CSED311 Computer Architecture

# Assignment 2 Report

**TEAM 28**

**20180154 전제민** jeminjeon@postech.ac.kr

**20180916 임경빈** kbini@postech.ac.kr

**2023/03/27**

## 1. Introduction

이번 Lab2을 수행하며 Single-cycle CPU를 RTL로 구현하였다. Single-cycle CPU는 메모리에 저장되어 있는 명령어를 한 클럭 사이클마다 하나씩 수행하는 형태로 설계되어 있는 CPU이다. 이번 보고서에서는 Single-cycle CPU의 논리 회로 설계를 다루고자 한다.

한 사이클에 하나의 명령어를 수행하는 특성으로 인해 Single-cycle CPU의 instruction per cycle 즉 CPI는 1이다. Cycle time이 가장 긴 시간 동작하는 명령어에 의해 결정되므로 성능이 다른 CPU에 비해 낮다고 볼 수 있다.

## 2. Design

Single-cycle CPU의 설계를 모듈별로 나누어 설명하고자 한다.

- **cpu** : cpu의 구성 모듈들을 연결하는 상위 모듈이다. top 모듈에서 들어온 클럭 입력, 리셋 입력, 중지 입력을 받아 들이고 메모리에 저장되어 있는 정해진 순서의 명령어가 순차적으로 수행될 수 있도록 한다. CPU가 수행해야 할 명령어는 opcode.v 파일에 명시되어 있다.
  - clk 입력을 받아 positive edge마다 한 사이클씩 올라갈 수 있도록 한다.
  - reset 입력이 들어오면 CPU의 state를 초기화시킨다.
  - Is_halt 입력이 들어오면 GPR[x17]의 값을 확인한 후 10과 같으면 시뮬레이션이 종료되도록 한다. 그렇지 않으면 NOP과 같은 역할을 수행하도록 한다.

- **ControlUnit** : fetch된 instruction을 받아오면 이를 비트 단위로 나누어 분석하여 어떤 타입의 명령어인지를 파악하고 이에 따라 적절하게 PC, ALU, Immediate Generator, Register, Memory의 실행 전반을 통제한다. 각 모듈에게 적절한 수행을 통제하기 위해 여러가지 플래그를 사용하고자 계획하였다.
  - **is_jal, is_jalr** : 명령어가 J타입임을 표현하는 플래그이다.
  - **branch** : 명령어가 B타입임을 표현하는 플래그이다.
  - **mem_read** : 메모리로부터 데이터를 읽어야함을 표현하는 플래그이다.
  - **mem_to_reg** : 메모리로부터 읽어온 데이터를 특정 레지스터에 write해야 함을 표현하는 플래그이다.
  - **mem_write** : 메모리에 특정한 데이터를 write해야 함을 표현하는 플래그이다.
  - **write_enable** : 레지스터에 value가 쓰여져야 하는지 표현하는 플래그이다.
  - **pc_to_reg** : 계산한 PC 값이 레지스터에 write되어야 함을 표현하는 플래그이다.
  - **is_ecall** : ecall 입력이 들어왔는지를 나타내는 플래그이다.
- **ALUcontrol** : ALUcontrol 모듈은 fetch된 instruction을 통해 ALU가 어떤 연산을 해야 하는지, 피연산자를 어디서 가져와야 하는지를 통제하는 역할을 한다. 또한 B-type 명령어가 들어왔을 때 명령어의 종류에 따라 연산을 하여 branch condition, 즉 bcond 변수를 set할지 여부를 결정하게 된다. 이때 연산자의 종류를 구분하도록 하는 ALU operation code는 opcodes.v 파일에 매크로로 선언하였다.
- **ALU** : ALU는 산술 논리 장치로서 주어진 두 수의 산술 연산과 AND, OR, XOR 같은 논리 연산을 수행하는 역할을 한다. rs1에서 가져온 데이터와 Control Unit에 따라 정해진 rs2 데이터 또는 Immediate에서 가져온 값을 ALUcontrol에서 결정한 연산자로 연산을 하게 된다. 만약 수행해야 할 명령어가 B-type이라면 rs1과 rs2의 값을 비교해서 branch가 taken 되어야 할 경우 이를 의미하는 bcond 플래그를 set하여 cpu 모듈에 전달한다.
- **ImmediateGenerator** : fetch된 instruction을 받아서 만약 Immediate를 사용해야 하는 type의 명령어일 경우 각 타입에 맞게 적절한 immediate를 생성하여 ALU에 전달하는 역할을 하는 모듈이다.
- **Memory** : 주소 또는 write할 데이터를 input으로 받고 메모리에서 가져온 데이터를 output으로 내보내는, 메모리에 접근하는 모듈이다. 접근할 메모리의 주소를 입력 받으면 {2'b00, addr >> 2}의 형태로 byte align을 하여 메모리에 접근한다. 데이터를 읽어야 할

경우에는 Asynchronous하게 데이터를 읽지만 데이터를 써야할 경우에는 Clk 시그널에 Synchronous하게 write하여야 다음에 해당 메모리에서 데이터를 가져올 때 최신 상태를 유지할 수 있다. ControlUnit에서 가져온 Mem_read 또는 Mem_write 플래그에 맞게 데이터를 적절하게 가져오거나 작성한다.

- **Register** : 레지스터에 접근하여 데이터를 쓰거나 읽는 역할을 하는 모듈이다. 데이터를 읽을 경우 Asynchronous하게 rf[rs1] 또는 rf[rs2] 또는 rf[17]에 접근하여 데이터를 가져온다. 데이터를 써야 할 경우 동시성을 보장하기 위해 clk 시그널에 synchronous하게 작성해야 할 데이터를 적절한 레지스터에 write한다. 만약 reset 입력이 있다면 모든 레지스터를 초기화하고 스택 포인터에 해당하는 rf[2]를 0x00002ffc로 리셋한다.

- **PC** : clk 시그널에 synchronous하게 Program counter, 즉 PC를 업데이트한다. 만약 reset 신호가 들어오면 current pc를 0으로 초기화하고, 그렇지 않을 경우 계산된 next_pc를 current_pc로 업데이트한다.

## 3. Implementation

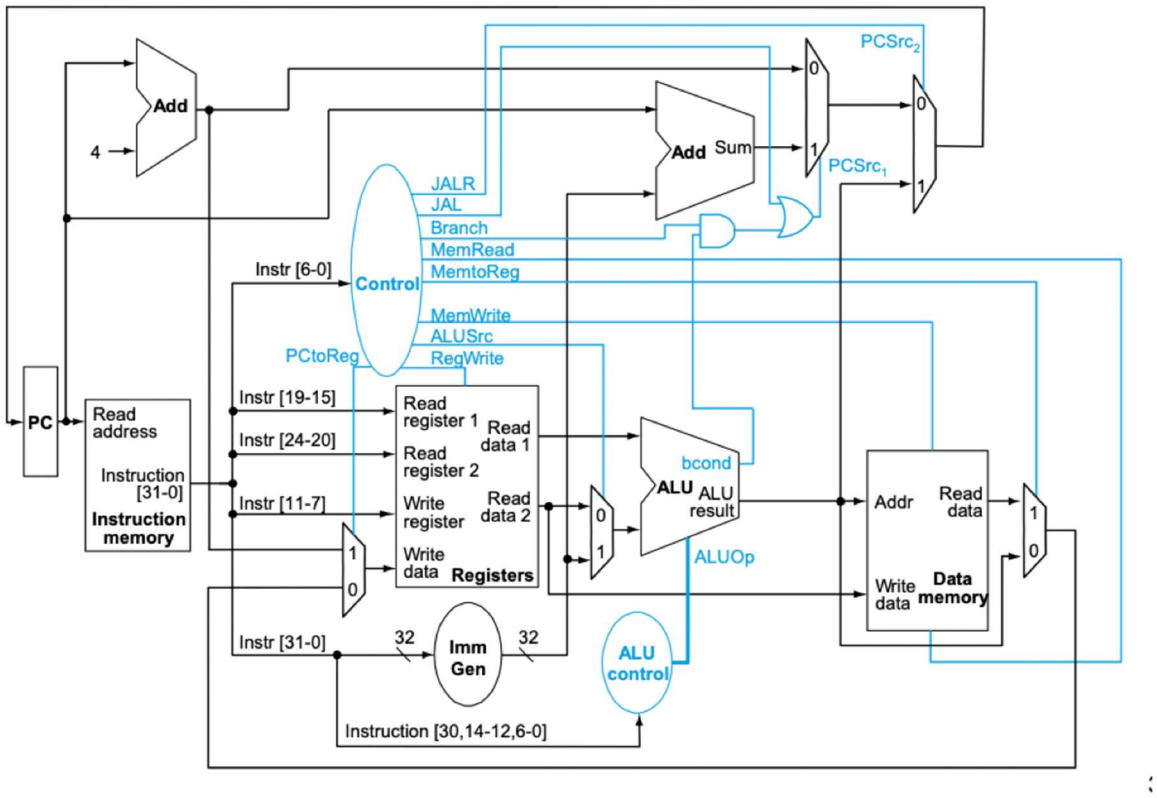


**그림 1 single cycle cpu**

- **CPU**

우리가 구현한 single cycle cpu의 Top module 이다. 하위 모듈들인 PC모듈, InstMemory 모듈, RegisterFile 모듈, ControlUnit 모듈, ImmediateGenerator모듈, ALUControlUnit 모듈, ALU 모듈, DataMemory 모듈의 연결을 여기서 구현하였다.

Reset signal, clock signal와 시뮬레이션을 끝낼지 여부를 나타내는 is_halted를 input으로 받는다. Instruction Fetch Stage는 Instruction Memory 모듈에서 일어나게 구현하였다. Instruction Decode and operand fetch Stage 는 Fetch된 Instruction을 ControlUnit에 넣어주고 opcode를 통해 구분된 type에 따라 RegisterFile모듈에 rs1, rs2, rd에 해당하는 instruction부분을 넣어주게 구현하였다. ALU/Execution Stage는 fetch된 operand를 ALU Control Unit에 alu 입력값 2 개를 전달하도록 구현하였다. Memory Access Stage는 LOAD, STORE 인스트럭션인 경우에 alu_result를 DataMemory 모듈에 주소로 전달하고 STORE 인스트럭션의 경우에는 din(저장해야하는 값)을 전달해 Memory read and Write 과정이 일어나도록 구현하였다. WriteBack Stage는 rd에 저장해야 할 값(LOAD인스트럭션의 경우에는 memory에서 읽은 값, JAL, JALR인스트럭션은 PC+4, ADD, ADDI 같은 ARITHMETIC, ARITHMETIC_IMM 인스트럭션의 경우에는 alu결과값을 전달해 레지스터에 write 하도록 구현하였다. 그리고 마지막으로 next_pc에 값을 업데이트하여 PC모듈로 전달해 PC가 업데이트되도록 구현하였다. 모든 과정은 clk asynchronous하게 이루어지게 구현하였다.

- **Register File**

레지스터 파일을 읽고 레지스터 파일에 데이터를 쓰는 모듈이다. Input으로 clk, rs1 register(번호), rs2 register(번호), rd register(번호), rd에 들어가야 할 input datadls rd_din, control_unit에서 레지스터 파일에 write 해야 하는지 판단하고 write해야 하면 1, 아니면 0이 output으로 내보내지는 write_enable을 가진다. 그리고 rs1 레지스터에 저장된 값을 rs1_dout, rs2 레지스터에 저장된 값을 rs2_dout에 저장해 내보내도록 구현하였다. 이때 rs1, rs2 레지스터 파일에 저장된 값을 rs1_dout, rs2_dout에 읽어오는 과정은 clk asynchronous 하게 진행되도록 구현하였다. 그리고 17번 레지스터에 값을 ecall에 저장하여 output으로 내보내도록 구현하였다. Rd에 data를 쓸 때는 clk이 rising edge 일 때 쓰도록 즉, clk synchronous 하게 진행되도록 구현하였고 register file을 initialize 할 때도 clk이 rising edge 일 때 레지스터 값을 2번을 제외한 0~31번 레지스터는 32'b0, 2번 레지스터는 32'h2ffc로 초기화하도록 구현하였다. 이때 write data 과정은 clk synchronous 하게 일어나도록 구현하였다.

- **Inst Memory**

Inst Memory 모듈은 Instruction Memory에서 인스트럭션을 읽는 모듈이다. Reset, clk, instruction

memory의 주소인 addr을 입력값으로 받고 Instruction Memory에서 주솟값으로 인스트럭션을 읽는다. 그리고 dout에 저장해서 내보내도록 구현하였다. Instruction Memory의 초기화는 reset이 1이 들어왔을 때 Instruction Memory를 전부 0으로 초기화하고 binary format의 instruction들로 이루어진 텍스트 파일(테스트 벤치 파일)을 Instruction 메모리에 저장하도록 구현하였다. Memory에 Write 하는 과정은 clk이 rising edge 일 때만 일어나도록(clk synchronous 하게) 구현하였다.

- **Data Memory**

Data Memory 모듈은 메모리에 데이터를 쓰고 읽는 기능을 하도록 구현하였다. Reset, clk, 데이터 메모리의 주소인 addr, 쓰여질 데이터인 din, 메모리에서 데이터를 읽을지 여부를 나타내는 mem_read, 메모리에 데이터를 쓸지 여부를 나타내는 mem_write를 입력값으로 받는다. Mem_read, mem_write는 Control Unit 모듈에서 나와서 Data Memory에 입력값으로 들어오도록 구현하였다. Reset이 0이 아닐때에 mem_read가 1인지 판단하고 1인 경우에 주소값에 해당하는 메모리에 저장된 값을 읽어와 dout에 저장하여 내보내도록 구현하였다. 메모리에서 데이터를 읽는 과정은 clk asynchronous하게 일어나도록 구현하였다. 메모리를 초기화 하는 과정은 reset이 1인경우에 메모리 값을 전부 0으로 초기화하고 아니면 din을 메모리에 write 하도록 구현하였다. 이 과정은 clk이 rising edge 일때만 일어나도록(clk synchronous하게) 구현하였다.

- **PC**

PC 모듈이다. 계산된 다음 PC 값인 next_pc, reset, clk을 input으로 가진다. Clk이 rising edge일 때마다 reset이 1이면 current_pc를 0으로 initialize하고 reset이 0이 아니라면 input으로 받은 계산된 next PC value 를 current_pc에 넣고 output으로 내보낸다. 이 과정은 clk이 rising edge일때만 일어나도록(clk synchronous하게) 구현하였다.

- **Immediate Generator**

인스트럭션을 통해 immediate value를 계산하는 모듈이다. 인스트럭션을 input으로 받고 이 인스트럭션을 통해 계산된 immediate value인 imm_gen_out를 output으로 가진다. 먼저 Opcode 부분에 해당하는 part_of_inst[6:0]으로 타입을 구분한다. 그리고 각 타입에 맞는 immediate value 위치를 인스트럭션에서 찾아 immediate value를 계산한 뒤 sign extension하여 imm_gen_out으로 내보낸다. 모든 과정은 clk asynchronous하게 이루어지게 구현하였다.

- **Control Unit**

ControlUnit은 인스트럭션을 입력받아 이 인스트럭션의 opcode 부분을 통해 이 인스트럭션이 특정 인스트럭션(예를 들어 B-type이거나 jal, jalr 인스트럭션) 인지 구분 하고 레지스터에 write 해야하는지, PC 값을 reg에 저장해야하는지, memory에서 read 하는지, alu 의 두번째 input으로 immediate value를 써야하는지 등을 제어하는 기능을 수행하도록 구현하였다. Is_jal은 이 인스트럭션이 JAL인스트럭션이면 1, 아니면 0 값을 가지도록 구현하였고 output으로 내보내진다. Is_jalr은 이 인스트럭션이 JALR인스트럭션이면 1, 아니면 0 값을 가지도록 구현하였고 output으로 내보내진다. Branch 는 B-type인스트럭션이라면 1, 아니면 0 값을 가지도록 구현하였고 output으로 내보내진다. Mem_read는 메모리에서 읽어와야하는 상황(LOAD 인스트럭션 인 경우)이면 1, 아니면 0 값을 가지도록 구현하였고 output으로 내보내진다. Mem_to_reg는 메모리의 값을 레지스터의 값에 저장해야 할 때(LOAD 인스트럭션인 경우)에 1, 아닌 경우에는 0을 가지도록 구현하였고 output으로 내보내진다. Mem_wirte는 메모리에 값을 저장해야 할 때(STORE인스트럭션인 경우)에 1, 그 외 다른 경우에는 0을 가지고 output으로 내보내지게 구현하였다. Alu_src는 immediate value 가 ALU에 두번째 입력값으로 들어가야 할 때(LOAD, STORE, ARITHMETIC_IMM의 인스트럭션)에 1, 아닌 경우에 0을 가지고 output으로 내보내지도록 구현하였다. Write_enable은 rd 에 값을 저장해야 할 때(ARITHMETIC, ARITHMETIC_IMM, LOAD, JALR, JAL)에 1, 아닌 경우에 0을 가지고 output으로 내보내지도록 구현하였다. Pc_to_reg는 (PC+4)값이 계산된 뒤 rd에 저장되어야 하는 경우(JAL, JALR) 에 1, 그 외에 0을 가지고 output으로 내보내지도록 구현하였다. Is_ecall은 opcode부분이 ECALL일때 1, 아닌 경우에 0을 가지고 output으로 내보내지도록 구현하였다. 모든 과정은 clk asynchronous하게 이루어지게 구현하였다.

- **ALU Control Unit**

어떤 연산을 해야 할 지 결정하는 모듈이다. Func7, funt3, opcode, 전체 instruction, 을 입력받아 어떤 연산을 해야하는지 구분 한 뒤 alu_op_out으로 내보낸다. 먼저 opcode를 통해 어떤 타입의 인스트럭션인지 구분하고 그 뒤 funct3를 통해 어떤 instruction인지 세부적으로 구분한다. 예를 들어 opcode를 통해 B-type 임을 판단 한 뒤에 funct3를 통해 BEQ, BNE, BLT 등을 구분하는 방식으로 구현하였다. ADD, SUB 의 구분은 funct7을 통해 구분하도록 구현하였다. 여기서 출력한 alu_op_out은 ALU 모듈로 input으로 들어가게 구현하였다. 모든 과정은 clk asynchronous하게 이루어지게 구현하였다.

- **ALU**

ALUControlUnit에서 어떤 연산을 해야되는지 나타내는 alu_op_out을 입력으로 받고 이를 통해 무슨 연산을 해야하는지 판단 한 후 입력받은 두 register value인 rs1(input1), rs2,(input2)으로 연산을 수행한다. 만약 Arithmetic 연산을 수행해야 한다면 input1, input2로 연산을 수행하고 연산

결과를 output reg [31:0] result_out으로 내보내고 b_cond(branch condition이 True인지 False인지)를 0으로 내보낸다. 만약 연산이 branch condition을 확인하기 위한 연산(BEQ, BNE 등 을 위한 연산) 이라면 branch conditon이 True일 때 b_cond를 1로 내보내고 아니면 b_cond를 0으로 내보내도록 구현하였다. 모든 과정은 clk asynchronous하게 이루어지게 구현하였다.

## 4. Discussion

- **Simulation을 통한 Debugging의 중요성** : 처음 구현을 하고 시뮬레이션을 실행하였을 때 기대와는 다른 레지스터 결과 값과 차이를 보였을 때 어떤 지점에서 문제가 생겼는지 알기 어려웠다. 때문에 Simulation을 실행하는 과정의 Wave 출력 결과를 보면서 차근차근 따라가며 어느 부분이 문제를 일으켰는지 찾아냈다. 이처럼 RTL 디자인은 하드웨어 레벨에서 동작하기 때문에 시뮬레이션을 통한 디버깅이 중요하다는 것을 깨닫게 되었다.
- **Asynchronous한 data reading과 Synchronous한 data writing** : postedge clk이라는 동시적인 시각에 메모리 또는 레지스터에 데이터 write가 이루어져야 데이터를 읽어올 때 같은 로케이션에서 같은 데이터를 일관되게 읽어올 수 있어 PVS를 여러 곳에서 동시에 접근할 수 있게 된다.

## 5. Conclusion

이번 Lab에서는 Verilog를 이용해 Single-Cycle CPU을 구현하였다. CPU의 디자인은 이전 Vending Machine에 비해 상당히 복잡해져서 늘어난 수의 모듈끼리 상호작용을 이해하고 디자인에 반영하는 것이 꽤 까다로웠다. 구현 과정에서 모듈들의 역할과 기능을 잘 파악하고 시뮬레이션 결과를 통해 디버깅을 하는 과정에서 RTL에 대한 스킬이 더욱 늘었다고 생각한다.

Single-cycle CPU를 디자인하면서 한 클럭 사이클에 하나의 instruction만을 수행하기 때문에 성능이 좋지 못하다는 점을 알게 되어 멀티 사이클 CPU나 파이프라인 CPU의 필요성을 몸소 체감하게 되었다. 이후 Lab에서도 이러한 점들을 고려하여 구현할 계획이다.