Computer Architecture and Engineering

CS152 Quiz #1

February 16th, 2010

Professor Krste Asanovic

Name: ANSWER KEY

This is a closed book, closed notes exam.

80 Minutes

9 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	**_____**	**29 Points**
Question 2	**_____**	**32 Points**
Question 3	**_____**	**18 Points**
TOTAL	**_____**	**80 Points**

NAME: _____

Problem Q.1: Microprogramming Bus-Based Architectures

[29 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 memory-indirect jump-and-link instruction in microcode. JALM writes the link register with the value PC+4, then fetches a word from memory, then jumps to the address held in the memory location. (This instruction could be used to accelerate virtual function calls in languages like C++ or Java.) The instruction has the following format:

JALM offset (r_s)

JALM performs the following operation:

temp ← R[rs] + sExt16(immediate)

R[31] ← PC+4

PC ← M[temp]

Q.1.A – Implement JALM [18 points]

Fill in Worksheet Q1-1 with the microcode for JALM. 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. Furthermore, assume no branch delay slots.

For this part of the problem, your implementation may make the simplifying assumption that **rs** is not equal to 31, i.e. that the order of the first two operations may be reversed.

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 ISA-visible registers (except the PC and the link register) in the course of executing the instruction. You will receive credit for elegance and efficiency.

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

Q.1.B – Implement JALM's Corner Case [11 points]

Now, fill in Worksheet Q1-2 with microcode for JALM that will work correctly even if **rs** is equal to 31. Compared to the simplified implementation from Q.1.A, how many additional cycles does your correct implementation take to execute?

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	μBr	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
JALM0:	R[31] <- A+4	0	31	1	1	*	*	INC_A_4	1	*	*	0	*	0	N	*
	A <- R[rs]	0	rs	0	1	1	*	*	0	*	*	0	*	0	N	*
	B <- sExt16(imm)	*	*	*	0	0	1	*	0	*	*	0	s16	1	N	*
	MA <- A+B	*	*	*	0	*	*	ADD	1	1	*	0	*	0	N	*
	PC <- Mem[MA]	*	PC	1	1	*	*	*	0	*	0	1	*	0	J	FETCH0

Worksheet Q1-1

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." With that in mind, one point was deducted per extra cycle your implementation took, and up to two points were deducted for not using don't-cares everywhere possible. Correctness issues cost two or more points apiece.

Some common errors:

Setting the link register to PC+8 (i.e. failing to notice that PC has already been incremented by 4)
Not taking advantage of the fact that A already contained PC
Not using don't-cares on ldIR, or employing don't-cares a cycle late

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
JALM0:	B <- R[rs]	0	rs	0	1	0	1	*	0	*	*	0	*	0	N	*
	R[31] <- A+4	0	31	1	1	*	0	INC_A_4	1	*	*	0	*	0	N	*
	A <- sExt16(imm)	*	*	*	0	1	0	*	0	*	*	0	s16	1	N	*
	MA <- A+B	*	*	*	0	*	*	ADD	1	1	*	0	*	0	N	*
	PC <- Mem[MA]	*	PC	1	1	*	*	*	0	*	0	1	*	0	J	FETCH0

Worksheet Q1-2

NAME: _____

Problem Q2: 6-Stage Pipeline

[32 points]

In this problem, we consider a modification to the fully bypassed 5-stage MIPS processor pipeline presented in Lecture 3 and Problem Set 1. Our new processor has a data cache with a two-cycle latency. To accommodate this cache, the memory stage is pipelined into two stages, M1 and M2, as shown in Figure 2-A. Additional bypasses are added to keep the pipeline fully bypassed.

Suppose we are implementing this 6-stage pipeline in a technology in which register file ports are inexpensive but bypasses are costly. We wish to reduce cost by removing some of the bypass paths, but without increasing CPI. The proposal is for all integer arithmetic instructions to write their results to the register file at the end of the Execute stage, rather than waiting until the Writeback stage. A second register file write port is added for this purpose. Remember that register file writes occur on each rising clock edge, and values can be read in the next clock cycle. The proposed change is shown in Figure 2-B.

In this problem, assume that the only exceptions that can occur in this pipeline are illegal opcodes (detected in the Decode stage) and invalid memory address (detected at the start of the **M2** stage). Additionally assume that the control logic is optimized to stall only when necessary. ***You may ignore branch and jump instructions in this problem.***

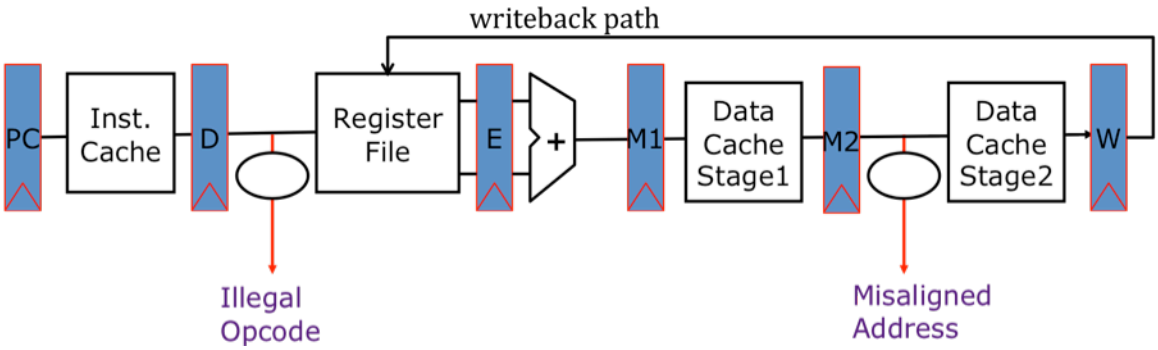

Figure 2-A. 6-stage pipeline. For clarity, bypass paths are not shown.

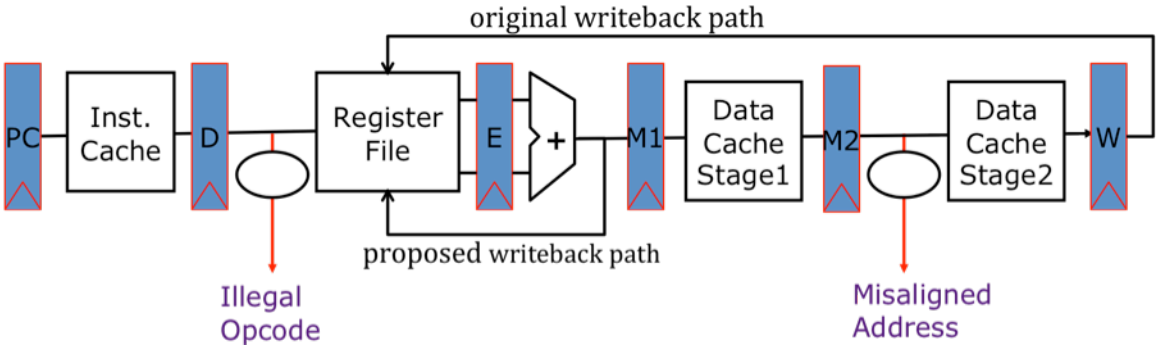

Figure 2-B. 6-stage pipeline with proposed additional write port.

NAME: _____

Problem Q.2.A-B

Hazards
[11 points]

Q.2.A – Second Write Port [6 points]
The second write port allows some bypass paths to be removed without adding stalls in the decode stage. Explain how the second write port improves performance by eliminating such stalls *and* give a short code sequence that would have required an interlock to execute correctly with only a single write port and with the same bypass paths removed.

The second write port improves performance by resolving some RAW hazards earlier than they would be if ALU operations had to wait until writeback to provide their results to subsequent dependent instructions. It would help with the following instruction sequence:

```
add r1, r2, r3
add r4, r5, r6
add r7, r1, r9
```

The important insight is that the second write port cannot resolve data hazards for immediately back-to-back instructions. (Recall that the RF is read in the ID stage, and when after the first instruction has written back, it is in M1, so the third instruction is in ID.)

Q.2.B – Bypasses Removed [5 points]
After the second write port is added, which bypass paths can be removed in this new pipeline without introducing additional stalls? List each removed bypass individually.

The bypass path from the end of M1 to the end of ID can be removed. (Credit was also given for the bypass path from the beginning of M2 to the beginning of EX, since these are equivalent.)

Additionally, ALU results no longer have to be bypassed from the end of M2 or the end of WB, but these bypass paths are still used to forward load results to earlier stages.

NAME: _____

Problem Q.2.C-D

Precise Exceptions
[14 points]

Q.2.C – Precise Exceptions [7 points]
Without further modifications, this pipeline may not support precise exceptions. Briefly explain why, and provide a minimal code sequence that will result in an imprecise exception.

Illegal address exceptions are not detected until the start of the M2 stage. Since writebacks can occur at the end of the EX stage, it is possible for an ALU op following a memory access to an illegal address to have written its value back before the exception is detected, resulting in an imprecise exception. For example:

```
lw r1, -1(r0)    // address -1 is misaligned
add r2, r3, r4   // r2 will be overwritten, even though preceding instruction has faulted
```

Q.2.D – Implementing Precise Exceptions with an Interlock [7 points]
Describe how precise exceptions can be implemented by adding a new interlock. Provide a minimal code sequence that would engage this interlock. Qualitatively, what is the performance impact of this solution?

Stall any ALU op in the ID stage if the instruction in the EX stage is a load or a store. The instruction sequence above engages this interlock.

Loads and stores account for about one-third of dynamic instructions. Assuming that the instruction following a load or store is an ALU op two-thirds of the time, and ignoring the existing load-use delay, this solution will increase the CPI by (1/3)*(2/3)==2/9. However, only a qualitative explanation was necessary for credit.

NAME: _____

Problem Q.2.E

Precise Exceptions
[7 points]

Q.2.E – Implementing Precise Exceptions with an Extra Read Port [7 points]
Suppose you are additionally given the budget to add a new register file *read* port. Propose an alternative solution to implement precise exceptions in this pipeline without requiring any new interlocks.

In addition to reading an instruction's source operands in the ID stage, also read the destination register, rd. If an early writeback occurs before a preceding exception was detected, then the old value of rd is preserved in the EX/M1 pipeline register and can be restored to the register file, maintaining precise state.

Problem Q.3: Iron Law of Processor Performance (Short Answer) [18 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
Reducing the number of registers in the ISA	Increase: values will more frequently be spilled to the stack, increasing the number of loads and stores.	Increase: more loads followed by dependent instructions, will cause stalls, which are likely to be difficult to schedule around. (-0.5 for no effect, the pipeline is unchanged)	Decrease: fewer registers means shorter register file access time. (-1 for no effect, because the pipeline is unchanged)
Adding a branch delay slot	Increase: NOPs must be inserted when the branch delay slot cannot be usefully filled.	Decrease: some control hazards are eliminated; also, additional NOPs execute quickly because they have no data hazards.	No effect: doesn't change pipeline. -or- Decrease: the branch_kill signal is no longer needed.
Merging the Execute and Memory stages (loads and stores use a separate adder to calculate base+offset)	No effect: this change is only microarchitectural, so is not ISA-exposed.	Decrease: the load-use hazard is eliminated, so fewer stalls will occur.	Increase: a combinational path through this stage includes both an adder and the data memory and is thus longer.

Changing implementation from a microcoded CISC machine to a RISC pipeline	Increase: it takes more RISC instructions than CISC instructions to encode the same program.	Decrease: microcoded machines take several clock cycles to execute an instruction, while the RISC pipeline should have a CPI near 1.	No effect: the amount of work done in one pipeline stage and one microcode cycle are about the same. -or- Increase: the RISC pipeline introduces longer control paths and adds bypasses, which are likely to be on the critical path.

END OF QUIZ