HW3.

2021154247 신지원

1.

a.

R-type = register read + ALU + register write

= 30ps + 200ps + 20ps = 250ps

lw = I-mem + register read + ALU + D-mem + register write

= 250 + 30 + 200 + 250 + 20 = 750

sw = I-mem + register read + ALU + D-mem

= 250 + 30 + 200 + 250 = 730

beq = I-mem + register read + ALU

= 250 + 30 + 200 = 480

old CPI = 0.52 × 250 + 0.25 × 750 + 0.11 × 730 + 0.12 × 480

= 455.4 ps

old CPU = 455.4 n ps

lw = 25% × 0.88 = 22%

sw = 11% × 0.88 = 9.68%

new CPI = 0.52 × 260 + 0.22 × 760 + 0.0968 × 740 + 0.12 × 490

= 432.912 ps

new CPU = 432.912 n ps

$$\text{speedup} = \frac{\text{old CPU}}{\text{new CPU}} = \frac{455.4}{432.912} \approx 1.052$$

∴ 5.2% 상승

b.

Total cost of CPU

	old	new
I-Mem	1000	1000
Register file	200	400
Mux	3×10 = 30	3×10 = 30
ALU	100	100
Adder	2×30 = 60	2×30 = 60
D-Mem	2000	2000
Single register	5	5
Sign Extend	100	100
Sign gate	1	1
Control	500	500
total CPU	3996	4196

$$\frac{\text{new CPU}}{\text{old CPU}} = \frac{4196}{3996} \approx 1.05$$

∴ 5% 상승

c.

2과 b의 답안을 통해 register를 추가하였을 때 비용이 증가하더라도 성능이 향상될 수 있음을 확인하였다. 이처럼 메모리 접근 횟수가 많거나 비용이 충분한 상황에서는 register 추가가 효율적일 수 있지만 비용 대비 성능이 효율성을 추구하는 프로젝트라면 합리적이지 않을 수 있다.

2.

a.

Program execution order \ cycle	1	2	3	4	5	6	7	8	9	10
sw $s5, 12($s3)	IF	ID	EX	MEM	WEB					
lw $s5, 8($s3)		IF	ID	EX	MEM	WEB				
sub $s4, $s2, $s1			IF	ID	EX	MEM	WEB			
beqz $s4, label				IF	ID	EX	MEM	WEB		
add $s2, $s0, $s1					IF	ID	EX	MEM	WEB	
sub $s2, $s6, $s1						IF	ID	EX	MEM	WEB

b.

Program execution order \ cycle	1	2	3	4	5	6	7	8	9	10
sw $s5, 12($s3)	IF	ID	EX	[MEM]	WEB					
lw $s5, 8($s3)		IF	ID	EX	MEM	WEB				
sub $s4, $s2, $s1			IF	ID	EX	MEM	WEB			
beqz $s4, label				[IF]	ID	EX	MEM	WEB		
add $s2, $s0, $s1					IF	ID	EX	MEM	WEB	
sub $s2, $s6, $s1						IF	ID	EX	MEM	WEB

단일 메모리를 사용하도록 코드를 수정하면 MEM과 IF가 서로 충돌하게 된다. 코드를 재배열하는 것은 지연이 발생하는 위치만 변경되는 것과 같기에 따라서 근본적인 문제를 해결하기 어렵다

c. 구조적 위험은 명령어 메모리와 데이터 메모리가 같은 메모리 자원을 공유할 때 발생한다. 그러나 NOP operation 또한 명령어 메모리에서 가져와야 하는 것이기에 NOP operation으로는 구조적 위험을 해결할 수 없다. 따라서 구조적 위험을 해결하기 위해서는 적절한 하드웨어 구현이 필요하며, 이는 명령어 메모리와 데이터 메모리를 분리하는 것이다.

d. 구조적 위험은 lw, sw 에서 주로 발생하기 때문에 (25+11 =36) 36%. 발생한다고 추정할 수 있다.

3.

a.

Program execution order \ cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $s0, 0($s3)	IF	ID	EX	MEM	WEB										
lw $s1, 8($s3)		IF	ID	EX	MEM	WEB									
add $s2, $s0, $s1			IF	ID	ST	EX	MEM	WEB							
addi $s3, $s3, -16				IF	ID	ST	EX	MEM	WEB						
bnez $s2, LOOP					IF	ID	ST	EX	MEM	WEB					
lw $s0, 0($s3)						IF	ID	EX	MEM	WEB					
lw $s1, 8($s3)							IF	ID	EX	MEM	WEB				
add $s2, $s0, $s1								IF	ID	ST	EX	MEM	WEB		
addi $s3, $s3, -16									IF	ID	ST	EX	MEM	WEB	
bnez $s2, LOOP										IF	ID	ST	EX	MEM	WEB

b.

add, addi 명령어는 MEM 단계를 사용하지 않으며, bnez 명령어는 MEM 과 WEB 단계를 사용하지 않는다. 이를 표기한 다이어그램은 아래와 같다.

(회색) = 사용 X

Program execution order \ cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $s0, 0($s3)	IF	ID	EX	MEM	WEB										
lw $s1, 8($s3)		IF	ID	EX	MEM	WEB									
add $s2, $s0, $s1			IF	ID	ST	EX	MEM	WEB							
addi $s3, $s3, -16				IF	ID	ST	EX	MEM	WEB						
bnez $s2, LOOP					IF	ID	ST	EX	MEM	WEB					
lw $s0, 0($s3)						IF	ID	EX	MEM	WEB					
lw $s1, 8($s3)							IF	ID	EX	MEM	WEB				
add $s2, $s0, $s1								IF	ID	ST	EX	MEM	WEB		
addi $s3, $s3, -16									IF	ID	ST	EX	MEM	WEB	
bnez $s2, LOOP										IF	ID	ST	EX	MEM	WEB

4.

a.

Program execution order \ cycle	1	2	3	4	5	6	7	8	9	10	11	12
add $s3, $s1, $s0	IF	ID	EX	MEM	WEB							
NOP												
NOP												
lw $s2, 4($s3)			IF	ID	EX	MEM	WEB					
lw $s1, 0($s4)				IF	ID	EX	MEM	WEB				
NOP												
or $s2, $s3, $s2					IF	ID	EX	MEM	WEB			
sw $s2, 0($s3)						IF	ID	EX	MEM	WEB		

b.

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

c.

Forwarding 이 있지만 Hazard Detection Unit 이 없으면 파이프라인이 데이터 위험을 인식하지 못하고 stall 을 실행하지 않기에 잘못된 데이터가 전달될 수 있다. 따라서 Hazard Detection Unit 이 필요하다.

d.

Program execution order \ cycle	1	2	3	4	5	6	7	8	9	10
add $s3, $s1, $s0	IF	ID	EX	MEM	WEB					
lw $s2, 4($s3)		IF	ID	EX	MEM	WEB				
lw $s1, 0($s4)			IF	ID	EX	MEM	WEB			
or $s2, $s3, $s2				IF	ID	EX	MEM	WEB		
sw $s2, 0($s3)					IF	ID	EX	MEM	WEB	

cycle 4 에서 ForwardingA = 10 (add 결과 전달) ForwardingB = 00

cycle 5 에서 ForwardingA = 00, ForwardingB = 00

cycle 6 에서 ForwardingA = 10, ForwardingB = 01 (web에서 EX로 전달)

cycle 7 에서 ForwardingA = 00, ForwardingB = 10 (MEM에서 EX로 전달)

e.

forwarding이 없으면 EX/MEM 및 MEM/WB 단계에서

결과를 가져올 수 없다. 따라서 최적의 입력 및 출력 신호가 필요하다.

EX/MEM, ID/EX, IF/ID 레지스터 값이 입력신호로 필요하며

해저드의 스톨 신호, NOP 실행 신호가 출력 신호로 필요하다

f.

cycle 1 . X

cycle 2 . X

cycle 3 . stall 신호

cycle 4 . stall 신호

cycle 5 . stall 신호