**Lab2 – Single-Cycle CPU**

1. **Introduction**

Lab2의 목표는 Single-Cycle RISC-V CPU(RV23I)를 구현하는 것이다. 이를 통해 우리는 single-cycle CPU의 구조와 작동 방식을 이해할 수 있다.

우리가 구현할 Single-Cycle CPU는 두 가지 부분으로 구성된다. 하나는 Datapath를 담당하는 부분으로 이는 ALU와 Register file로 이루어진다. 그리고 다른 하나는 Control unit으로 이는 control signals를 사용해 datapath를 생성한다.

또한 우리 구현하는CPU는 RISC-V로 우리는 제공 받는 opcode.v file에 존재하는 RV32I instruction들에 대해 작동하는 CPU이며, machine를 정지하는 것은 아래와 같이 프로그램의 마지막에 ECALL instruction을 사용하면 정지하도록 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **Design**

CPU의 전체적인 구조는 아래와 동일하게 디자인했다.

도표, 개략도이(가) 표시된 사진

자동 생성된 설명

그리고 디자인을 할 때에 CPU가 작동하는 한 cycle의 전체적인 흐름은 아래와 같이 생각하고 구현했다.

PC에서 instruction의 address를 보내준다. InstMemory에서 해당 address의 instruction을 RegisterFile, ControlUnit, ImmediateGenerator, ALUControlUnit으로 보내준다. RegisterFile에서는 받은 instruction을 ALU에 보내주는 data로 write한다. ControlUnit에서는 받은 instruction을 가지고 각 instruction에 맞는 control bit를 보내준다. ImmediateGenerator에서는 받은 instruction에서 어떤 instruction이냐에 따라 그에 맞는 immediate를 보내준다. ALUControlUnit는 어떤 instruction이냐에 따라 그에 맞는 alu\_op를 ALU에 보내준다. 그 다음 ALU는 alu\_op를 통해 연산의 종류를 결정해 RegisterFile로부터 받은 data를 연산한다. 그 다음 load나 store instruction의 경우 DataMemory에서 memory에 접근하여 이를 RegisterFile로 보내준다. 그리고 한편 동시에 PC의 경우, Control bit와 ALU의 cond값을 통해 instruction이 jump나 branch라면 해당 위치로, 그렇지 않다면 PC+4로 이동하도록 한다.

1. **Implementation**

우리는 Single-Cycle CPU를 구현할 때에 기본적으로 기능에 따라 modularize를 하여 구현했다.이때 사용된 module의 종류와 각 module별 기능, 구현 등의 내용은 아래와 같다.

1. **PC**

PC에 다음PC로 불러오는 기능을 한다. Clock에 asynchronous하다.

1. **Plus\_Four\_PC**

현재 PC에 4를 더해주는 기능으로, jump나 branch instruction 외의 다른 instruction들을 수행하는 일반적인 경우에 사용된다. Clock에 synchronous하다.

1. **jumpPC**

현재 PC에 immediate를 더해주는 기능으로, jump나 branch instruction을 수행할 때에 사용된다. Clock에 synchronous하다.

1. **PC\_MUX**

다음 PC로 넘어갈 때에 Plus\_Four\_PC로 넘어갈지 jumpPC로 넘어갈지 결정하는 MUX이다. 선택 기준은 ControlUnit과 ALU의 output을 input으로 받아 이를 기준으로 결정한다. Clock에 synchronous하다.

1. **InstMemory**

Instruction memory를 가지고 있는 module이다. address를 통해 memory에 access한다. Clock에 asynchronous하다.

1. **ALU\_Write\_MUX**

Register file로 Write-back할 데이터를 결정하는 MUX이다. pc\_to\_reg 값을 기준으로 결정하며 jump instruction일 때는 next\_pc를, 그 외의 instruction의 경우 dout을 사용한다. Clock에 synchronous하다.

1. **RegisterFile**

Register file을 읽어오고 데이터를 register file에 쓰는 기능을 한다. Clock에 synchronous하게 write한다.

1. **ControlUnit**

input으로 opcode를 받아 opcode에 따라 output하는 control bit를 결정하는 식으로 control 기능을 수행한다. Clock에 synchronous하다.

1. **ImmediateGenerator**

input으로 전체 instruction을 받고, 이것을 opcode에 따라 각 opcode에 필요한 형태로 output에 보내는 기능을 한다. Clock에 synchronous하다.

1. **ALUControlUnit**

input으로 들어온 opcode, funct3, funct7의 값에 따라 output의 alu\_op를 결정하여, ALU가 어떻게 작동할지 control한다. Clock에 synchronous하다.

1. **ALU\_MUX**

ALU의 input data를 결정하는 MUX이다. RegisterFile에서 output으로 나온 rs2\_out, ImmediateGenerator에서 output으로 나온 imm\_gen\_out 중 하나를 선택한다. Clock에 synchronous하다.

1. **ALU**

연산 기능을 수행한다. alu\_op 값에 따라 어떤 연산을 할지 결정한다. Clock에 Synchronous하다.

1. **DataMemory**

memory로부터 data를 read하고, 다시 memory에 data를 write하는 기능을 한다. Clock에 synchronous하게 write한다.

1. **Memory\_MUX**

write-back할 memory를 결정하는 MUX이다. ALU의 output과 DataMemory의 output 중 하나를 결정한다. Clock에 synchronous하다.

1. **Discussion and Conclusion**

기본적으로 수업 PPT에 있는 Single-Cycle CPU의 구조를 따라 디자인하고 구현했다. 이때 조금 다른 점은 다음 PC를 정하는 과정에서 ADD module과 AND, OR gate를 사용하지 않고 Plus\_Four\_PC와 jumpPC라는 module을 새로 만들어 각 module에서 PC가 이동할 다음 위치를 연산했다. 그리고 PC\_MUX를 통해 if문을 활용하여 Control bit에 따라 Plus\_Four\_PC로 이동할지, jumpPC로 이동할지 결정했다.

이번 Lab2을 통해 Single-Cycle CPU의 구조와 instruction 별로 각 stage에서 어떻게 작동하는지 이해할 수 있게 되었다.