Report: Lab4

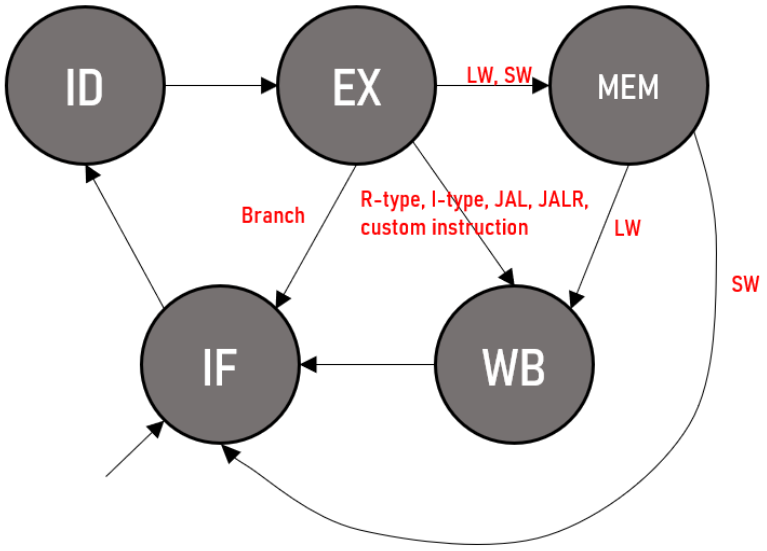
20190533 Hyemin Lee
20190235 Gangtae Park

1. Introduction

이번 Lab4는 지난번 Lab3의 Single-cycle CPU에 이어서 Multi-cycle CPU를 verilog를 이용해 구현하는 Lab이다. Single-cycle CPU를 Multi-cycle CPU로 바꾸기 위해서는 각각의 instruction에 대해서 state별로 수행하는 역할을 적절히 나눠주는 것이 중요하다. Control Unit을 구현함에 있어 두 가지 방법이 있는데 FSM을 기반으로 control unit을 디자인하거나 “microprogram + microcoding”을 기반으로 한 컨트롤러를 구현하는 것이다. 우리는 좀 더 익숙한 방법인 FSM을 기반으로 control unit을 디자인하는 방법을 선택했다.

2. Design

기본적으로 전체적인 module의 구성, control & data path의 분리는 이전 lab3에서 구현한 single-cycle cpu와 동일하다. 우리가 구현한 Multi-cycle CPU는 clock의 positive edge를 만날 때마다 state가 변화한다. 본 CPU가 입력 받는 Instruction들을 LW, SW, Branch 그리고 나머지들로 구분하였고, 어떤 그룹에 속하느냐에 따라 state는 다르게 변한다. state diagram과 transition table은 아래과 같다.



state label	R/I-type, custom,JAL,JALR	LW	SW	Branch
IF	ID	ID	ID	ID
ID	EX	EX	EX	EX
EX	WB	MEM	MEM	IF

MEM		WB	IF	
WB	IF	IF		

3. Implementation

우리는 state를 전환하는 방법으로 control flow에서 이를 작동하게 했다. state와 next_state라는 변수를 만들어서 control flow에서 opcode를 통해 어떤 instruction을 수행하는지 알 고 난 후엔 next_state를 통해서 한 cycle 후에 다음 state로 전환될 수 있도록 하였다. 또한, RISCV_TOP에서 현재 state를 받으면 아래 표에 따라서 각각에 맞는 역할을 수행하도록 MUX들이 컨트롤한다. 아래 표는 우리가 작성한 코드의 변수명을 그대로 사용하였으며, I_MEM_ADDR이 현재 PC의 역할을, PC가 next_PC의 역할을, rs1이 ALU 첫 번째 입력값의 역할을, rs2가 ALU 두 번째 입력값의 역할한다.

R(+custom)-type, I-type, JAL ,JALR	
IF	I_MEM_ADDR <= PC IR <= I_MEM_DI
ID	PC <= I_MEM_ADDR + 4 A <= RF_RD1 B <= RF_RD2 calculate target
EX	rs1 <= A rs2 <= B(R) or imm(I) ALU_out <= rs1 (op) rs2 if (JAL, JALR) PC = target
WB	RF_WE <= 1 RF_WD <= ALU_out (R,I) or I_MEM_ADDR + 4(JAL, JALR)

LW	
IF	I_MEM_ADDR <= PC IR <= I_MEM_DI
ID	PC <= I_MEM_ADDR + 4 A <= RF_RD1 B <= RF_RD2

EX	rs1 <= A rs2 <= imm(I)
MEM	D_MEM_DOUT <= B D_MEM_ADDR <= A + imm
WB	RF_WE <= 1 RF_WD <= D_MEM_DI

SW	
IF	I_MEM_ADDR <= PC IR <= I_MEM_DI
ID	PC <= I_MEM_ADDR + 4 A <= RF_RD1 B <= RF_RD2
EX	rs1 <= A rs2 <= imm(I)
MEM	D_MEM_WEN <= 0 D_MEM_DOUT <= B D_MEM_ADDR <= A + imm RF_WD <= A + imm

Branch	
IF	I_MEM_ADDR <= PC IR <= I_MEM_DI
ID	PC <= I_MEM_ADDR + 4 A <= RF_RD1 B <= RF_RD2 calculate target
EX	rs1 <= A rs2 <= B if (bcond) PC <= target RF_WD <= (bcond) ? 1 : 0

4. Evaluation

ModelSim을 이용해 TB_RISCV_inst, TB_RISCV_forloop, 그리고 TB_RISCV_sort testbench들을 시뮬레이션하였다. 그 결과, TB_RSICV_inst의 경우 모든 testcase(26개)를 통과하였고 총 111cycle의 결과를 얻었다.

```
# Test #   25 has been passed
# Test #   26 has been passed
# Finish:        111 cycle
# Success.
# ** Note: $finish    : C:/Users/rkdxo/Desktop/KAIST/3-2/EE312/lab4/testbench/TB_RISCV_inst.v(179)
#    Time: 1215 ns  Iteration: 1  Instance: /TB_RISCV
```

TB_RISCV_forloop의 경우 모든 testcase(17개)를 통과하였고 총 310cycle의 결과를 얻었다.

```
# Test #   16 has been passed
# Test #   17 has been passed
# Finish:        310 cycle
# Success.
# ** Note: $finish    : C:/Users/rkdxo/Desktop/KAIST/3-2/EE312/lab4/testbench/TB_RISCV_forloop.v(167)
#    Time: 3205 ns  Iteration: 1  Instance: /TB_RISCV
```

TB_RISCV_sort의 경우 또한 마찬가지로 모든 testcase(40개)를 통과하였고 총 41478cycle의 결과를 얻었다. 모든 testbench에 대해서 testcase를 모두 통과하였고 cycle 수 또한 적절히 나온 것을 알 수 있다.

```
# Test #   39 has been passed
# Test #   40 has been passed
# Finish:      41478 cycle
# Success.
# ** Note: $finish    : C:/Users/rkdxo/Desktop/KAIST/3-2/EE312/lab4/testbench/TB_RISCV_sort.v(193)
#    Time: 414885 ns  Iteration: 1  Instance: /TB_RISCV
```

5. Discussion

PC를 update함에 있어서 문제가 발생했었다. JAL instruction을 수행할 때, PC에 target값을 대입하였음에도 불구하고 PC+4로 update되었었다. 항상 PC를 update할 때는 alwyas @ (*)을 통해서 update를 해주었는데 이를 negative edge에서 update되도록 변경해주었더니 문제를 해결 할 수 있었다.

6. Conclusion

Lab4는 verilog을 이용해 Multi-cycle CPU를 구현하는 랩이였다. 이번 랩을 통해서 Single-cycle에 비해서 Multi-cycle이 얼마나 더 효율적인지 알 수 있었으며 state를 바꿔주는 과정이 어떻게 진행되는지, 각 state에서 어떤 역할을 수행하는지를 직접 확인 할 수 있었다. 따라서 Multi-cycle CPU에 대해서 더욱 잘 이해할 수 있게 되었다.