EE312 <Lab 4: Multi-cycle CPU Lab >

20140727 이동현 20170616 정희진

1. Introduction

Lab 4는 verilog를 이용하여 RISC-V의 RV321 architecture를 가지고 있는 Multi-cycle CPU를 설계해보는 lab이다. 우리가 설계할 Multi-cycle CPU는 하나의 clock cycle마다 하나의 stage가 진행된다. Multi-cycle CPU는 어떤 instruction이 실행 되느냐에 따라 필요한 cycle의 개수가 다르고 instruction을 실행하기 위해 필요한 과정을 모두 마치면 처음 상태로 돌아가기 때문에 single-cycle CPU보다 더 좋은 latency를 가질 수 있다는 장점이 있다. Lab 4를 통해 수업시간에 배운 MIPS Multi-cycle CPU와는 조금 다른 방식의 Multi-cycle CPU를 설계해보며 Multi-cycle CPU의 작동 방식을 이해할 수 있다.

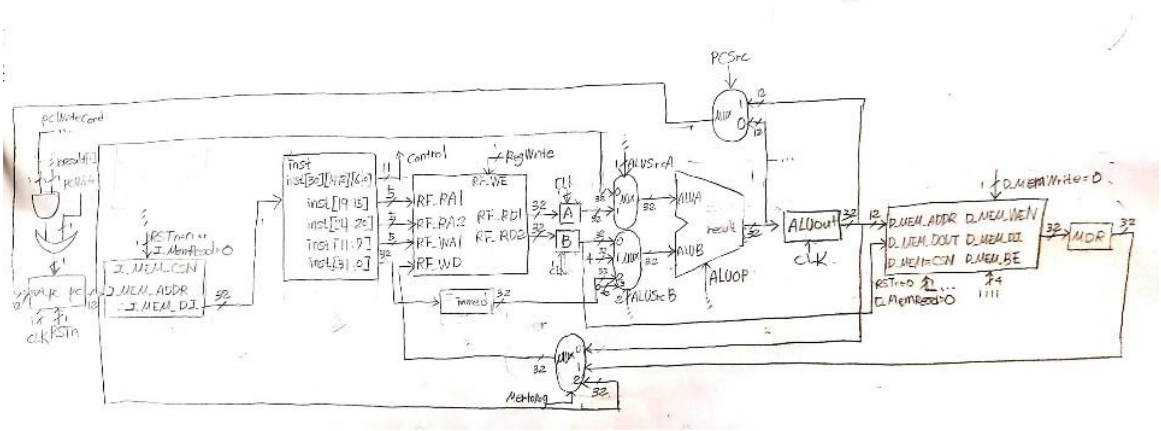
2. Design

RISC-V TOP에서 새롭게 정의한 몇몇 변수들의 의미는 다음과 같다.

pc	program counter
nxt_pc	특정 조건을 만족시킬 경우 clk의 rising edge에서 pc가 되는 값
PCWrite	pc값을 바꿀지 결정하는 control signal
PCWriteCond	조건에 따라 pc값을 바꿀지 결정하는 control signal, branch instruction에서 사용된다.
result	ALU 연산의 결과
state	instruction이 진행하고 있는 단계, IF(000), ID(001), EX(010), MEM(011), WB(100)로 나누어 진다.
nxt_state	clk의 rising edge에서 state가 되는 값
inst	instruction
I_MemRead	instuction memory에서 input된 address에 위치된 instruction을 output할지 결정하는 control signal
immed	instruction을 통해 만든 immediate
RegWrite	register에 data를 넣을지 결정하는 control signal
A, B	register에서 나온 data가 clk의 rising edge가 될 때 A, B로 옮겨진다.
ALUSrcA, ALUSrcB	ALUA, B에 넣을 값을 선택
ALUA, ALUB	ALU에 operand로 들어가는 값
ALUOP	ALU 연산을 선택해주는 control signal
ALUOut	ALU 연산을 통해 나온 result값이 clk의 rising edge가 될 때 ALUOut으로 옮겨진다.
PCSrc	pc source, nxt_pc를 결정하는 control signal
MDR	memory에서 나온 data

D_MemRead	data memory에서 input된 address에 위치된 data를 output할지 결정하는 control
D_MemWrite	memory에 data를 넣을지 결정하는 control signal
MemtoReg	register에 넣을 data를 결정하는 control signal. jalr을 컨트롤 하기 위해서 pc를 추가해주었다.

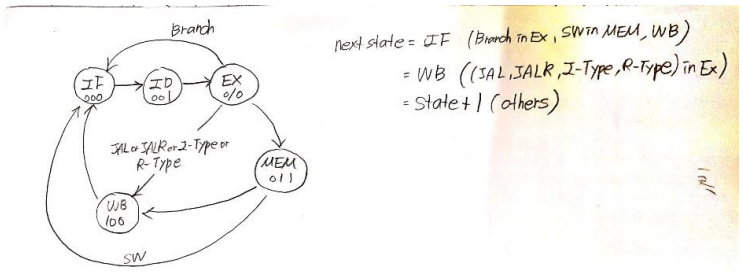
설계한 Multi-cycle CPU의 그림은 다음과 같다.

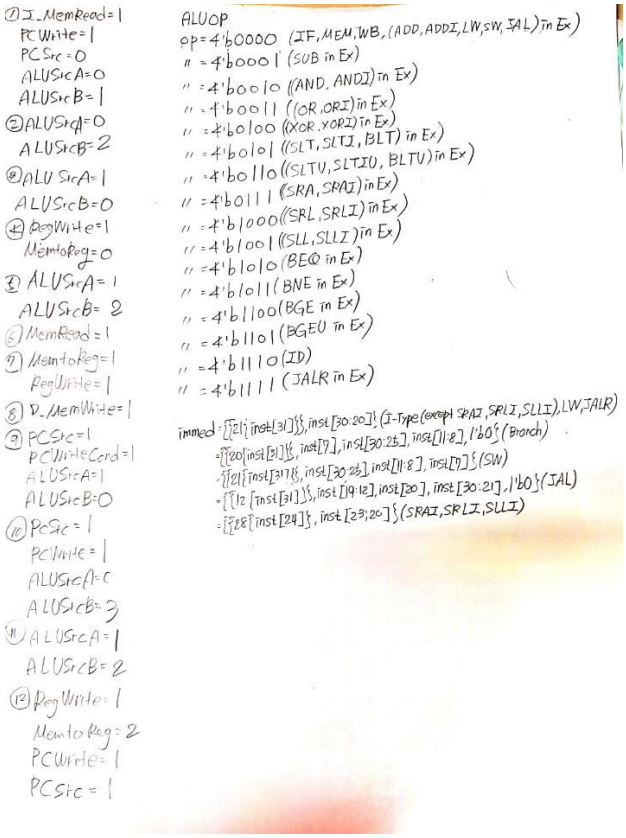


3. **Implementation**

(1)I-type, R-type, lw, sw, branch, jal, jalr의 각 stage에서 행해져야 하는 것들은 아래의 그림과 같다. 중복되는 명령이 많기 때문에 같은 명령의 경우 숫자를 붙여서 간략하게 표현하였다.

(2)각 명령들이 수행될 때 control part에 의해 정해지는 변수 값들과 instruction에 따른 state의 변화는 다음과 같이 설정해주었다.





4. Evaluation

모든 testcase를 통과하였으며 속도도 빨랐다.

(1) TB_RISCV_inst.v

```
# Loading work.REG_FILE
VSIM 2> run -all
# Test #    1 has been passed
# Test #    2 has been passed
# Test #    3 has been passed
# Test #    4 has been passed
# Test #    5 has been passed
# Test #    6 has been passed
# Test #    7 has been passed
# Test #    8 has been passed
# Test #    9 has been passed
# Test #   10 has been passed
# Test #   11 has been passed
# 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
# Test #   18 has been passed
# Test #   19 has been passed
# Test #   20 has been passed
# Test #   21 has been passed
# Finish:         90 cycle
# Success.
# ** Note: $finish    : C:/modelsim_project/multicycle lab/testbench/TB_RISCV_inst.v(175)
#    Time: 1005 ns  Iteration: 1  Instance: /TB_RISCV
```

(2) TB_RISCV_forloop.v, TB_RISCV_sort.v

```
# Start time: 19:20:25 on Nov 06,2019
# Loading work.TB_RISCV
# Loading work.RISCV_CLKRST
# Loading work.RISCV_TOP
# Loading work.SP_SRAM
# Loading work.REG_FILE
VSIM 4> run -all
# Finish:         310 cycle
# Success.
# ** Note: $finish    : C:/modelsim_project/multicycle lab/testbench/TB_RISCV_forloop.v(166)
#    Time: 3205 ns  Iteration: 1  Instance: /TB_RISCV
# 1
# Break in Module TB_RISCV at C:/modelsim_project/multicycle lab/testbench/TB_RISCV_forloop.v line 166
# Compile of TB_RISCV_sort.v was successful.
VSIM 5> vsim -gui work.TB_RISCV
# vsim
# Start time: 19:21:15 on Nov 06,2019
# Loading work.TB_RISCV
# Loading work.RISCV_CLKRST
# Loading work.RISCV_TOP
# Loading work.SP_SRAM
# Loading work.REG_FILE
VSIM 6> run -all
# Finish:       41478 cycle
# Success.
# ** Note: $finish    : C:/modelsim_project/multicycle lab/testbench/TB_RISCV_sort.v(192)
#    Time: 414885 ns  Iteration: 1  Instance: /TB_RISCV
```

5. Discussion

각 stage에서 변수들이 어떤 값을 가질지 고려하며 설계하는데 많은 시행착오가 있었다.

1. 처음에 test를 실행했을 때, OUTPUT_PORT 값이 나오지 않았다. 이는 코드를 보면서 다시 debugging을 하니 pc의 초기값을 잘 설정해주지 않아서 나타난 에러라는 것을 깨달았다. 초기값을 잘 설정해주니 OUTPUT_PORT가 잘 나왔다.

2. instruction마다 stage(clock의 수)가 다른데, test file은 1clock마다 test하였다. test file은 OUTPUT_PORT에 나온 값을 보며 test하는데, OUTPUT_PORT는 어느정도의 stage 통과 후에 만들어지므로 OUTPUT_PORT값이 만들어 지기 전 stage에서는 fail을 하고 만들어진 후에 success를 하였다. (ex. addi instruction을 받으면 test는 첫번째, 두번째, 세번째는 실패하고 4번째 성공한다) 이는 NUM_INST를 잘못설정해주었기 때문에 발생한 에러였다. 처음에는 NUM_INST를 instruction의 IF 단계에서 +1를 해주었다. 가장 나중 stage에서 +1를 해주니 에러가 발생하지 않았다.

3. jalr에서 A값이 정해지기 전에 먼저 사용해서 에러가 발생했다. 이 경우 A값은 ID단계 때 정해주고, EX단계 때 사용하면서 debugging을 하였다.

4. memory에 값을 저장해주는 control signal인 D_MemWrite를 memory module과 연결해주지 않아 lw와 sw instruction을 실행하는데 있어서 에러가 발생했다. 신호를 파악하면서 알아냈고, 연결해주니 잘 실행이 되었다.

6. Conclusion

모든 testcase를 잘 통과한 것으로 보아 의도한 대로 잘 설계된 것으로 보인다. 이 과정에서 FSM, single-cycle CPU와 multi-cycle CPU의 차이와 multi-cycle CPU를 설계할 때 고려해야 하는 것들에 대해 잘 이해할 수 있었다.

이번 lab을 하면서 instruction의 stage를 좀 더 줄일 수 있지 않을까라는 생각이 들었다. 제출된 코드에서는 ALU를 1개만 사용했기 때문에 pc값을 계산할 때와 data값을 계산할 때 많은 충돌이 일어나면서 stage가 많아질 수 밖에 없었다. 만약 ALU가 2개였다면(or ALU1개와 adder 1개) 충돌이 일어나지 않으면서 stage를 줄일 수 있을 것이다. 또한 lab를 하면서 lecture에서 배웠던 것들 중 몇가지 불필요한 register와 constrol signal을 발견했다. 예를 들어 제출된 코드에서는 lecture에 있는 IorD signal과 IRWrite signal은 사용하지 않았다. register 또한 자세히 살펴보면 필요가 없는 것들이 몇가지 있다(ex. MDR). 다만 이번랩에서는 input, output된 data값들을 잘 명시하기 위해서 register을 사용한 경우가 많다. lecture에 나와있는 대로 2개의 memory를 통합시킬 수 있겠다라는 생각도 들었다. 만약 이런 경우에는 control singal이 더 필요할 것으로 예상된다.

다음에 이와 관련된 coding을 하게 된다면 1. 불필요한 register, control signal을 제외하거나 2. pc와 관련된 stage를 미리 연산함으로써 instruction의 clock 수를 줄이거나 3. 사용된 2개의 memory를 하나로 통합해서 보다 더 간단한 디자인을 완성할 수 있을 것이다.