20200052 김규빈

2024-1 Computer Architecture Homework #3

Due: 6/7 (Fri) 11:59 p.m.

1. [25] When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are beginning with the datapath from Figure 4.21, the latencies from Exercise 4.7, and the following costs:

I-Mem	Register File	Mux	ALU	Adder	D-Mem	Single Register	Sign extend	Sign gate	Control
1000	200	10	100	30	2000	5	100	1	500

Latencies from Exercise 4.7

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Singlegate	Register Read	Register Setup	Sign extend	Control
250ps	150ps	25ps	200ps	150ps	5ps	30ps	20ps	50ps	50ps

Instruction mix from Exercise 4.8

R-type/I-type (non-lw)	lw	sw	beq
52%	25%	11%	12%

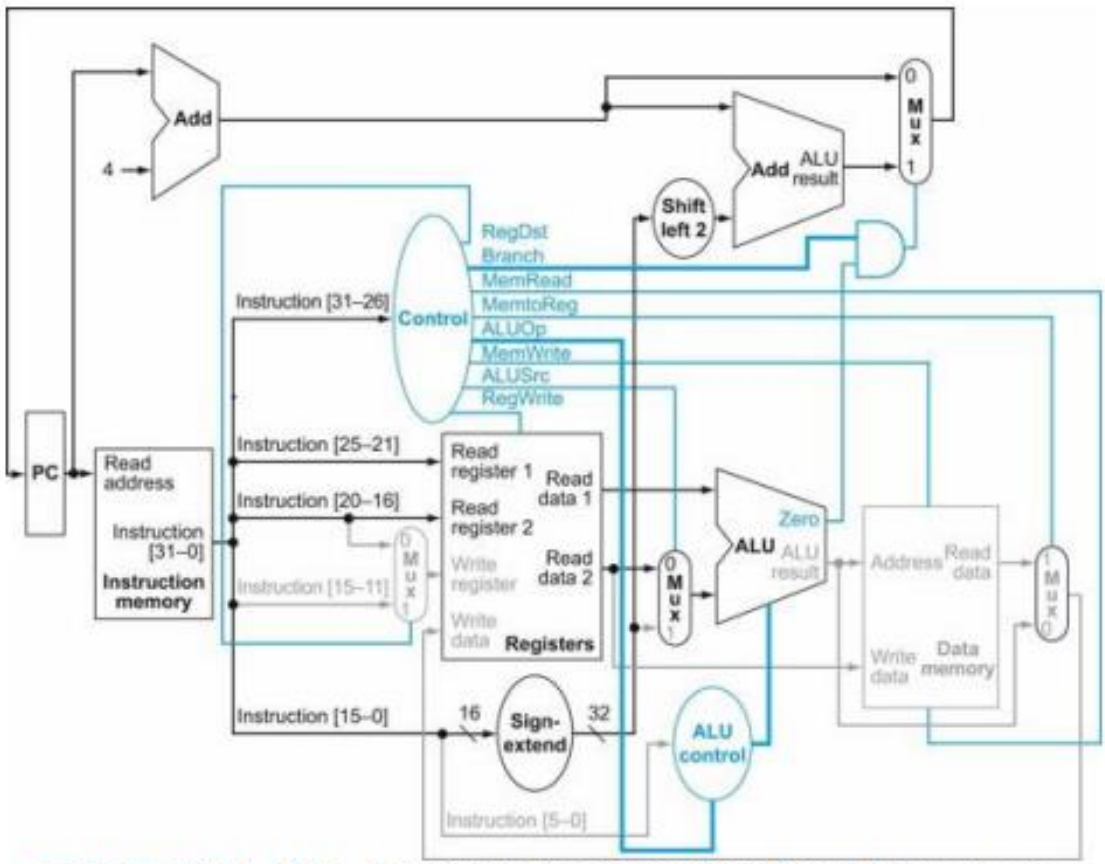


FIGURE 4.21 The datapath in operation for a branch-on-equal instruction.
The control lines, datapath units, and connections that are active are highlighted. After using the register file and ALU to perform the compare, the Zero output is used to select the next program counter from between the two candidates.

Suppose doubling the number of general purpose registers from 32 to 64 would reduce the number of lw and sw instruction executed by 12%, but increase the latency of the register file from 150 ps to 160 ps and double the cost from 200 to 400. (Use the instruction mix from Exercise 4.8 and ignore the other effects on the ISA discussed in Exercise 2.18.)

a. [10] What is the speedup achieved by adding this improvement?

b. [10] Compare the change in performance to the change in cost.

c. [5] Given the cost/performance ratios you just calculated, describe a situation where it makes sense to add more registers and describe a situation where it doesn't make sense to add more registers.

2. [20] Consider the fragment of MIPS assembly below:

```
sd $s5, 12($s3)
Id $s5, 8($s3)
sub $s4, $s2, $s1
beqz $s4, label
add $s2, $s0, $s1
sub $s2, $s6, $s1
```

Suppose we modify the pipeline so that it has only one memory (that handles both instructions and data). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

a. [5] Draw a pipeline diagram to show were the code above will stall.

b. [5] In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

c. [5] Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding NOPs to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

d. [5] Approximately how many stalls would you expect this structural hazard to generate in a typical program? (Use the instruction mix from Exercise 4.8.)

3. [20] Consider the following loop.

```
LOOP: ld $s0, 0($s3)
      ld $s1, 8($s3)
      add $s2, $s0, $s1
      addi $s3, $s3, -16
      bnez $s2, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.

a. [10] Show a pipeline execution diagram for the first two iterations of this loop.

b. [10] Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the addi is in the IF stage. End with the cycle during which the bnez is in the IF stage.)

4. [35] Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add $s3, $s1, $s0
lw $s2, 4($s3)
lw $s1, 0($s4)
or $s2, $s3, $s2
sw $s2, 0($s3)
```

a. [5] If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

b. [5] Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register $t0 can be used to hold temporary values in your modified code.

c. [5] If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

d. [10] If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.59.

e. [5] If there is no forwarding, what new input and output signals do we need for the hazard detection unit in Figure 4.59? Using this instruction sequence as an example, explain why each signal is needed.

f. [5] For the new hazard detection unit from 4-e(above question), specify which output signals it asserts in each of the first five cycles during the execution of this code.

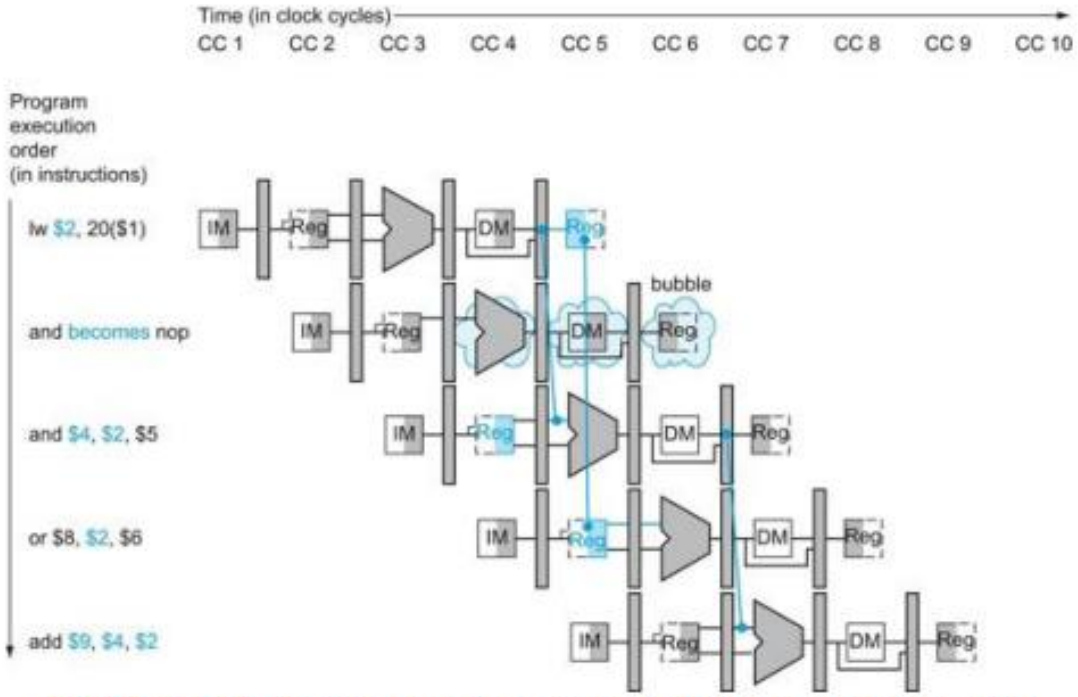


FIGURE 4.59 The way stalls are really inserted into the pipeline.

A bubble is inserted beginning in clock cycle 4, by changing the `and` instruction to a nop. Note that the `and` instruction is really fetched and decoded in clock cycles 2 and 3, but its EX stage is delayed until clock cycle 5 (versus the unstalled position in clock cycle 4). Likewise the `OR` instruction is fetched in clock cycle 3, but its ID stage is delayed until clock cycle 5 (versus the unstalled clock cycle 4 position). After insertion of the bubble, all the dependences go forward in time and no further hazards occur.

1. [25] When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are beginning with the datapath from Figure 4.21, the latencies from Exercise 4.7, and the following costs:

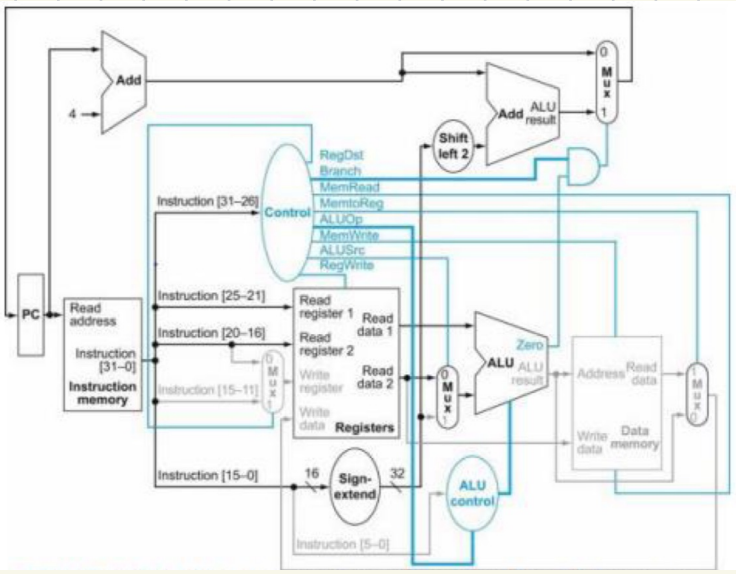
I-Mem	Register File	Mux	ALU	Adder	D-Mem	Single Register	Sign extend	Sign gate	Control
1000	200	10	100	30	2000	5	100	1	500

Latencies from Exercise 4.7

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Singlegate	Register Read	Register Setup	Sign extend	Control
250ps	150ps	25ps	200ps	150ps	5ps	30ps	20ps	50ps	50ps

Instruction mix from Exercise 4.8

R-type/I-type (non-lw)	lw	sw	beq
52%	25%	11%	12%



Suppose doubling the number of general purpose registers from 32 to 64 would reduce the number of lw and sw instruction executed by 12%, but increase the latency of the register file from 150 ps to 160 ps and double the cost from 200 to 400. (Use the instruction mix from Exercise 4.8 and ignore the other effects on the ISA discussed in Exercise 2.18.)

a. [10] What is the speedup achieved by adding this improvement?

a. doubling the number of general purpose register

→ reduce the number of lw and sw instruction executed by 12%

→ reduce the number of all instruction to $(0.12)(0.25+0.11) \approx 4.3\%$

After change, total number of instructions : $100 - 4.3 = 95.7\%$.

given CPU : single-cycle, critical path : load instruction

* lw (ex. lw $s1, 100 ($s2))

Register read + I-mem + Register file + MUX + ALU + D-mem + MUX + Register Setup

$= 30 + 250 + 150 + 25 + 200 + 250 + 25 + 20$

$= 950$ (ps)

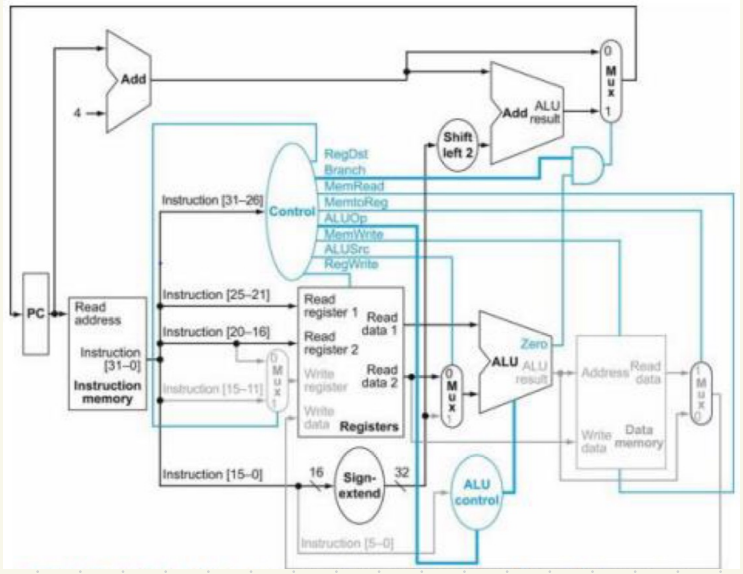
After change, latency of register file : 150 ps → 160 ps.

∴ minimum clock period : 960 ps

$$\therefore \text{speed up} = \frac{950}{960 \times 0.957} \approx 1.03.$$

b. [10] Compare the change in performance to the change in cost.

I-Mem	Register File	Mux	ALU	Adder	D-Mem	Single Register	Sign extend	Sign gate	Control
1000	200	10	100	30	2000	5	100	1	500



b. before

Single Register (PC) 5

I-Mem 1000

Register File 200

ALU 100

Control 500

ALU Control 500

D-Mem 2000

Sign-extend 100

adder 30×2=60

MUX 10×4=40

Sign gate 1

Before : $5+1000+200+100+500+500+2000+100+60+40+1=4506$

after change, cost of Register File : 200 → 400

After : $4506+200=4706$

change in cost : $\frac{4706}{4506} \approx 1.044$

∴ The change in cost is approximately 4.4%.

c. [5] Given the cost/performance ratios you just calculated, describe a situation where it makes sense to add more registers and describe a situation where it doesn't make sense to add more registers.

C.

performance (latency)는 약 3% 개선

but. cost는 약 4.4% 증가

∴ cost가 중요한 상황에서는 register를 2배로 늘리면 cost가 약 4.4% 증가하므로, 레지스터를 늘리는 것은 타당하지 않음.

but. performance (latency)가 중요한 상황에서는 register를 2배로 늘리면 performance가 약 3% 개선되므로, register를 늘리는 것이 타당함.

2. [20] Consider the fragment of MIPS assembly below:

```
sd $s5, 12($s3)
ld $s5, 8($s3)
sub $s4, $s2, $s1
beqz $s4, label
add $s2, $s0, $s1
sub $s2, $s6, $s1
```

Suppose we modify the pipeline so that it has only one memory (that handles both instructions and data). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

a. [5] Draw a pipeline diagram to show were the code above will stall.

--: stall

a.

	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12
sd $s5, 12($s3)	IF	ID	EX	MEM	WB							
ld $s5, 8($s3)		IF	ID	EX	MEM	WB						
sub $s4, $s2, $s1			IF	ID	EX	MEM	WB					
beqz $s4, label				--	--	IF	ID	EX	MEM	WB		
add $s2, $s0, $s1							IF	ID	EX	MEM	WB	
sub $s2, $s6, $s1								IF	ID	EX	MEM	WB

b. [5] In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

b. Reordering instructions won't reduce the number of stalls/NOPs. Since every instruction fetch and data access causes a structural hazard. The conflict will still occur, just between different pairs of instructions. Thus, reordering the code doesn't solve the issue of stalls.

c. [5] Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding NOPs to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

C.

No. we cannot solve structural hazard in hardware.
structural hazard는 여러개의 instruction이 프로세서의 동일한 resource에 대해 동시에 access하는 상황에서 발생한다.
이러한 상황은 instruction이 실행되는 도중에 발생하기 때문에 hardware에서 handle 불가능.

However. we can solve structural hazard by adding NOPs to the code.
Structural hazard가 발생한 부분에 NOP을 삽입하고 hazard가 발생하지 않을 때 instruction 다시 실행하면 된다.

d. [5] Approximately how many stalls would you expect this structural hazard to generate in a typical program? (Use the instruction mix from Exercise 4.8.)

d. Using the given instruction mix, structural hazards from data access instructions (lw and sw) cause stalls. The percentage of lw is 25%, and sw is 11%, summing up to 36%.
Therefore, approximately 36% of cycles in a typical program would experiance stalls due to these hazards.

3. [20] Consider the following loop.

```
LOOP: ld $s0, 0($s3)
      ld $s1, 8($s3)
      add $s2, $s0, $s1
      addi $s3, $s3, -16
      bnez $s2, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.

a. [10] Show a pipeline execution diagram for the first two iterations of this loop.

O: the stage that do not perform useful work.

--: stall

A.

```
ld $s0, 0($s3)
ld $s1, 8($s3)
add $s2, $s0, $s1
addi $s3, $s3, -16
bnez $s2, LOOP
ld $s0, 0($s3)
ld $s1, 8($s3)
add $s2, $s0, $s1
addi $s3, $s3, -16
bnez $s2, LOOP
```

	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13	cc14	cc15	cc16
ld $s0, 0($s3)	IF	ID	EX	MEM	WB											
ld $s1, 8($s3)		IF	ID	EX	MEM	WB										
add $s2, $s0, $s1			IF	ID	--	EX	MEM	WB								
addi $s3, $s3, -16				IF	--	ID	EX	MEM	WB							
bnez $s2, LOOP					--	IF	ID	EX	MEM	WB						
ld $s0, 0($s3)							IF	ID	EX	MEM	WB					
ld $s1, 8($s3)								IF	ID	EX	MEM	WB				
add $s2, $s0, $s1									IF	ID	--	EX	MEM	WB		
addi $s3, $s3, -16										IF	--	ID	EX	MEM	WB	
bnez $s2, LOOP											--	IF	ID	EX	MEM	WB

b. [10] Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the addi is in the IF stage. End with the cycle during which the bnez is in the IF stage.)

b.

◯ : the stage that do not perform useful work.

Pipeline stages are not performing useful work when stalls occur or when instructions are not doing any useful work.

As shown in the diagram, there are no cycles where all five pipeline stages are performing useful work simultaneously. Therefore, while the pipeline is full, there are no cycles with all stages doing useful work.

4. [35] Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add $s3, $s1, $s0
lw $s2, 4($s3)
lw $s1, 0($s4)
or $s2, $s3, $s2
sw $s2, 0($s3)
```

a. [5] If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

a.

	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13	cc14
add $s3, $s1, $s0	IF	ID	EX	MEM	WB									
nop		--	--	--	--	--								
nop			--	--	--	--	--							
lw $s2, 4($s3)				IF	ID	EX	MEM	WB						
lw $s1, 0($s4)					IF	ID	EX	MEM	WB					
nop						--	--	--	--	--				
or $s2, $s3, $s2							IF	ID	EX	MEM	WB			
nop								--	--	--	--	--		
nop									--	--	--	--	--	
sw $s2, 0($s3)										IF	ID	EX	MEM	WB

b. [5] Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register $t0 can be used to hold temporary values in your modified code.

b. We cannot reduce NOP by changing and/or rearranging the code.
In the given sequence, reducing the number of NOPs is impossible because of inherent data hazards. Each instruction depends on the result of the previous one, necessitating NOPs to avoid conflicts. Using register $t0 does not eliminate the need for these NOPs due to the unavoidable delays from data dependencies.

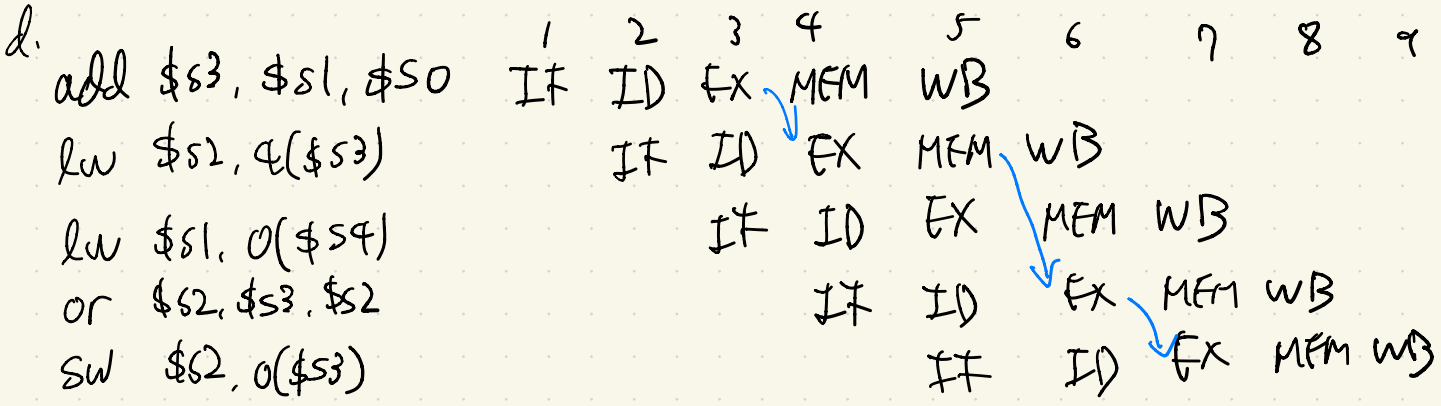
c. [5] If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

C. With forwarding implemented but without a hazard detection unit, the code executes correctly. Hazard detection is only needed to insert stalls when an instruction following a load uses the load's result immediately. In this sequence, no such situation occurs, so no stalls are required.

```
add $s3, $s1, $s0   IF ID EX MEM WB
lw  $s2, 4($s3)        IF ID EX MEM WB
lw  $s1, 0($s4)           IF ID EX MEM WB
or  $s2, $s3, $s2            IF ID EX MEM WB
sw  $s2, 0($s3)                 IF ID EX MEM WB
```

d. [10] If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.59.

d.



There are no stalls

→ PCWrite, IF/IDWrite = 1 } hazard detection unit
control MUX = 0

- EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and EX/MEM.RegisterRd = ID/EX.RegisterRs ForwardA = 10
- EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and EX/MEM.RegisterRd = ID/EX.RegisterRt ForwardB = 10
- MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and MEM/WB.RegisterRd = ID/EX.RegisterRs ForwardA = 01
- MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and MEM/WB.RegisterRd = ID/EX.RegisterRt ForwardB = 01

cc1. ForwardA = X, ForwardB = X
cc2. ForwardA = X, ForwardB = X
cc3. ForwardA = 00, ForwardB = 00
cc4. ForwardA = 10, ForwardB = 00 (10: base register taken from result of previous instruction)
cc5. ForwardA = 00, ForwardB = 00
cc6. ForwardA = 00, ForwardB = 01 (01: base register taken from result of two instructions previous)
cc7. ForwardA = 00, ForwardB = 10 (10: base register taken from result of previous instruction)

e. [5] If there is no forwarding, what new input and output signals do we need for the hazard detection unit in Figure 4.59? Using this instruction sequence as an example, explain why each signal is needed.

e. The hazard detection unit needs the rd value from the MEM/WB register. This is because the instruction in the ID stage might depend on the results from either the instruction in the EX stage or the MEM stage.
The existing hazard detection unit already has the rd value from the EX/MEM register. No additional output signals are needed since the current output signals can handle stalling the pipeline. The rd value from EX/MEM detects hazards between the first lw and the or instruction.
rd of EX/MEM : detect hazard between add $s3, $s1, $s0 and lw $s2, 4($s3)
rd of MEM/WB : detect hazard between lw $s2, 4($s3) and or $s2, $s3, $s2

f. [5] For the new hazard detection unit from 4-e(above question), specify which output signals it asserts in each of the first five cycles during the execution of this code.

f.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
add $s3, $s1, $s0	IF	ID	EX	MEM	WB		
lw $s2, 4($s3)		IF	ID	--	--	EX	MEM
lw $s1, 0($s4)			IF	--	--	ID	EX
or $s2, $s3, $s2						IF	ID
sw $s2, 0($s3)							IF

CC1 : PCWrite = 1, IF/ID Write = 1, control mux = 0

CC2 : PCWrite = 1, IF/ID Write = 1, control mux = 0

CC3 : PCWrite = 1, IF/IDWrite = 1, control mux = 0

CC4 : PCWrite = 0, IF/IDWrite = 0, control mux = 1

CC5 : PCWrite = 0, IF/IDWrite = 0, control mux = 1