EE312 Lab4 Report

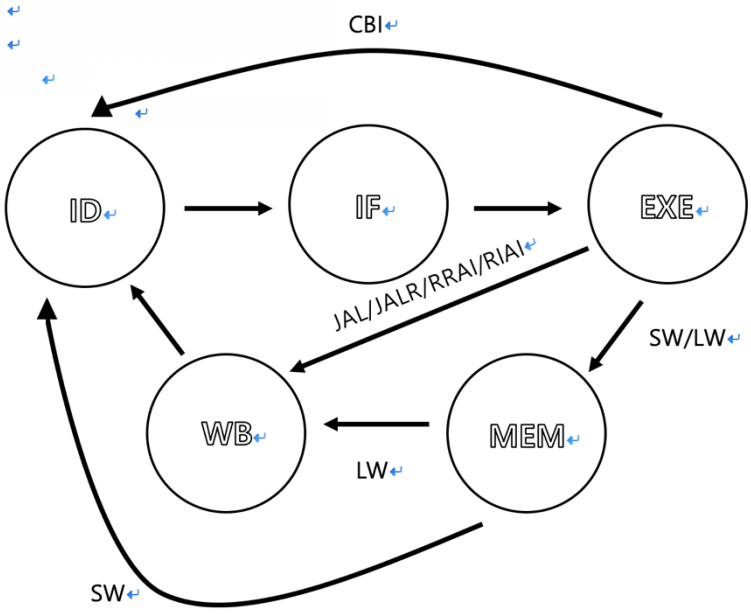
20180155 김준범
20160791 권용빈

Ⅰ. Introduction

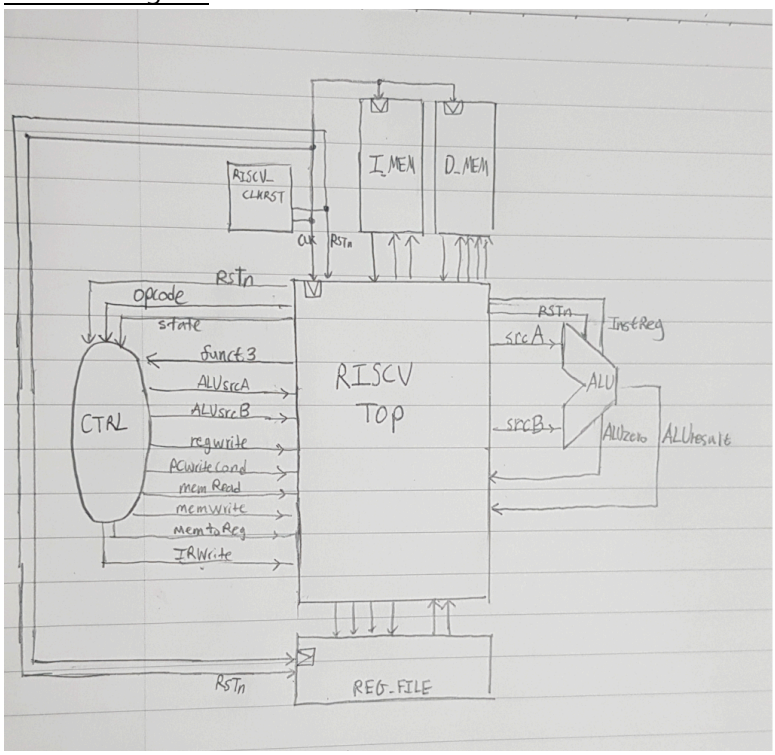
Lab4는 Lab3에서 공부하였던 RISC- V의 uArchecture의 instruction들을 수행할 수 있는 Multi-cycle cpu구현하는 과제였다. CPU를 구성하는 Register, Memory, Clock , ALU, PC, Control Signal에 대한 상호작용을 이해해야 하며 각 instruction마다 필요로 하는 state/control signal 업데이트, ALU/PC 연산에 대한 부분을 implement해야 했다.

Ⅱ. Design

State Diagram



Module Diagram



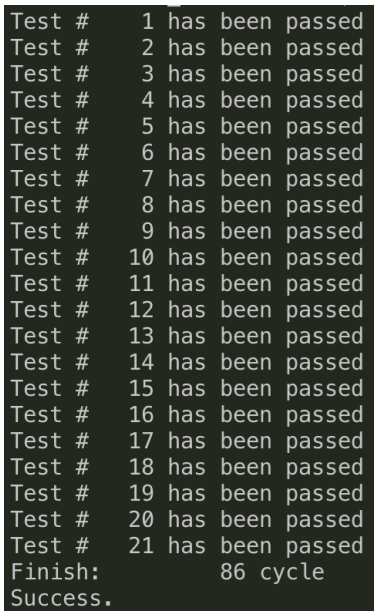
위 design 부분의 그림에서 볼 수 있듯이 구현한 multi-cycle cpu 는 기존 코드에 총 3 개의 module 을 더하였다. RISCV_TOP module 을 통하여 32-bit instruction 을 각 opcode 별로 분류하여 finite-state machine 구조로 실행시켜주었다. IF, ID, EXE, MEM 그리고 WB stage 로 총 5 개의 stage 를 opcode 에 따라 state 를 순차적으로 update 해주었다. RISCV_TOP 에서 state 를 CTRL module 에 넘겨줌으로써 각 state 별로 작동을 제어해주는 control signal 을 update 해주었다. 또한 EXE state 에서 각 instrunction 마다 요구하는 연산은 srcA 와 srcB 값을 control signal 을 통해 적절히 정해준 뒤 ALU module 의 input 으로 넘겨주어 연산을 한 뒤 결과값을 다시 받아오는 형식으로 하였다.

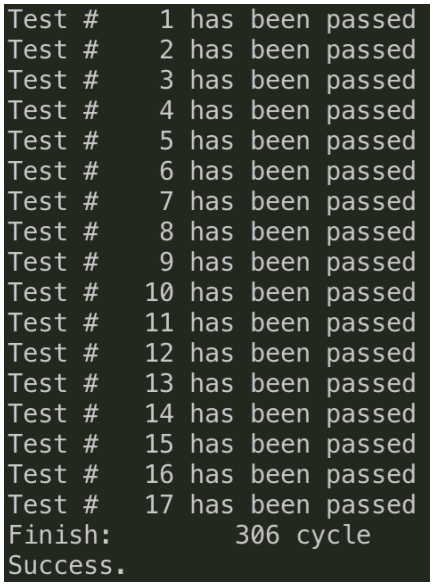
Ⅲ. Implementation

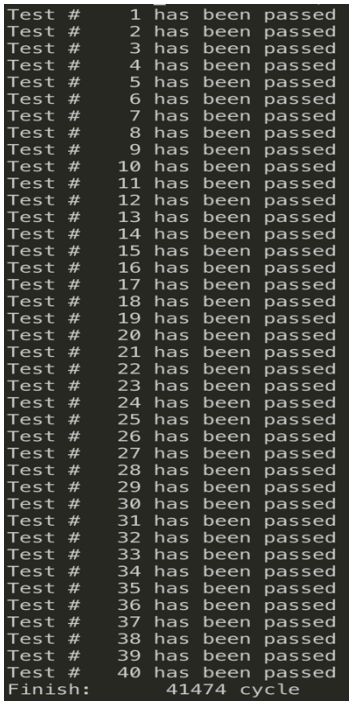
본 과제에서는 위에 Design에서도 언급했듯이 3개의 모듈- RISCV_TOP, ALU, CTRL을 제작했다. RISCV_TOP 모듈은 5개의 state로 구성된 finite state machine의 틀을 가지고 있으며 해당되는 state machine은 always(negedge CLK) 구문안에 있기 때문에 한 clock cycle 마다 실행된다. Always 구문의 실행이 끝날때마다 instruction의 종류에 따라서 state를 update하여 다음 stage로 넘어갈 수 있게 implement했다. ALU와 CTRL 모듈은 모두 combinational 구문(always (*))을 사용하며 모든 input과 output이 wire로 RISCV_TOP에 연결되어있다. CTRL모듈은 opcode와 현재의 state, funct3 인자를 받아서 현재 state에 필요한 control signal을 RISCV_TOP에 asynchronous하게 제공해준다. RISCV_TOP에서는 signal을 받고 ALU의 srcA, srcB에 연결할 값을 선택하거나, register file이나 memory 참조 등의 결정을 내린다. ALU에 연결될 값은 이미 RISCV_TOP 모듈에서 결정이되기 때문에 ALU

자체는 현 instruction이 요구하는 연산을 파악하는 작업만 실행한 후 계산결과인 ALUresult와 branch를 탈지 안탈지의 여부인 ALUzero를 RISCV_TOP에 전달한다.

IV. Evaluation







위의 각 test bench실행 결과의 스크린샷에서 볼 수 있듯이 모든 test case들을 통과하였다. 3개의 test bench들이 이번 과제에서 cover해야하는 모든 instruction을 포함하고 있진 않지만 각 opcode에 해당하는 부분들이 instruction이 제대로 실행 되고 있음을 확인하였기 때문에 큰 문제가 될 거라고 생각하진 않는다.

Inst: 86 cycle
Forloop: 306 cycle
Sort: 41474 cycle

또한 위에 적혀있듯이 inst의 총 cycle갯수는 총 88cycle RRAI와 RIAI instruction에 해당하는 21개의 test case + halt case라고 계산했을 경우 현재의 Instrunction과 이전의 instruction이 halt조건을 만족한다면 halt signal을 1로 setting 해주는 식으로 구현하였기 때문에 (21*4+2)cycle=86로 계산과 결과값이 일치한다. 그러므로 성공적으로 구현되었다고 볼 수 있다. Forloop과 sort 또한 single cycle과 동일한 testcase들을 사용하였고 single cycle에서 나온 instruction갯수와 실행된 instruction의 평균 cycle수를 곱하였을 때 상식적인 오차범위안에 cycle결과값이 존재하기 때문에 성공적으로 구현했다고 볼 수 있다.

Ⅴ. Discussion

.

본 과제에서는 ALU의 src에 필요한 값을 RISCV_TOP의 EX stage에서 설정해주면 asynchronous하게 ALUresult를 얻어내는 디자인을 구상했었는데 ALUresult의 value는 계획한대로 잘 얻을 수 있었지만 ALUzero의 값이 한 cycle 늦게 얻어지는 문제에 마주했다. 그래서 일시적으로 branch에 관련한 ALU 연산만 한 cycle 전 단계인 ID stage에서 alu의 source 값들을 설정해 주었는데 이 부분에 대해서는 discussion이 아직 진행 중이다.

VI. Conclusion

이전 과제인 Lab3 report에서 말했듯이 기존의 단일 Single-cycle cpu 코드를 module화를 진행한뒤에 multi cycle cpu를 구현하여서 상당히 수월하게 진행되었다. Multi cycle cpu를 직접 구현함으로써 module화와 cpu의 관한 이해도가 상승하였다. 하지만 다음 과제는 pipeline을 구현하는 것이 다소 걱정이 된다.