CpE 313
Microprocessor System Design
Solution to Exam 2

Instructions

Read each individual problem appearing on this exam carefully and do only what is specifically stated. Each question is worth 12 points.

This exam is designed to be completed by a well-prepared student in approximately **75 minutes**; most students are expected to finish it within the allotted time. On your initial pass through this exam, skip any problems that appear to be overly difficult.

You can choose to turn in this exam after 24 hours or before. However, you will get 100% of the points you score if you turn in your exam after 75 minutes. You will get 80% of the points you score if you turn in within 24 hours after the exam.

IMPORTANT: Write your initials at the TOP of EACH page. Also, be sure to read and sign the Academic Honesty Statement that follows:
"In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing this exam. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action."

Printed Name: . . . . Signature: . . . .

Date: . . . .

DO NOT BEGIN UNTIL INSTRUCTED TO DO SO.

Precise Exceptions

Problem 1.

(a) Outline the steps that must be performed to stop and restart a pipeline in the event of an interrupt if the interrupt handling must be precise.

1. Stop fetching further instructions.
2. Force a "trap" instruction into the pipeline.
3. Let all instructions before the offending instruction complete.
4. Turn off all writes for the offending instruction and the instructions issued after it.
5. The trap instruction must change the PC so that an "interrupt service routine (ISR)" can be started. The ISR will save PC of the offending instruction so that it can be restarted. It will then service the interrupt, and finally reload the address of offending instruction into the PC.

(b) Assume that we want to have precise interrupts for the MIPS floating point pipeline. Why does it not suffice to handle interrupts in the WB stage the way it is done for the integer pipeline?

Response Example 1: *In the FP pipeline, it is possible for a short (fast) instruction to enter WB before a previous slow instruction. This would violate the rule of precise interrupts, if all are processed in WB.* ***(Adapted from Jordan's response.)***

Response Example 2: *In a FP pipeline, each instruction type may take a different number of clock cycles to complete. Therefore, it is still possible for a later instruction to reach its WB stage before a previous instruction so that if both have exceptions, the later instruction is serviced first.* ***(Adapted from Russell's response.)***

Dynamically Scheduled Pipelines

Problem 2.

What is dynamic scheduling? How is it different from static scheduling?

Please see the solution to the practice test.

(b) Illustrate with a simple example piece of code (consisting of no more than two instructions) where dynamic scheduling is preferable to static scheduling.

```
LD R0, 0(R1)
ADD R2, R0, R3
```

The load instruction can take a variable number of clock cycles to execute depending on whether `LD R0, 0(R1)` *hits in the cache or if it has to go to the memory or the hard disk to retrieve the data. The compiler (static scheduling) cannot know how many independent instructions it needs to find and place between* `LD` *and* `ADD`*, but with dynamic scheduling the hardware can allow later instructions to execute as necessary.* ***(Adapted from Esteban's response.)***

Thornton's Scoreboard

Problem 3.

Consider the instruction sequence given below. These instructions are to be executed by dynamically scheduled single-issue pipeline that is managed by Thornton's scoreboard.

```
mul F2, F1, F4
add F6, F2, F7
sub F7, F3, F1
```

Assume that instruction `mul F2, F1, F4` is already being executed during clock cycle x, and will finish execution at the end of clock cycle $x + 10$.

(a) Will `add` be fetched in clock cycle $x + 1$? If yes, will `add` be issued in clock cycle $x + 2$? *Explain* your answers.

Yes, `ADD` will be fetched and issued as stated above. The only two conditions which would prevent the completion of the issue stage are: the required functional unit is not available or the destination register of `ADD` is already in use by an active instruction. Neither of these conditions apply. ***(Adapted from Boone's response.)***

(b) Describe the state of `sub` at the end of clock cycles $x+2$ to $x+8$? Assume `sub` takes 2 cycles of execution.

`SUB` has been fetched ($x+2$), issued ($x+3$), finished execute stage ($x+5$), but its write back is stalled until `ADD` can read its F7 operand, which will not happen until one cycle after `MUL` finishes executing. ***(Adapted from Esteban's response.)***

Tomasulo's Algorithm

Problem 4.

List three ways in which a dynamically scheduled single-issue pipeline managed by Tomasulo's algorithm differs from a dynamically scheduled single-issue pipeline managed by Thornton's scoreboard.

Please see the solution to the practice test.

Tomasulo's Algorithm

Problem 5. For the code sequence below, show the register file and reservation station states for the first 18 clock cycles. Please assume that:
1) floating point add/sub take 2 cycles of execution
2) floating point mult takes 10 cycles of execution
3) there are three FP adder units and two mult units
4) WR takes one clock cycle.

You will need to fill in the work sheets on the next four pages. If you leave an entry blank, I will assume that its value is the same as that shown in the work sheet for the previous stage.

Omit the clock cycles for which the reservation station and register file states do not change. Do not feel obligated to use all work sheets. There are more work sheets here than you need.

The initial states of reservation stations and register file are shown in the work sheet below.

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1	n					
Add2	2	n					
Mult1	3	n					
Mult2	4	n					

Integer Reg #	Q	V
1	0	100
2	0	50
3	0	31
4	0	1000
5	0	2000
6	0	3000
7	0	49

Figure 1: Initial State

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 2: Clock cycle . . . . . .

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 3: Clock cycle . . . . . .

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 4: Clock cycle . . . . . .

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 5: Clock cycle . . . . . .

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 6: Clock cycle . . . . . .

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 7: Clock cycle . . . . . .

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	*RS Tag*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 8: Clock cycle . . . . . .

Instruction		j	k
MULTD	R2	R1	R4
ADDD	R7	R2	R7
SUBD	R7	R7	R1

finishing times

IS	RPO	EXE	WR

RS	*RS Tag*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
Mult1	1						
Mult2	2						
Add1	3						
Add2	4						

Integer Reg #	Q	V
1		
2		
3		
4		
5		
6		
7		

Figure 9: Clock cycle . . . . . .

Re-Order Buffer

Problem 6.

(a) Define load speculation as implemented in Alpha 21264.

The processor will load a value from cache before determining if it is a hit or miss, and begin using it to execute later consumer instructions. ***(Adapted from Craig's response.)***

(b) Describe how you can use a re-order buffer to implement speculative load execution.

Response Example 1: *This is possible because of the ROB. If the load was determined to be a miss, the CPU can scrap everything in the re-order buffer after the load instruction and re-execute these instructions when the correct value has been loaded.* ***(Adapted from Craig's response.)***

Response Example 2: *The load instruction will wait in ROB until the HIT/MISS information is determined. Only after a HIT, the load can be deleted from the ROB. If a MISS is determined, the ROB is used to roll back the result of later instructions (after load), and the execution of load is resumed with reading the correct data from memory or hard disk.* ***(Adapted from Maciej's response.)***

Wide-Issue Microprocessors

Problem 7.

VLIW and superscalar are two types of wide-issue processors. Give details of issue logic for both of these types.

Superscalar is dynamic multiple issue processor. N instructions are fetched and hardware decides which instruction is issued based on dependencies with instructions that are prior in fetch order and those that are in execution.

VLIW is a static multiple issue processor. The compiler decides which instruction to issue in which clock cycle. Then the compiler packages instructions into N-instruction packets and sends them to the processor. ***(Adapted from Russell's response.)***

Branch Prediction

Problem 8.

Draw a block diagram that shows how a branch target buffer works. Clearly show how the address from program counter is used. What is the branch penalty for a correctly predicted branch?