EE312 Computer Architecture

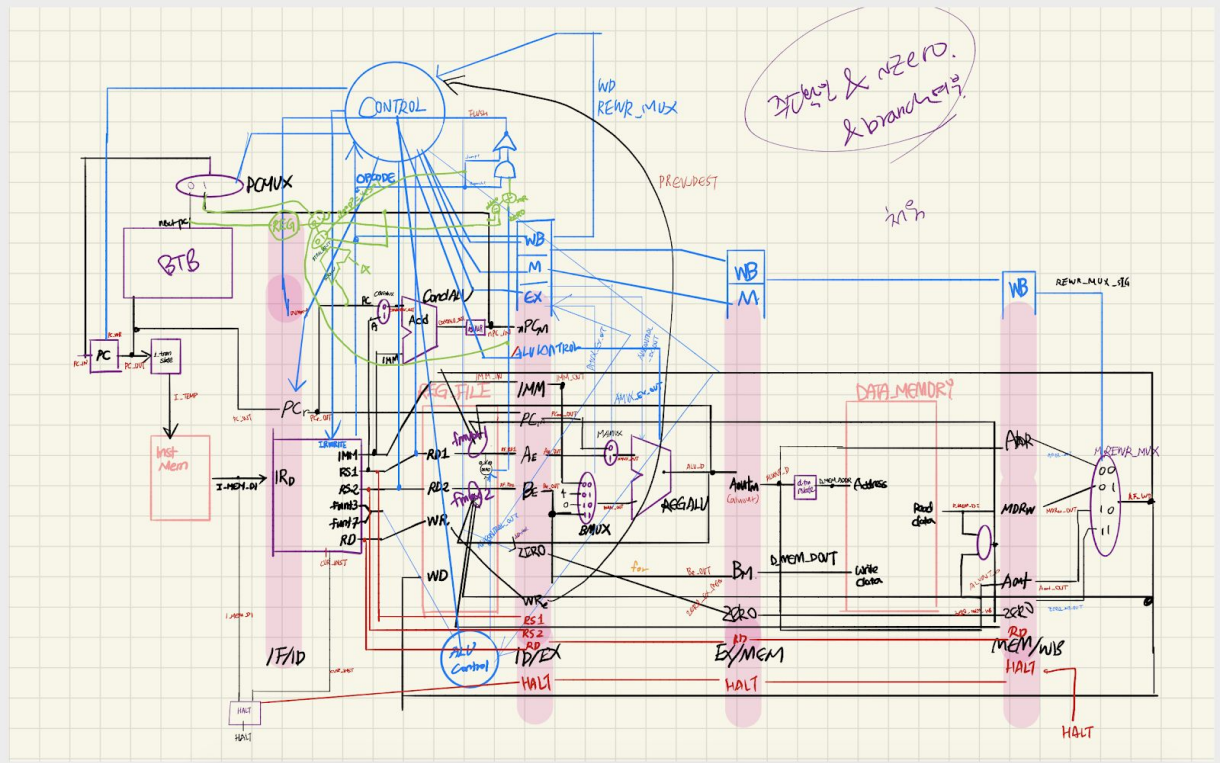
LAB6 Report

20160542 이현지, 20160707 형준하

1. Introduction
    - 이번 랩에서는 5stage pipeline cpu를 구현하는 과제를 맡게 되었다. 이는 현대적인 cpu의 구조를 구현해본다는 점에서 과거 랩들과 차이가 있고, 조금 더 난이도가 있었던 실험이었다. Pipeline을 제대로 구현하기 위해서는 branch나 load등에서 stall이 필요한데, 이는 cpu의 성능을 저하시키기 때문에 branch prediction이나 forwarding등의 기술들이 필요하고, 이는 과제의 난이도를 더욱 높였다.
    - 우리는 5 stage pipeline cpu를 구현하였고, 거기에 forwarding을 구현하였다. 2bit saturation counter with BTB는 구현하다가 멈추었다.

2. Design
    - 5stage pipelined CPU, with forwarding and branch always not taken prediction.
    - 5stage pipelined CPU를 구현하였습니다. ID stage에서 instruction을 디코드하고, 이를 파이프라인 레지스터에 흘려 보냄으로서 control signal 이 필요한 스테이지에 가서 동작하도록 하였습니다. 포워딩은 ID스테이지에서 이루어지고, 거기서 한개, 두개, 세개의 인스트럭션 앞에서 값을 가져오게 구현하였습니다.
    - LW가 EX에 있을때 조건이 맞다면 스톨을 하게 하였고, 브랜치는 always not taken으로 예측하고 만약 taken이라면 1개의 인스트럭션을 flush하도록 하였습니다.

-



3. Implementation

- Controller의 경우 싱글사이클 cpu 와 비슷한 형태를 가졌다. 하지만 여기서 다른 점은, controller에서 나온 control signal 들이 바로 효과를 내는 것이 아니라, pipeline register를 타고 가다 필요한 stage에 가서 효과를 발휘한다는 점이다. 각각의 pipeline register들은 clock 의 posedge 에서 latch 하게 했다. 이를 통해 control signal들과 데이터들을 다음 스테이지로 넘겨주었다.
- Flush: 처음 구현시 branch always not taken으로 했을 때는 Jump or branch taken이라면 flush를 하게 하였고, BTB를 사용한 후에는 ((predicted addr == target addr) ^ branch taken) or Jump로 flush조건을 설정해주었다. flush를 하면 posedge에서 해당사이클에 더미 인스트럭션(컨트롤 시그널)이 채워지도록 했다.
- stall: 스톨은 ex stage에 LW 가 있고, ID 스테이지에 LW에 dependent한 instruction이 있을때 실행된다. 스톨도 마찬가지로 더미 인스트럭션으로 하나를 채우고, 그 뒤에 따라오는 인스트럭션들은 pcwrite 과 irwrite 을 0으로 설정하므로서 멈춰있도록 하였다.

-

- Forwarding: 포워딩은 ID스테이지에서 값을 받도록 했으며, EX, MEM, WB에서 forwarding 된 값을 가져온다. Register 주소를 비교하고, 해당 register를 사용하는지를 확인하며 dependency를 체크하고 만약 있다면 forwarding 이 일어나도록 한다.
- Zero: ID stage에서 branch가 taken인지 아닌지를 결정하는 역할을 한다.
- BTB: Two bit saturated BTB를 구현하려다가 실패하고, 원래대로 always untaken으로 prediction을 진행하도록 하였다.
- NumInst 값을 변화시키는 always문이 TODO 바깥에 정의되어 있었지만, classum에서 Num Inst를 필요에 따라 변경하여도 된다하여 저희도 코드 상 위치와 값 변화시키는 방법을 바꿨습니다. 바꾼 방법은, stall 이나 flush가 일어난 인스트럭션이 아니라면 wb에서 1씩 더해가도록 하였습니다.

4. Evaluation
    - Test 를 모두 통과했습니다.
    - cycle 수는 각각 inst: 25cycles, forloop: 97cycles, 13991 cycles가 걸렸습니다.

5. Discussion
    - 디버깅이 너무 힘들었습니다. 인생을 배울 수 있는 과제였습니다. 특히 sort의 경우 인스트럭션이 너무 길고 많아서 디버깅하는게 매우 어려웠습니다. 테스트케이스가 좀더 많고 촘촘하게 있으면 좋겠다는 생각을 했습니다.

6. Conclusion
    - 이번 과제를 하며 파이프라인을 정말 제대로 알게되었고 고생을 많이 해서 이 개념을 까먹지 않을 것 같습니다. 파이프라인에 대해 수업시간에 배웠지만 직접 구현해보고 자잘한 디테일들을 배워가며 더 심도있고 정확하게 파이프라인에 대해 알게 된 것 같습니다.