed to eliminate stalls? Justify your answer.

The most straightforward way to arrive at the correct answer of 3 threads is to construct a stall-free schedule using 3 threads (which is a straightforward exercise), then convincingly argue that it's not possible to create such a schedule with only 2 threads.

_____3_____ Threads

NAME:_____

Problem Q4.3.C **6** points

Suppose main memory latency is 200 cycles, so we decide to add a write-allocate, write-back data cache to the multithreaded processor from Q4.3.B. We want to execute the SAXPY code using 8 threads. Each thread will operate on a contiguous chunk of the X and Y vectors. Assuming that the X and Y arrays are too large to fit in cache, circle the cache parameter(s) that will be most critical to performance. For each parameter that you circle, what value do you recommend?

Number of Sets _____ Sets

Associativity _____16_____ Ways

Line Size _____8_____ Words

We want an associativity of 16, because we have 16 streams occurring at once (8 threads times 2 streams per thread, namely the X and Y arrays). Credit was given for 8 ways, too; depending on the addresses of X and Y and the particular thread interleavings, 8 may be sufficient.

Line size is tricky! To cover a memory latency of 200 cycles, each of the 8 threads must average 25 instructions without a cache miss. We'll round that up to 4 loop iterations. In 4 loop iterations, we access 8 unique words. Since the streams are unit stride, we'll average one cache miss per 4 loop iterations with 8-word cache lines.

Problem Q4.4: Iron Law of Processor Performance (Short Answer) [18 points]

Consider a five-stage in-order RISC pipeline as a baseline. Mark whether the following modifications will cause each of the categories to **increase, decrease**, or whether the modification will have **no effect**. **Explain your reasoning** to receive credit.

	Instructions / Program	Cycles / Instruction	Seconds / Cycle
Widening the processor to dual-issue superscalar	No effect: this is purely a microarchitectural change.	Decrease: we can now execute up to twice as many instructions in the same amount of time.	Increase: additional register file ports and bypass paths may be on the critical path
Migrating to a traditional VLIW architecture	Net decrease, because single instructions express multiple operations. Total code size is much greater, though (NOPs).	Decrease: traditional VLIW machines lack interlocks, so CPI=1.	No effect, to first order: all new hardware executes in parallel; no dependence cross-checks.
Multithreading the processor	No effect for multiprogrammed workloads (but slight increase in the OS code); slight increase if we parallelize a program. For credit, had to mention the intended workload.	Decrease: we can avoid interlocking on some hazards by switching threads.	Increase: additional architected state, namely a larger register file, may be on the critical path.
Adding a single-lane vector unit	Decrease (if program is vectorizable): one vector instruction encodes several operations.	Increase: vector instrucutions will take several cycles to execute on a single-lane implementation	No effect: the 5-stage pipeline will still contain the critical path. (Or, decrease: the vector unit can be deeply pipelined and run at a higher clock rate than the control processor.)

END OF QUIZ