4.4.1

Instruction memory 에서 instruction을 읽어오는 것은 200ps 가 걸린다. PC+4는 I-Mem과 동시에 일어남.

4.4.2

Instruction: PC relative branch (unconditional)

1) I-memory 에서 instruction fetch

2) offset value (sign extended)

3) shift left 2

4) Add (PC + 4, offset)

5) MUX

∴ 200 + 15 + 10 + 70 + 20 = 315ps.

4.4.3

conditional PC-relative branch

1) I-memory 에서 instruction fetch

2) register 불러오기

3) MUX

4) ALU

5) MUX

∴ 200 + 90 + 20 + 90 + 20 = 420ps

4.8.1

pipelined : 가장 소요시간이 긴 350ps

non-pipelined : 모든 stage의 시간을 다 더해야 함.

250 + 350 + 150 + 300 + 200 = 1250ps

4.8.2

pipelined :

latency (한 명령어의 총 실행속도)

가장 소요시간이 긴 step : 350 ps

총 5 steps

∴ 350 × 5 = 1750 ps

non-pipelined :

lw : 5 steps 모두 필요

∴ total latency = 250 + 350 + 150 + 300 + 200 = 1250ps

4.8.3

가장 소요시간이 긴 step에 의해 latency가 결정되므로 이를 split 한다. split 이후에는 2 번째로 소요시간이 긴 300ps로 clock cycle time이 결정될 것이다. 혹은 350ps를 스플릿 한 결과가 300ps보다 크다면 ($x > 300ps$) 이로 clock cycle time 결정.

4.8.4

ALU ⇒ utilization of data memory X

Beq ⇒ 〃 X

stall이 없기 때문에

lw & sw ⇒ utilizes the data memory.

∴ utilization of the data memory

= 20 + 15 = 35%

4.8.5

write register port ⇒ lw, ALU 가 수행할 수 있음.

(ALU는 레지스터 간의 계산에서 발생 가능)

∴ 45 + 20 = 65%

4.8.6

multi-cycle:

lw : 5 cycles 필요

sw : 4 cycles 필요 (WB X)

ALU : 4 cycles 필요 (MEM X)

Beq : 4 cycles 필요 (WB X)

⇒ 5×0.2 + 4(0.15 + 0.45 + 0.2) = 4.2

∴ multi-cycle 4 times slower than
pipelined

single cycle:

$$\frac{\text{Total cycle time}}{\text{cycle time of pipeline}} = \frac{1250}{350} \simeq 3.57$$

∴ single-cycle 3.57 times slower than
pipelined.

4.9.1

1번: or r1, r2, r3

2번: or r2, r1, r4

3번: or r1, r1, r2

1) RAW가 r1에 존재
1번에서 r1에 쓰고 2번, 3번에서 r1을 read

2) RAW가 r2에 존재
2번에서 r2 write, 3번에서 r2 read

3) WAR이 r2에 존재
2번에서 r2 write, 1번에서 r2 read.

4) WAR이 r1에 존재
3번에서 r1 write, 2번에서 r1 read

5) WAW가 r1에 존재.

1번에서 r1 write, 3번에서 r1 write.

4.9.2.

1번과 2번 명령어 간에 data hazard가 발생할 수 있다. (r1)

2번과 3번 명령어 간에 data hazard가 발생할 수 있다 (r2)

∴ or r1, r2, r3

Nop

Nop

or r2, r1, r4

Nop

Nop

or r1, r1, r2

4.9.3

```
IF  ID  EX ↘ MEM  WB
    IF  ID   EX ↘ MEM  WB
        IF   ID    EX   MEM  WB
```

forwarding을 이용할 수 있다면 hazards가 없고
따라서 NOP를 추가할 필요도 없음.

4.9.4

total execution time

= (num of cycles + num of stalls) x clock cycle time

no forwarding :

total execution time = $(7+4) \times 250 = 2750$ ps

full forwarding ;

total execution time = $(7+0) \times 300 = 2100$ ps

speedup ⇒ $2750/2100 = 1.3095$ times

4.9.5

(1번)
or r1, r2, r3

(2번)
or r2, r1, r4

(3번)
or r1, r1, r2

∴ NOP 불필요

1번의 ALU에서 계산된 결과를 forwarding을 통해 2번 ALU쪽으로 넘김

2번의 ALU에서 계산된 결과를 forwarding을 통해 2번 ALU쪽으로 넘김

1번 ALU에서 계산된 r1 결과를 3번 ALU쪽으로 넘김

4.9.6

No forwarding : total execution time = 2750 ps

ALU - ALU forwarding :

total execution time = 7 × 290 = 2030 ps

speedup => 2750/2030 = 1.35 times

4.13.1

```
add r5, r2, r1                add r5, r2, r1
lw  r3, 4(r5)                 nop
lw  r2, 0(r2)                 nop
or  r3, r5, r3       =>       lw  r3, 4(r5)
sw  r3, 0(r5)                 lw  r2, 0(r2)
                              nop
                              or  r3, r5, r3
                              nop
                              nop
                              sw  r3, 0(r5)
```

4.13.2

```
add r5, r2, r1                add r5, r2, r1
lw  r2, 0(r2)                 lw  r2, 0(r2)
lw  r3, 4(r5)                 nop
or  r3, r5, r3       =>       lw  r3, 4(r5)
sw  r3, 0(r5)                 nop
                              nop
                              or  r3, r5, r3
                              nop
                              nop
                              sw  r3, 0(r5)
```

⇒ performance improvement X

temporary value를 저장할 수 있는 r7 필요없다.

4.13.3

hazard detection이 존재하지 않는다면 stall이 필요해진다. instruction이 old value나 stale data를 가질 수 있기 때문이다.
∴ hazard detection은 forwarding이 가능해도 필요함.

4.13.4

Cycles	signals		
	PCwrite	ALUin1	ALUin2
IF ID EX MEM WB	1	X	X
IF ID EX MEM	1	X	X
IF ID EX	1	0	0
IF ID	1	1	0
IF	1	0	0

4.13.5

∴ EX or MEM에서 나올 value에 instruction이 depend on 한다면 ID에서 stall이 필요.

∴ for instruction in EX, load나 R-type에서 Rd를 check 해야함.

∴ for instruction in MEM, destination 레지스터가 이미 chosen / selected.

∴ register number만 체크해준다.

∴ ID/EX stage에서 rd에 의한 input 또한 필요.

. EX/MEM stage 에서 output number of output register check 필요.

∴ input we need

- ID/EX stage에서의 rd
- the output number of the output register from EX/MEM stage
- ID/EX에서의 rt 레지스터.

∴ output we need

추가적으로 필요하지 않다. 현재 존재하는 3개의 output으로 stall 할 수 있음.

A.13.6

Cycles	signals
	pcwrite
IF ID EX MEM WB	1. pcwrite =1
IF ID ··· ···	2. pcwrite = 1
IF ···	3. pcwrite = 1
···	4. pcwrite =0
	5. pcwrite =0