Assignment 2
CMSC611-101
Advanced Computer Architecture, Fall 2009

Sandipan Dey

Question 1: *(80 Points)*

The goal of this exercise is to compare how a loop runs on a variety of pipelined versions of MIPS. The loop implements the vector operation Y = a × X + Y. Here is the MIPS code for the loop:

```
foo:    L.D     F2, 0(R1)       ; Load X(i)
        MULT.D  F4, F2, F0      ; multiply a * X(i)
        L.D     F6, 0(R2)       ; load Y(i)
        ADD.D   F6, F4, F6      ; add a * X(i) + Y(i)
        S.D     0(R2), F6       ; store Y(i)
        ADDIU   R1, R1, #8      ; increment X index
        ADDIU   R2, R2, #8      ; increment Y index
        SLTIU   R3, R1, done    ; test if done
        BEQZ    R3, foo         ; loop if not done
```

Assume that the results are fully bypassed. The conditional branches are resolved in the ID stage. Use the FP latencies shown in the following table but assume that the FP unit is fully pipelined:

Functional unit	Latency
Integer ALU	0
Data memory (integer and FP loads)	1
FP add	3
FP multiply	7

A) Using the standard single issue MIPS pipeline show the number of stall cycles for each instruction and what clock cycle each instruction begins execution (i.e., enters its first EX cycle) on the first iteration of the loop. How many clock cycles does each loop iteration take?

B) Unroll the loop to make four copies of the body and schedule it for the standard MIPS pipeline. Re-order the instructions in order to maximize performance. How many clock cycles does each loop iteration take?

C) Consider running the loop on a CDC scoreboard. What would be the state of scoreboard when the SLTIU instruction reaches the write result stage in the first iteration? Assume that issue and read operands stages each take one cycle. Assume that there are one integer functional unit, one FP multiplier, and one FP adder.

D) Assume Tomasulo based CPU with one integer unit, one FP multiplier and one FP adder. Show the state of the reservation stations and register-status tables when the SLTIU writes its result on the CDB in the first loop iteration.

Answer:

A)

Instruction	Clock Cycle Number																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L.D F2, 0(R1)	IF	ID	EX	MEM	WB															
MULT.D F4, F2, F0		IF	ID	Stall	M1	M2	M3	M4	M5	M6	M7	M8	MEM	WB						
L.D F6, 0(R2)			IF	Stall	ID	EX	MEM	WB												
ADD.D F6, F4, F6					IF	ID	Stall	Stall	Stall	Stall	Stall	Stall	A1	A2	A3	A4	MEM	WB		
S.D 0(R2), F6						IF	Stall	Stall	Stall	Stall	Stall	Stall	ID	EX	Stall	Stall	MEM	WB		
ADDIU R1, R1, #8													IF	ID	Stall	Stall	EX	MEM	WB	
ADDIU R2, R2, #8														IF	Stall	Stall	ID	EX	MEM	WB
SLTIU R3, R1, done																	IF	ID	EX	MEM
BEQZ R3, foo																		IF	Stall	ID
Branch Delay																				Stall

Forwarding stages are highlighted: MEM →M1 (L.D to MULT.D), M8 →A1 (MULT.D to ADD.D), A4 →MEM (ADD.D to S.D), EX →ID (SLTIU to BEQZ, since **branch is resolved in the ID stage**, the operands must be ready before that)

		Clock Cycle Issued	
foo:	L.D F2, 0(R1)	1	
	Stall	2	;to prevent RAW hazard for F2
	MULT.D F4, F2, F0	3	
	L.D F6, 0(R2)	4	
	Stall	5	;to prevent RAW hazard for F4, F6
	Stall	6	
	Stall	7	
	Stall	8	
	Stall	9	
	Stall	10	
	ADD.D F6, F4, F6	11	
	Stall	12	;to prevent RAW hazard for F6
	Stall	13	
	S.D 0(R2), F6	14	
	ADDIU R1, R1, #8	15	
	ADDIU R2, R2, #8	16	
	SLTIU R3, R1, done	17	
	Stall	18	;to prevent RAW hazard for R3,
	BEQZ R3, foo	19	;branch is resolved in ID stage
	Stall	20	;delayed branch

Hence, total number of clock cycles required = 20 per loop.

Optimized / reordered by a smart / sophisticated compiler, the loop looks like the following:

		Clock Cycle Issued
foo:	L.D F2, 0(R1)	1
	L.D F6, 0(R2)	2
	MULT.D F4, F2, F0	3
	ADDIU R1, R1, #8	4
	ADDIU R2, R2, #8	5
	SLTIU R3, R1, done	6
	Stall	7
	Stall	8
	Stall	9
	ADD.D F6, F4, F6	10
	Stall	11
	BEQZ R3, foo	12
	S.D -8(R2), F6	13

B)
Unrolling the loop and making 4 copies of the body (here we assume the number of times the loop iterates is a multiple of 4),

Instruction	**Clock Cycle Issued**
L.D F2, 0(R1)	1
MULT.D F4, F2, F0	3
L.D F6, 0(R2)	4
ADD.D F6, F4, F6	11
S.D 0(R2), F6	14
L.D F8, 8(R1)	15
MULT.D F10, F8, F0	17
L.D F12, 8(R2)	18
ADD.D F12, F10, F12	25
S.D 8(R2), F12	28
L.D F14, 16(R1)	29
MULT.D F16, F14, F0	31
L.D F18, 16(R2)	32
ADD.D F18, F16, F18	39
S.D 16(R2), F18	42
L.D F20, 24(R1)	43
MULT.D F22, F20, F0	45
L.D F24, 24(R2)	46
ADD.D F24, F22, F24	53
S.D 24(R2), F24	56

ADDIU R1, R1, #32	57
ADDIU R2, R2, #32	58
SLTIU R3, R1, done	59
BEQZ R3, foo	61
Branch Delay	62

Reordering, we get the following:

Instruction	**Clock Cycle Issued**
L.D F2, 0(R1)	1
L.D F6, 0(R2)	2
MULT.D F4, F2, F0	3
L.D F8, 8(R1)	4
L.D F12, 8(R2)	5
MULT.D F10, F8, F0	6
L.D F14, 16(R1)	7
L.D F18, 16(R2)	8
MULT.D F16, F14, F0	9
L.D F20, 24(R1)	10
L.D F24, 24(R2)	11
MULT.D F22, F20, F0	12
ADD.D F6, F4, F6	13
ADD.D F12, F10, F12	14
ADDIU R1, R1, #32	15
SLTIU R3, R1, done	16
ADD.D F18, F16, F18	17
S.D 0(R2), F6	18
S.D 8(R2), F12	19
ADD.D F24, F22, F24	20
S.D 16(R2), F18	21
ADDIU R2, R2, #32	22
BEQZ R3, foo	23
S.D -8(R2), F24	24

Since 4 times the body of the loop takes 24 clock cycles, each iteration of the loop takes 6 clock cycles.

C) Scoreboard status:

Instruction Status

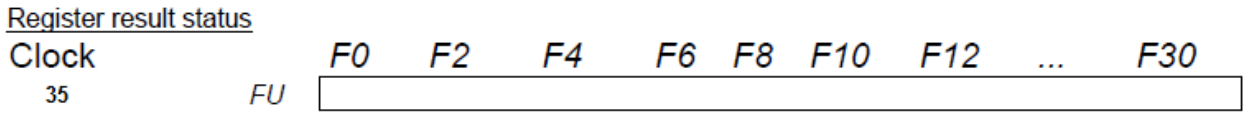
Instruction	i	j	k	Issue	Read Operands	Execution Comp	Write Result
L.D	**F2**	0	R1	1	2	3	4

MULT.D	F4	**F2**	F0	2	5	13	14
L.D	**F6**	0	R2	5	6	7	8
ADD.D	**F6**	F4	**F6**	6	15	19	20
S.D	0(R2)	**F6**		9	21	22	23
ADDIU	**R1**	R1	8	24	25	26	27
ADDIU	R2	R2	8	28	29	30	31
SLTIU	**R3**	**R1**	done	32	33	34	35
BEQZ		**R3**	foo				

As seen from above, there is no WAR/WAW hazard, only structural/RAW hazards are there to be resolved by the scoreboard.

Functional Unit Status

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	BEQZ		R3	foo			Yes	Yes
	Multiply	No								
	Add	No								



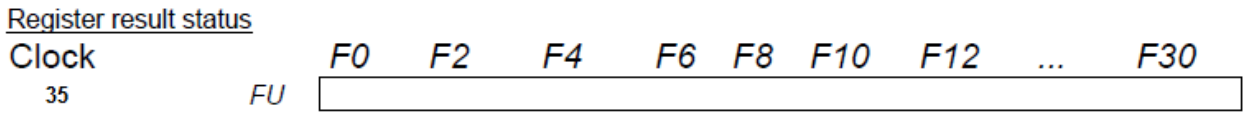
D)

Instruction Status

Instruction	i	j	K	Issue	Read Operands	Execution Complete	Write Result
L.D	**F2**	0	R1	1	2	3	4
MULT.D	F4	**F2**	F0	2	5	13	14
L.D	**F6**	0	R2	5	6	7	8
ADD.D	**F6**	F4	**F6**	6	15	19	20
S.D	0(R2)	**F6**		9	21	22	23
ADDIU	**R1**	R1	8	24	25	26	27
ADDIU	R2	R2	8	28	29	30	31
SLTIU	**R3**	**R1**	done	32	33	34	35
BEQZ		**R3**	foo				

Functional Unit Status (immediately before write result stage, just after clock cycle 34)

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	SLTIU	R3	R1	done			Yes	Yes
	Multiply	No								
	Add	No								



Question 2: *(20 Points)*
It is critical that the scoreboard be able to distinguish RAW and WAR hazards, since a WAR hazard requires stalling the instruction doing the writing until the instruction reading an operand initiates execution, while a RAW hazard requires delaying the reading instruction until the writing instruction finishes-just the opposite. For example, consider the sequence:

MULT.D	F0, F6, F4
SUB.D	F8, F0, F2
ADD.D	F2, F10, F2

The SUB.D depends on the MULT.D (a RAW hazard) and thus the MULT.D must be allowed to complete before the SUB.D; if the MULT.D were stalled for the SUB.D due to the inability to distinguish between RAW and WAR hazards, the processor will deadlock. This sequence contains a WAR hazard between the ADD.D and the SUB.D, and the ADD.D cannot be allowed to complete until the SUB.D begins execution. The difficulty lies in distinguishing the RAW hazard between MULT.D and SUB.D, and the WAR hazard between the SUB.D and ADD.D.
Describe how the scoreboard avoids this problem and show the scoreboard values of the above sequence assuming the ADD.D is the only instruction that has completed execution(though it has not written its result). (Hint: Think about how WAW hazards are prevented and what this implies about active instruction sequences.)

Answer

The scoreboard can distinguish the RAW and WAR hazard (dependency and anti-dependency) from the functional unit status, by checking the flags R_j and R_k that indicate whether the source registers F_j and F_k (corresponding functional units Q_j and Q_k) respectively are ready or not, along with checking the destination register F_i.

If FU be the functional unit used by an instruction I, then I will not write its result until $(\forall f)((F_j(f) \neq F_i(FU) \vee R_j(f) = No) \wedge (F_k(f) \neq F_i(FU) \vee R_k(f) = No))$ is true. This prevents WAR hazard and also the condition on $R_j(f)$ differentiates it from RAW hazard.

If the destination register of *I* (from *FU*) is same as a source register of an instruction *J* from another functional unit *f, i.e.,*

If $(\exists f)F_j(f) = F_i(FU)$, it will be

- Anti-dependence (WAR hazard) if $R_j(f) = Yes$
  (the corresponding source register is ready).
- Dependence (RAW hazard) if $R_j(f) = No$
  (the source register is still waiting to be written).

It is important to note that if the source register is still waiting, it must be waiting for *I* only. Since WAW hazards can't happen (prevented at issue stage), the case that the source $R_j(f)$ is waiting for some other instruction (not *I*) is ruled out (since if it is true then both *I* and the other instruction (both active) would have same destination $R_j(f)$).

Assuming that the scoreboard has 2 FP-Add units (if there is only 1 FP-Add unit, due to in-order issue and structural hazard ADD.D can never be issued until SUB.D completes execution and writes it result) and 1 FP-Multiply unit, and assuming execution cycles for Add and Multiply 4 and 8 clock cycles respectively,

Instruction Status

Instruction	i	j	K	Issue	Read Operands	Execution Complete	Write Result
MULT.D	**F0**	F6	F4	1	2		
SUB.D	F8	**F0**	**F2**	2			
ADD.D	**F2**	F10	F2	3	4	8	

Functional Unit Status

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Mult	Yes	Mul	F0	F6	F4			Yes	Yes
	Add1	Yes	Sub	F8	F0	F2	Mul		No	Yes
	Add2	Yes	Add	F2	F10	F2			Yes	Yes

As we can see from above,

The relation between SUB.D and ADD.D is anti-dependence (leading to WAR hazard), since, $F_k(Add1) = F_i(Add2)$ and $R_k(Add2) = Yes$.

The relation between MULT.D and SUB.D is dependence (leading to RAW hazard), since, $F_j(Add1) = F_i(Mult)$ and $R_j(Add1) = No$.