컴퓨터구조 (CSED311)

Lab 2 Report

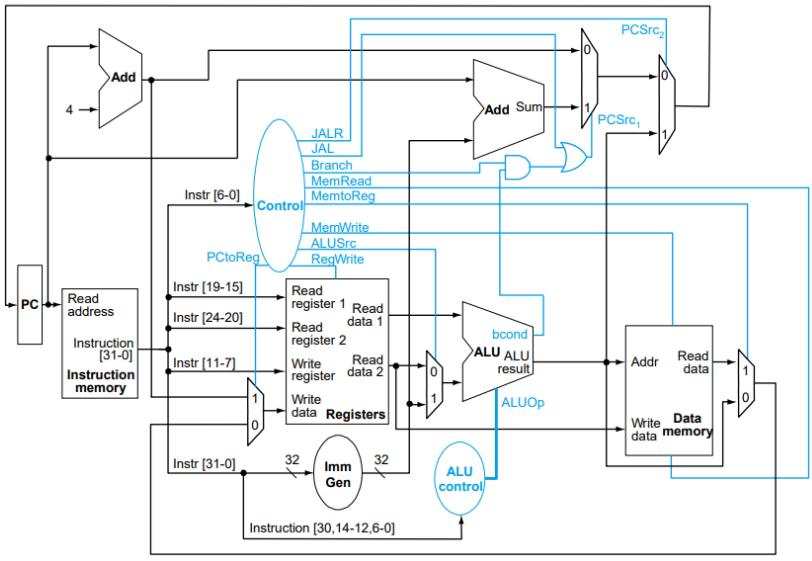
20200220 오상윤

(1) Introduction

이번 lab에서는 베릴로그를 이용하여 single cycle RISC-V cpu를 구현하였다. 주어진 cpu는 input으로 reset과 clk를, output으로 machine의 중지를 나타내는 is_halted를 갖는다. clk의 positive edge가 한 cycle을 나타내며, cpu module은 ECALL instruction을 만났을 때 x17 레지스터의 값이 10인 경우 is_halted의 값을 1로 출력하여 machine의 중지를 알린다. 처리 가능한 instruction은 R-type, I-type, S-type, B-type, J-type이며, LOAD instruction은 LW, STORE instruction은 SW만을 처리 가능하다. 파일 구성은 cpu module을 담고 있는 cpu.v, RegisterFile module을 담고 있는 RegisterFile.v, InstMemory module과 DataMemory module을 담고 있는 Memory.v, 나머지 PC module, ControlUnit module, ImmediateGenerator module, ALUControlUnit module, ALU module을 담고 있는 modules.v, constant의 define한 값들을 담고 있는 opcodes.v로 구성되어 있다.

(2) Design

수업 자료에 있는 Single-Cycle Datapath Design을 토대로 하여 전체적인 module을 Design 하였다.



top module로부터 호출되어 cpu로써 동작하는 cpu module안에서 각 module이 IF, ID, EX, MEM, WB를 위한 logic을 수행하도록 하였다. PC, InstMemory module이 IF를, ControlUnit, RegisterFile, ImmediateGenerator, ALUControlUnit module이 ID를, ALU module이 EX를, DataMemory module이 MEM을, RegisterFile module이 WB를 수행하는 데에 사용된다. cpu module 내부의 submodule들에 대한 구체적인 내용은 아래와 같다.

1. PC module

   매 cycle마다 pc 값을 변경하는 module이다. 초기의 default pc값은 0x0으로 설정하며, 이후 매 cycle 마다 logic을 통해 새롭게 계산된 새로운 pc 값 next_pc를 current_pc 값으로 넣어준다. cpu는 current_pc를 받아 InstMemory module에 input으로 전달하여 해당 line의 instruction을 얻어낸다.

2. InstMemory module

   메모리와 동일하게 동작하도록 설계되어 있으며 cpu로부터 addr(pc 값)을 받아 해당 addr의 instruction을 읽어 dout으로 내보낸다. 초기에 지정된 txt 파일로부터 instruction을 읽어와 메모리와 같이 동작하는 배열에 저장해 놓는다. cpu는 dout으로 나온 instruction을 받아 다른 sub module들로 연결시켜 logic을 수행하도록 한다.

3. ControlUnit module

   cpu로부터 instruction에 따른 opcode를 받아 opcode에 따라 control signal을 적절하게 설정한다. opcode가 JAL인 경우 is_jal을, JALR인 경우 is_jalr을, BRANCH인 경우 branch를, LOAD인 경우 mem_read와 mem_to_reg를, STORE인 경우 mem_write를, I-type또는 S-type인 경우 alu_src를, S-type도 아니고 B-type도 아닌 경우 write_enable을, JAL 또는 JALR인 경우 pc_to_reg를, ECALL인 경우 is_ecall을 1로 설정하는 combinational logic이다. cpu는 각각의 signal들을 다른 module들에 적절하게 연결하여 signal에 따라 올바르게 동작하도록 한다.

4. RegisterFile module

   register들의 값을 담고 있는 module로, source register 1, 2의 번호를 받아 각각의 register의 값을 output으로 내보낸다. 또한 이번 lab에서 ecall instruction이 실행되었을 때 x17 register가 값이 10인 경우에만 machine을 halt하도록 설계되었으므로, x17 register의 값이 10인 경우 x17_is_10이라는 signal을 내보내도록 하였다. write_enable이 1인 경우 destination register의 번호에 해당하는 register에 접근하여 rd_din을 통해 입력으로 들어온 값을 register에 기록하도록 하였다.

5. ImmediateGenerator module

instruction에서 opcode를 뽑아내 각 opcode에 따라 적절하게 immediate value를 생성하는 module이다. I-type, S-type, B-type, J-type에 따라 immediate value의 생성 방법이 다르며, 구체적인 생성은 아래 implemetation에서 설명할 것이다. cpu는 생성된 immediate value를 적절하게 이용하여 ALU의 입력 값으로 주거나 다음 PC 값 계산을 수행하도록 한다.

6. ALUControlUnit module

   instruction에서 opcode, func3, func7을 뽑아내 ALU가 수행할 operator를 적절하게 지정한다. 이때 B-type은 sub 연산을, LOAD, JALR, STORE는 add 연산을 수행하도록 지정하였다. ALUControlUnit module이 alu_op를 통해 operator를 지정하면, ALU module이 이를 input으로 받아 해당하는 연산을 수행한다.

7. ALU module

   ALUControlUnit으로부터 alu_op를 받아 input들에 대해 적절한 연산을 수행한 다음, 연산 결과를 내보낸다. 이때 B-type의 instruction인 경우 bcond를 지정해줘야 하므로, sub 연산인 경우 func3의 값을 참고하여 연산의 결과에 따라 bcond를 적절하게 지정해주도록 하였다. cpu에서 bcond를 Branch의 taken 여부 결정에 이용하도록 한다.

8. DataMemory module

   메모리와 동일하게 동작하도록 설계되어 있는 데이터를 저장하는 메모리이다. 초기에 모든 메모리 공간을 0으로 초기화 해놓고, 이후 cpu로부터 전달받는 mem_read, mem_write signal에 따라서 addr에 해당하는 memory 값을 읽거나 memory에 값을 쓰는 기능을 수행한다.

(3) Implementation

1. PC module

   input으로 reset, clk, next_pc 를 받으며, output으로 current_pc를 갖는다. clock synchronous 하게 동작하며 clk의 positive edge마다 current_pc의 값을 next_pc로 바꾸어 준다. default PC의 값이 0x0이므로 reset이 1인 경우, pc를 32'b0으로 초기화하도록 하였다.

2. InstMemory module

   input으로 reset, clk, addr를 받으며, output으로 dout을 갖는다. mem이라는 reg 배열을 가지고 있어 실제 메모리처럼 동작하도록 구현되어 있으며, 초기에 (reset이 1일

때) readmemh를 통해 지정된 경로에 존재하는 txt 파일에 접근하여 txt 파일 속 binary format의 instruntion들을 mem 배열에 저장한다. 이후 asynchrounous하게 addr 값을 받아 해당 메모리에 접근하여 dout으로 내보내준다.

3. ControlUnit module

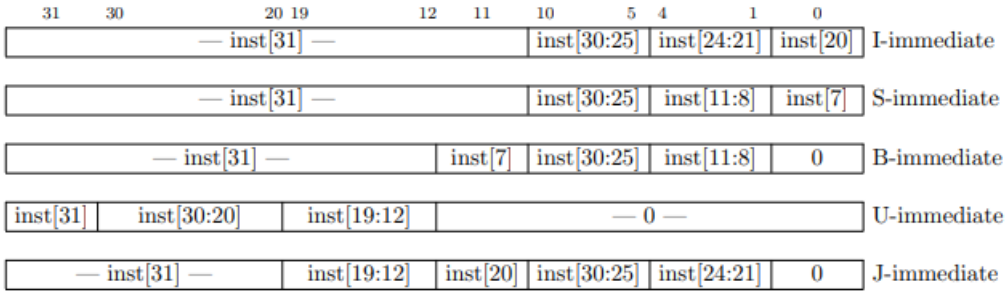
   input으로 part_of_inst을 받으며, output으로 is_jal, is_jalr, branch, mem_read, mem_to_reg, mem_write, alu_src, write_enable, pc_to_reg, is_ecall 을 갖는다. cpu로부터 받는 part_of_inst는 instruction의 하위 비트 7개로 opcode를 나타내며, opcode에 따라 combinational logic을 통해 asynchronous하게 control signal들을 적절하게 바꿔 output으로 내보낸다.

4. RegisterFile module

   intput으로 reset, clk, rs1, rs2, rd, rd_din, write_enable을 받으며, output으로 rs1_dout, rs2_dout, x17_is_10을 갖는다. rf라는 reg배열을 가지고 있어 이 배열이 register들의 값들을 담고 있다. 초기에 (reset이 1일 때) rf 배열을 모두 0으로 초기화하고, x2 register는 그 값을 0x2ffc로 초기화한다. rs1, rs2, rd는 각각 source register 1, 2, destination register의 number를 나타낸다. asynchronous하게 rs1과 rs2값에 따라 해당 register의 값을 rs1_dout과 rs2_dout에게 넘겨준다. x17_is_10은 x17 register의 값이 10인 경우 1의 값을 갖도록 한다. 또한 clock synchronous하게 clk가 positive edge일 때 write_enable의 값이 1인 경우 rd에 해당하는 register에 rd_din으로 들어온 값을 기록한다. 만약 x0에 값을 기록하고자 하는 경우에는 x0는 항상 0의 값을 가지므로 값을 기록하지 않는다.

5. ImmediateGenerator module

   input으로 part_of_inst, output으로 imm_gen_out을 갖는다. cpu로부터 받는 part_of_inst는 32 bit의 instruction이며 하위 비트 7개로 opcode를 뽑아내 opcode에 따라 asynchronous하게 instruction으로부터 immediate value를 적절하게 생성하여 imm_gen_out으로 내보낸다. opcode를 통해 type을 얻어낸 다음 각 type에 따라 immediate value의 생성방법은 아래의 riscv-spec 자료의 그림을 따른다.



6. ALUControlUnit module

   input으로 part_of_inst, output으로 alu_op를 갖는다. cpu로부터 받는 part_of_inst는 32 bit의 instruction이며, instruction에서 opcode, func3, func7을 뽑아내 각 type과 func7, func3를 고려하여 asynchronous하게 alu에게 전달할 operator인 alu_op를 적절하게 지정하여 내보낸다.

7. ALU module

   input으로 alu_op, alu_in_1, alu_in_2, funct3를 받으며, output으로 alu_result, alu_bcond를 갖는다. asynchronous하게 동작하는 combinational logic이며, alu_op에 따라 operator에 맞게 alu_in_1과 alu_in_2간의 연산을 수행하여 alu_result로 내보낸다. 이 때 sub 연산을 수행한 경우, funct3의 값에 따른 Branch 유형에 따라 bcond를 적절하게 지정하도록 하였다. sub 연산 이외의 연산의 경우 연산의 결과와 상관없이 bcond를 0으로 지정한다.

8. DataMemory module

   input으로 reset, clk, addr, din, mem_read, mem_write를 받으며, output으로 dout을 갖는다. mem이라는 reg 배열을 갖고 있어 실제 메모리와 동일한 기능을 수행한다. 초기에 (reset이 1일 때) mem 배열 내 원소를 모두 0으로 초기화한다. mem_read가 1인 경우, addr 값에 따라 asynchronous하게 mem 배열에 접근하여 해당하는 index의 원소 값을 dout으로 전달한다. mem_write가 1인 경우, addr 값에 따라 clock synchronous하게 clk가 positive edge일 때 mem 배열에 접근하여 해당하는 index의 원소에 din 값을 넣어준다.

(4) Discussion

이번 lab 에서 CPU 를 구현하면서 reg 를 최소한으로 사용하고 wire 를 주로 사용하여 code 를 최대한 단순하게 짜고자 노력하였다. 값을 담고 있는 reg 가 많아질수록 code 가 복잡해지고 생각해야 하는 부분이 많아진다는 점을 몸소 느낄 수 있었다. ALU module 을 구현하는 과정에서 BRANCH instruction 을 위한 bcond output 을 설정하기 위해 연산 결과를 적절하게 비교하여야 했는데, reg 에 담긴 값은 기본적으로 unsigned 값으로 처리가 되어 연산 결과가 음수 값인지 0 보다 큰 값인지를 여부를 단순히 0 과 비교 연산해서는 얻어낼 수가 없었다. BLT, BGE 를 처리하기 위해서는 연산 결과가 음수 값인지 알아야 했고, 연산 결과를 저장한 reg 의 MSB 가 1 인지 확인함으로써 음수 값인지 확인하고 bcond 를 적절하게 지정해줄 수 있었다.

(5) Conclusion

주어진 skeleton code 에서 적절하게 code 를 디자인하여 Single-Cycle CPU 를 구현해보면서 Single-Cycle CPU 의 동작 방식을 완전히 이해하게 되었다. 또한 synchronous log 과 asynchronous logic 을 적절하게 사용하는 방법을 숙지하였으며, CPU 구현에 있어서 Modularization 의 중요성을 확인할 수 있었다.