Computer Organization (001)

Lab01-ALU Report

전기정보공학부

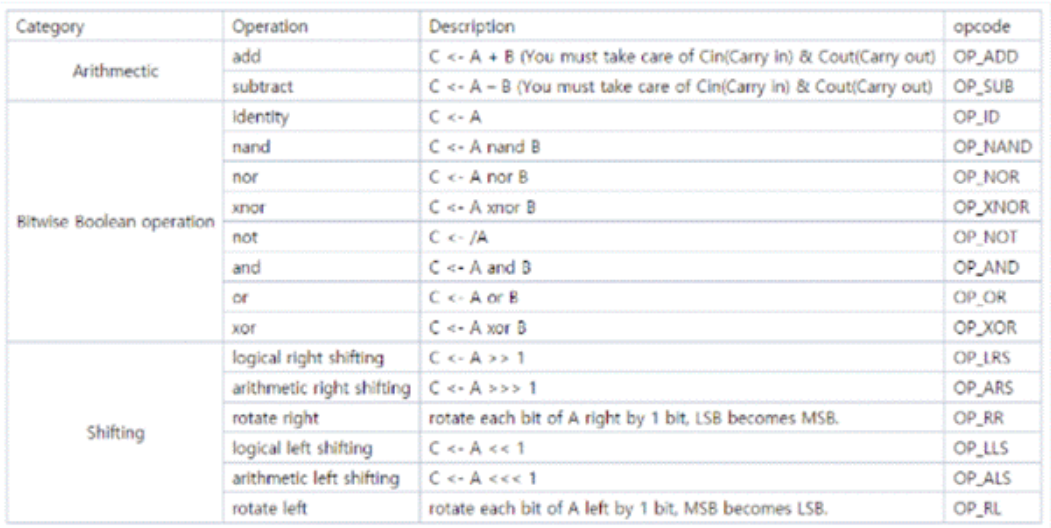
2017-17497김현규

1. Introduction

이번 Lab01 에서는 Cpu를 이루는 가장 중요한 요소 중 하나인 ALU를 Verilog 와 Vivado를 이용하여 구현한다. 최대 16개의 연산자를 지원하는 16-bit ALU를 구현하면서 cpu의 ALU에서 어떤 방식으로 명령어들이 처리하는가에 대해 알아본다. 또한 앞으로의 lab에도 지속적으로 사용될 Verilog와 Vivado등의 도구에도 익숙해지는 것이 본 lab의 목표이다.

1. Desing

이번 프로젝트에서는 16-bit의 ALU를 구현하며 아래와 같은 16개의 연산자를 지원한다.



ALU로 입력되는 값들은 16bit크기의 피연산자 2개(A, B), add와 substract의 경우에만 사용되는 1bit 크기의 Cin, 연산자를 나타내는 4bit의 OP, 출력되는 값들로는 결과를 나타내는 16bit 크기의 C와 carry out을 나타낼 1bit의 Cout이 존재한다.

우선 연산자에 따라서 다른 연산이 이루어지므로 Switch문을 이용하여 OP값에 따라서 다른 동작을 하도록 구현해야 한다. Switch문 내부에서는 코드의 가독성을 올리기 위해 `define을 통해 매크로 변수로 지정된 값을 사용한다. add연산자와 substract 연산자의 경우 연산 도중 16bit의 범위를 초과하는 일이 발생할 수 있어 17bit의 임시적인 reg를 만들어서 구현한다. Bitwise 연산자들의 경우 verilog에서 지원하는 기본적인bitwise 연산자들로 충분히 간단하게 구현할 수 있다. Shifting 중 logical shifting은 verilog의 <<와 >>연산자와 같은 동작을 하므로 이러한 연산자를 사용하면 되고, left arithmetic shifting 은 logical arithmetic shifting과 같으며, right의 경우에는 MSB가 같은 값으로 유지되도록 구현한다. Rotate left와 right은 각각 MSB와 LSB의 값을shifting 이후에 더해주도록 구현한다.

1. Implementation

이번 lab에서 구현한ALU는 별도의 submodule없이 구현하였습니다. 반복되는 코드도 없었으며 연산자마다 많아야 두 세줄 정도의 짧은 코드로도 충분히 구현이 가능했기에 submodule의 필요성을 느끼지 못하였습니다. 우선 각 연산자마다 할당한 매크로 상수는 위 표의 위에서 아래 순서대로 4b`0000 ~ 4b`1111까지 지정해 주었고 각 코드에 해당하는 연산이 진행되도록 switch문을 사용하였습니다. C가 wire이기 때문에 switch문 내부에서 assign이 불가능하여 alu\_result라는 reg를 선언한 후 그 둘을 assign해 주었습니다.

design에서 서술한 바와 같이 bitwise 연산자들의 경우에는 verilog에서 지원하는 연산을 조합하여 간단히 구현할 수 있었습니다. Add연산의 경우에는 17bit의 temp변수에 A+B+Cin값을 대입하여 carry out이 발생하는지 여부를 1<<16값과 비교하여 구하였고, substract에서도 B+Cin의 값을 temp에 저장한 이후 A값과 비교하여 A와 B+Cin의 값을 비교하였습니다. 이외에 shifting 연산에서는 부호비트와 MSB, LSB의 값이 어떻게 변화하는지를 유의하여 코딩하였습니다. ARS연산에서 부호 비트는 그대로 유지되도록, rr연산에서는 LSB가 MSB로 이동하고 ll연산에서는MSB가 LSB로 이동하도록 구현하였습니다.

1. Duscussion

간단한 프로젝트라는 것은 알고 있지만, verilog를 처음 써보는 것이라서 굉장히 많은 시간이 걸렸습니다. 물론 랩시간에 베릴로그에 대해 들었지만, 그때에는 이해가 된 것 같았는데 막상 프로젝트를 시작해보니 끊임없이 빨간 줄과 에러가 발생했습니다. 아직 2학년이라 그런지 찾아봐도 이게 무슨 말 인지 잘 이해도 안되고 주변에 물어볼 사람도 없어서 꽤 많은 시행착오를 겪었습니다. 그래도 어찌어찌 잘 코딩을 한 것 같아서 다행이라는 생각이 들긴 합니다. 그런데 한가지 걱정되는 점은 이번 랩에서 진행한 코드가 계속 사용될 것인데, 혹시 제가 작성한 코드에서 예상치 못한(테스트 벤치에서 발견하지 못한) 버그가 있을 수 있다는 점입니다. 지금이야 코드가 짧으니 쉽게 파악할 수 있겠지만 나중에 더 많은 모듈을 사용하고 코드도 길어지게 될 때가 걱정입니다. 특히 저와 같이 독강을 하는 사람의 경우 코드를 다른 사람과 비교하며 버그를 찾기도 힘들 것인데 대안이 있었으면 좋겠다는 생각이 들었습니다.

1. Conclusion

이번 랩에서 구현하고자 하는 것들은 모두 구현할 수 있었다.