2025 Spring

CSED311

컴퓨터구조

Lab 2 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**

이 과제에서 우리는 Single cycle cpu를 베릴로그로 구현했다. CPU를 구성하는 기본 모듈들을 설계하고 구현하였으며 주어진 테스트를 이용해 확인한 결과 RIPES 시뮬레이터 상의 레지스터 값들과 동일한 레지스터 값을 가졌다. 이후 과제들을 통해 더 최적화 된 형태의 cpu를 개발하게 될텐데, 이번 과제를 통해 기본적인 cpu 구조를 명확히 이해함으로써 이후 더 개선된 구조의 cpu를 구현하는데 있어 어려움이 없도록 하는 것이 목표라고 할 수 있다.

1. **디자인**

이 과제에서 기본적인 모듈은 강의 슬라이드 CH5의 34페이지를 참고했다.

텍스트, 도표, 평면도, 개략도이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

각 모듈의 목록은 아래와 같다.

* pc
* calculate\_next\_pc
* control\_unit
* parse\_instructions
* instruction\_memory
* immediate\_generator
* register\_file
* alu\_control\_unit
* alu
* data\_memory

아래는 각 모듈의 디자인에 대한 설명이다.

1. **pc**

Input

Output:

1. **calculate\_next\_pc**

Input

Output:

1. **control\_unit**

Input

Output:

1. **parse\_instructions**

Input

Output:

1. **instruction\_memory**

Input

* **reset** (Instruction memory를 모두 0으로 재설정 하기 위한 신호)
* **clk** (cpu의 clock 신호)
* **addr** (명령어의 주소)

Output: **dout** (Instruction memory의 addr 주소에 저장된 명령어 반환)

* 현재 PC가 가리키고 있는 명령어가 어떤 명령어에 해당하는지를 Instruction memory에 대한 참조로 구한 후 반환하는 모듈이다.
* 이 모듈은 clock에 동기화 되지 않는다.

1. **data\_memory**

Input

* **reset** (data memory를 모두 0으로 재설정 하기 위한 신호)
* **clk** (cpu의 clock 신호)
* **addr** (명령어의 주소)
* **din** (write을 할 경우 저장될 데이터
* **mem\_read, mem\_write** (read 또는 write을 할지 말지 결정하는 신호)

Output: **dout** (data memory의 addr 주소에 저장된 데이터 반환)

* 메모리에 접근해 저장된 데이터를 레지스터에 저장하거나, 레지스터(din)에 저장된 값을 메모리에 저장하는 모듈이다.
* read 과정은 clock에 동기화 되지 않으며, write 과정만 clock에 동기화 된다.

1. **register\_file**

Input

* **reset** (register file을 모두 0으로 재설정 하기 위한 신호)
* **clk** (cpu의 clock 신호)
* **rs1, rs2, rd** (source1, source2, destination에 해당하는 레지스터)
* **rd\_din** (write을 할 경우 레지스터rd 에 저장될 데이터)
* write\_enable (1로 설정될 경우 write 처리)

Output: **ecall\_reg\_cond, rs1\_dout, rs2\_dout, print\_reg**

(control unit에 넘길 cond 신호와 출력할 레지스터 값들)

* read 과정은 clock에 동기화 되지 않으며, write 과정만 clock에 동기화 된다.

1. **alu\_control\_unit**

Input

* **funct3**
* **funct7**
* **opcode**

Output: **alu\_op** (ALU가 실행해야 할 연산의 종류를 알려주는 값)

* 이 모듈은 alu 모듈이 정확히 어떤 연산(e.g., ADD, LLS, etc)을 해야 할지 정해서 alu에게 알려주는 역할을 수행한다. 연산의 종류는 funct3, funct7, opcode를 알아야 결정할 수 있으므로 입력을 위와 같이 받고 4비트로 이루어진 alu\_op를 출력한다. alu\_op에 대한 규칙은 alu\_func.v에 정의되어 있다.
* clock에 동기화 되지 않은 채 동작한다.

1. **alu**

Input

* **alu\_op** (alu가 수행해야 할 연산의 종류)
* **alu\_in\_1, alu\_in\_2** (alu 연산의 피연산자에 해당하는 레지스터 값들)

Output: **alu\_result, alu\_bcond** (연산 결과와 branch condition을 만족시켰는지 여부)

* Lab1에서 제작했던 alu.v와 비슷한 구조로 작성되었지만 BEQ 등 branch를 할 수도 있는 연산을 처리할 경우 alu\_bcond의 값을 결정하는 로직이 추가되었다.
* clock에 동기화 되지 않은 채 동작한다.

1. **immediate\_generator**

Input

Output:

1. **구현**

**\*\*control\_unit 설명추가\*\***

Single cycle cpu의 5가지 단계를 거치며 어느 모듈이 사용되는지 설명하는 방식으로 구현 방법을 설명하도록 하겠다.

도표, 라인, 평면도, 직사각형이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. **IF**

Instruction fetch 단계에서 cpu는 pc가 가리키는 값이 어떤 명령어를 의미하는지 판단한다. pc가 어떤 값을 갖는지는 pc.v에서 다음과 같은 모듈로 구현되었다**. \*\*pc.v, calculate\_next\_pc.v 설명 추가\*\***

이후 instruction\_memory 모듈로 주소를 넘기면 Instruction memory에 저장된 명령어를 다음 stage로 전달한다. 이때 reset이 1로 설정되지 않은 경우는 아래와 같은 코드로 간단히 구현할 수 있다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

reset이 1로 설정된 경우 초기화를 진행하고 이는 주어진 코드에서 모두 구현이 되어 있으므로 설명은 생략한다.

1. **ID**

Instruction decode 단계에서는 register\_file 모듈이 사용된다. 이 모듈은 레지스터의 값을 읽거나 쓰는 작업을 수행해야 하는데, write 과정은 **WB** 단계에서 일어나므로 뒤에서 설명하겠다. read 과정은 주어진 rf 레지스터를 참조하는 것으로 구현할 수 있다. 또한 ecall\_reg\_cond의 경우 **\*\*이거 설명 부탁..\*\*** 따라서 코드는 아래와 같다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

또한 I-type 연산에 사용되는 즉시 값을 만들어내는 immediate\_generator 모듈도 사용된다. **\*\*설명 추가\*\***

1. **EX**

Execute 단계에선 실제 연산을 수행해야 한다. 먼저 alu\_control\_unit 이 alu가 처리할 연산의 종류를 구분해준다. 가장 먼저 opcode를 이용해 연산의 종류가 arithmetic, arithmetic\_imm, load,store, jalr, jal, branch인지 구분한다. 다음으로 funct3의 값을 이용해서 연산의 종류를 결정짓고, 몇가지 경우에선 funct7의 값까지 이용해야 연산의 종류를 결정지을 수 있으므로 삼항 연산자를 이용해 구분을 지어줬다. 코드는 아래와 같다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

이때 R-type 연산의 경우 ADD와 SUB을 funct3만으로는 구분할 수 없어 funct7이 무슨 값인지를 확인한다. 비슷하게 SRL과 SRA, 그리고 I-type의 SRL과 SRA도 funct7 값을 알아야 하기 때문에 삼항 연산자를 이용했다.

LOAD, STORE, JALR, JAL의 경우 모두 alu가 ADD 연산을 하면 된다(점프도 결국 주소를 더하는 것이기 때문). 따라서 위 네 가지 경우 모두에서 FUNC\_ADD로 alu\_op를 설정해줬다.

BRANCH의 경우 alu는 모두 동일하게 SUB 연산을 한다고 볼 수도 있지만, SUB 결과에 따라 bcond를 어떻게 설정할지는 달라지기 때문에 각각 다른 function code를 할당해줬다. 이때 alu\_func에서 사용하지 않는 값들을 주석처리 하고 해당 값을 각각의 function code에 할당했다.

다음으로는 alu 모듈에서 실제 연산을 수행해야 한다. 이 모듈은 Lab1에서 작성한 alu 모듈에 약간의 수정을 가해 완성했다. 우선 사용하지 않는 명령들을 주석처리 했다. 이후 branch 명령어들에 대해 언제 bcond를 1로 설정할지에 대한 코드를 추가했다. 완성된 alu 코드는 아래와 같다.

텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. **MEM**

Memory 접근은 명령이 LOAD이거나 STORE인 경우에만 일어나며, 각각 메모리에 있는 값을 레지스터에 저장하거나 레지스터에 있는 값을 메모리에 저장하는 과정이 필요하다. 이는 data\_memory 모듈에서 처리하며, read가 실행되는 경우 dout의 값을 mem의 dmem\_addr번째 값으로 설정한다. 만약 mem\_read가 1로 설정되지 않았다면 0을 출력하도록 삼항 연산자를 이용했다(if문을 사용하지 않고 구현하기 위해서이다).



write이 실행되는 경우 din으로 입력받은 데이터를 mem의 dmem\_addr번째 위치에 저장한다. 코드는 다음과 같다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. **WB**

WB 단계는 레지스터의 값을 변경해야 할 경우 일어나며, register\_file 모듈에서 처리한다. 다시 말하지만 이 모듈은 read와 write을 처리하고 read과정은 ID단계에서 일어난다. WB에서 일어나는 write 과정은 input으로 들어온 write\_enable이 1로 설정된 경우에만 일어난다. 또한 write을 하려고 하는 레지스터가 x0, 즉 항상 0으로 유지되어야 하는 레지스터가 아닌 경우에만 write을 할 수 있다. 따라서 코드는 다음과 같다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**\*\*parse\_instructions는 어디에 설명해야 할지 모르겠음\*\***

1. **논의사항**

**\*\*뭘 써야 하지\*\***

1. **결론**

최종적으로 작성된 코드를 이용해 테스트를 해본 결과, 각 테스트에 대해서 아래와 같은 레지스터 값들을 얻었다.

텍스트, 스크린샷, 패브릭, 패턴이(가) 표시된 사진

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

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

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

<basic\_mem.txt> <loop\_mem.txt> <non-controlflow\_mem.txt>

Ripes simulator로 확인한 결과 세가지 테스트에서 모두 동일한 사이클 수와 레지스터 값을 얻을 수 있었다. 이를 통해 과제의 목표였던 single cylcle cpu를 정확히 구현하는 것에 성공했다고 결론지을 수 있겠다.

처음 single cycle cpu에 대한 수업을 들을 때는 이해하기 어려웠다. 어떻게 각 모듈들이 동작하는 것인지 파악하기 힘들었고 각 신호가 무슨 역할을 하는지도 한눈에 들어오지 않았다. 하지만 이 과제를 끝내고 나니 해당 내용들에 대한 이해도가 매우 올라간 느낌이다. 컴퓨터 구조에 대한 내용을 단순히 이해하는 것에서 그치지 않고 실제 작동하는 cpu를 구현해 보는 경험을 통해 HDL 능력까지 키울 수 있었던 매우 흥미로운 과제였다고 생각한다.

다음 단계는 자연히 multi cycle cpu를 구현하는 것일 것이다. Single cycle cpu는 구조상 매우 느릴 수밖에 없다. 앞으로 남은 과제를 통해 사람들이 어떻게 cpu를 발전시켰는지 알아가보고 싶다.