

20203090 한용옥

1. Explain how control signals in Slide 20 (Chapter 4) work.

PC Src

조건 분기 명령어 ('beq', 'bne')의 결과에 따라 False : PC+4로 다음 명령어 순차 실행

True : 계산된 분기 주소 ($PC + 4 + 4 \cdot offset$)로 이동

'PC Src'는 ALU의 'Zero' 신호를 보고 분기할지 제어

RegWrite

명령어가 레지스터 파일에 값을 저장할 필요가 있는 경우에만 활성화되어 정확한 타이밍에 레지스터에 값이 쓰일 수 있게 함

ALU Src

add \$s0 \$s1 \$s2 같은 레지스터 2개에 대한 연산을 수행할 것인지

add: \$s0 \$s1 | 같은 레지스터 하나와 상수에 대한 연산을 수행할 것인지

ALU operation

ALU는 덧셈, 뺄셈, 논리비트연산 등 다양한 연산을 수행 가능핚데

어떤 연산을 수행할 것인지 제어한다

Mem Write / Mem Read

메모리에 값을 쓸 것인지 (Mem Write), 메모리에서 값을 읽어올 것인지 (Mem Read) 결정한다

Mem to Reg

add \$s0 \$s1 \$s2 같이 ALU의 연산 값을 레지스터에 저장할 것인지

lw \$s0 0(\$s1) 같이 메모리에서 읽어온 값을 레지스터에 저장할 것인지 제어한다

2. What is the minimum number of cycles needed to completely execute n instructions on a CPU with a k stage pipeline? Find a formula.

최소 사이클을 찾는 것이므로 파이프라인 스트림은 고려하지 않는다

t 는 특정 시점 CPU cycle, $t+1$ 은 다음 CPU cycle 가하자

첫 명령어의 첫 stage 가 $t=1$ 에 실행된다 가정

CPU pipeline 과정에 의해 n 번째 명령어의 첫 stage는 $t=n$ 에 실행된다

이후 n 번째 명령어의 stage 단계는 $k-1$ 개 남았으므로 $n+k-1$ 시점에 n 번째 명령어가 수행완료됨

필요한 최소 사이클 수는 $n+k-1$ 개

$$n+k-1$$

3. Add NOP instructions to the code below so that it will run correctly on a pipeline that does not handle data hazards.

1. addi \$s0, \$s1, 5

2. add \$s2, \$s0, \$s1

3. addi \$s3, \$s0, 15

4. add \$s4, \$s2, \$s1

data hazard를 처리 못하므로 포워딩 같은 것을 사용하지 못한다

2에서 1의 연산 결과인 \$s0을 필요로 하므로 1의 RB가 끝난 다음 시점이 2의 ID가 되어야 함

따라서 2의 IP는 1의 RB와 동시 진행되고 1과 2 사이의 NOP는 3개가 되어야 함

3에서 \$s0을 쓰기 위해 3의 ID는 1의 RB 이후에 이루어져야 하지만

위에서 2의 IF가 1의 RB와 동시에 진행되어 3은 stall 없이 진행되어도 된다

4에서 2의 연산 결과인 \$s2를 필요로 하므로 2의 RB가 끝난 다음 시점이 4의 ID가 되어야 한다

4의 IP는 2의 RB와 동시에 진행되고 2와 4 사이의 nop는 2개 필요하다

수정된 명령어는 다음과 같다

1. addi \$s0, \$s1, 5

2. nop

3. nop

4. nop

5. add \$s2, \$s0, \$s1

6. addi \$s3, \$s0, 15

7. nop

8. nop

9. add \$s4, \$s2, \$s1

4. Explain the condition of data hazards in slide 69.

명령어 A, B, C 가 순차 실행되고 B, C는 A의 연산결과를 필요로 한다고 하자

일단 stall 없이 파이프라인 과정에 순차적으로 넣으면

A가 EX 단계 일때 B는 ID 단계이다 이때 A의 연산 결과가 저장될 레지스터 번호가 EX/MEM, Register Rd에 저장되고 B의 피연산자 레지스터가 ID/EX, Register Rs, ID/EX, Register Rt에 저장된다

B는 A의 연산 결과를 필요로 하므로

ID/EX, Register Rs, ID/EX, Register Rt 둘 중 하나는 EX/MEM, Register Rd oder

A가 MEM 단계 일때 C는 ID 단계이다 이때 A의 연산 결과가 저장될 레지스터 번호가

MEM/WB, Register Rd에 저장되고 C의 피연산자 레지스터가

ID/EX, Register Rs, ID/EX, Register Rt에 저장된다

C는 A의 연산 결과를 필요로 하므로

ID/EX, Register Rs, ID/EX, Register Rt 둘 중 하나는 MEM/WB, Register Rd oder

B, C는 A의 연산 결과를 필요로 하지만 현재 A의 RB 전에 이미 B, C의 ID가 실행되었고

data hazard와 위조건을 다시 쓰면

$ID/EX, Register Rs = EX/MEM, Register Rd$

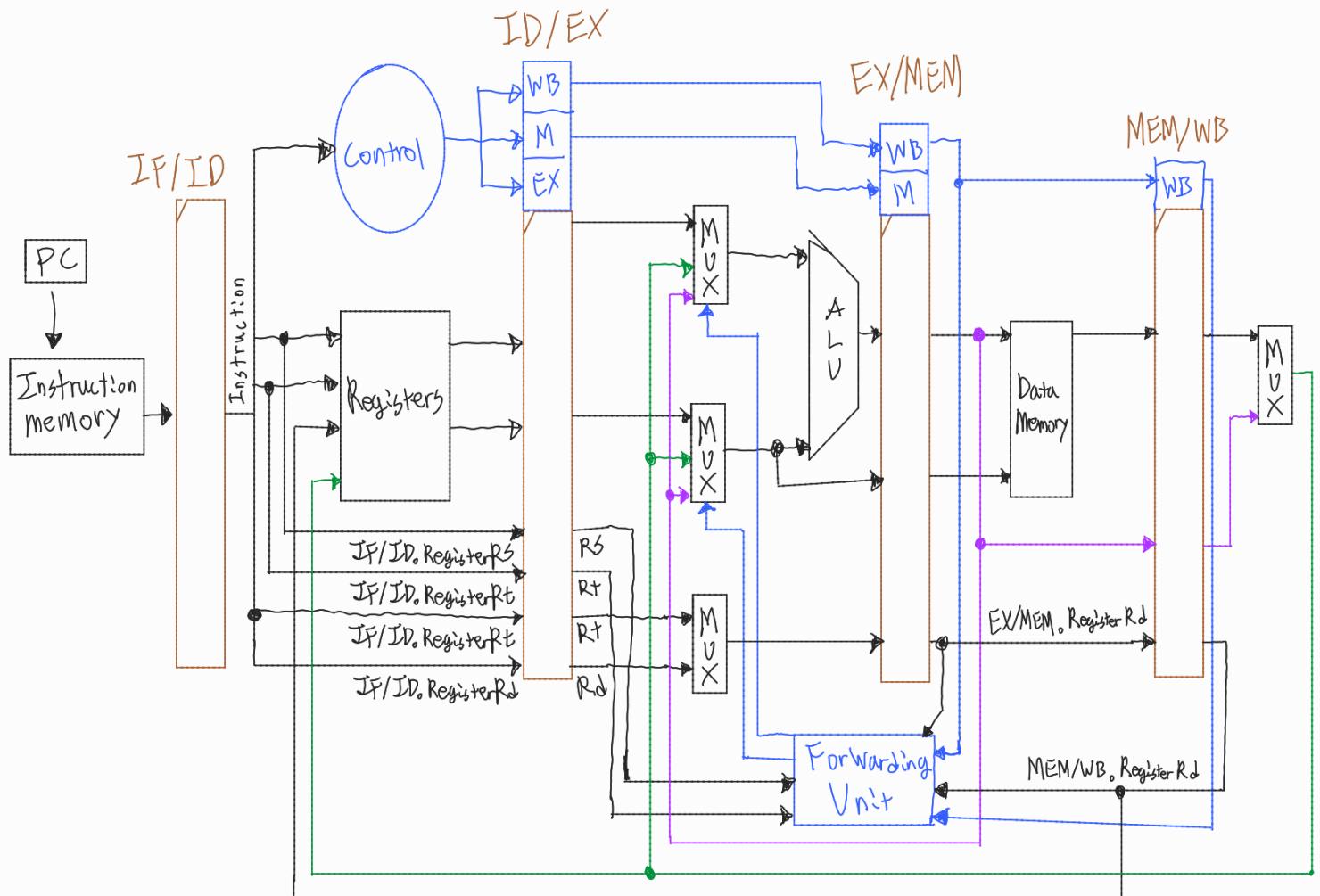
OR $ID/EX, Register Rs = MEM/WB, Register Rd$

OR $ID/EX, Register Rt = EX/MEM, Register Rd$

OR $ID/EX, Register Rt = MEM/WB, Register Rd$

위 조건이 만족되면 현재 명령어가 넣으려는 값이 아직 register file에 쓰이지 않았기 때문에
포워딩 유통은 EX/MEM 또는 MEM/WB 레지스터에서 값을 직접 전달해야 한다

5. Draw the figure in Slide 75.



6. Explain the condition for load-use hazard in Slide 77.

명령어 A, B가 순차 실행된다 하자 A는 메모리에서 값을 읽어 레지스터에 로드하는 명령

B는 A의 명령에 의해 값이 로드된 레지스터를 피연산자로 쓰다라고 하자

일단 stall 없이 파이프라인 과정에 순차적으로 넣으면

A가 ID 단계 수행할 때 A는 메모리에서 값을 레지스터로 불러오는 명령임이 결정되어

ID/EX₀ Mem Read 가 활성되고, 또한 값을 쓸 레지스터도 결정되어 ID/EX₀ Register R_t에 저장됨

A가 ID 단계 B는 IF 수행 중이고 이 때 가져온 명령어로부터 어느 레지스터를 피연산자로 쓸지

결정되어 그 들을 IF/ID₀ Register RS, IF/ID₀ Register RT에 저장한다

B는 A의 실행 결과를 필요로 하므로 IF/ID₀ Register RS, IF/ID₀ Register RT 들 중 하나는

ID/EX₀ Register R_t 이다

A는 메모리에서 값을 읽어 레지스터에 로드하는 명령

B는 A의 명령에 의해 값이 로드된 레지스터를 피연산자로 쓰는 상황의 조건을 대체쓰면

ID/EX₀ Mem Read AND (ID/EX₀ Register R_t = IF/ID₀ Register RS

OR ID/EX₀ Register R_t = IF/ID₀ Register RT)

조건이 만족된 상태에서 그대로 진행시 A의 RB 전에 B의 EX가 실행되어 data hazard다

A는 메모리 로드 연산이고 B는 이를 쓰므로 이는 load-use data hazard다

A의 MEM 단계에서 쓸 값이 결정되므로 저 조건이 만족되면

B를 한번 stall 하고 포워딩으로 값을 넘겨야 한다