20170699 최진성 20170080 김동규 Lab Report

Lab 4. Multi-cycle CPU Lab

1. Introduction

Multi-Cycle CPU를 Verilog를 사용하여 실제로 구현한다. 지난 Lab 3에서 구현하였던 Single-cycle CPU를 기반으로, 기존에는 하나의 클럭 사이클에서 5단계의 동작 (IF, ID, EX, MEM, WD)이 모두 이루어졌다면 이번 랩에서는 Instruction마다 필요한 동작과 그에 따라 필요한 clk만 사용하도록 하는 CPU를 설계한다. 이를 위해서는 lab3와 마찬가지로 Control path와 Data path가 각각 어떤 동작을 해야 하는지, ISA instruction들의 사전 이해에 더하여, Multi-cycle CPU에서 각 클럭마다 현재 상태에 기반한 control signal과 다음 상태를 결정하는 finite-state machine (FSM)에 대한 이해가 필요하다. 해당 랩을 위해서 Single-cycle CPU를 어떻게 state로 구분하여 다음 state를 결정할 것인지에 대한 사전 설계를 진행하였고, 이를 기반으로 실제 구현을 하였다.

2. Design

Multi-Cycle CPU에는 IF (Instruction Fetch), ID (Instruction Decode), EX (Execute), MEM (Memory Operation), WB (Write Back)가 각각의 state로, 주어진 instruction과 현재 state에 따라 state를 이동할 수 있도록 설계하여야 한다. 해당 instruction에서 필요한 단계가 모두 지났으면 pc를 업데이트하는 신호를 보내 다음 instruction으로 넘어가도록 설계하였다. 해당 CPU의 state diagram은 다음과 같다.

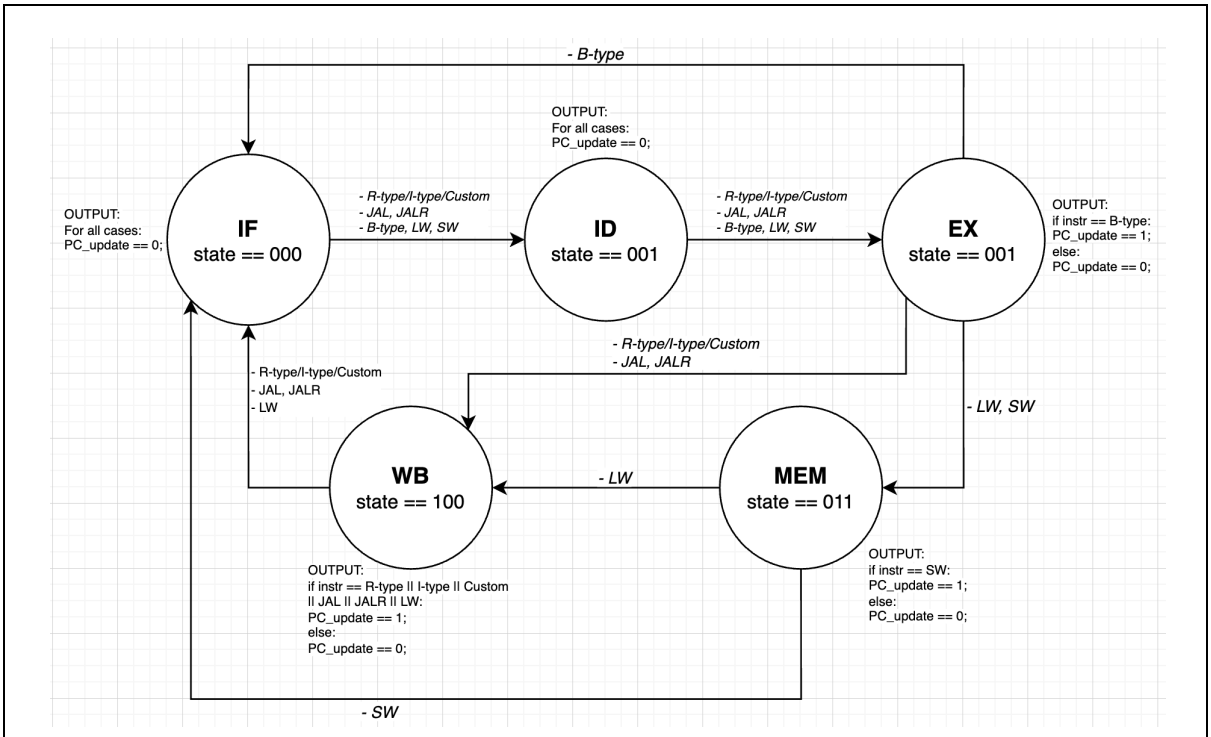


Figure 1. Lab 4에서 구현하고자 하는 Multi-cycle CPU의 State Diagram

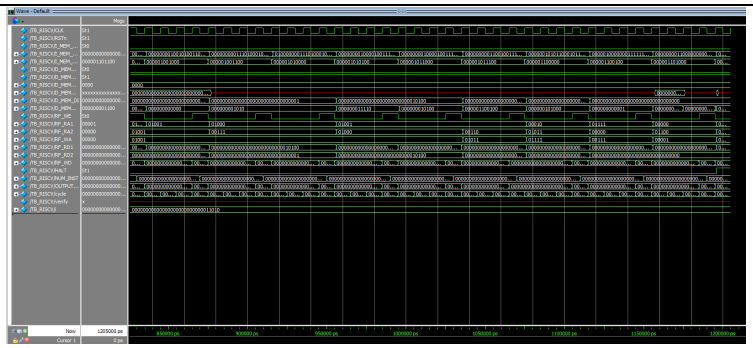
각 state에서는 해당하는 control signal에 올바른 신호를 넣어주어서 각 stage마다 올바른 연산이 가능하도록 설계하였다. 각 연산에 대한 Control signal은 single-cycle lab과 동일하게 설계하였다.

3. Implementation

Control signal의 경우 각 state마다 필요한 control signal을 전달하도록 수정하였다. 모듈들의 연결과 더불어 FSM의 state control이 가능하도록 RISCV_TOP를 모두 구현하였다. 이 때 PC update를 위 state diagram과 같이 하나의 instruction이 모두 끝나고 PC_update가 1일 경우에만 진행하고 다음 instruction을 가져올 수 있도록 구현하였다. 세부적인 각각의 연산 구조는 Single-Cycle과 유사한 방식을 채택하였다. 이전 랩에서 구현하였던, Instruction에 포함되어 있는 immediate value를 출력으로 생성하는 IMM 모듈과 연산을 수행하는 ALU를 활용하였다.

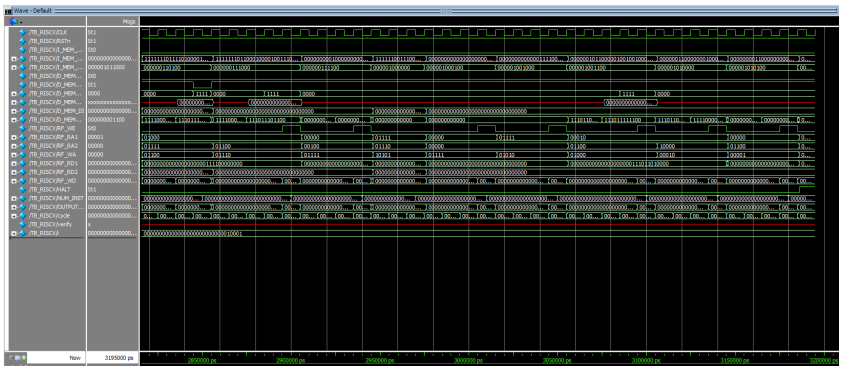
4. Evaluation

완성된 코드를 컴파일한 이후 주어진 3 개의 testbench 파일을 사용하여 시뮬레이션 결과를 확인하였다. 그 결과 모든 테스트에 대해서 다음과 같이 정상적인 결과값이 나오고, reference cycle 과 유사한 값들이 나온다는 사실을 확인하였다.



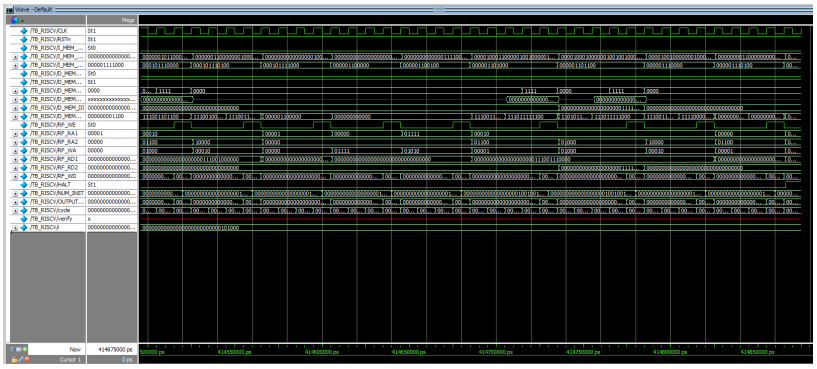
```
# Test #    20 has been passed
# Test #    21 has been passed
# Test #    22 has been passed
# Test #    23 has been passed
# Test #    24 has been passed
# Test #    25 has been passed
# Test #    26 has been passed
# Finish:         110 cycle
# Success.
# ** Note: $finish    : C:/modelsim_project/lab4/TB_RISCV_inst.v(179)
#    Time: 1205 ns  Iteration: 1  Instance: /TB_RISCV
```

Figure 2. 주어진 Testbench 파일로 실행한 시뮬레이션 결과 – inst



```
# Test #    12 has been passed
# Test #    13 has been passed
# Test #    14 has been passed
# Test #    15 has been passed
# Test #    16 has been passed
# Test #    17 has been passed
# Finish:         309 cycle
# Success.
# ** Note: $finish    : C:/modelsim_project/lab4/TB_RISCV_forloop.v(167)
#    Time: 3195 ns  Iteration: 1  Instance: /TB_RISCV
```

Figure 3. 주어진 Testbench 파일로 실행한 시뮬레이션 결과 – forloop



```
# Test #    35 has been passed
# Test #    36 has been passed
# Test #    37 has been passed
# Test #    38 has been passed
# Test #    39 has been passed
# Test #    40 has been passed
# Finish:       41477 cycle
# Success.
# ** Note: $finish    : C:/modelsim_project/lab4/TB_RISCV_sort.v(193)
#    Time: 414875 ns  Iteration: 1  Instance: /TB_RISCV
```

Figure 4. 주어진 Testbench 파일로 실행한 시뮬레이션 결과 - sort

이를 통해 구현한 Multi-Cycle CPU가 정상적으로 잘 작동한다는 사실을 파악할 수 있다.

5. Discussion

구현해야 하는 Multi-Cycle CPU가 동작해야 하는 Instruction마다 어떤 Stage들을 거치고, 이를 어떻게 Finite State Machine으로 설계하여 동작하게 할 것인지에 대한 고민이 필요하였다. 이전에 수행한 과제로부터의 경험이 도움이 되어 큰 어려움 없이 수행하였다. 이전 과제들과 마찬가지로 Classum을 통한 조교님과 교수님의 질의응답이 큰 도움이 되었다.

6. Conclusion

Single-Cycle CPU의 단점을 보완한 Multi-Cycle CPU를 구현하며 두 구현 방식의 장단점을 더 명확하게 이해하게 되었다. Stage로 나눈다는 concept을 이해하고 실제로 구현했다는 점에서 Lab5에서 Pipeline를 구현하기 위한 필수적인 준비 과정을 거쳤다고 생각된다.