Computer Architecture and Engineering

CS152 Quiz #1

February 12th, 2009

Professor Krste Asanovic

Name: ANSWER KEY

This is a closed book, closed notes exam.

80 Minutes

10 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 get 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	_____	26 Points
Question 2	_____	31 Points
Question 3	_____	10 Points
Question 4	_____	12 Points
TOTAL	_____	80 Points

NAME: _____

Problem Q.1: Microprogramming Bus-Based Architectures

[26 points]

In this problem, we explore microprogramming by writing microcode for the bus-based implementation of the MIPS machine described in Handout #1 (Bus-Based MIPS Implementation), which we have included at the end of this quiz for your reference.

You are going to implement a fetch-and-add instruction in microcode. Fetch-and-add reads a memory location, adds to it the contents of a source register, stores the sum back to the same memory location, and returns what was originally in memory in the destination register. (This instruction is commonly used to provide synchronization in multiprocessor systems.) The instruction has the following format:

XAdd r_d, r_s

XAdd performs the following operation:

$$temp \leftarrow M[r_d] + r_s$$

$$r_s \leftarrow M[r_d]$$

$$M[r_d] \leftarrow temp$$

Fill in Worksheet Q1-1 with the microcode for XAdd. Use *don't cares* (*) for fields where it is safe to use don't cares. Study the hardware description well, and make sure all your microinstructions are legal. To further simplify this problem, *ignore* the busy signal, and assume that the memory is as fast as the register file.

Please comment your code clearly. If the pseudo-code for a line does not fit in the space provided, or if you have additional comments, you may write in the margins as long as you do it neatly. Your code should exhibit "clean" behavior and not modify any registers (except r_s) in the course of executing the instruction. You will receive credit for elegance and efficiency.

Finally, make sure that the instruction fetches the next instruction (i.e., by doing a microbranch to FETCH0 as discussed in the Handout).

NAME: _____

State	PseudoCode	ld IR	Reg Sel	Reg Wr	en Reg	ld A	ld B	ALUOp	en ALU	ld MA	Mem Wr	en Mem	Ex Sel	en Imm	μB r	Next State
FETCH0:	MA <- PC; A <- PC	0	PC	0	1	1	*	*	0	1	*	0	*	0	N	*
	IR <- Mem	1	*	*	0	0	*	*	0	0	0	1	*	0	N	*
	PC <- A+4	0	PC	1	1	0	*	INC_A_4	1	*	*	0	*	0	D	*
. . .																
NOP0:	microbranch back to FETCH0	0	*	*	0	*	*	*	0	*	*	0	*	0	J	FETCH0
XADD0:	MA <- R[rd]	0	rd	0	1	*	*	*	0	1	*	0	*	0	N	*
	B <- R[rs]	0	rs	0	1	*	1	*	0	0	*	0	*	0	N	*
	A <- Mem A <- R[rs]	*	rs	1	1	1	0	*	0	0	0	1	*	0	N	*
	Mem <- A+B	*	*	*	0	*	*	ADD	1	*	1	1	*	0	J	FETCH0

There are many valid ways to solve this problem. The problem description both said to use don't cares (*) and to use "elegance and efficiency" so some (but not too many) points were deducted for extra cycles or not using don't cares everywhere possible.

With these tables, you can think of the signals taking effect at the next rising edge, which is the next row in the table (except with jumps). For example in the last row when writing to memory, the contents of MA don't matter afterwards. The last row sets ldMA to *, because the next rising edge memory will grab the old address from MA, and at that same rising edge the don't care will take affect and MA may or may not load a value for the next cycle.

Another important detail is that the IR is important because it contains things like rs, rd, and immediates. Without it the register file wouldn't know where things like R[rs] are. It needs to be kept around at least until it is no longer needed. In the example above, it is the last row where rs is used.

Actual wrong values in the table (like *'s where it should be 0 or 1) had larger deductions because they result in incorrect rather than just less efficient operation.

NAME: _____

Problem Q2: Mem-ALU Pipeline

[31 points]

In this problem, we consider further modifications to the fully bypassed 5-stage MIPS processor pipeline presented in Lecture 3 and Problem Set 1. We will re-order the stages so the Execute (ALU) stage comes after the Memory stage. After this change we will only support register indirect addressing. This change will let us use the contents of memory as one of the operands for arithmetic operations. For example, something like the CAdd instruction could be implemented:

CAdd r_d, r_{s0}, r_{s1}

CAdd performs the following operation:

$$r_d \leftarrow M[r_{s0}] + r_{s1}$$

In this problem, assume that the control logic is optimized to stall only when necessary, and that the pipeline is fully bypassed. ***You may ignore branch and jump instructions in this problem.***

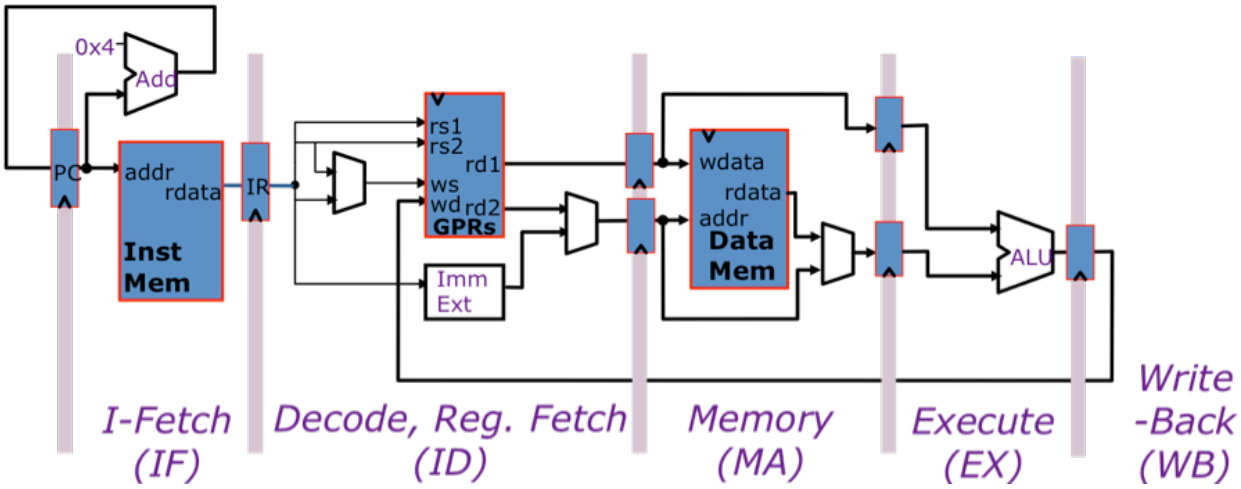

Figure 2-A. Mem-ALU Pipeline

NAME: _____

Problem Q.2.A-B

Hazards
[16 points]

Besides enabling the CAdd instruction, this pipeline modification will change which hazards exist in the pipeline. We want to compare the pipeline from lecture and the problem set (old) to this modified pipeline with the ALU after memory (new). Assume both the old and new pipelines are fully bypassed with correct control logic. *For each problem below, give a sample instruction sequence to clarify your explanation.*

Q.2.A – Hazards Removed [8 points]
Give an example where the old pipeline will stall, but the new pipeline will not.

A load followed by an arithmetic instruction in the next two cycles would have had to stall, but not anymore.

```
LW    r2, 0(r3)        or even        LW    r2, 0(r3)
ADD   r4, r2, r5                      ADD   r4, r5, r6
                                      ADD   r4, r4, r2
```

Q.2.B – Hazards Added [8 points]
Give an example where the new pipeline will stall, but the old pipeline will not.

An arithmetic instruction followed by a dependent memory access.

```
ADD   r2, r3, r4
LW    r5, 0(r2)
```

NOTE: Branches were not a valid solution to this problem. On a branch mis-predict, the instructions following the branch are flushed, not stalled. The problem description also said to not consider branches and jumps.

NAME: _____

Problem Q.2.C-D

Memory Addressing
[15 points]

Q.2.C - Hardware Changes [7 points]
As an architect you want to maintain full compatibility for the old ISA, including support for register-immediate indirect addressing (e.g., LW r2, 8(r3)). How would you modify the new pipeline (and associated control logic) to support this efficiently?

Option A: Include an adder at end of the decode stage (register file) or at the beginning of the memory access stage to compute the effective address. This will increase the cycle time but keep the same CPI.

Option B: Include an adder in between the decode stage (register file) and the memory access stage to compute the effective address. This will keep the same cycle time but increase the CPI because there will be more hazards.

Option C: Add an adder in parallel in the memory access stage. When an address needs to be calculated the memory stage will take two cycles (the result of the adder will be looped back. This will also keep the same cycle time but increase the CPI because there will be more hazards.

Option D: Add a bypass from the output of the ALU back to beginning of the memory stage to route the computed address. This solution adds very little hardware, keeps the same cycle time, but adds complexity and increases CPI with the stage re-use and additional hazards caused by more stages.

There are more ways to solve this.

Q.2.D – Implementing Precise Exceptions [8 points]
Describe a problem that might arise when implementing precise exceptions and propose a simple solution. Hint: consider store instructions…

The problem is that it is hard to draw a single commit line for processing exceptions with the current pipeline because if it is drawn at the WB stage a store (following an exception causing instruction) could still write to memory before being flushed and change architectural state. If the commit line is put in the MA stage, an arithmetic exception could happen after the commit point.

Option A: Buffer stores until all instructions ahead of it are guarantied to commit without exception.

Option B: Kill (undo) write if exception detected. This is worth only 6 pts as this will take much more time.

There are more ways to solve this. If the response correctly described the problem, 4 pts were given. If some notion of the problem with stores was given, 2 pts were awarded.

NAME: _____

Problem Q3: ISA Visible? (Short Yes/No Questions)

[10 points]

The following questions describe two variants of a processor that are otherwise identical. In each case, circle "**Yes**" if the difference should be visible to software in the ISA, and circle "**No**" otherwise. You must also **briefly explain** your reasoning and any assumptions you are making. *Ignore differences in performance.*

Problem Q3.A **More stages**

Pipelined processor A has more stages than pipelined Processor B, but both have full bypassing.

No. Pipeline implementations are normally below the ISA. Sometimes it appears to affect it with things like delay slots, but full bypassing and the problem's claim that the processors are otherwise equal implies that they will have the same amount of delay slots (if any).

Problem Q3.B **Control type**

Processor A uses microcoded control while Processor B uses hardwired control.

No. This is an implementation detail, that shouldn't be ISA visible.

Problem Q3.C **CISC/RISC**

Processor A is considered to be a CISC machine while Processor B is a RISC machine.

Yes. One of the main ways RISC and CISC machines are distinguished is by the types of instructions they provide. Modern implementations could even seem somewhat similar, but the abstractions they provide from above will look different.

Problem Q3.D **Microcode type**

Machine A has very vertical microcode while machine B has a more horizontal microcode.

No. Once again this is an implementation detail (like Q3.B), that shouldn't be ISA visible.

Problem Q3.E **Stack depth**

Stack machine A has more physical registers to implement its stack than stack machine B.

No. This shouldn't be visible if the ISA says the machine will automatically spill into memory if the stack fills up.

One could argue the case for yes if the system provides no mechanism to automatically spill because then the programmer will need to structure their code to not overflow.

Problem Q.4: Iron Law of Processor Performance (Short Answer) [12 points]

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
Dividing up a pipeline stage into two stages	**No effect** if pipeline adds no new hazards that must be solved with programmer added NOP's (like delay slots). **Increase** if it does add new need for NOP's to be added.	**Increase** since more stages will cause more hazards and thus more stalls to deal with them.	**Decrease** since this should reduce critical path. **No effect** if the stage split is not on the critical path.
Adding a complex instruction	**Decrease** if the added instruction can replace a sequence of instructions. **No effect** if it is unusable.	**Increase** if implementing the instruction means adding or re-using stages. **No effect** if the number of cycles is kept constant but it just lengthens the logic in one stage.	**Increase** since more logic and thus longer critical path. **No effect** if it is implemented by more or re-used stages but each stage gets no longer.
Reducing the number of bypass paths	**No effect** if the processor can detect the hazards and interlock automatically. **Increase** if the programmer needs to add NOP's for the hazards.	**Increase** since more cycles will be spent waiting from the added NOP's.	**Decrease** since bypasses are almost always the critical path. **No effect** if the bypass removed happens to not be on the critical path. Meaning another bypass or part of the system is.
Improving memory access speed	**No effect** since instructions make no assumption about memory speed.	**Decrease** if access to memory is pipelined (>1 cycle) since it will now take less cycles. **No effect** if memory access is done in a single cycle.	**Decrease** if memory access is on the critical path or memory was 1 cycle. **No effect** if memory is pipelined and just takes less cycles.

Notice how for this problem in many cases more than one way could be argued if the reasoning was there. There are 12 boxes so 1 point was given per box, and a full explanation was needed to get the point (with no partial credit per box). The boxes were too small, but some explanations were on the right track but really just not sufficient and thus not given the point.

Comments:
Splitting up pipeline stage, CPI – Many people wrote something to like "more stages more cycles." It is important to remember that CPI is a rate, not a latency. The reason that adding a stage will increase CPI is that it will increase the number of hazards and thus the likelihood of stall cycles. In an ideal pipeline with no hazards, adding a stage will not increase the CPI.

Adding a complex instruction, CPI & cycle time – This one varies depending on how the complex instruction is implemented. If the complex instruction is implemented by adding or re-using stages it could be done without changing the cycle time. The cycle time could increase even if this is done because of the additional logic to control and route. Alternatively, the complex instruction could be implemented without additional cycles, but this will almost certainly mean that more logic was added somewhere, increasing the critical path length and increasing the cycle time.

Reducing # of bypasses, cycle time – Many people said no effect since it wasn't on the critical path. Such a claim (that its not on the critical path) should be clearly stated or even justified. Bypasses are usually the critical path, so much so that some CPU's even have to pipeline their bypass paths to get reasonable cycle times.

END OF QUIZ