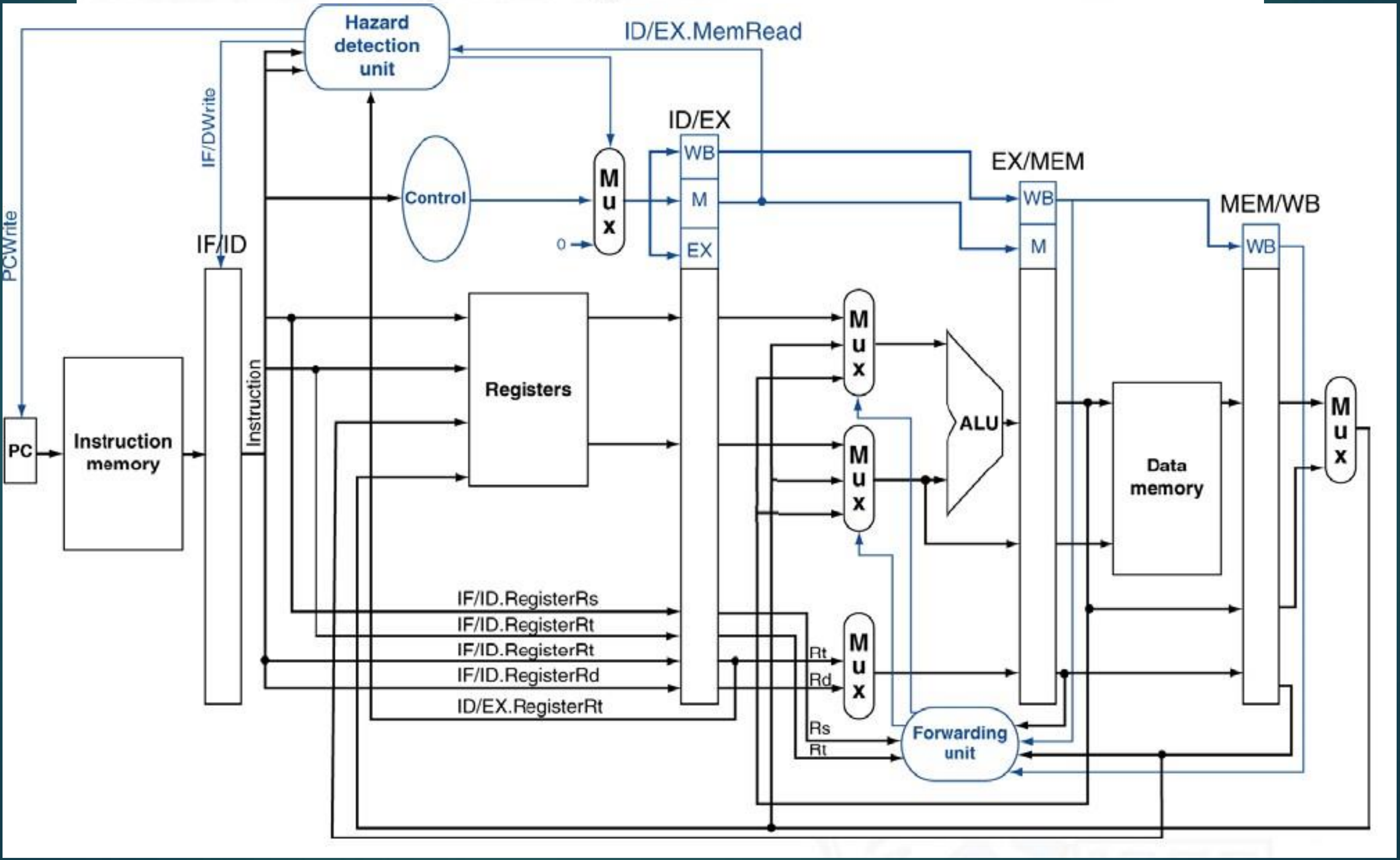
VE370 RC6

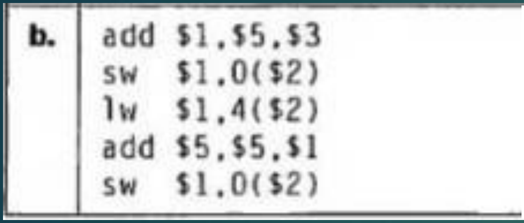
SOLUTION FOR HW6

4.21.4(b)

b.	add $1,$5,$3 sw $1,0($2) lw $1,4($2) add $5,$5,$1 sw $1,0($2)

4.21.4 [20] <4.7> If there is forwarding, for the first five cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.60.





b.
```
add $1,$5,$3
sw  $1,0($2)
lw  $1,4($2)
add $5,$5,$1
sw  $1,0($2)
```

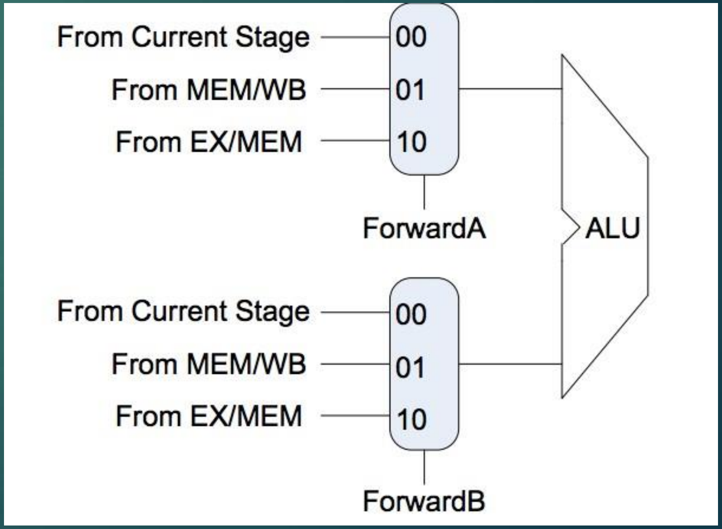
Load-use hazard when

- ID/EX.MemRead and
  ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
  (ID/EX.RegisterRt = IF/ID.RegisterRt))

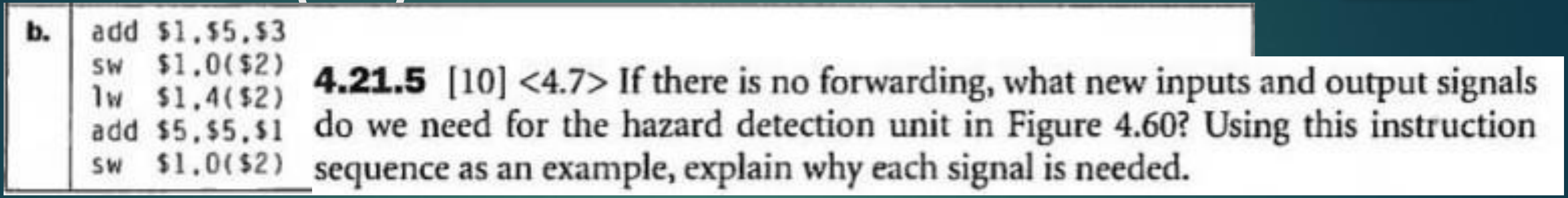
Hazard Detection is in **ID** stage

IF	ID	EX	MEM	WB
Add 1, 5, 3	/	/	/	/
Sw 1, 0(2)	Add 1, 5, 3	/	/	/
Lw 1, 4,(2)	Sw 1, 0(2)	Add 1, 5, 3		
Add 5, 5, 1	Lw 1, 4(2)	Sw 1, 0(2)	Add 1, 5, 3	
Sw…	Add 5, 5, 1	Lw 1, 4,(2)	Sw 1, 0(2)	Add 1, 5, 3

PCWrite	ALUin1	ALUin2
1	x	x
1	x	x
1	00	00
1	00	10
0	00	00



4.21.5(b)



b.	add $1,$5,$3 sw $1,0($2) lw $1,4($2) add $5,$5,$1 sw $1,0($2)

4.21.5 [10] <4.7> If there is no forwarding, what new inputs and output signals do we need for the hazard detection unit in Figure 4.60? Using this instruction sequence as an example, explain why each signal is needed.

Need input and output signal that judge register dependency and insert stalls to the cycle.

In the ID stage, we need to read the values from the register file. If no forwarding exists, additional input is needed to detect if there is dependency on the instruction in the **EX** and **MEM** stage.

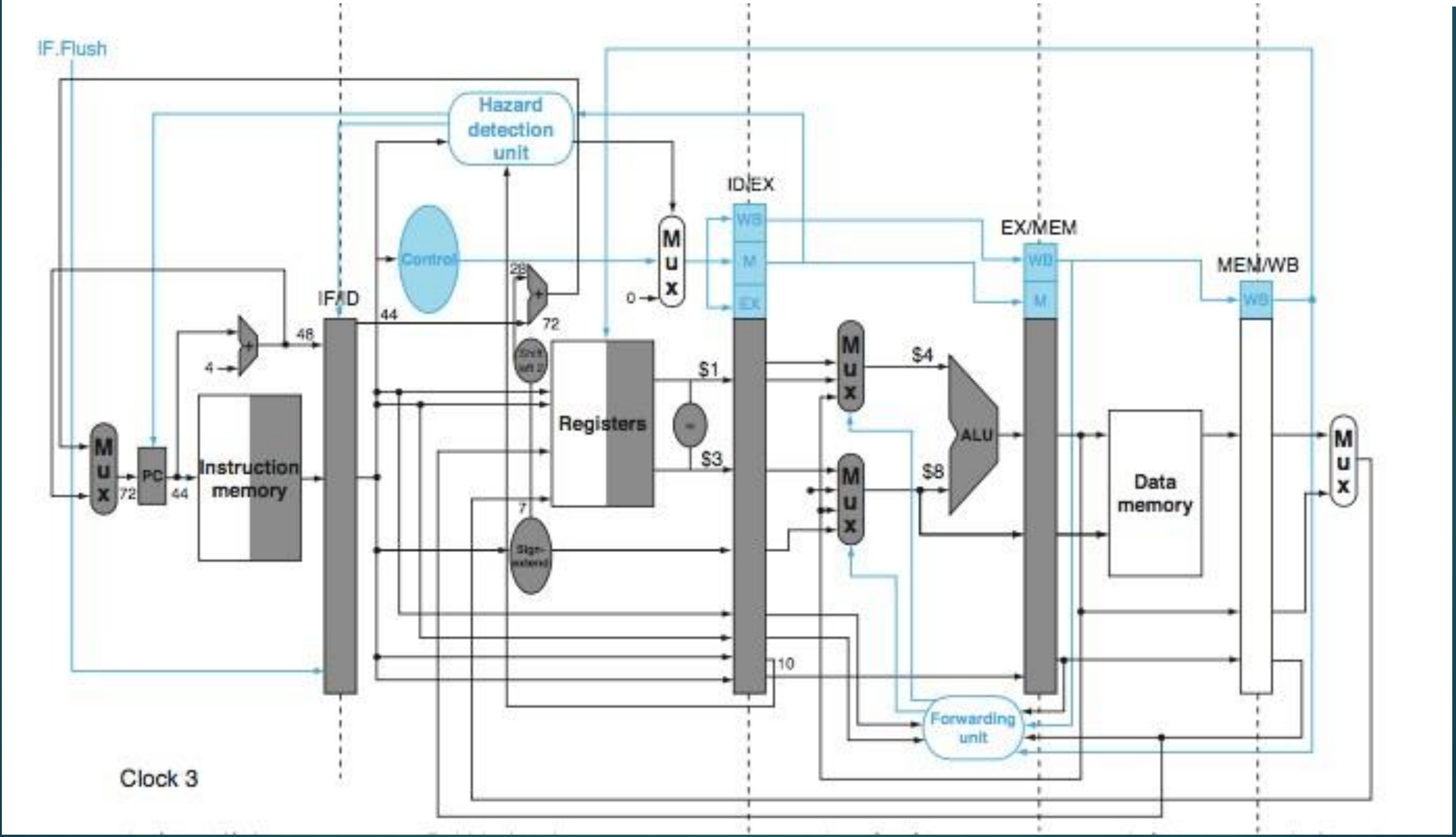
No additional output is needed.

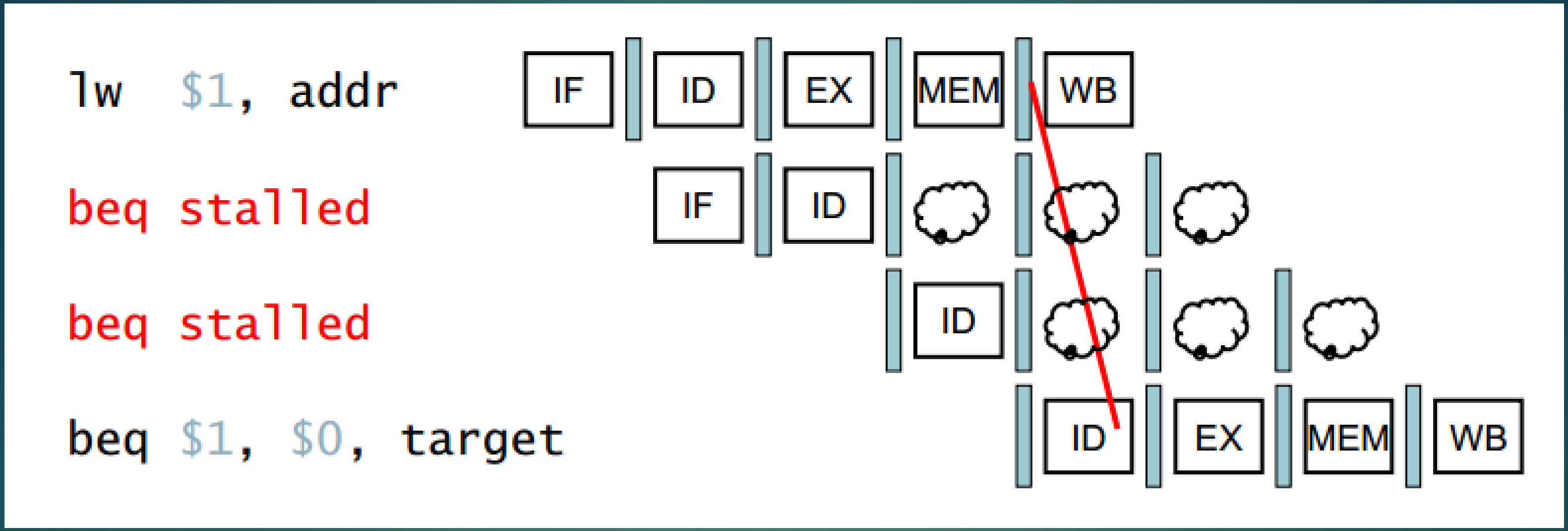
4.22.4

b.

```
         LW  R2,0(R1)
Label1:  BEQ R2,R0,Label2 ; Not taken once, then taken
         LW  R3,0(R2)
         BEQ R3,R0,Label1 ; Taken
         ADD R1,R3,R1
Label2:  SW  R1,0(R2)
```

4.22.4 [10] <4.8> Using the first branch instruction in the given code as an example, describe the hazard detection logic needed to support branch execution in the ID stage as in Figure 4.62. Which type of hazard is this new logic supposed to detect?

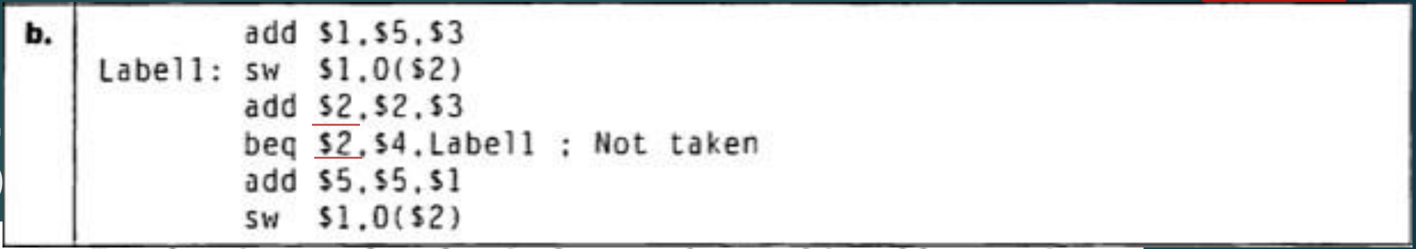




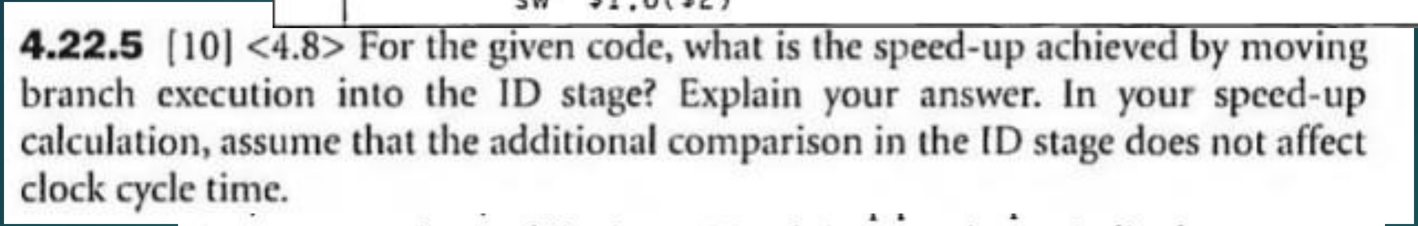
Need add several nops when there are register dependency.
They are data hazard.

e.g. Lw $1 ….
beq $1, …..
then need to stall for 2 cycles

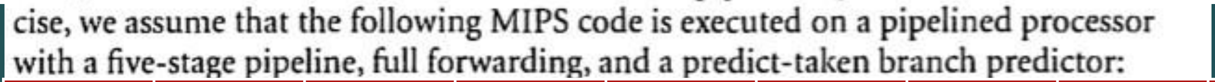
4.22.5



```
b.          add  $1,$5,$3
    Label1: sw   $1,0($2)
            add  $2,$2,$3
            beq  $2,$4,Label1 ; Not taken
            add  $5,$5,$1
            sw   $1,0($2)
```



4.22.5 [10] <4.8> For the given code, what is the speed-up achieved by moving branch execution into the ID stage? Explain your answer. In your speed-up calculation, assume that the additional comparison in the ID stage does not affect clock cycle time.

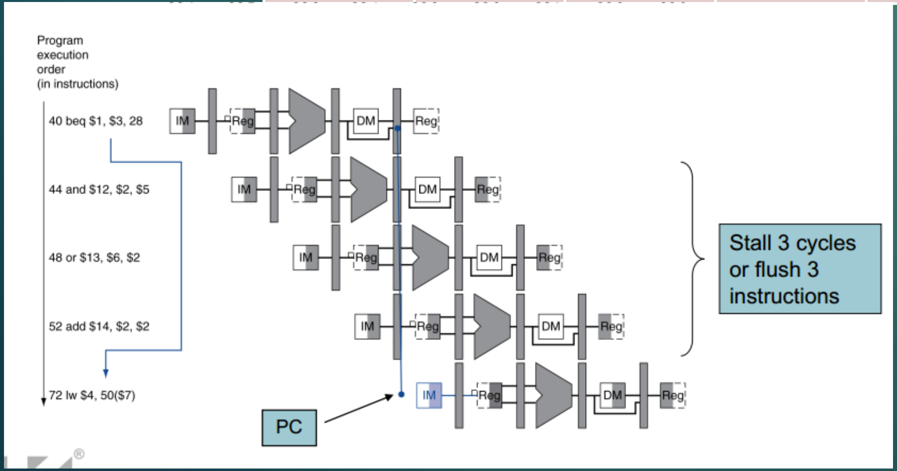


cise, we assume that the following MIPS code is executed on a pipelined processor with a five-stage pipeline, full forwarding, and a predict-taken branch predictor:

Original:

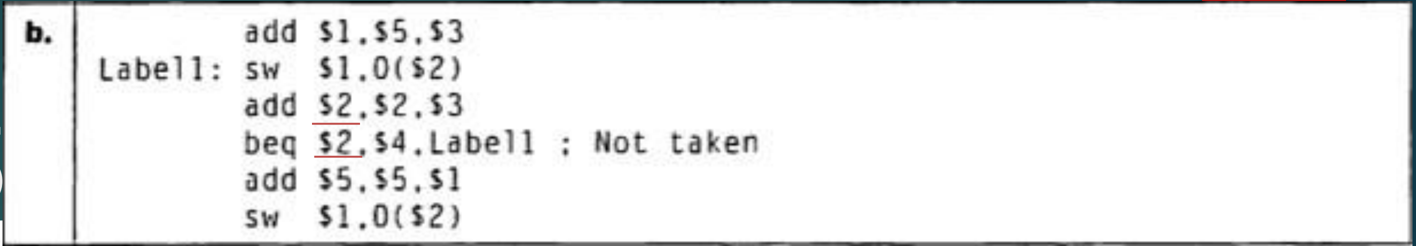
13 cycle

add	IF	ID	EX	MEM	WB				
sw		IF	ID	EX	MEM	WB			
add			IF	ID	EX	MEM	WB		
beq				IF	ID	EX	MEM	WB	
add								IF	ID
sw									IF

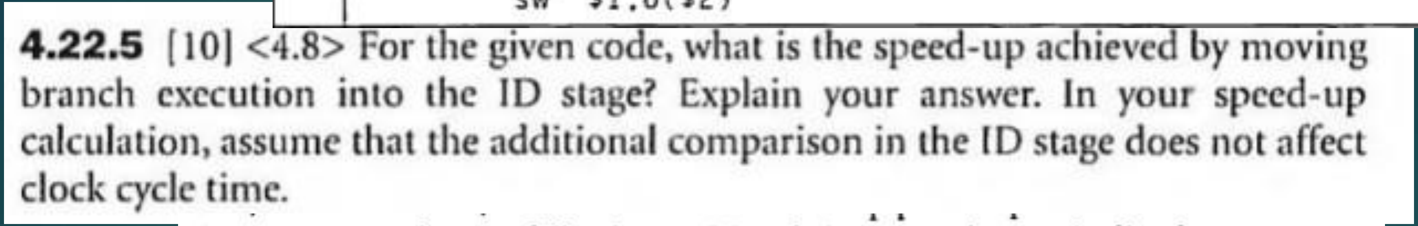


If determined in MEM

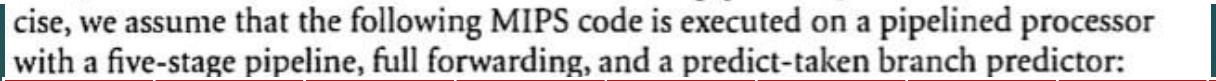
4.22.5



```
b.          add  $1,$5,$3
   Label1:  sw   $1,0($2)
            add  $2,$2,$3
            beq  $2,$4,Label1  ; Not taken
            add  $5,$5,$1
            sw   $1,0($2)
```



4.22.5 [10] <4.8> For the given code, what is the speed-up achieved by moving branch execution into the ID stage? Explain your answer. In your speed-up calculation, assume that the additional comparison in the ID stage does not affect clock cycle time.



cise, we assume that the following MIPS code is executed on a pipelined processor with a five-stage pipeline, full forwarding, and a predict-taken branch predictor:

Original:

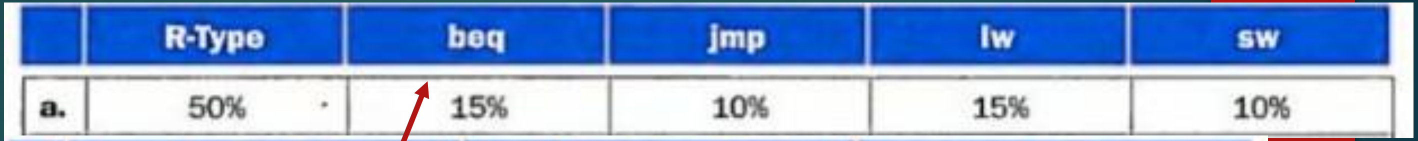
13 cycle

add	IF	ID	EX	MEM	WB				
sw		IF	ID	EX	MEM	WB			
add			IF	ID	EX	MEM	WB		
beq				IF	ID	EX	MEM	WB	
add								IF	ID
sw									IF

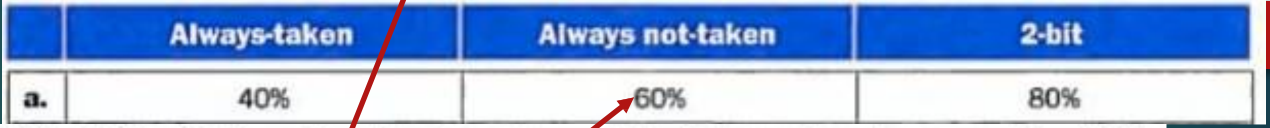
Now:
11 cycle

add	IF	ID	EX	MEM	WB				
sw		IF	ID	EX	MEM	WB			
add			IF	ID	EX	MEM	WB		
beq				IF	nop	ID	EX	MEM	WB
add						IF	ID	EX	MEM
sw							IF	ID	EX

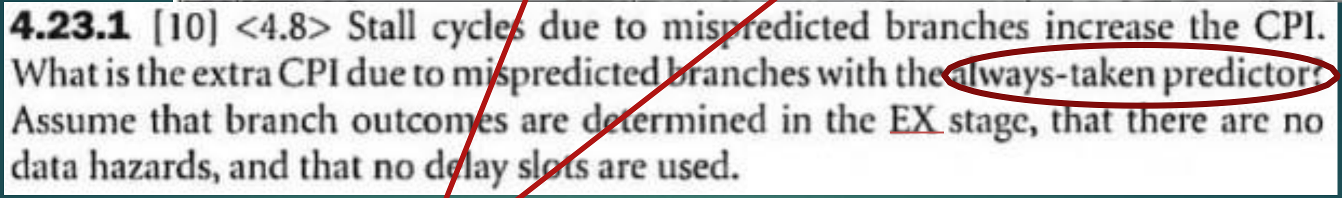
4.23.1



	R-Type	beq	jmp	lw	sw
a.	50%	15%	10%	15%	10%



	Always-taken	Always not-taken	2-bit
a.	40%	60%	80%

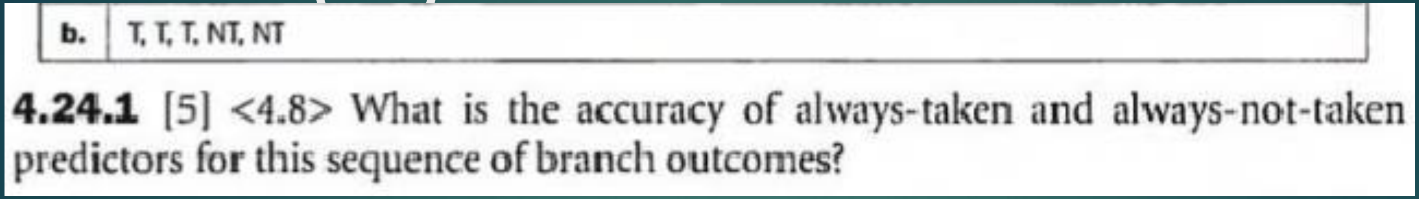


4.23.1 [10] <4.8> Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the EX stage, that there are no data hazards, and that no delay slots are used.

Branch penalty: 2

2 * 0.6 * 0.15 = 0.18

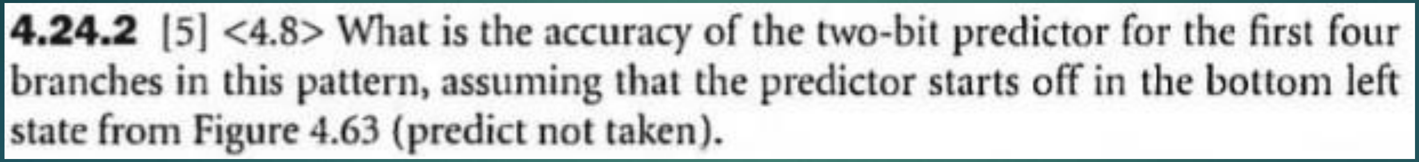
4.24.1(b)



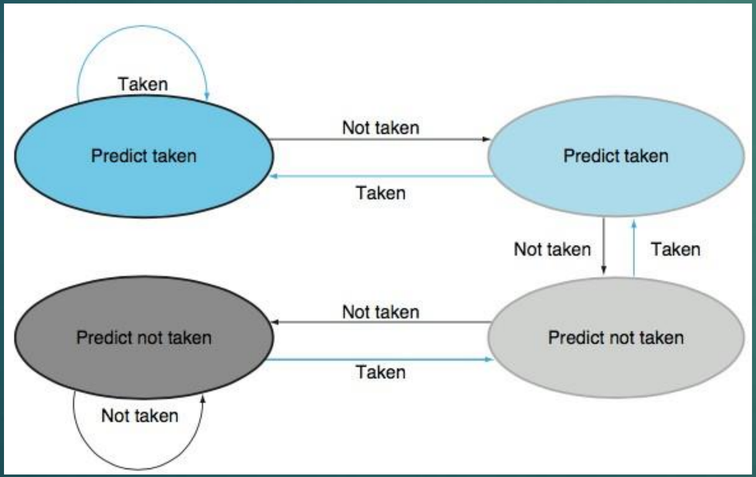
b.	T, T, T, NT, NT

4.24.1 [5] <4.8> What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

Always-taken: 3/5 = 60%
Always-not-taken = 2/5 = 40%



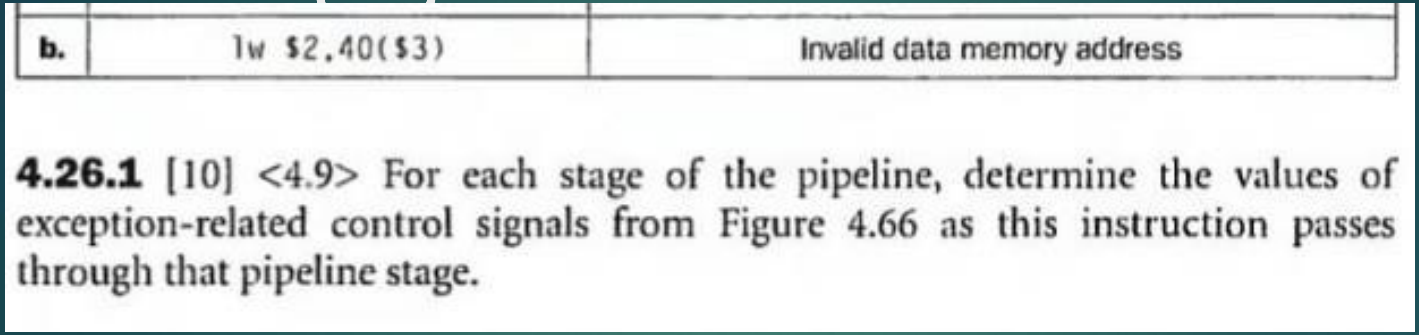
4.24.2 [5] <4.8> What is the accuracy of the two-bit predictor for the first four branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.63 (predict not taken).



Prediction: NT, NT, T, T

Accuracy = ¼ = 25%

4.26.1(b)



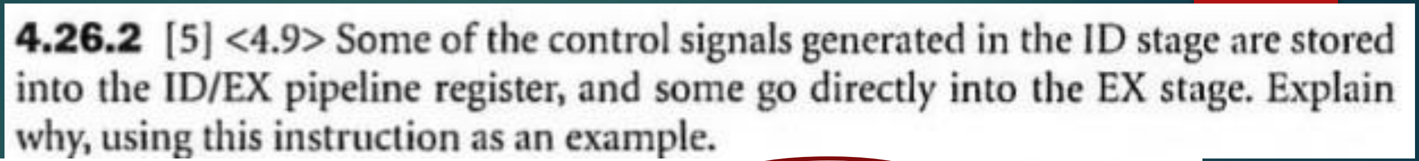
b.	lw $2,40($3)	Invalid data memory address

4.26.1 [10] <4.9> For each stage of the pipeline, determine the values of exception-related control signals from Figure 4.66 as this instruction passes through that pipeline stage.

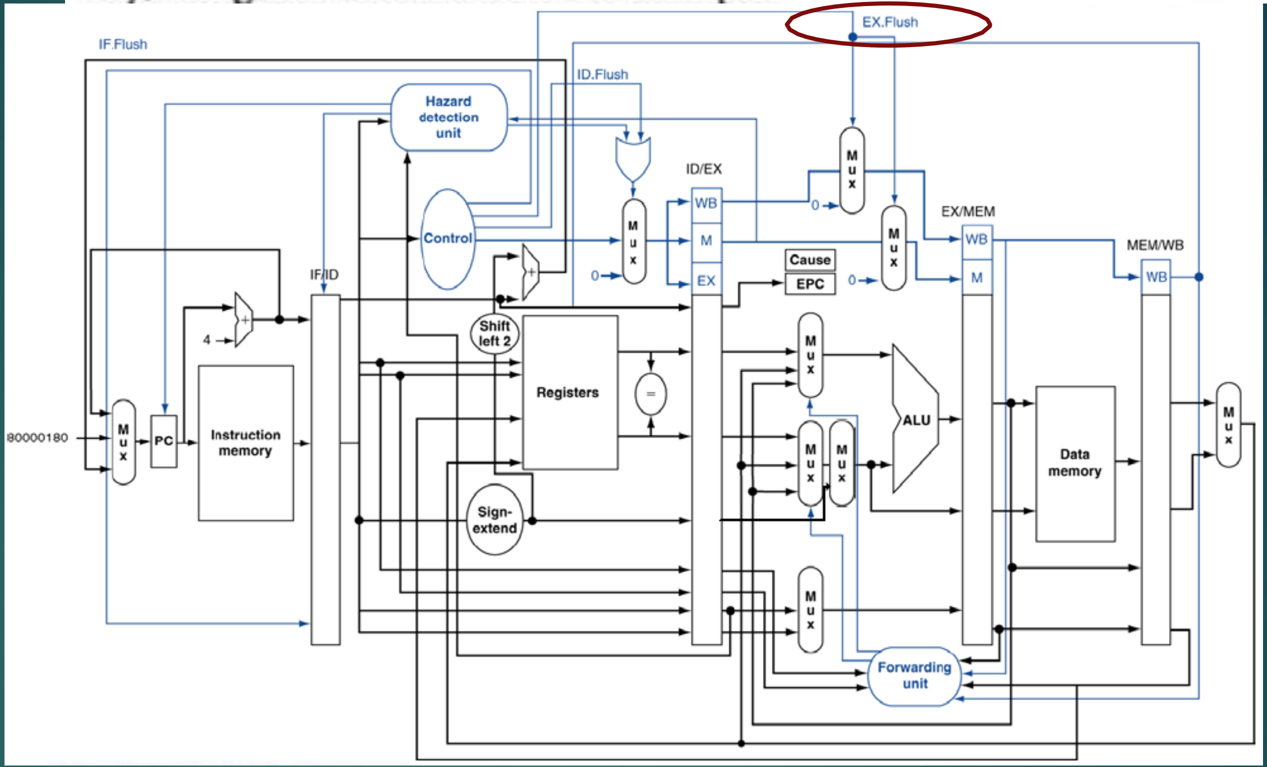
IF	ID	EX	MEM	WB
flush	flush	flush	flush	/
			exception	

Set PC to handler, Save current address in EPC

4.26.2



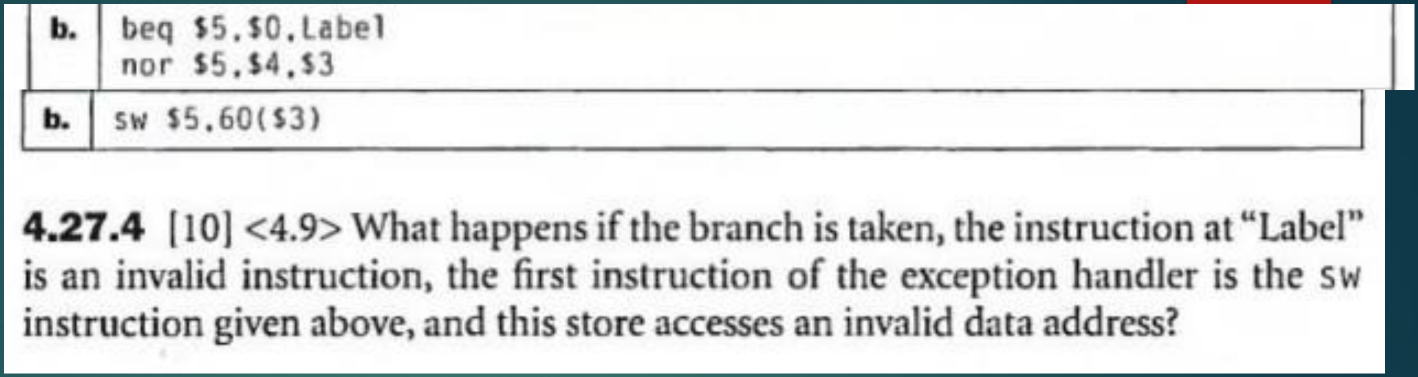
4.26.2 [5] <4.9> Some of the control signals generated in the ID stage are stored into the ID/EX pipeline register, and some go directly into the EX stage. Explain why, using this instruction as an example.



Ex.flush goes immediately from ID to EX. Because we need to flush all instruction after the one that exception occurred.

For this MEM exception, it tell the control in ID and then flush ins in IF,ID,EX,MEM and handle the exception

4.27.4



b.	`beq $5,$0,Label`
	`nor $5,$4,$3`

b.	`sw $5,60($3)`

4.27.4 [10] <4.9> What happens if the branch is taken, the instruction at "Label" is an invalid instruction, the first instruction of the exception handler is the `sw` instruction given above, and this store accesses an invalid data address?

The Processor can not go back to original beq instruction after handling the exceptions.

Lost the addr of beq, since it is overwritten

PC	INS	EPC	HANDLER
Addr of beq	beq…	Addr of beq	Sw…
Addr of sw	sw…	Addr of sw	Something
something	…	Addr of sw	/