Adel Danandeh

Homewordk#2

Q1.A Baseline

Answer has been provided in the first row of the table.

Q1.B Faster Memory

The overall performance would change, because the clock cycle changes. Since the largest value of latency of each stage has decreased from 400ps to 320ps (which is the latency of Decode stage) therefore the value of frequency increases from 2.5 GHz to 3.125 GHz. Ultimately, throughput would be improved as well. Performance is improved, since time has been improved.

Q1.C Proposed Scheme

In order to achieve maximum frequency we find the average cycle time of the pipeline (just how it was mentioned during lecture).

Cycle Time for each stage = Total Cycle Time / number of stages = 280ps

Since frequency is reverse of time, we have 1/280ps = 3.571 GHz.

The smaller Cycle Time we have, the larger frequency we get. In order to improve frequency we must decrease Cycle Time (since Cycle Time is in the denominator) therefore value of the fraction increases. In other words, we would have a better frequency (which this is what we are trying to achieve).

Processor	**Cycle Time**	**Max Clock Frequency**	**Latency of Instr**	**Throughput**
a) baseline	400ps	2.5 GHz	3200ps	0.0025 Instr/ps
b) faster memory	320ps	3.125 GHz	2560ps	0.003125 Instr/ps
c) proposed scheme	280ps	3.571 GHz	280*8=2240ps	0.003571 Instr/ps

Q2.A Comparison

In order to compare each option we will calculate the value of time using Iron's Law:

Time(original) = IC*CPI*Period = IC*1*Period

Perf(original) = 1/Time(original) = 1/IC*Period

The Time and performance of option1:

Time(opt1) = IC*1*2*Period

Perf(opt1) = 1/Time(opt1) = 0.5/IC*Period

The Time and performance of option2:

Time(opt2) = 1.17*IC*1*Period

Perf(opt2) = 1/Time(opt2) = 0.85/IC*Period

The Time and performance of option 3:

Time(opt3) = IC*1.20*1*0.5Period

Perf(opt3) = 1/Time(opt3) = 1.66/IC*Period

------- Explanation ----------

Based on the comparison of their Time, performance of each option could be found, which we could conclude that option 3 has a better performance compare to the other options (option1 & option2).

Q2.B

We compare the Time for option3 and the option in which software team provides:

Time(opt3) = IC*1.20*1*0.5Period

Time(softOpt) = X*IC*1.05*0.5Peroid

We calculate the speedup, which has to be greater than 1 so the option that software guys are offering is faster than option3.

Time(opt3)/Time(softOpt)>1

(1.20*IC*1*0.5Period)/(X*IC*1.05*0.5Peroid)>1

X*1.05 < 1.20 which it means the max increase is 14.28%

Q3.A

CPI(ave) = 3*0.35 + 2*0.2 + 7*0.03 + 1*0.42 = 2.08

Q3.B

The average CPI for each option is:

CPI(ave-option1) = 2*0.35 + 1*0.20 + 7*0.03 + 1*0.42 = 1.53

CPI(ave-option2) = 3*0.35 + 2*0.20 + 1*0.03 + 1*0.42 = 1.9

Let's compare the time using Iron's Law:

Time(opt1) = IC*CPI*Period = IC*1.53*Period

Time(opt2) = IC*CPI*Period = IC*1.9*Period

Time is inverse to performance, so the smaller Time we have the better performance we get.

Therefore option1 must have a better performance.

Speedup= Time(opt1)/Time(opt2) = IC*1.53*Period/ IC*1.9*Period = 0.80

Q.4A

BRANCH:

If branch is taken:

F0	F1	D	X0	X1	W						
	F0	F1	D	X0	X1	W					
		F0	F1	D	X0	X1	W				
			F0	F1	D	X0	X1	W			
				F0	F1	D	X0	X1	W		
					F0	F1	D	X0	X1	W	

Yellow highlights mean: We can't fetch any instruction in the highlighted range; therefore we flush out the highlighted segment above.

Since branches are resolved at execute stage, we have to wait 5 cycles to be able to fetch the next instruction.

Therefore if branch is taken: CPI = 5

If branch is not taken:

F0	F1	D	X0	X1	W						
	F0	F1	D	X0	X1	W					
		F0	F1	D	X0	X1	W				
			F0	F1	D	X0	X1	W			

CPI = 1 in this case.

CPI(ave) = 80% * 5 + 20% * 1 = 4.2

JUMP:

F0	F1	D	X0	X1	W						
	F0	F1	D	X0	X1	W					
		F0	F1	D	X0	X1	W				
			F0	F1	D	X0	X1	W			

Yellow highlights mean: Instruction can't be fetched until jump is resolved at Decode stage. Since we can't fetch any instructions in the highlighted range above, we have to waste 3 cycles. Therefore, CPI = 3 in this case.

LOAD:

F0	F1	D	X0	X1	W						
	F0	F1	D	X0	X1	W					
		F0	F1	D	X0	X1	W				
			F0	F1	D	X0	X1	W			

There is no flushing happening in this case, since every instruction could be fetched one after another.

Therefore CPI = 1

STORE:

F0	F1	D	X0	X1	W						
	F0	F1	D	X0	X1	W					
		F0	F1	D	X0	X1	W				
			F0	F1	D	X0	X1	W			

There is no flushing happening in this case, since every instruction could be fetched one after another.

Therefore CPI = 1

Instruction Type	Frequency	CPI
Branch	15%	4.2
Jump	5%	4
Load	30%	1
Store	30%	1
Other	20%	1

CIP(ave) = 4.2 * 0.15 + 0.05 * 4 + 0.30 * 1 + 0.30 * 1 + 0.20 * 1 = 1.63

Q.4B

If we take branches always (100% of the time), we have:

CIP(ave) = 5 * 0.15 + 0.05 * 3 + 0.30 * 1 + 0.30 * 1 + 0.20 * 1 = 1.70

Q.4C

We have 4 cases to deal with:

Case 1: Branch is taken with a useful instruction

F0	F1	D	X0	X1	W	
	F0	F1	D	X0	X1	W -> useful instruction

CPI = 1 (branch is taken with useful instruction)

CPI = 1 * 0.40 (taken 40% of the time) = 0.4

Case 2: Branch is taken with a nop instruction

F0	F1	D	X0	X1	W	
	F0	F1	D	X0	X1	W -> nop instruction

This adds up to value of CPI by 1 => CPI = 5, basically one cycle would be wasted.

CPI = 1 * 0.40 = 0.40

Case 3: Branch is not taken with a useful instruction:

CPI = 0.1 * 1 = 0.10

Case 4: Branch is not taken with a nop instruction:

CPI = 0.10 * 2 = 0.20

CPI(ave) = 0.20 + 0.10 + 0.4 + 0.4 = 1.1

Q4.D

We have two cases:

Case 1: Next instruction is not dependent on the previous instruction:

Load followed by Add instruction.

F0 F1 D X0 W

F0 F1 D X0 X1 W

CPI = 1 * 25% + 50% * 2 = 1.25

CPI(Load) = 1.25

Case 2: Next instruction dependent on the previous instruction:

F0 F1 D X0 X1 W

F0 F1 D X0 W

CPI = 1 * 25% + 2 * 50% = 1.25

CPI(Other- Add) = 1.25

Instruction Type	Frequency	CPI
Branch	15%	4.2
Jump	5%	4
Load	30%	1.25
Store	30%	1
Other	20%	1.25

CIP(ave) = 4.2 * 0.15 + 0.05 * 4 + 0.30 * 1.25 + 0.30 * 1 + 0.20 * 1.25 = 1.755

Q4.E

In case of branch taken:

CPI(Branch Taken) = 0.5*0.2*1 + 0.5*2 + 0.2*3 = 1.7

Q5.

Microarchitecture	Num Instr (#)	Cycle Time	B2B Instr Latency(cyc)	B2B Instr Latency(T)	Throughput CPI	Total Execution Time (T)
a) 1-cycle	4	63	1	63	1	252
b) Iterative	4	21	4	84	4	336
c) 2-cycle	4	33	2	66	5/4	165
d) 4-cycle	4	18	4	72	7/4	126
e) Var-cyle	4	23	2.5	57.5	8/4	184

Q5.A

	C0	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
I0	X0	X0	X0	X0												
I1					X0	X0	X0	X0								
I2									X0	X0	X0	X0				
I3													X0	X0	X0	X0

Q5.B

	C0	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
I0	X0	X1														
I1		X0	X1													
I2			X0	X1												
I3				X0	X1											

Q5.C

	C0	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
I0	X0	X1	X2	X3												
I1		X0	X1	X2	X3											
I2			X0	X1	X2	X3										
I3				X0	X1	X2	X3									

Q5.D

	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
I0	Y	X0	X1	X2	X3											
I1		Y	Y	X0	X1	X2										
I2				Y	Y	X0	X1									
I3						Y	Y	X0								

Average Latency (cycle) = 4+3+2+1 = 10/4 = 2.5

Average Latency (T) = 4*23 + 3*23 + 2*23 + 1*23 = 57.5

Q5.E

Based on the Iron's Laws formula we have:

Time = IC*CPI*Period

Time(a) = 4*1*63 = 252

Time(b) = 4*4*21 = 336 (5.part (a))

Time(c) = 4*5/4*33 = 165 (5.part (b))

Time(d) = 4*7/4*18 = 126 (5.part (c))

Time(e) = 4*8/4*23 = 184 (5.part (d))

Since we know the fact that performance is the inverse of Time, the best performance is the one, which it has the smallest Time. Therefore, based on the calculations above (comparing Iron's Law) part (d) has the smallest Time, so it has the best performance.

For (a), we find that when the multiplication is performed by only one stage the clock cycle for it is very large. Therefore, we are not able to do parallel instructions, because there is only one stage in which it could process one instruction per cycle. Since the clock cycle is very big, the throughput is small. However, we could avoid having additional latency due flops by having only one stage.

For (b), we have a small cycle time, because the instruction could take less time to be processed. Since we iterate only when an instruction has finished, the other one could be started. Ultimately, we cannot overlap them and this microarchitecture turns out to be the worst in terms of performance.

For (c), we reduce the cycle time with the increase of the number of stages which, it gives capability of performing instructions in parallel. Therefore, we have a better performance than the previous two cases (a) and (b). However, we have a tradeoff of pipelines such as hazards.

For (d), the microarchitecture is somewhat like part (c), but with a smaller cycle time, because the pipeline has more stages. But, the latency of an instruction is larger than in part (c). What this shows is that; even

though we have divided the pipeline in more stages to reduce the clock cycle, the number of stages is not sufficient enough to improve the latency of an instruction. However, the total execution is improved, because of the clock cycle time. Therefore, we find the best one. We should also take into consideration the hazards for pipeline here as a disadvantage.

We could consider a 20 cycle pipelined multiplier if having that many stages reduces the clock cycle so much that it is beneficial. However, going through that many stages would increase the clock cycles per instruction. This will increase the Time, and therefore decrease the performance. So, as long as value of CPI increases and period of the clock decreases in such a way that both of them compensate in Iron's Law. Therefore performance could be maintained or increased and it could be a good option to have that many stages. In addition, we must take into account that extra latency due to the additional flip flop that is introduced for new stages.

Q5.F

	C0	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**	**C16**	**C17**
I0	F	D	X0	X1	X2	X3	M	W										
I1		F	D	D	D	D	X0	X1	X2	M	W							
I2			F	F	F	F	D	D	D	X0	X1	M	W					
I3							F	F	F	D	D	X0	M	W				

CPI = 14 cycles/ 4 instructions = 3.5 cycles/instr.

Q6.A

Instruction 2 depends on instruction 1 because of r3.

Instruction 4 depends on instruction 1 because of r3.

Instruction 6 depends on instruction 5 because of r4.

Instruction 7 depends on instruction 6 because of r5.

Instruction 9 depends on instruction 4 because of r3.

Instruction 10 depends on instruction 9 (because of r3) and instruction 2 (because of r2).

Q6.B

Instruction	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
xor r3, r3,r3	F	D	X	M	W															
addiu r2, r3, 1		F	D	D	D	X	M	W												
j L1			F	F	F	D	X	M	W											
mul r3, r3, r3						F	-	-	-	-	Gets flushed out									
bne r2, r3, -28							F	D*	X	M	W									
END								F	-	-	-	-	Gets flushed out							
mul r3, r3, r3									F	D	X0	X1	X2	M	W					
add r4, r1, r4										F	D	D	D	X	M	W				
lw r5, 0(r4)											F	F	F	D	D	D	X	M	W	
sub r5, r5, 1														F	D	D	D	D	D	X
sw r5, 0(r6)	I am stopping here, because I can't completely finish an instruction														F	F	F	F	F	D
addiu r3, r3, 1																				F
bne r2, r3, -28																				
END																				

Q6.C

Instruction	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22
xor r3, r3,r3	F	D	X	M	W	Bypass data to execute of the next instruction.																	
addiu r2, r3, 1		F	D	X	M	W																	
j L1			F	D	X	M	W																
FLUSHED				F	-	-	-	-															
bne r2, r3, -28					F	D*	X	M	W														
END-FLUSHED						F	-	-	-	-													
mul r3, r3, r3							F	D	X0	X1	X2	M	W										
add r4, r1, r4								F	D	D	D	X	M	W									
lw r5, 0(r4)									F	F	F	D	X	M	W								
sub r5, r5, 1												F	D	D	X	M	W						
sw r5, 0(r6)					Highlighted red and green colors are when we bypass to execute stage of next instruction.								F	F	D	X	M	W					
addiu r3, r3, 1															F	D	X	M	W				
bne r2, r3, -28												Data hazard X and D				F	D	X	M	W			
END																	F	-	-	-	-		

Yellow highlight is when we have control hazard, in case of branch and jump next instruction must be flushed.

Q6.D

1 xor r3, r3, r3

2 j L1

3 addiu r2, r3, 1 3

4 loop: add r4, r1, r4

5 lw r5, 0(r4)

6 addiu r3, r3, 1

7 sub r5, r5, 1

8 sw r5, 0(r6)

9 L1: bne r2, r3, -28

10 mul r3, r3, r3

11 END

with switching instructions j L1 and addiu r2, r3, 1 3 we could save one cycle, since we don't have to stall at decode stage anymore. With moving instruction addiu r3, r3, 1 to be fetched right after lw r5, 0(r4) we save another cycle. We could put mul r3, r3, r3 after branch (this works in this specific code) and save one cycle.

Overall we could save 3 cycles with reordering the code.

Q6.E

Instruction	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22
xor r3, r3,r3	F	D	X	M	W																		
addiu r2, r3, 1		F	D	X	M	W																	
j L1			F	F	D	X	M	W															
FLUSHED					F	-	-	-	-														
bne r2, r3, -28						F	D*	X	M	W													
END-FLUSHED							F	-	-	-	-												
mul r3, r3, r3								F	D	X0	X1	X2	M	W									
add r4, r1, r4									F	D	D	D	X	M	W								
lw r5, 0(r4)										F	F	F	D	X	M	W							
sub r5, r5, 1													F	D	X	M	W						
sw r5, 0(r6)			We can't fetch an instruction due to structural hazard- we have to stall.											F	F	D	X	M	W				
addiu r3, r3, 1																F	D	X	M	W			
bne r2, r3, -28																	F	D	X	M	W		
END																		F	D	X	M	W	

PURPLE: by passing from execute stage to next instruction's execute stage

GREEEN: structural hazard, because instruction in the fetch stage conflicts with a load instruction in the Memory stage.

GRAY: F F D X M W

In here on fetch is extra but I did not have time to shift everything one to left. In J L1 we don't need to stall a fetch. I have made a mistake and did not have enough time to fix it. So everything after JL1 (third row) has to be shifted to left by one. (SORRY FOR THE CONFUSION).

In this case we have to waste one extra cycle to stall fetch to avoid structural hazard.