* 1. MAX(...) = 25ns
  2. 25% load, 10%가 depend이다 -> 버블을 추가해줘야한다.

10% branch. 90%은 TAKEN -> 90%가 점프한다

나머지는 버블 추가 안해도 됨.

IF는 낫테이큰. 점프 안한다고 예상. 근데 점프하면 다시 돌아와서 계산해야 하니까 딜레이 생김.

이상적으로는 CPI = 1.

여기에 load, branch 패널티 더해주면 됨.

0.25 \* 0.1 \* 2(포워딩 없을 땐 load는 스톨 2개)

0.1 \* 0.9 \* 1 ( ID에서 resolve -> 1개 stall 해줘야함.)

Load

포워딩 없으면

IF id ex mem wb

If id ...기달 값 받음 exe mem wb

있으면 wb 때 exe로 바로 이동

2.1 Assume that the two source operands of an ALU instruction can be read with one access to register file: -> 한번에 2개 빼낼 수 있다. (sub $3, $1, $2같을 때)

Instruction Memory도 포함해야함.

그러므로 instruction 6개. lw할때랑 sw할때, 2개 2개 해서 총 10개.

Register File은 8번. Sub, add는 2번씩이니까. ALU는 sub, add 때만 이니까 2번.

2.2 lw 2번, sw 2번으로 총 40사이클,

Add, sub 각각 한번으로 총 2사이클

10 \* 4 + 1 \* 2= 42.

3.1 바이어스는 2^-32 ~ 2^31을 나타내야 할 때 음수 지수 표현을 양수로 하기 위함이니, 바이어스는 31가 되어야 한다.

3.2.a 4.25는 100.01 = 1.0001 \* 2^2 이므로 0 | 100001 | 000100000

3.2.b 4096은 100000000000 = 1.0 \* 2^12 이므로 0 | 101011 | 00000000

3.3 1.000000000 ( 43 – 31)

0.00000...10001 (43 – 31)

1.0000... 10001 ( 43 – 31)

저기 0이 9개. Significont 비트가 9개인데 guard, round, stiky 순서로 각각 101이 됨.

Guard가 1이니까 올림 1함. Stiky도 1이긴 한데 이건 문서 참고하고.

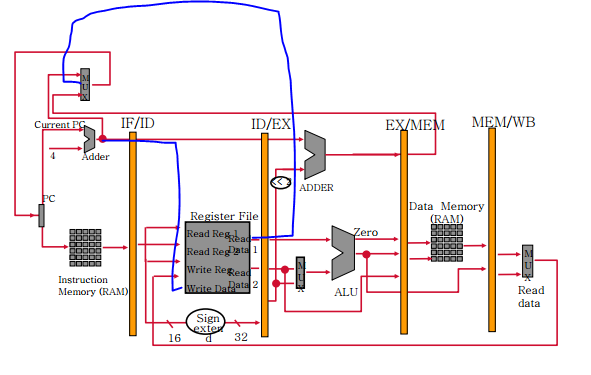
여하튼 1.0000000.... 001 \* 2^(43-31) 이 됨

가드 없으면 1.00000 ..000 이되겠지

4.1 ajl $1 은 $1값으로 점프 하면서 Reg$31에 PC+4를 넣겠다는 거임.

PC쪽 MUX에 들어가서 그걸 PC에 쓰게 할거고.

넣는건 PC+4값을 PC 위 에더에서 가져와서 그걸 register File에 write data쪽으로 넣어주면 됨. 아래 그림 참조.



4.2 이걸 수행하려면 버블이 얼마나 필요하겠는가?

위 그림처럼 ID/EX 뒤에서 뽑아낼거면 2번, 아니면 ID/EX 전에서 뽑아낼거면 1번.

5.1 루프 돌때 몇사이클 걸리는지.

Forwarding 하고 있고 nop도 있으니까 bubble 생길 일이 없다는걸 설명하고.

그러면 총 9 + (5-1) = 13번 걸린다고 설명하면 됨.

5.2 위에 3 명령어는 아래 명령어들이랑 dependency가 없으니까 그대로 옮겨주면 됨.

Beq 뒤에 넣어도 되는데, 그래도 잘 돌아감. 수업시간에 나왔다고 함. 영상에 있다고 하니까 꼭 볼것.

Delay slot이 뭔지도 검색해볼것. 그건 beq랑 관련있음.

6.1 버블 생겨야하는거 만들어주고 (add 랑 add 사이 1개, lw 다음 2개) 그다음에 clock 새서 instruction 갯수로 나눠주면 CPI 나옴.

기억할것. lw에서 다음 명령어의 ID는 WB랑 동시에 하는거임.

6.2 add랑 add 사이는 버블 안생기고, lw는 다음에 버블 1개만 생기니까 그거 고려해서 clock 새서 instruction 갯수로 나눠주면 끝.