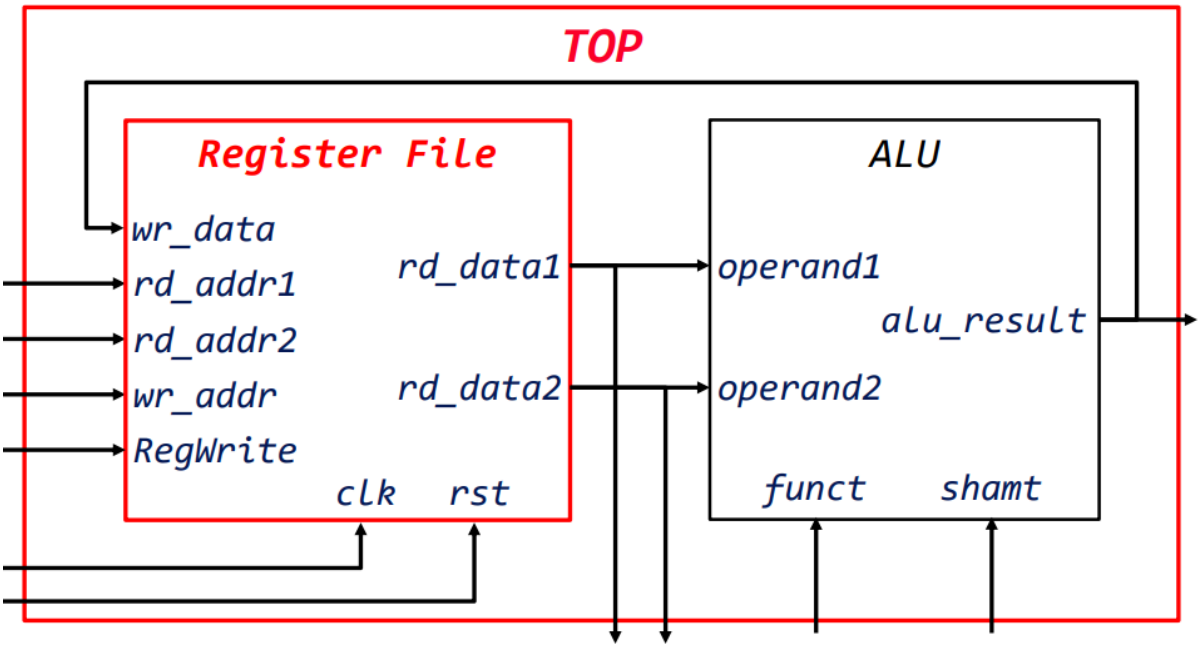
컴퓨터구조론 HW2 wiki

2021045505 김보국

1. Goal

32-entry 32-bit 2-read/1-write register file을 구현하고, Top을 구현하여 HW1에서 제작하였던 ALU와 register file을 연결한다.

2. Design



Register File(RF)

Input

- rd_addr1, rd_addr2 - read하는 대상 주소를 받는다.
- wr_data - write 할 데이터를 받아온다.
- wr_addr - write 할 대상 주소를 받는다. wr_data를 그 주소에 write한다. 단, Regwrite 값이 1일 경우에만 write를 실행한다.
- Regwrite - 1비트로, 이 값에 따라 write의 여부가 결정된다.
- clk - cpu의 클럭 신호. positive일 때 RF unit이 동작한다.
- rst - 이 값이 참이라면 RF의 값을 모두 initial로 초기화한다.

Output

- rd_data1, rd_data2 - rd_addr에서 read한 값들을 output한다.

ALU

- operand1, operand2 - 연산할 데이터를 받는다.
- 구조 - operand1, operand2, funct, shamt를 통해 alu를 실행하고, 그 결과값을 alu_result에 담는다.

TOP

- rd_data1, rd_data2를 각각 operand1, operand2에 연결한다.
- alu_result를 wr_data에 연결한다.
- 최종적으로, RF와 ALU를 하나로 묶는 역할을 한다.

3. Implementation

CPP

- RF.cpp

RF::read는 register_files의 rd_addr들에 저장된 값을 읽어 rd_data에 넣어주는 것이므로 아래와 같이 구현하였다.

```
void RF::read(uint32_t rd_addr1, uint32_t rd_addr2, uint32_t *rd_data1, uint32_t
    *rd_data1 = register_files[rd_addr1];
    *rd_data2 = register_files[rd_addr2];
}
```

RF::write는 wr_data 값을 register_files의 wr_addr에 저장하는 것이므로 아래와 같이 구현하였다.

```
void RF::write(uint32_t wr_addr, uint32_t wr_data, uint32_t RegWrite) {
    if(RegWrite==1){
        register_files[wr_addr] = wr_data;
    }
}
```

- TOP.cpp

TOP은 RF와 ALU를 하나로 design에 따라 묶어주었다. tick에서는 데이터를 read한 후 alu로 넘겨 compute를 실행한 후 다시 RF에 결과값을 넣어주는 방식으로 동작한다.

```
void TOP::tick(uint32_t rd_addr1, uint32_t rd_addr2,
        uint32_t wr_addr, uint32_t shamt, uint32_t aluop, uint32_t RegWrite,
        uint32_t *rd_data1, uint32_t *rd_data2, uint32_t *wr_data) {
    rf.read(rd_addr1,rd_addr2,rd_data1,rd_data2);
    alu.compute(*rd_data1, *rd_data2, shamt, aluop, wr_data);
    rf.write(wr_addr,*wr_data,RegWrite);
}
```

Verilog

- RF.v

rd_data1, rd_data2는 reg로 선언하였다. 따라서 asynchronous로 동작하게 하기 위해서 always 구문을 작성하였고, 실제로 값을 넣어주는 부분은 non-blocking으로 구현하였다.

```
  // Fill in the asynchronous functions
    always @(*) begin
        rd_data1 <= register_file[rd_addr1];
        rd_data2 <= register_file[rd_addr2];
    end

    always @(posedge clk) begin
        if (rst) begin
            $readmemh("initial_reg.mem", register_file);
        end
        // FILL what happens synchronously
        if (RegWrite) begin
            register_file[wr_addr] = wr_data;
        end
    end
```

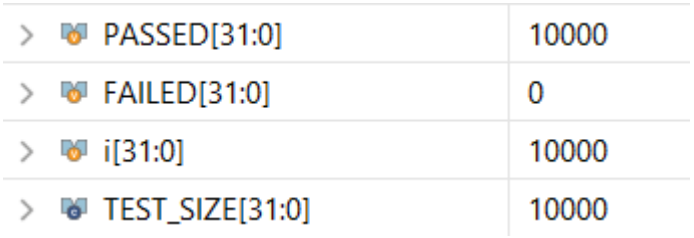
- TOP.v

RF와 ALU유닛을 묶어주는 부분이므로, 그저 변수에 주의하며 함수를 사용하였다. Design과 같이 rd_data1, rd_data2, wr_data로 두 유닛을 묶어주었다.

```
RF rf_u(clk, rst, rd_addr1, rd_addr2, rd_data1, rd_data2, RegWrite, wr_addr, wr_
ALU alu_u(rd_data1, rd_data2, shamt, funct, wr_data);
```

4. Result

cpp 코드를 실행했을 때 생성된 메모리로 검증했을 때 모두 이상 없이 design에 맞게 동작하는 것을 확인할 수 있었다.



5. Troubleshooting

verilog 코드를 검증하는 도중, 저번 과제에서 성공했던 ALU를 가지고 테스트를 진행했음에도 오류가 발생하는 경우가 유의미하게 존재하였다. 따라서 Failed가 나온 첫 번째 testcase를 분석한 결과, ALU의 SRA 연산에서 issue가 발생한다는 것을 확인할 수 있었다.

Trouble

```
`ALU_SRA : alu_result=operand2>>>shamt;
```

위는 verilog에서 ALU.v에 이전에 작성하였던 SRA연산이다. 이 연산에서 “>>>”연산자를 사용하였는데, 이를 잘못 이해하고 있었다.

Solve

```
`ALU_SRA : alu_result=$signed(operand2)>>>shamt;
```

위와 같이 바꾸어 주니 정상적으로 동작하는 것을 확인할 수 있었다. “>>>”연산의 경우 확인 결과 unsigned에서 연산을 진행할 경우 MSB를 무조건적으로 0으로 판단하고, signed 상황에서 연산을 진행할 경우 부호에 따라 MSB가 바뀌는 것을 확인할 수 있었다. 따라서 SRA연산에서 operand2를 signed로 바꾸어 연산을 진행해주어 문제를 해결하였다.