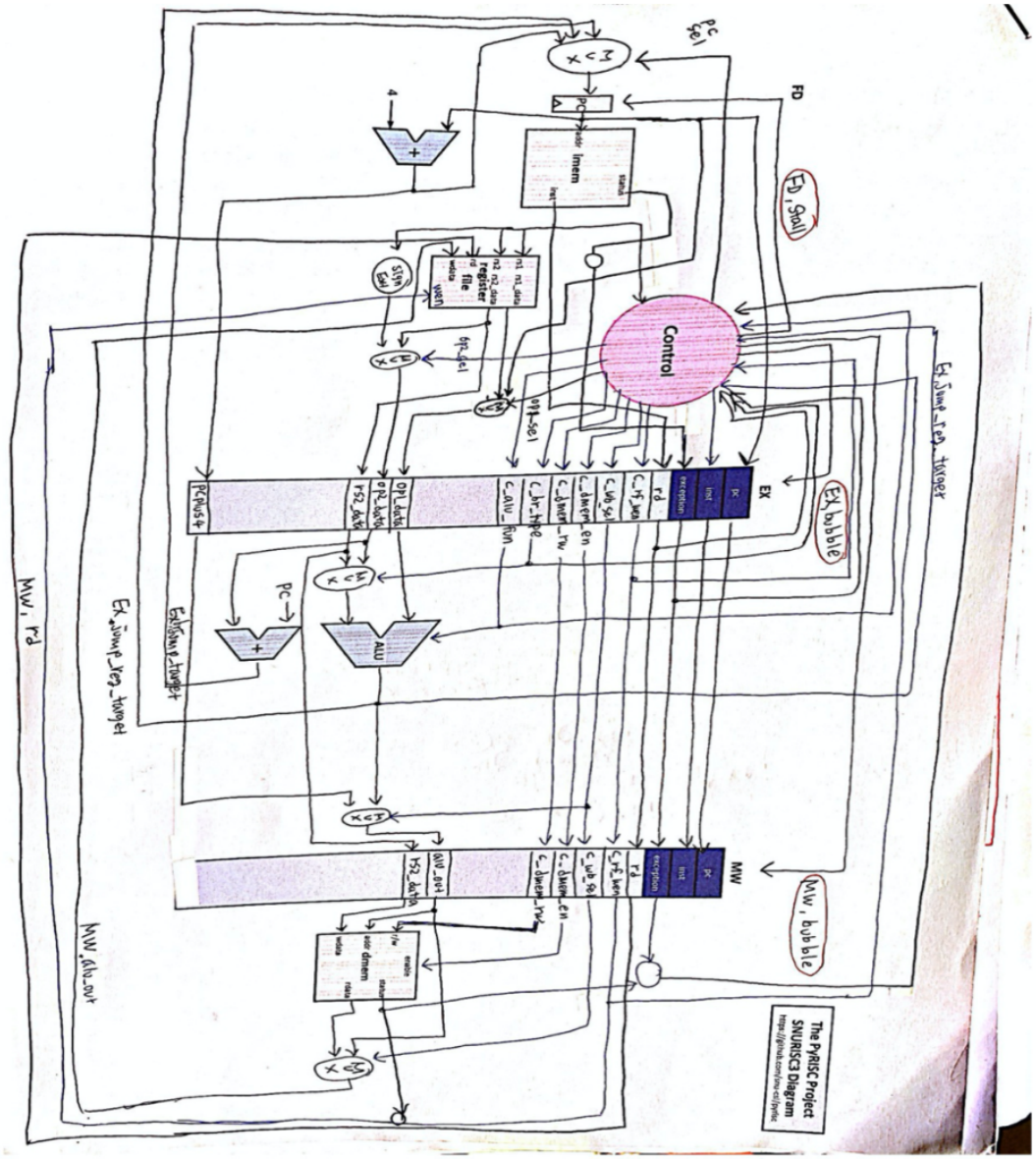
Computer Architecture Project4

독어교육과 정은주

2014-19498

1. What does the overall pipeline architecture look like?



2. When do data hazards occur and how do you deal with them?

* Show all the possible cases when data hazards can occur and your solutions to them.

Data Hazard는 FD, EX, MW 3개의 단계 중 1)FD-EX depend 그리고 2)FD-MW depend에 의해서 발생한다. 만약 FD의 rs1, rs2 register의 값이 EX/MW 단계 Instruction의 rd register와 같은 경우 FD 단계에서는 stall이 그리고 EX/MW 단계에서는 Bubble이 필요하다. 이때, data hazard의 조건이 몇 가지 있는데, 먼저, rd register 값이 0이 아니어야 한다.(write를 해도 0이다) 더불어 write back 단계에서 register에 값을 쓰는 instruction이어야 한다. 가능한 경우의 수는 다음과 같다.

<FD-EX or FD-MW depend>

FD : R-type Instruction(add, and, or, sll, slt, sra, srl, sub, xor, etc), SB-type Instruction(beq, blt, bne, bge), S- type Instruction(store) & load Instruction(rs1의 경우만 고려), I-type Instruction(immediate 값의 존재로, rs1의 경우만 고려) Pipeline이 5단계 일 때의 loaduse hazard는 3단계로 줄면서, 앞서 언급한 경우들에 포함된다.

EX_Instruction/MW_Instruction : rd != 0 and register에 write를 하는 Instruction

1) EX or MW.RegisterRd = FD.RegisterRs1
   EX or MW.RegisterRd = FD.RegisterRs2
2) EX or MW.RegiWrite
3) EX or MW.RegisterRd != 0

Sample Example

1) FD_rs2 == EX_rd (FD: S-type - store)

FD: sw t6, 0(t0)
EX: addi t6, zero, 3
MW: lui t0, 0x80010000

FD: sw t6, 0(t0)
EX: BUBBLE
MW: addi t6, zero, 3

FD: sw t6, 0(t0)
EX: BUBBLE
MW: BUBBLE

2) FD_rs1 == MW_rd (FD: SB-type - bne)

FD: bne a4, a3, 0x80000248
EX: addi a2, zero, 33
MW: addi a4, a4, 1

FD: bne a4, a3, 0x80000248
EX: BUBBLE
MW: addi a2, zero, 33

3) FD_rs2 == EX_rd (FD: R-type - sub)

FD: sub t1, t3, t2
EX: sub t2, t3, t1
MW: sub t5, t5, t1

FD: sub t1, t3, t2
EX: BUBBLE
EX: sub t2, t3, t1

FD: sub t1, t3, t2
EX: BUBBLE
MW: BUBBLE

*Load와 I-Type의 경우에는 rs2의 값이 아닌 rs1과의 dependency만 고려한다.

4) FD_rs1 == EX_rd (FD: load)

FD: ld t0, 0(t6)
EX: addi t6, zero, 3
MW: lui t0, 0x80010000

FD: ld t0, 0(t6)
EX: BUBBLE
MW: addi t6, zero, 3

FD: ld t0, 0(t6)
EX: BUBBLE
MW: BUBBLE

5) FD_rs1 == MW_rd (FD: I-type, addi)

FD: addi a3, a4, 1
EX: addi a2, zero, 33
MW: addi a4, a4, 1

FD: addi a3, a4, 1
EX: BUBBLE
MW: addi a2, zero, 33

이를 해결하기 위한 코드는 아래와 같다. 먼저, 위에서 언급한 제약 조건들을 만족하면서, EX단계 혹은 MW의 register rd register value와 FD 단계의 rs1 혹은 rs2의 register와 같은지 확인한다.

```
# FD_ dependency with EX_ : when FD stage's inst is either branch or R-type or store/load
FD_EX_depend = True if((EX.reg_rd == Pipe.FD.rs1 and rs1_oen) \
                    or (EX.reg_rd == Pipe.FD.rs2 and rs2_oen)) \
                    and EX.reg_rd != 0 and Pipe.EX.c_rf_wen else False

# FD_ dependency with MW_ : when FD stage's inst is either branch or R-type or store/load
FD_MW_depend = True if((MW.reg_rd == Pipe.FD.rs1 and rs1_oen) \
                    or (MW.reg_rd == Pipe.FD.rs2 and rs2_oen)) \
                    and MW.reg_rd != 0 and Pipe.MW.c_rf_wen else False
```

그리고, FD 단계에서는 stall, EX 단계에서는 bubble을 만든다. 그러면, dependency가 해소될 때까지, 즉, Execute-Write Back 단계에서 register에 값이 쓰일 때까지 기다리고, 그 후에 Instruction을 fetch & decode 하게 된다.

```
# FD_stage should be stalled until the MW_stage write its WB_data in Register
self.FD_stall = (FD_MW_depend or FD_EX_depend) and not EX_brjmp
self.EX_bubble = EX_brjmp or FD_EX_depend or FD_MW_depend
```

3. When do control hazards occur and how do you deal with them?

* Again, show all the possible cases when control hazards can occur and your solutions to them.

Control Hazard는 Branch Hazard로도 불리며, branch instruction(SB-type)과 관련해서 발생한다. Pipe3에서는 "always Branch Not taken" 방법을 택해서 stall을 최소화한다. 만약, Ex 단계에서 branch prediction이 틀린 경우에 즉, branch가 taken 되는 경우에 이전에 fetch한 값 대신 새로운 주소 값을 forwarding해서 FD 단계에서 다시 Fetch and Decode 하는 과정이 필요하다. 만약 EX 단계를 Bubble로 만들지 않는다면, not taken일 때 예측 값으로 실행되어서는 안 되는 Instruction이 잘못해서 EX 단계로 갈 수 있기 때문에 이 경우에 Control Hazard가 발생한다. Jal와 Jalr도 이와 비슷하게 동작한다. 즉, 총 Stall과 Bubble을 만드는 조건은 아래코드와 같다.

```
# Control signal to select the next PC
self.pc_sel = PC_BRJMP if (EX.reg_c_br_type == BR_NE  and (not Pipe.EX.alu_out)) or \
                          (EX.reg_c_br_type == BR_EQ  and Pipe.EX.alu_out) or \
                          (EX.reg_c_br_type == BR_GE  and (not Pipe.EX.alu_out)) or \
                          (EX.reg_c_br_type == BR_GEU and (not Pipe.EX.alu_out)) or \
                          (EX.reg_c_br_type == BR_LT  and Pipe.EX.alu_out) or \
                          (EX.reg_c_br_type == BR_LTU and Pipe.EX.alu_out) or \
                          (EX.reg_c_br_type == BR_J) else \
              PC_JALR if EX.reg_c_br_type == BR_JR else \
              PC_4

EX_brjmp = (self.pc_sel != PC_4)
```

연산이 일어나는 Execution 단계의 Instruction이 SB type(bne, blt, bge etc) 혹은 Jal, Jalr 등인 경우 뛰어야 하는 Program Counter가 변경될 수 있다. Execution 단계에 있는 Branch Instruction의 경우는 Branch not taken 방법을 사용하기 때문에, Execution 단계에 Branch Instruction이 있을지라도, FD 단계에서 stall을 하지 않는다. 다만 Execution 단계에서 뛸 PC 값이 정해지면, 그 때, FD 단계에서 Stall이 발생하고, Execution 단계에서 Bubble이 생긴다.

```
# FD_stage should be stalled until the MW_stage write its WB_data in Register
self.FD_stall = (FD_MW_depend or FD_EX_depend) and not EX_brjmp
self.EX_bubble = EX_brjmp or FD_EX_depend or FD_MW_depend
```

FD_Stall은 EX 단계의 Instruction이 Jal, Jalr 혹은 Branch type이 아닌 경우이면서, Data Hazard가 생길 때 가능하며, EX_Bubble의 경우는 Forward 된 값이 다시 Decode, Fetch 단계를 거칠 수 있도록 Ex 단계가 Bubble이 된다.

Sample Example

1) EX => Jal	2) EX => Jalr	3) EX => SB-type
FD: add a0, s1, a0	FD: addi a2, zero, 0	FD: addi sp, sp, -16
EX: jal ra, 0x80000010	EX: jalr zero, ra, 0	EX: bge a5, a0, 0x80000058
MW: addi a0, s0, -2	MW: addi a0, a2, 0	MW: add a1, a1, 1
FD: addi a5, zero, 1	FD: lw a1, 0(t3)	FD: addi a0, zero, 1
EX: BUBBLE	EX: BUBBLE	EX: BUBBLE # if branch taken
MW: jal ra, 0x80000010	MW: jalr zero, ra, 0	MW: bge a5, a0, 0x80000058