CS60003: High Performance Computer Architecture
CLASS-TEST — ② [Spring 2023-2024]
Model Solution

1) (a) $T_{pipelined}|_{min} = T_{longest\_stage}|_{min} + d|_{register} = 2us + 0.1us = \underline{\underline{2.1us}}$

(b) In the steady state, in every 5 clocks, 4 instructions could ~~[illegible]~~ be comple

$$\therefore CPI|_{pipelined} = \frac{5 cycles}{4 instructions} = \underline{\underline{1.25}}$$

(c) Note than in a non-pipelined machine, there is no possibility of any stall
Suppose, a program consists of "I" instructions

∴ time taken to execute the program on the non-pipelined machine:

$$\tau|_{non\text{-}pipelined} = I * CPI * T|_{non\text{-}pipelined} = I \cdot 1 \cdot 7 \; us$$

Similarly, $\tau|_{pipelined} = I * 1.25 * T|_{pipelined} = I * 1.25 * 2.1 \underline{us}$.

$$\therefore Speedup = \frac{\tau_{non\text{-}pipelined}}{\tau_{pipelined}} = \frac{7I}{1.25 * 2.1 * I} = \underline{\underline{2.67}}$$

(d) If the # of pipeline stages become infinite, then, the delay of the pipelined machine is only the delay of the registers, i.e.,

$$\lim_{\# of stages \to \infty} T|_{pipelined} = d|_{register} = \underline{\underline{0.1us}}$$

$$\therefore Speedup|_{ignoring\ extra\ \phi\ cycles} = \frac{7I}{1 * 0.1 I} = 70.$$

$$Speedup|_{considering\ extra\ stall\ \phi\ cycles} = \frac{7I}{1.25 * 0.1} = 56$$

Both answers could be accepted.

This answer also makes sense since the clock cycle time-period is now almost zero (because of infinite # of pipeline stages). Hence, the only pipeline delay now is because of the register delay.

(a) When there is no branch misprediction, the only stalls are because of RAW hazards, which cannot be resolved (∵ there is no forwarding), except those which can be resolved because of the nature of the register file where WB → ID register file reading is possible.

```
Loop: ld,   a1, 0(a2)
      addi, a1, a1, 1
      sd    a1, 0(a2)
      addi  a2, a2, 4
      sub   a4, a3, a2
      bnez  a4, Loop
```

→ : True Dependencies

① : Cannot be resolved (even if there had been forwarding)

② : Cannot be resolved (∵ there is no forwarding.) ~~However, by delaying instruction fetch for sd (see pipeline timing diagram below), they do not effectively affect the performance~~ (WB → ID "forwarding through register file" helps)

③ : Same as for #2. ④ : Same as for #2

Note that since branch outcomes are known in EX stage there are 2 cycles penalty in case of branch mispredictions

~~1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16~~

S: Stall → denotes "forwarding through register file"

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ld	F	D	X	M	W													
addi		F	S	S	D	X	M	W										
sd					F	S	S	D	X	M	W							
addi								F	D	X	M	W						
sub									F	S	S	D	X	M	W			
bnez												F	S	S	D	X	M	W

(b) In the 1st ¢ cycle, assuming the 1-bit branch predictor is in $\boxed{\text{Not Taken}}$ state, there is a misprediction ⇒ 2 ¢ cycles penalty because of pipeline flush.

Again, in the last ¢ cycle, there is similarly a 2 ¢ cycle penalty.

The loop runs total $\frac{396}{4} = 99$ times ~~in all the~~

The middle $99-1-1 = 97$ iterations take ~~18~~ 16 ¢ cycles each (because of overlapped execution)

∴ total # of ¢ cycles $= 97 * \cancel{18}\,16 + 20 + \cancel{20}\,18 = \underline{\underline{\cancel{1786}\ 1590}}$ ¢ cycles

Note: if the 2 cycle stall due to pipeline flushes in the last iteration is ignored, then # of ¢ cycles $=$ ~~1784~~ ~~[illegible]~~ 1588.

Both answers are acceptable.

Note: if initially 1-bit Branch Predictor is in "Taken" state, then there is no misprediction in 1st iteration.

∴ total # of ¢ cycles $= 97*16 + 18 + 18 = \underline{\cancel{[illegible]}\ 1586}$ ¢ cycles (or ~~[illegible]~~ 1584 ¢ cycles)

3) Note: only specific snapshots are shown.

Clock cycle #4

Instr. Status

	Issue	Read Op	Ex. Comp.	Write Res.
ld	1	2	3	4
fsub.d	2			
fdiv.d	3			
fadd.d				

RAW Hazard ← fsub.d, fdiv.d

Structural hazard ← fadd.d

FU Status

	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int.	Yes	Load	f6		x5				
Mult1	No								
Mult2	No								
Add/Sub	Yes	Sub	f8	f6	f2	Int.		No	Yes
Divide	Yes	Div.	f10	f0	f6	Int.		Yes	No

Reg. Res. Status

Clock cycle #		f0	f2	f5	f6	f8	f10
4	FU				Int	Add/Sub	Div

Clock Cycle #7

	Issue	Read Op	Ex. Compl.	Write Res
ld	[illegible] 1	2	3	4
fsub.d	2	5	7	
fdiv.d	3	5		
fadd.d				

Structural hazard ← fadd.d

FU Status

	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
M1	No								
M2	No								
Add/Sub	Yes	Sub	f8	f6	f2			Yes	Yes
Divide	Yes	Div	f10	f0	f6			Yes	Yes

Reg. Reg. Status

Clock Cycle #		f0	f2	f5	f6	f8	f10
4	FU				~~Int~~	Add/Sub	Div

Clock Cycle #10.

	Iss.	Read Op	Ex Comp.	Wr. Res
ld	1	2	3	4
fsub.d	2	5	7	8
fdiv.d	3	5		
fadd.d	9	10		

	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
M1	No								
M2	No								
Add/Sub	Yes	Add	f5	f8	f6			Yes	Yes
Div.	Yes	Div.	f10	f0	f6			Yes	Yes

Reg. Res. Status

C Cycle#		f0	f2	f5	f6	f8	f10
10	FU			Add/Sub			Div.

Clock Cycle #13.

	Iss.	Read Op.	Ex. Comp.	Wr. Res.
ld	1	2	3	4
fsub.d	2	5	7	8
fdiv.d	3	5		
fadd.d	9	10	12	13

	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int.	No								
M1	No								
M2	No								
Add/Sub	Yes		f5	f8	f6			Yes	Yes
Div.	Yes	Div.	f10	f0	f6			Yes	Yes

Reg. Status.

Clock Cycle#		f0	f2	f5	f6	f8	f10
13	FU			Add/Sub			Div.

~~Division ends execution on Cycle #45~~.

Division ends execution ~~from~~ on Clock #45

" " Write Register on Clock #46.