CSED311 Lab3: Single Cycle CPU

컴퓨터공학과 강덕형 20170684
컴퓨터공학과 이승준 20170735

Introduction

설계 및 구현 목표

이번 Lab 과제는 TSC ISA 를 동작시킬 수 있는 single cycle cpu 를 구현하는 것이 목표이다. 이 과정에서 instruction processing FSM인 cpu를 설계하고, cpu를 구성하는 다양한 component들에 대해서 이해하고 구현한다.

학습목표

이번 Lab 과제를 통해 instruction processing FSM인 cpu를 이해하고 이를 구성하고 있는 component인 datapath와 control unit의 역할과 특성에 대해 학습하는 것이 목표이다. 이 과정에서 모듈화를 익히고 cpu와 memory간의 상호작용이 어떻게 일어나는지 이해한다.

Design

cpu는 data의 path에 관여하는 unit들인 datapath와 instruction을 decode하고 control signal들을 생성하는 control unit으로 구성되어 있다. 우선 datapath의 경우 ALU, Register file, Extend Delegator 및 MUX들로 디자인 하였다. control unit의 경우 instruction decoder, Control, PcMuxSelector로 구성하였다. 각각의 모듈들은 다음과 같은 역할을 수행한다.

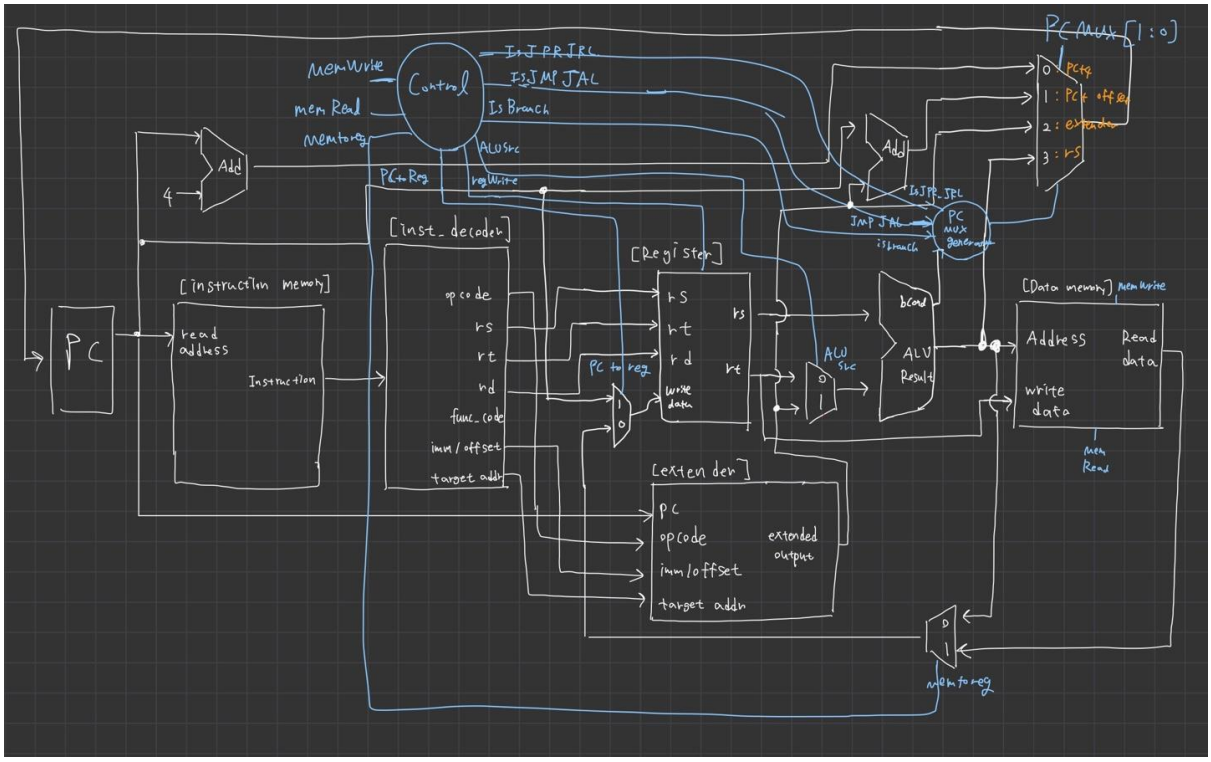
Data Path:

- ALU: register - register 혹은 register - immediate value간의 연산을 수행한다.
- RegisterFile: 4개의 레지스터를 가지고 있으며 외부에서 입력되는 신호와 주소에 따라 register의 값을 쓰거나 읽는다.
- ExtendDelegator: TSC에서 수행하는 다양한 연산들은 immediate value나 offset을 다양한 방법으로 extend하기를 요구한다. 이에 해당 모듈이 연산 종류에 맞게 extend 전체를 관장해준다.
- MUX: datapath에서 control signal에 따라 data를 취사 선택해서 연결해준다.

Control Unit:

- InstDecoder: 16 bit instruction 을 받아, opcode, func_code, register 주소 등을 파싱해 control 모듈 및 Data Path 로 뿌려준다.

- Control: InstDecoder가 제공한 opcode와 func_code를 받아 연산 종류에 따라 control signal을 생성하여 Data Path로 뿌려준다.
- PcMuxSelector: Control 모듈에서 signal들을 받아서 next pc를 결정하는 mux에 제어 입력을 제공해 준다.



<Fig 1. Diagram of CPU>

Implementation

[Data Path]

ALU: func code와 opcode 그리고 alu_input_1, alu_input_2를 받아 opcode와 func_code에 해당하는 연산을 case 문을 이용해 수행한다. 연산 결과인 alu_output과 연산이 branch type인 경우 branch condition이 충족이 되었는지 여부를 나타내는 bcond를 output으로 가진다.

RegisterFile: 입력으로 레지스터 주소를 받고, 해당 주소의 레지스터에 저장되어 있는 데이터를 output으로 출력한다. 만약 pos clock 에 reg_write_signal이 1인 경우, 입력으로 전달받은 write_addr에 해당하는 레지스터에 write_data를 써준다.

ExtendDelegator: TSC 의 ISA 를 구현하기 위해서는, immediate value or offset, target address 를 다양한 형태로 extend 할 수 있어야 했다. 미리 다양한 형태의 extend 를 해주는 submodule 들을 ExtendDelegator 내에 instantiation 해놓은 후, pc 와 opcode 에 따라 extended_output 을 적합하게 내보내 주었다.

MUX: 몇 개의 input 중에 선택해야하는지에 따라, 모듈을 나누어 구현하였다. 2개, 3개, 4개의 input 중 취사선택할 수 있도록 모듈을 3개 구현했으며, 3항 연산자를 이용해 sel 값에 따라 out 으로 적절하게 선택된 값을 전달하였다.

[Control]

InstDecoder: instruction을 받아 instruction을 구성하는 여러 요소들을 bit 위치에 따라 구별하여 parsing해 주었다. write address의 경우 JAL/JRL 연산의 경우 TSC ISA에 따라 2, I type 연산 및 LWD, SWD 의 경우 inst[9:8], 그 외 연산의 경우 inst[7:6] 로 결정되었다.

Control: 입력으로 전달받은 opcode와 func_code에 따라 control signal들을 적절히 생성하여 주었다.

PcMuxSelector: Next pc를 결정할 때, 현재 수행하고 있는 연산의 종류에 따라 next pc는 다르게 계산된다. 현재 연산이 branch type이고 bcond가 true일때, next pc 를 결정하는 mux 에 0 을 제어입력으로 제공한다. 현재 연산이 JMP/JAL의 경우 1를 제어입력으로 제공한다. 현재 연산이 JPR/JRL의 경우 2을 제어입력으로 제공한다. 나머지 연산의 경우 3을 제공한다.

[CPU와 메모리간의 데이터 교환]

본 과제의 경우 instruction memory와 data memory가 구분이 안되어 있고, inout port 로 선언된 data wire를 통해 memory와 cpu간의 data를 교환한다. 이에 우리는 posedege clk 일 때, data 에 pc 를 넣어주고 readM 을 1 로 만들어주었다. 이에 CPU 는 memory 에게 데이터를 읽어달라고 요청하고, memory 는 데이터를 읽어주고 inputReady 를 1 로 만든다. CPU는 inputReady가 1이 될때까지 기다리고 있었고 inputReady가 1이 되는 순간 data는 memory에서 fetch한 instruction이 된다. 이후 CPU 는 instruction 변수에 받아온 데이터를 저장하고, readM signal 을 0 으로 만들어준다. CPU 는 negedge clk 일 때, LWD or SWD 연산일 경우 메모리와 추가적으로 데이터교환을 해야한다. 데이터 교환에 사용되는 address 는 alu_output 이고, 데이터 교환을 시작하기 위해 CPU 는 memory 쪽으로 데이터를 읽거나 써달라고 signal 을 킨다. 그리고 memory 로부터 데이터 작업이 끝났다는 signal 을 wait 을 이용해 기다린다. 이후, 다시 CPU 는 다시 memory signal 을 끈다.

[CPU]

위 모듈들을 적절히 연결해 우리는 cpu를 구현하였다. cpu는 clock의 posedge때 instruction fetch를 수행한다. memory로부터 가져온 instruction을 instruction decoder는 parsing하고 parsing 결과를 Control/ExtendDelegator/RegisterFile/ALU에 제공한다. 각 모듈들은 제공받은 input에 맞게 작동을 수행한다. 만약 연산의 종류가 LWD/SWD인 경우 추가적인 메모리간의 데이터 교환을 필요로 하는데, 이는 negedge에서 일어난다. 우리는 nextPC를 이러한 연산을 수행한 이후에 결정할 수 있으며, PcMuxSelector에게 받은 제어입력에 따라 nextPC가 결정된다.

Discussion

Data wire delcared by inout port

inout port 는 input 과 output 모두 될 수 있는 wire 이며, 모듈 간에 데이터 교환이 양방향으로 이루어질 때 사용된다. 그리고, inout port 로 선언된 변수를 이용해 input 을 받기 위해선, 해당 변수가 zzz...z 값이 되어 있어야 한다.

본 과제에서는 testbench 안의 memory 와, 우리가 구현해야할 cpu 가 데이터를 교환하기 위해 inout data 가 이용되었다.

Testbench 의 메모리 입장에서는, readM or inputReady 가 켜져 있을 때가, 메모리 자신이 CPU 로 데이터를 출력해줘야할 때이다. 그러므로 (readM || inputReady) 일 때 memory 는 data 를 loadedData 로 assign 해 output 을 내놓는다. 그 외의 경우에는 CPU 로부터 data input 을 받기 위해 `WORD_SIZE'bz 로 assign 해놓는다.

CPU 입장에서는, writeM or ackOutput 이 켜져 있을 때가, CPU 자신이 메모리로 데이터를 출력해줘야 할 때이다. 그러므로 그 경우 cpu 는 data 를 reg_data2 로 assign 해준다. 그 외의 경우는 memory 로부터 data input 을 받기 위해 `WORD_SIZE'bz 로 assign 해놓는다.

Use of `wait` statement and both posedge and negedge of clk

우리는 posedge와 negedge에서 wait문을 사용해 주었다. 그 이유는 특정 조건이 만족된 이후에만 일부 assignment가 실행되게 하기 위함이다. posedge의 경우, wait문 안에 inputReady == 1 이라는 조건을 집어넣어 memory에서 data read가 완료되고 inout port의 data가 memory로부터 읽어 온 data(이 경우에는 instruction)이 되어야만 instruction <= data, readM <= 0이 실행되어 읽기 모드를 종료한다. negedge의 경우 pc update를 수행한다. 또한 LWD/SWD의 경우 negedge clk 일 때 메모리와의 데이터 교환을 한 번 더 수행하도록 구현했다. 물론 wait문을 적절히 활용하면

posedge만으로도 모든 연산을 다 수행할 수 있지만, instruction fetch와 memory data 교환을 구분하기 위해 우리는 posedge와 negedge clk를 모두 이용하였다.

Conclusion

우리는 본 과제에서 instruction processing FSM 인 CPU 를 이해하고, 이를 모듈화를 사용해 구현함으로서 cpu를 구성하고 있는 component인 datapath와 control unit의 역할과 특성에 대해 학습할 수 있었다. 또한, cpu와 memory간의 상호작용이 어떻게 일어나는지도 이해할 수 있었다.