Lab2_Report

이름: 권도현

학번: 2023065350

학과: 컴퓨터소프트웨어학부

Project Design

	High-level simulation	Verilog Testbench
Programming language	C++	Verilog
Environment	Ubuntu	Vivado
Objective	ALU 연산 구현 및 .mem 테스트 벡터 생성	RF + ALU 연동 확인, .mem 기반 연산 검증
Simulation Output	.mem 파일 생성	ALU 결과 비교 및 PASS/FAIL 출력

Overall Structure

1. CPP

ALU 연산은 HW01의 ALU 코드를 그대로 사용하였다.

RF.cpp

- read(): Register file의 두 개의 5-bit read address에 접근하여 데이터를 읽어오도록 구현하였다.
- write(): write data를 write address에 맞는 register file에 저장할 수 있도록 구현하였다. RegWrite가 1인 경우에만 write할 수 있도록 if문을 통해서 미리 RegWrite를 확인하도록 하였다.

TOP.cpp

- 구현 목표: Register file로부터 데이터를 operand에 가져와서, ALU 연산을 진행한 후에 Register file에 write 한다.
- tick(): Read -> ALU operation -> Write 순으로 구현되어야 한다.
- Parameter는 alu.compute()의 operand에 *rd_data를 넣어 register file에서 읽어온 값을 사용하도록 하였고, alu_result에 wr_data를 적어 wr_data에 alu_result를 미리 할당해놓았다.

```cpp
// Edited
void RF::read(uint32_t rd_addr1, uint32_t rd_addr2, uint32_t *rd_data1, uint32_t *rd_data2) {
        *rd_data1 = register_files[rd_addr1];
        *rd_data2 = register_files[rd_addr2];
}

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

RF.cpp

```cpp
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);
}
```

TOP.cpp

2. Verilog

ALU 연산은 HW01의 ALU 코드를 그대로 사용하였다.

대부분의 코드를 CPP에서와 비슷하게 작성하였지만, Verilog에서는 read()는 비동기화 작업, write는 동기화 작업에서 동작하도록 작성하였다.

TOP.v에서 module 인스턴스를 생성할 때, operand(rd_data)를 통해 alu opreand에 읽어온 데이터를 사용하고, alu_result(wr_data)를 통해 wr_data가 alu_result와 연결될 수 있도록 하였다.

```v
// FIXME
// Instantiate modules and connect them!
RF rf (
    .clk(clk),
    .rst(rst),
    .RegWrite(RegWrite),
    .rd_data1(rd_data1),
    .rd_data2(rd_data2),
    .rd_addr1(rd_addr1),
    .rd_addr2(rd_addr2),
    .wr_data(wr_data),
    .wr_addr(wr_addr)
);

ALU alu(
    .operand1(rd_data1),
    .operand2(rd_data2),
    .shamt(shamt),
    .funct(funct),
    .alu_result(wr_data)
);
```

TOP.v

```
// 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
    if (RegWrite) begin
        register_file[wr_addr] <= wr_data;
    end
end
```

RF.v

Results

1. CPP

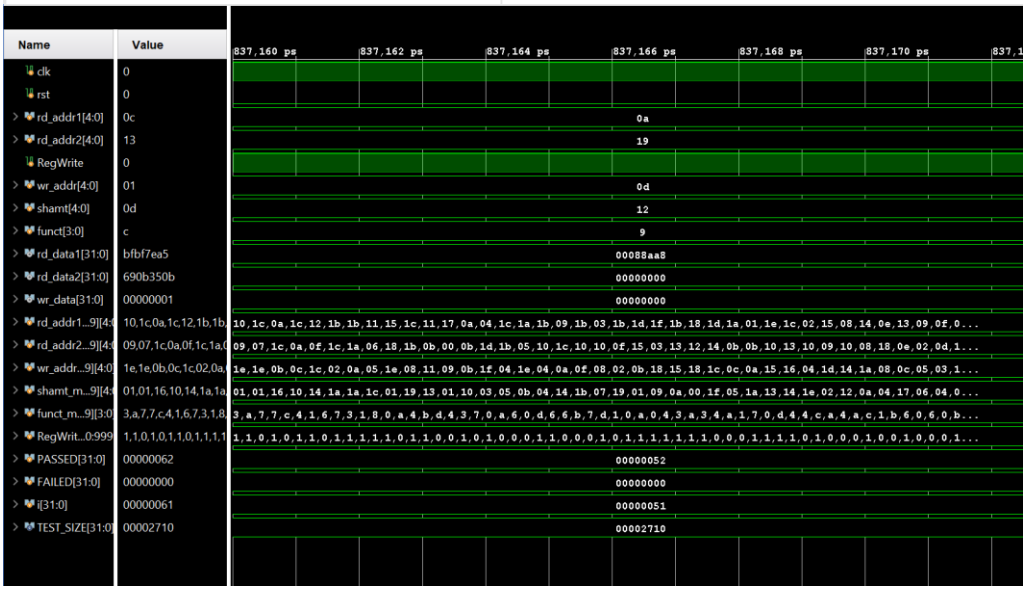
make를 하고 생긴 실행파일 top을 실행하였더니 .mem 파일이 생성되었다.

```
kwondh3236@NOTKDH:~/computer_architecture/HW02/cpp$ ls
ALU.cpp   Makefile  RF.o      TOP.o      initial_reg.mem  rd_addr1.mem  rd_data2.mem  top
ALU.h     RF.cpp    TOP.cpp   funct.mem  main.cpp         rd_addr2.mem  regwr.mem     wr_addr.mem
ALU.o     RF.h      TOP.h     globals.h  main.o           rd_data1.mem  shamt.mem     wr_data.mem
```

2. Verilog

.mem 파일을 testbench로 하여 시뮬레이션 하였다.

PASSED[31:0]	10000
FAILED[31:0]	0
i[31:0]	10000
TEST_SIZE[31:0]	10000



Difficulties

HW01에서 Vivado tool을 다루는 것이나, Verilog를 처음 다루어 봐서 관련된 어려움이 많았는데, HW01을 한번 해보니 관련된 어려움이 적어 과제를 수행하는 데에 있어 큰 어려움은 없었던 것 같다.

실수했던 부분은 TOP.v의 ALU 인스턴스를 생성하는 부분에서 .RegWrite(Regwrite)로 w를 소문자로 써서 결과가 제대로 나오지 않았다. 웨이브폼을 통해 RegWrite와 Write 작업이 제대로 수행되지 않는 것을 확인하고 수정할 수 있었다.

Hardware Modules