Homework-2

Advanced Computer Architecture (611)

Sandipan Dey

85

A) 20
B) 20
C) 17
D) 6
E) 20

Question 1: *(80 Points)*

The goal of this exercise is to compare how a loop runs on a variety of pipe... The loop implements the vector operation $Y = a \times X + Y$. 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:

Assumptions:

Instructions	Latency	#Clock cycles to get result after the execution has started
ADDIU, SLTIU (integer instructions)	0	1
ADD.D	3	4
MULT.D	7	8
L.D	1	2

It's to be noted that for L.D, the result is available after MEM cycle (hence has latency 1). Also, since **branch is resolved in the ID stage**, the operands must be ready before that.

A) With the above assumptions, we show the following pipeline stages.

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), and using the fact that all the FP units are pipelined,

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

By 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) Assuming that L.D takes 2 clock cycles (latency 1), ADD.D takes 4 clock cycles (latency 3), MULT.D takes 8 clock cycles (latency 7), the scoreboard status when SLTIU writes:

Instruction Status							
Instruction	I	j	k	Issue	Read Operands	Execution Comp	Write Result
L.D	**F2**	0	R1	1	2	4 3	5
MULT.D	**F4**	**F2**	F0	2	6	14	15
L.D	**F6**	0	R2	6	7	9	10
ADD.D	**F6**	**F4**	**F6**	7	16	20	21
S.D	**F6**	0	R2	11	22	23	24
ADDIU	**R1**	R1	8	25	26	27	28
ADDIU	R2	R2	8	29	30	31	32
SLTIU	**R3**	**R1**	done	33	34	35	36

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

Clock cycle	Scoreboard Action
1	1st L.D issued (assuming no structural hazard being the 1st instruction)
2	MULT.D issued (no structural hazard, different FU), 1st L.D reads operands
3-5	1st L.D completes execution (2 cycles) and writes result (1 cycle). The 2nd L.D can't be issued (structural hazard for busy Integer FU), no further instruction will be issued till clock cycle 4 (guarantees in-order issue), also MULT.D can't read operand, RAW hazard will be resolved after 1st L.D writes F2 at clock cycle 4
6	2nd L.D issued, MULT.D reads operands
7	ADD.D issued, MULT.D starts execution, 2nd L.D reads operands
8-10	2nd L.D completes execution and writes result. S.D can't be issued (structural hazard for busy Integer FU), no further instruction will be issued till clock cycle 8 (guarantees in-order issue), MULT.D continues execution, ADD.D can't read operand because of RAW hazard (data-dependence) on F4 (MULT.D), F6 (2nd L.D)
11	S.D issued, ADD.D can't read operand because of RAW hazard (data-dependence) on F4 (MULT.D) , MULT.D continues execution
12-15	MULT.D completes execution and writes result. ADD.D can't read operand because of RAW hazard (data-dependence) on F4 (MULT.D), S.D can't read operand since data-dependent (RAW hazard) on F6 (ADD.D), ADDIU can't be issued (structural hazard for busy Integer FU), no further instruction will be issued till clock cycle 24 (guarantees in-order issue)
16-21	ADD.D reads operand, completes execution and writes result. S.D can't read operand since data-dependent (RAW hazard) on F6 (ADD.D), ADDIU can't be issued (structural hazard for busy Integer FU), no further instruction will be issued till clock cycle 24 (guarantees in-order issue)
22-24	S.D reads operand, completes execution and writes result. ADDIU can't be issued (structural hazard for busy Integer FU), no further instruction will be issued till clock cycle 24 (guarantees in-order issue)
25	1st ADDIU issued

26-28	1st ADDIU reads operand, completes execution and writes result. 2nd ADDIU can't be issued (structural hazard for busy Integer FU), no further instruction will be issued till clock cycle 27 (guarantees in-order issue)
29	2nd ADDIU issued
30-32	2nd ADDIU reads operand, completes execution and writes result. SLTIU can't be issued (structural hazard for busy Integer FU), no further instruction will be issued till clock cycle 31 (guarantees in-order issue)
33	SLTIU issued
34-36	SLTIU reads operand, completes execution and writes result. No other instructions corresponding to Integer FU can be issued (structural hazard) till clock cycle 36.

Functional Unit Status (after clock cycle 35, right before FLTIU writes its result)

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

Functional Unit Status (after clock cycle 36, right after FLTIU writes its result)

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult	No								
	Add	No								

Register result status

Clock 36

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU									

D) Load and Stores are treated as FUs with Reservation Stations. Assuming that the Tomasulo has 3 load buffers and 3 store buffers, L.D with 2 clock cycles, ADD.D with 4 clock cycles and MULT.D with 8 clock cycles execution time, with 1 integer, 1 FP-Mult and 1 FP-Add reservation station,

1 ~~Reservation Station for~~ Integer ALU
1 - - - Mult (FP)
1 - - - Add (FP)
} Assumption

Instruction Status

Instruction	i	J	K	Issue	Execute	Write Result
L.D	**F2**	0	R1	1	3	4
MULT.D	**F4**	**F2**	F0	2	14	?
L.D	**F6**	0	R2	3	5	6
ADD.D	**F6**	**F4**	**F6**	4	?	?
S.D	**F6**	0	R2	5	?	?
ADDIU	**R1**	R1	8	6	7	8
ADDIU	R2	R2	8	9	10	11
SLTIU	**R3**	**R1**	done	12	13	14

ADD.D can't start execution until MULT.D writes result, then F4 is available

S.D can't start execution until the operand is written by ADD.D

Can't issue the 2nd ADDIU immediately after 1st since Integer unit is busy

(broadcasted) to CDB

Reservation Stations (right after SLTIU write-result stage, just after clock cycle 14)

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
0	Mult1	Yes	MULT.D	R(F0)	M(R1)		
	Add1	Yes	ADD.D		M(R2)	Mult1	

Register Status

Field	F0	F2	F4	F6	...	...	F30
Qi			Mult1	Add1			

Load Buffers

Field	Load1	Load2	Load3
Address			
Busy	No	No	No

Store Buffers

Field	Store1	Store2	Store3
Address			
Busy	No	No	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 (data-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.

f 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.

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

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

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

It is important to note the following: if the instruction J is still waiting for its (one of) source register to be written by some other instruction, it must be waiting for instruction I only and on no other instruction. 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 the same destination $R_j(f)$, resulting a WAW hazard, a contradiction).

To explain simply, two instructions are related to each other if 1st instruction's destination is same as (at least) one of the sources of the 2nd instruction. The relation is data-dependence (leading to RAW hazard) if the 2nd instruction waits for the source to be written by 1st instruction (indicated by the corresponding source flag value 'not ready'). The relation is, on the other hand anti-dependence (leading WAR hazard) if the 2nd instruction does not wait on the 1st instruction at all (indicated by the corresponding source flag value 'ready'), the other source register of the 2nd instruction which is different from the 1st instruction's destination register may typically be 'not ready', waiting on some other instruction.

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
2	Mult	Yes	Mul	**F0**	F6	F4			Yes	Yes
	Add1	Yes	Sub	F8	**F0**	**F2**	**Mul**		**No**	**Yes**
0	Add2	Yes	Add	**F2**	F10	F2			Yes	Yes

As can be seen 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 $R_k(Add1) = Yes$.

$\therefore f = Add1,\ FU = Add2 \Rightarrow (F_k(f) \neq F_i(FU) \vee R_k(f) = No) = false$. This makes the condition $(\forall f)((F_j(f) \neq F_i(FU) \vee R_j(f) = No) \wedge (F_k(f) \neq F_i(FU) \vee R_k(f) = No))$ false (also, $Q_k(Add1)$ indicates that it is not waiting for Add2) and becomes a potential candidate for WAR hazard, Add2 waits until F2 is read by Add1, then writes F2 only after it is read by Add1.

On the contrary, the relation between MULT.D and SUB.D is data-dependence (leading to RAW hazard), since, $F_j(Add1) = F_i(Mult)$ and $R_j(Add1) = No$. Also, $F_k(Add1) \neq F_i(Mult)$ and $R_k(Add1) = Yes$.

$\therefore f = Add1,\ FU = Mult \Rightarrow (F_j(f) \neq F_i(FU) \vee R_j(f) = No) = true$, also, $(F_k(f) \neq F_i(FU) \vee R_k(f) = No) = true$. This makes the condition $(\forall f)((F_j(f) \neq F_i(FU) \vee R_j(f) = No) \wedge (F_k(f) \neq F_i(FU) \vee R_k(f) = No))$ true (also, $Q_j(Add1)$ indicates that it is waiting for Mult) and scoreboard becomes aware of the fact that this is NOT a candidate for WAR hazard (rather it is a candidate for RAW hazard), hence Mult does not wait and writes F0, avoiding deadlock.