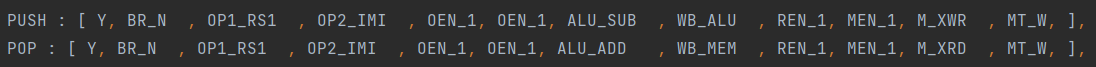
Computer Architecture Lab 4 Report

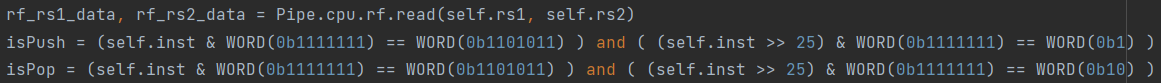
공과대학 컴퓨터공학부 2021-18641 이하동

Part 1. New instruction – Push & Pop

csignals



ID.compute



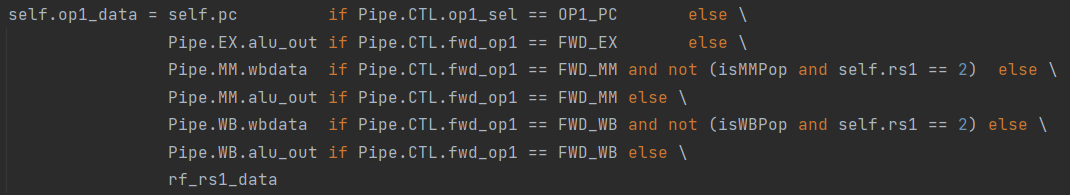
텍스트이(가) 표시된 사진

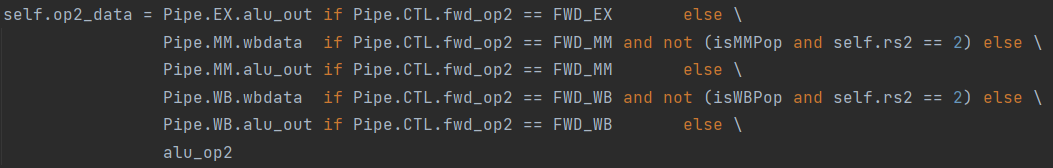
자동 생성된 설명

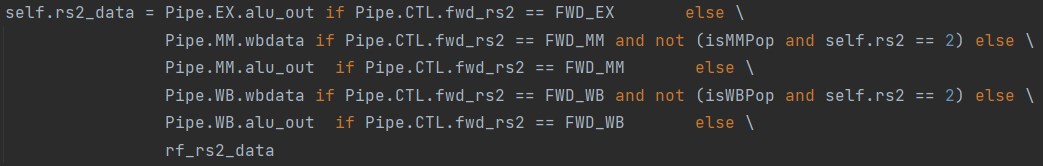
1. new Data Hazard

Push의 경우 rs1와 rd를 sp로 맞춰주었기 때문에, 기존 파이프라인 로직을 재활용할 수 있다.

따라서 Data hazard가 새롭게 생기는 경우는 rs1 혹은 rs2가 sp일 때, Pop의 rd가 sp가 아님에도 sp 값에 변화가 생기는 것이 문제가 된다. 이 경우를 예외처리하여 Forwarding을 수행해주어야 한다. 또한 기존 로직을 따르면 Pop에서 MM.wbdata, WB.wbdata는 메모리 상의 sp주소가 갖는 값이 된다. 따라서 sp+4를 포워딩하기 위해, 새로운 파이프라인 레지스터를 만들어 포워딩해주어야 한다.







2. Changes in datapath

1) alu\_out의 값을 저장하기 위한 MM.alu\_out과 WB.alu\_out이라는 레지스터 변수를 새로이 선언, 업데이트 하는 로직을 만들었음(이는 기존 update로직에 한 줄 추가하면 끝남, 생략하였음)

2) op1\_data와 op2\_data, rs2\_data를 정하는 mux에 MM.alu\_out과 WB.alu\_out을 추가로 연결함.

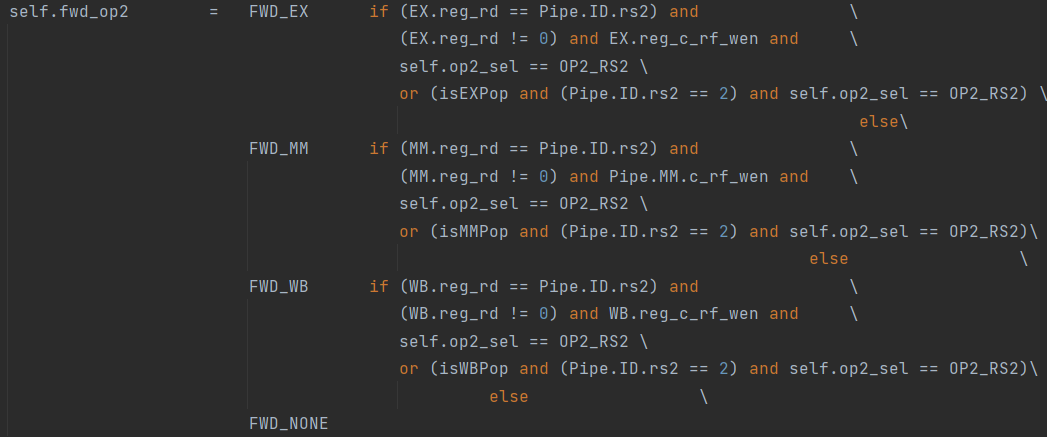
3) Push, Pop 명령어의 경우 R타입이나 사용하지 않는 값이 있었음. Push는 rs1, rd = 2, Pop은 rs1 = 2로 맞추기 위해 Register file 앞에 mux를 설치하였음.

3. Changes in Control Signals

CTL.gen()

텍스트이(가) 표시된 사진

자동 생성된 설명



텍스트이(가) 표시된 사진

자동 생성된 설명

CTL에서는 어떤 단계에서 forwarding을 할지만 정하고, Pop과 관련하여 alu\_out / wb\_data 중 하나를 정하는 것은 ID단계에서 처리하였다.

두 단계를 종합하면 아래와 같은 Control signal이 나온다. (코드 상의 실제 상수 값과는 다를 수 있다.)

|  |  |  |
| --- | --- | --- |
| Mux control | Source | Explanation |
| ForwardA = 000 | Self.pc /rf\_rs1\_data | The first ALU operand comes from Register file or PC  (실제 구현에선 구분해야하지만, 여기선 생략함.) |
| ForwardA = 001 | EX.alu\_out | The first ALU operand comes from ALU result of EX |
| ForwardA = 010 | MM.wbdata | The first ALU operand comes from WB data of MM |
| ForwardA = 011 | MM.alu\_out | The first ALU operand comes from ALU result of MM |
| ForwardA = 100 | WB.wbdata | The first ALU operand comes from WB data of WB |
| ForwardA = 101 | WB.alu\_out | The first ALU operand comes from ALU result of WB |
| ForwardB = 000 | Alu\_op2 | The second ALU operand comes from original op2 value |
| ForwardB = 001 | EX.alu\_out | The second ALU operand comes from ALU result of EX |
| ForwardB = 010 | MM.wbdata | The second ALU operand comes from WB data of MM |
| ForwardB = 011 | MM.alu\_out | The second ALU operand comes from ALU result of MM |
| ForwardB = 100 | WB.wbdata | The second ALU operand comes from WB data of WB |
| ForwardB = 101 | WB.alu\_out | The second ALU operand comes from ALU result of WB |
| ForwardC = 000 | rf\_rs2\_data | The second rs2 value comes from ID stage rs2 value |
| ForwardC = 001 | EX.alu\_out | The second rs2 value comes from ALU result of EX |
| ForwardC = 010 | MM.wbdata | The second rs2 value comes from WB data of MM |
| ForwardC = 011 | MM.alu\_out | The second rs2 value comes from ALU result of MM |
| ForwardC = 100 | WB.wbdata | The second rs2 value comes from WB data of WB |
| ForwardC = 101 | WB.alu\_out | The second rs2 value comes from ALU result of WB |

ForwardA = 001 if ((EX.reg\_rd == Pipe.ID.rs1) and rs1\_oen and   
 (EX.reg\_rd != 0) and EX.reg\_c\_rf\_wen)   
 or (isEXPop and (Pipe.ID.rs1 == 2) and rs1\_oen)  
 else

010 if (MM.reg\_rd == Pipe.ID.rs1) and rs1\_oen and   
 (MM.reg\_rd != 0) and Pipe.MM.c\_rf\_wen   
 else

011 if (isMMPop and (Pipe.ID.rs1 == 2) and rs1\_oen)  
 else

100 if (WB.reg\_rd == Pipe.ID.rs1) and rs1\_oen and   
 (WB.reg\_rd != 0) and WB.reg\_c\_rf\_wen

else

101 if (isWBPop and (Pipe.ID.rs1 == 2) and rs1\_oen)   
 else  
000

ForwardB = 001 if ((EX.reg\_rd == Pipe.ID.rs2) and rs1\_oen and   
 (EX.reg\_rd != 0) and EX.reg\_c\_rf\_wen and self.op2\_sel == OP2\_RS2)   
 or (isEXPop and (Pipe.ID.rs2 == 2) and self.op2\_sel == OP2\_RS2)  
 else

010 if (MM.reg\_rd == Pipe.ID.rs2) and self.op2\_sel == OP2\_RS2 and   
 (MM.reg\_rd != 0) and Pipe.MM.c\_rf\_wen   
 else

011 if (isMMPop and (Pipe.ID.rs2 == 2) and self.op2\_sel == OP2\_RS2)  
 else

100 if (WB.reg\_rd == Pipe.ID.rs2) and self.op2\_sel == OP2\_RS2 and   
 (WB.reg\_rd != 0) and WB.reg\_c\_rf\_wen

else

101 if (isWBPop and (Pipe.ID.rs2 == 2) and self.op2\_sel == OP2\_RS2)   
 else  
000

ForwardC = 001 if ((EX.reg\_rd == Pipe.ID.rs2) and rs2\_oen and   
 (EX.reg\_rd != 0) and EX.reg\_c\_rf\_wen)   
 or (isEXPop and (Pipe.ID.rs2 == 2) and rs2\_oen)  
 else

010 if (MM.reg\_rd == Pipe.ID.rs2) and rs2\_oen and   
 (MM.reg\_rd != 0) and Pipe.MM.c\_rf\_wen   
 else

011 if (isMMPop and (Pipe.ID.rs2 == 2) and rs2\_oen)  
 else

100 if (WB.reg\_rd == Pipe.ID.rs2) and rs2\_oen and   
 (WB.reg\_rd != 0) and WB.reg\_c\_rf\_wen

else

101 if (isWBPop and (Pipe.ID.rs2 == 2) and rs2\_oen)   
 else  
000

Part 2.Branch Prediction using BTB

1. BTB implementation

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

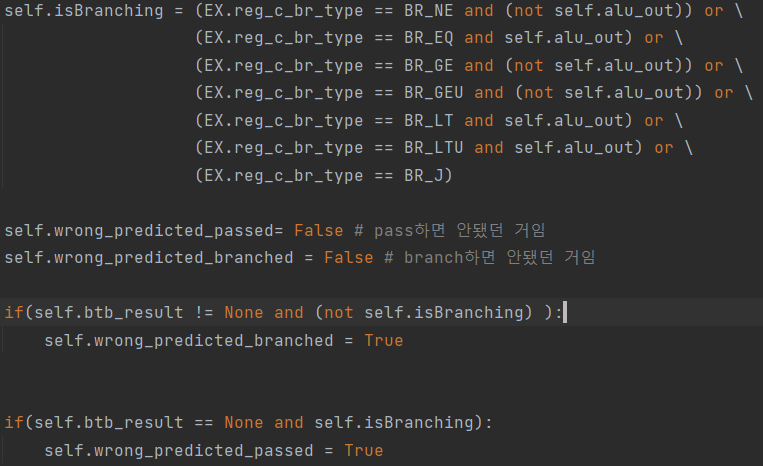
텍스트이(가) 표시된 사진

자동 생성된 설명

Valid bit, tag, target\_address를 객체 변수로 갖는 클래스를 정의하였다. 해당 클래스의 객체를 담는 리스트를 BTB로 이용하였다. 리스트의 index를 index 값으로 이용하였고, 비트 연산을 통해 index, tag, address를 pc 값에서 추출하였다. Lookup의 결과는 실패 시에 None을 반환하도록 구현하였다.

2. Detecting misprediction

EX.compute()



EX.compute()단에서 misprediction을 감지하였다. self.isBranching은 inst 종류가 branch이면서 브랜칭해야하는지를 나타내는 부울 변수이다. self.btb\_result는 IF단에서 얻은 예측 결과이다. 이를 이용하여 예측 결과와 분기해야하는지 여부가 다르면 wrong\_predicted를 True로 만든다.

i) 만약 btb\_result는 not branching & 계산결과는 branching -> wrong\_predicted\_branched = True

ii) 만약 btb\_result는 branching & 계산결과는 not branching -> wrong\_predicted\_passed= True

i은 잘못 분기한 경우를 detect, ii는 잘못 분기하지 않은 경우를 detect한다.

3. Handle misprediction

다음 pc값을 결정하는 우선순위는 다음과 같다.

1순위: misprediction의 수정 / jalr 명령어의 수행

2순위: btb\_result

3순위: PC + 4

만약 wrong\_passed면 branch target으로 jump해야되고, wrong\_branched면 EX단의 레지스터에 저장된 pc값에 4를 더한 값으로 pc값을 업데이트해야한다.

CTL.gen()

텍스트이(가) 표시된 사진

자동 생성된 설명

우선순위에 맞게 CTL.gen()을 수정

IF.compute에서 CTL.gen()을 호출한 뒤, 아래와 같이 pc\_next를 업데이트함.

텍스트이(가) 표시된 사진

자동 생성된 설명

위에 세 조건이 1순위, 4번째 조건이 2순위, 마지막 pcplus4가 3순위를 갖도록 코딩하였다.

4. Changes in the datapath

1) inst의 btb 예측 결과를 저장할 btb\_result 레지스터 변수를 선언하였다. EX 레지스터까지 연결하였음. (그 이후론 쓰이지 않는다.)

2) pc값에 대한 MUX에 EX.pcplus4, EX.brjmp\_target, EX.jump\_reg\_target, btb\_result, pcplus4를 연결하도록 바꿈 (Forwarding)

3) EX의 예측 결과를 CTL에 연결함.

5. Changes in the control signals

control signal의 변화는 4번의 코드에서 나타나 있는 pc\_sel에 대한 내용이다.

|  |  |
| --- | --- |
| Mux control | Source |
| pc\_sel = 000 | pcplus4 |
| pc\_sel = 001 | btb\_result |
| pc\_sel = 010 | EX.pcplus4 |
| pc\_sel = 011 | EX.brjmp\_target |
| pc\_sel = 100 | EX.jump\_reg\_target |

pc\_sel = 011 if Pipe.EX.wrong\_predicted\_passed else   
 010 if Pipe.EX.wrong\_predicted\_branched else   
 100 if EX.reg\_c\_br\_type == BR\_JR else   
 001 if self.btb\_result != None else   
 000