EE312 Lab Report – Lab 5. Pipelined CPU

20160030 고은석, 20160680 추헌호

1. Introduction

본 과제에서는 Verilog HDL을 통해 RV32I 기반의 5-stage Pipelined CPU를 구현한다. Pipelined CPU는 Multi Cycle CPU와 같이 하나의 instruction에 대해 한 클럭당 IF, ID, EX, MEM, WB 5개의 stage 중 하나를 처리한다. 하지만 차이는 Multi Cycle CPU가 하나의 instruction이 끝나야 다음 instruction을 실행하는 반면 Pipelined CPU는 하나의 stage가 끝나면 바로 다음 instruction을 실행한다는 것이다. 이러한 특성은 CPI를 획기적으로 줄여준다는 장점이 있지만, 여러가지 hazard가 발생하기 때문에 이를 forwarding이나 branch prediction 등을 구현해서 해결해야 한다.

2. Design

Multi Cycle CPU의 기본적인 구조는 Figure 1과 같다. 기본적으로 Multi Cycle CPU에서 발전된 형태로, 많은 레지스터들이 추가되어 특정 스테이지에서의 데이터를 다음 스테이지로 전달한다. FWRD 모듈은 조건이 충족되면 포워딩을 위한 신호를 발생해 하늘색 MUX들을 통해 포워딩을 실행하여 Data hazard를 해결한다. 또한 always-not-taken branch prediction이 구현되어 있어, branch가 taken되면 flush 신호가 발생해 flush를 실행하여 Control hazard를 해결한다. 자세한 로직은 3. Implementation에서 설명한다.

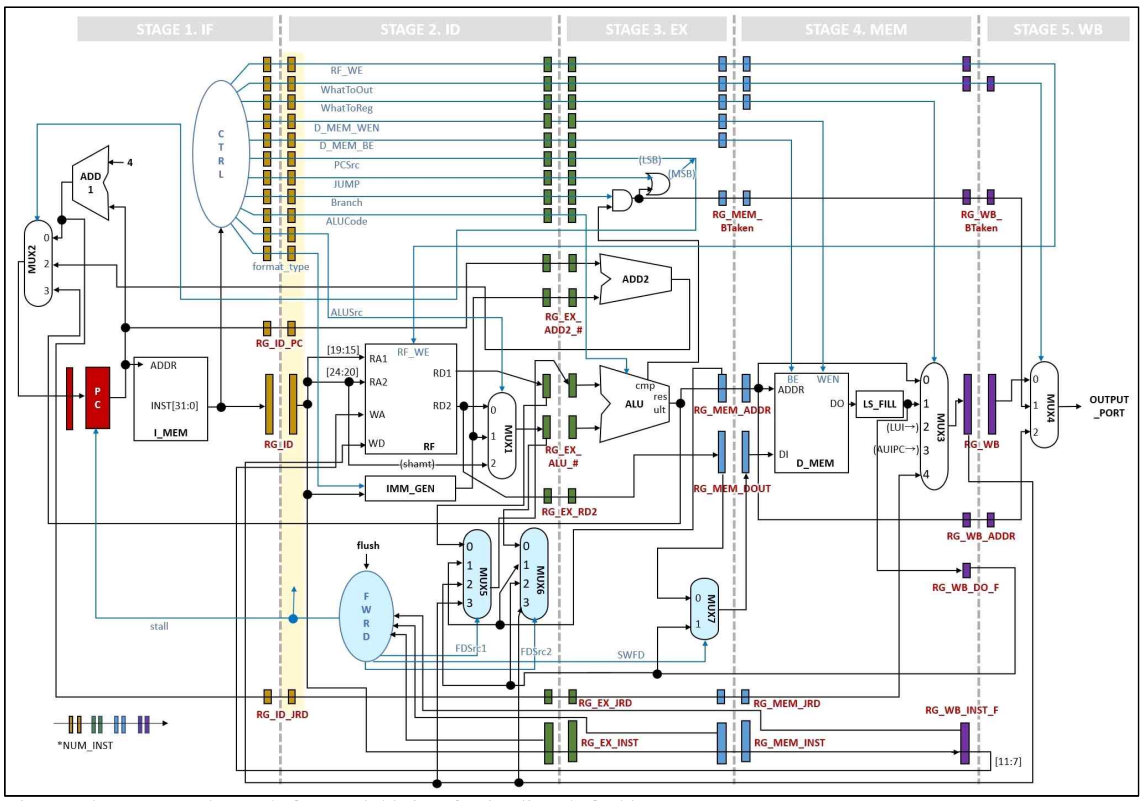


Figure 1. Datapath and Control Unit of Pipelined CPU

3. Implementation

1) CTRL, ALU module

기본적으로 Lab 3, 4에서 사용한 것과 유사하다. 특정 신호가 어떤 조건 하에서 발생하는지는 Lab 3, 4의 Lab report에 기술되어 있으므로 생략한다.

2) Pipeline Registers

Figure 1과 같이, Lab 4와 비교했을 때 훨씬 많은 레지스터들이 추가되었다. 5가지 색으로 구분해 놨지만, 이는 구현의 편의를 위한 것이고 사실 같은 색이 같은 stage를 의미하지는 않는다. 같은 색의 레지스터들 중 앞쪽의 것은 이름에 접미사 _F를, 뒤쪽의 것은 접미사 _L를 붙였고, 인접한 다른 색의 RG_??_L과 RG_??_F가 하나의 세트로서 같은 stage에 해당한다. _F는 기본적으로 posedge CLK에, _L은 negedge CLK에 갱신되며, _L이 갱신되면 그 stage의 실행이 시작된다고 볼 수 있다. 각 레지스터의 구체적인 명칭은 Figure 1을 참조하라. 명칭의 기본 구조는 RG_(stage)_(role)_(F/L) 이다.

3) Forwarding Unit

Pipelined CPU가 이전 Lab과 다른 가장 큰 이유 중 하나로, FWRD 모듈이 추가된다. 일단 flush 신호에 따라 flush 되는 instruction은 고려하지 않는다. 어차피 없는 셈 쳐야 하는 instruction이기 때문이다. flush 중이 아닌 정상적인 상황이라면, forwarding 받는 instruction이 EX_F 레지스터에 존재하는 경우와 MEM_F 레지스터에 존재하는 경우 각각을 조사한다. 전자가 대부분의 경우고 후자는 store의 경우이다. store은 MEM stage가 시작될 때 포워딩을 받기 때문이다.

전자의 경우 바로 전 instruction과 현재 instruction을 비교해 적절한 신호(FDSrc5, FDSrc6)을 MUX5와 MUX6에 전달하고, 해당하지 않으면 2단계 앞의 instruction과도 비교해 마찬가지로 신호를 전달한다. 이 때 어디에서 forwarding을 받는지는 Figure 1의 MUX 5, 6을 참조하라. Store이 forwarding을 받는 경우는 SWFD 신호가 MUX7에 전달됨으로써 처리된다.

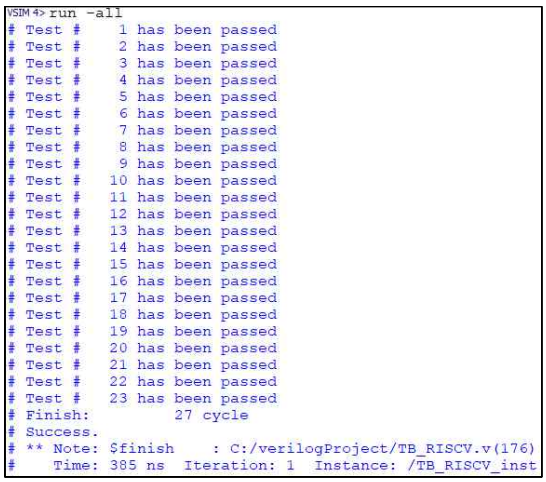
부가적으로 FWRD 모듈은 stall 신호도 발생한다. 이는 LW(load) instruction으로부터 forwarding을 받을 때 필요한데, LW 뒤에 나오는 두 신호는 각각 ID, IF stage에서 stall되어야만 한다. 이는 LW의 경우 포워딩할 데이터가 MEM stage 이후에 나오기 때문이다. LW로부터 forwarding을 받아야 하는 경우에만 stall 신호가 LW로부터 두 stage 뒤의 PC_L 및 ID_L 레지스터들의 갱신을 막는 방식(pause and remain)으로 stall을 실행한다.

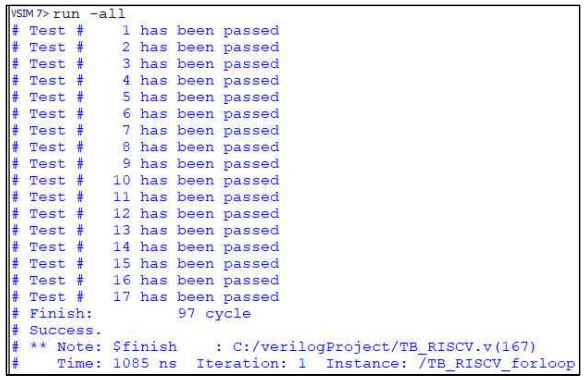
4) Always-Not-Taken Branch Prediction

말 그래도 Branch가 Taken되지 않을거라 가정하고 계속 instruction을 실행하는 방식으로, 사실 그냥 아무것도 안 하고 냅뒀다가 EX stage가 끝나고 branch가 taken되면 나중에 수습하는 방식이다. 따라서 branch가 taken된다고 결정되기 전까지는 전과 같이 PC에 4씩 더해지다가, taken 되었다는 신호(BTaken)가 발생하면 0이었던 flush에 1씩 더해 두 cycle동안 flush가 0이 아니게 만든다. 이 동안은 RG_EX_L 레지스터들을 갱신하지 않음으로서 flush 돼야 할 두 instruction의 데이터를 overwriting을 통해 삭제한다.

4. Evaluation

작성한 코드의 평가는 주어진 3개의 testbench 파일을 통해 하였으며, 각각의 경우에 포함된 모든 테스트를 통과하였다. 또한 waveform를 분석한 결과에 기반하여, 구현 의도에 맞게 성공적으로 5-stage pipelined fashion으로 작동하고 있다고 판단했다. CPI는 1보다는 크지만(이는 flush 및 stall 때문이다), Multi Cycle CPU와 비교했을 때 획기적으로 줄어들었고 이는 waveform과 마찬가지로 성공적인 5-stage Pipelined CPU가 구현되었다는 방증이다.





(L) Figure 2. Result of simulation when 'TB_RISCV_inst.v' was used

(R) Figure 3. Result of simulation when 'TB_RISCV_forloop.v' was used

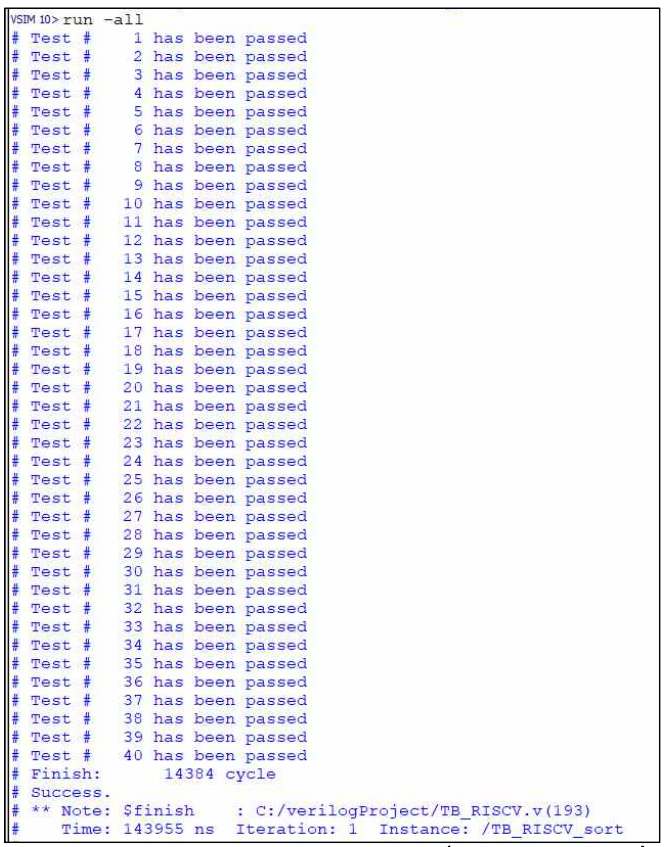


Figure 4. Result of simulation when 'TB_RISCV_sort.v' was used

5. Discussion

본 과제에서 어려웠던 점은 디버깅이었다. 이전 랩들에 비해 레지스터의 수가 훨씬 많아졌고, 특정 instruction에 해당하는 레지스터가 어느 위치에 있는지도 헷갈려서 디버깅이 훨씬 어려웠다. 또 테스트벤치는 테스트케이스의 Pass/Fail이 결정되는 지점에서 모든 요소를 다 판단하지 못하고, 단지 output port로 나오는 값만으로 성공 여부를 출력하기 때문에 우연히도 잘못된 지점부터 몇 개의 테스트는 성공으로 뜨는 경우가 있었다. 그래서 디버깅을 할 때 실패 지점만이 아니라 그 전 몇 개의 instruction도 같이 검토해야 했다. 결과적으로 대략 60시간 이상을 투입하여 구현에 성공했다.

6. Conclusion

작성한 코드를 이용하여 확인해본 결과 실패한 testcase가 없었고 waveform 및 CPI도 적절했기에 Pipelined CPU가 성공적으로 구현되었다고 판단했다. 본 과제를 통해 강의를 통해 배운 Pipelined CPU의 구조에 대해 확실히 이해할 수 있게 되었고, 현대 전자공학 산업의 가장 기본적인 조각 중 하나를 이해했다는 점에서 큰 의미가 있다고 생각한다.