2025 Spring

CSED311

컴퓨터구조

Lab 3 report

Team ID: 15

팀원 1: 20230345 이성재

팀원 2: 20230355 정지성

목차

1. 서론
2. 디자인
3. 구현
4. 논의 사항
5. 결론
6. **서론**

지난번에 single cycle cpu를 구현한데 이어 이번 과제에선 multi cycle cpu를 구현하는 것이 목표였다. 가장 큰 차이점이라면 FSM을 설계해서 각 명령어가 클럭에 맞춰 state을 이동하는 방식으로 구현해야 한다는 점이다. Single cycle cpu보다 짧은 클럭시간을 이용할 수 있고, 필요한 만큼만 시간을 할당할 수 있기 때문에 더 효율적이다.

1. **디자인**

CPU 디자인은 수업자료를 참고했다.

도표, 평면도, 기술 도면, 개략도이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Ch6, 14p 에 나와있는 그림으로, 필요한 모듈들과 연결되어야 할 control value를 알 수 있다. 또 다른 중요 포인트는 FSM을 어떻게 구현할지에 대한 것인데 이는 아래의 그림대로 구현하였다.

그림, 스케치, 라인 아트, 클립아트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**!!! 여기 그림에는 ID\_ECALL 이 없는데, 추가해서 넣어야 할 듯 !!!**

특히 ID 단계는 branch의 경우 다른 컨트롤이 필요하기 때문에 별도의 state이 필요했다.

다음으로 주요 모듈들의 입력, 출력 값과 모듈에 대한 설명을 하도록 하겠다.

1. **ALUControlUnit**

Input

* **functs** (instruction[30], funct3 으로 총 4비트)
* **opcode** (I-type의 연산을 처리하기 위해 추가했다)
* **ALUOp**

Output: **ALUControl**

* Control unit으로부터 ALUOp를 받고 3가지 케이스에 대하여 ALU 모듈이 실행해야 할 연산의 종류를 functs와 opcode를 이용해 계산하는 모듈이다.

1. **ALU**

Input

* **ALUControl**
* **alu\_in\_1**
* **alu\_in\_2**

Output: **alu\_result, alu\_bcond**

* ALUControl 값에 따라 적절한 연산 결과를 alu\_result로 출력하고 branch인 경우 alu\_bcond도 적절하게 저장한다.

1. **구현**
2. **ALUControl**

먼저 ALUOp를 기준으로 결과가 달라지기 때문에 case 문을 사용해줬다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

위의 두 경우는 각각 ALUOp가 00과 01일때를 다룬다. 특히 00인 경우에는 addition 연산으로 결정되기 때문에 추가적인 로직 없이 ALUControl 값을 정할 수 있었다. 01인 경우에는 funct3값에 따라 바꿔줘야 하는 branch문이다. 지난 랩과 마찬가지로, ALU가 해야할 연산 자체는 SUB이지만 alu\_bcond를 어떤 조건에 1로 설정할지는 달라지기 때문에 funct3, 즉 functs[2:0]의 값에 따라 다른 ALUControl 값을 갖도록 했다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

다음으로 ALUOp가 10인 경우엔 branch가 아니면서, funct3에 따라 해야할 연산의 종류가 달라진다. 따라서 funct3값으로 case문을 만들어 처리했다. 특히, funct7의 5번째 비트(instruction[30]) 값이 1로 설정되면 ADD는 SUB으로, LRS는 ARS로 바꿔줘야 하기 때문에 각 경우에 조건문을 달아주었다. 특이한 부분은 opcode가 I-type을 의미하는지 확인한다는 점인데, 이 부분이 없으면 addi 등의 인자로 음수가 온 경우 잘못 실행되기 때문에 추가해주었다. 자세한 사항은 논의사항에서 다루도록 하겠다.

**ALU**

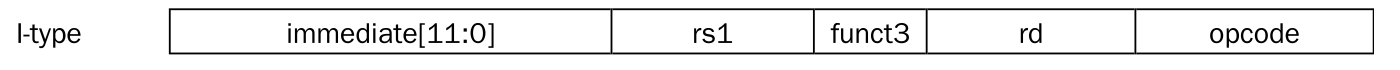
ALU 모듈은 single cycle cpu를 구현할 때와 크게 다르지 않았다. 입력의 이름을 opcode에서 ALUControl로 바꾼 것과 bcond를 alu\_bcond로 바꾼 정도가 전부이다. ALUControl이 어떤 값을 갖는지에 따라 다른 연산을 실행하도록 case문을 작성했고, 코드는 다음과 같다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. **논의 사항**

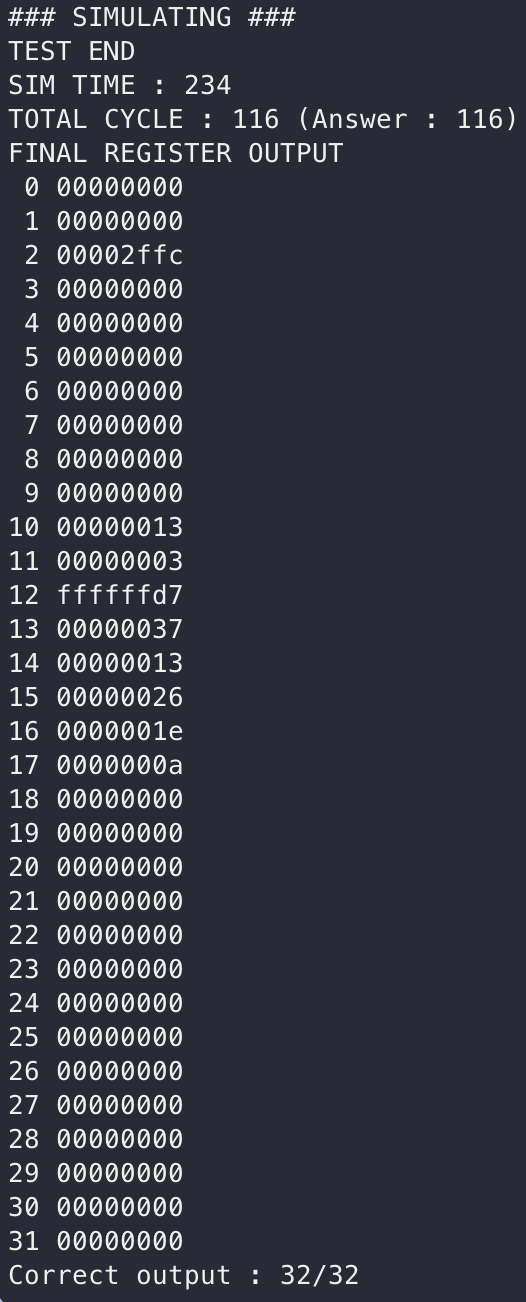
ALUControl에서 ALUOp가 10인 경우 opcode로 명령어가 I-type인지 확인해야 하는 이유를 설명하겠다.



I-type과 R-type의 가장 큰 차이점은 funct7이 없다는 것인데, 이 모듈에서 functs는 instruction[30]을 MSB로 갖고 있다. 여기서 문제가 되는 부분은 imm 부분에 음수가 오는 경우이다. -32 같은 수는 2의 보수로 1111 1110 0000 이라는 값을 갖는데 이때 instruction[30] = 1이 되어버린다. 하지만 이는 R-type에서의 경우와는 달리 sub을 의미하는 것이 아니다. opcode가 I-type인지 확인하는 저 로직이 없을 때는 addi x2, x2, -32 와 같은 명령이 x2에 -32를 빼는 명령이 되었었다(결과적으로 32가 더해짐). 따라서 I-type인 경우 무조건 ADD가 실행되도록 수정이 필요했다.

1. **결론**

5개의 Test bench를 실행해본 결과는 아래와 같다.

 텍스트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 책, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

최종적으로 정확한 레지스터 값과 정답과 같거나 작은 횟수의 cycle을 사용했다는 것을 근거로 multi cycle cpu를 구현하는 데 성공했다고 볼 수 있겠다.

수업 시간에 들은 설명만으로는 multi cycle cpu의 작동 방식을 이해하기 힘들었다. 특히 FSM과 관련된 부분을 설명하실 때는 따라갈 수가 없었던 기억이 있다.