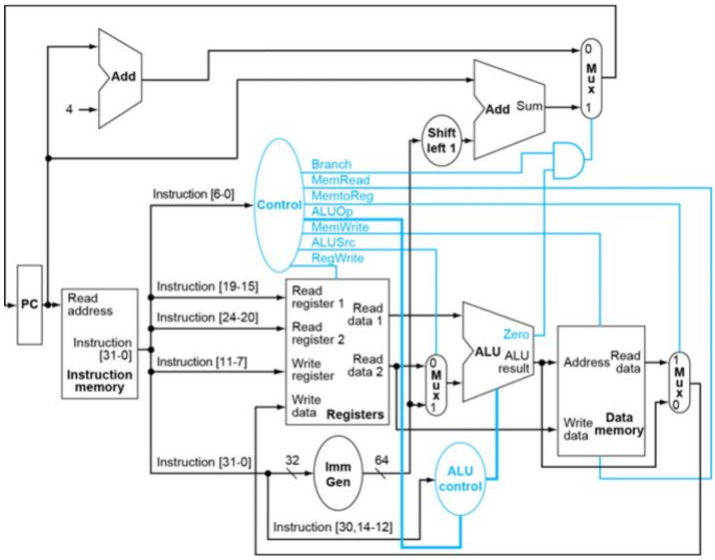
Assignment 5

201820873 윤현찬



1. PC

```
module PC(ADDR, ADDROUT, CLK, RST);
    input CLK;
    input [31:0] ADDR;
    input RST;
    output reg [31:0] ADDROUT;


    always @(posedge CLK) begin
        if (RST) ADDROUT <= 0; else begin
        ADDROUT <= ADDR;
        end
    end


endmodule
```

PC Moulde은 명령어 따라 입력되는 값이 다르며 Branch 명령어일 경우 Immediate값만큼 Counter값이 증가되고, 나머지의 경우에는 4만큼 증가한다. 이러한 연산은 다른 Module에서 진행되며 PC의 역할은 CLK에 따라 동기화 시키는 역할을 한다. 따라서 CLK의 상승 Edge가 발생할 때 저장되어 있는 값이 변한다.

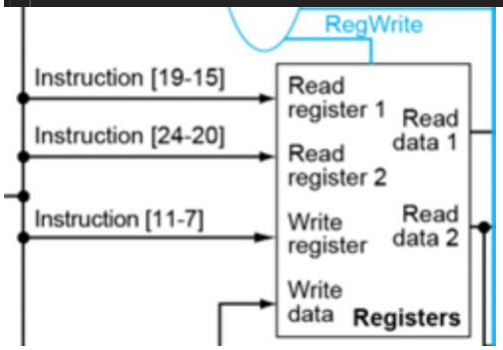
2. 명령어 읽기

```
always @(*) begin
    // $display("\n%d, %b %b %b %b %b %b", ADDR, INST[31:25], INST[24:20], INST[19:15], INST[14:12], INST[11:7], INST[6:0]);
    ControlIn = INST[6:0];
    RR1 = INST[19:15]; // Read Register1 입력 => RD1
    RR2 = INST[24:20]; // Read Register2 입력 => RD2
    WR = INST[11:7]; // Write Register 입력 (데이터를 쓸 레지스터 주소)
    sign_in = INST[31:0];
    ALU_control_IN = {INST[30], INST[14:12]};
    OUT = INST;

end
```

Vr_inst_mem.v 파일은 주어진 파일이었다. 이 Moduled은 PC의 출력인 ADDR의 값에 따라 case문을 통해 저장된 명령어를 INST라는 변수로 출력하는 파일이다. 출력된 INST를 SingleCPU.v 파일에서 always문을 활용하여 각 Module이 필요요 하는 명령어의 일부를 나누어 주었다.

3. Register

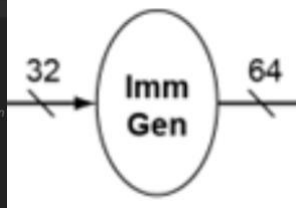
```
module Vr_register_file(CLK, RST, RR1, RR2, WR, WD, WE, RD1, RD2);
    input CLK;
    input RST;
    input [4:0] RR1; // Read Register1
    input [4:0] RR2; // Read Register2
    input [4:0] WR; // Write Register
    input [31:0] WD; // Write date
    input  WE; // Wriet Enable
    output [31:0] RD1; // Read data1
    output [31:0] RD2; // Read data2

    reg [31:0]     register_file [0:31];
    assign RD1 = register_file[RR1];
    assign RD2 = register_file[RR2];
    integer      i;
```



Register 파일도 주어진 파일이다. Instruction Memory에서 rs1에 해당하는 데이터를 RR1에 rs2에 해당하는 데이터를 RR2에 입력한다. 또한 연산 후 데이터를 저장할 Register 위치인 rd2는 WR에 입력된다. RR1에 저장되어 있는 값이 RD1으로 출력되고, RR2에 저장되어 있는 값이 RD2로 출력된다. 또한 Control Module에서 Opcode를 본 후 Register의 동작을 제어하는데 WE를 통해 WE가 1이면 저장되고 0이면 저장되지 않는다.

4. Immediate

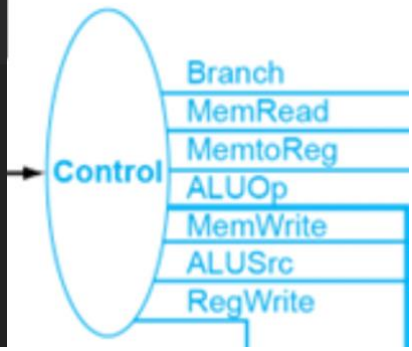
```
module SignedExtension(sign_in, sign_out, RST);

    input RST;
    input [31:0] sign_in;
    output reg [31:0] sign_out;
    reg [11:0] immediate;



    always @(*) begin
        if(RST) immediate = 0;
        else begin
        if(sign_in[6:0] == 7'b1100011)
            immediate = {sign_in[31], sign_in[7], sign_in[30:25], sign_in[11:8]};
        else if (sign_in[6:0] == 7'b0100011)
        begin
            immediate = {sign_in[31:25], sign_in[11:7]};
        end
        else immediate = sign_in[31:20];


        sign_out[11:0] = immediate;
        sign_out[31:12] = immediate[11] ? 20'b1111_1111_1111_1111_1111: 20'b0; // Signed-Extension
        // $display("sign_in:%b, immediate: %b, sign_out: %b", sign_in, immediate, sign_out);
        end
    end

endmodule
```



Immediate는 Register에 저장된 값이 아닌 상수를 사용하여 연산하고자 할 때 필요한 Module이다. RISC-V에서 Immediate에 해당하는 위치는 명령어의 type마다 다르므로 opcode를 통해 명령여의 type을 분류하여 immediate값을 저장하였다. Immediate는 RISC-V에서 12-bit로 나타내어 지는데 ALU의 입력은 32bit이므로 Sign-Extension을 해주어야 한다.

5. Control

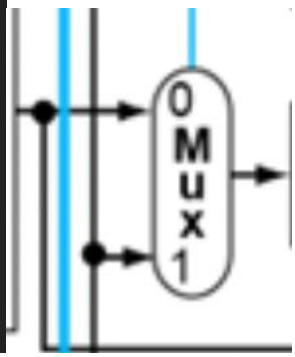
```
module Control(ControlIn, branch, memRW, MemtoReg, ALUOp, ALUSrc, RegWrite, RST);


input [6:0] ControlIn;
input RST;
output reg branch;
output reg memRW;
output reg MemtoReg;
output reg [1:0] ALUOp;
output reg ALUSrc;
output reg RegWrite;

always @(*) begin
    if(RST == 1) begin…
    end
    else
    case(ControlIn)
    7'b1100011: begin // B-type…
    7'b0110011: begin // R-Type…
    7'b0100011: begin // SW-type…
    7'b0010011: begin // I-type…
    7'b0000011: begin // LW-type
        ALUOp = 2'b00;
        ALUSrc = 1;
        branch = 0;
        RegWrite = 1;
        memRW = 0;
        MemtoReg = 0;
        // $display("ControlIn: %b, RegWrite: %b, LW-Type", ControlIn, RegWrite);

        end
    endcase
end
endmodule
```



Control Module은 명령어를 해석한 후 각 Module이 해야 할 동작을 제어해주는 Module이다. 명령어를 해석하는 방법은 RISC-V에서 명령어 INST[6:0]이 Opcode에 해당하므로 이를 활용하여 명령어의 동작을 파악할 수 있다. Branch는 Branch명령어일 때 Counter의 변화를 제어하고, MemRead와 MemWrite는 memRW로 합쳐져서 0이면 메모리의 데이터를 읽고 1이면 메모리에 데이터를 저장하는 동작을 제어한다. 또한 ALUOp는 ALU Controller에 입력되어 ALU의 동작을 제어하고 ALUSrc는 ALU의 입력을 Register의 저장되어 있는 값으로 할지 Immediate값으로 할지 제어한다. RegWrite는 Write Register 위치에 Write Data 값을 저장하는 동작을 제어한다. 각 명령어에 따른 변수들의 변화는 Code에 첨부되어 있다.

6. MUX1

```
module mux(MUXIN1, MUXIN2, MUXOUT, SEL, RST);

input RST;
input SEL;
input [31:0] MUXIN1;
input [31:0] MUXIN2;

output reg [31:0] MUXOUT;

always @(*) begin
    if(RST == 1) begin
        MUXOUT = 32'b0;
    end
        else

        MUXOUT = SEL ? MUXIN2 : MUXIN1;


end

endmodule
```



MUX는 두 입력 중 SEL 변수에 따라 출력을 선택하는 동작을 한다. MUX는 3번 사용되는데 첫번째로 사용되는 MUX는 Register에 저장되어 있는 값 또는 Immediate값을 선택하는 동작을 한다. 이 동작은 Control Module의 ALUSrc를 통해 제어된다.

7. ALUControl

```
module ALUcontrol(ALUOp, ALU_control_IN, ALU_control_OUT);

input [1:0] ALUOp;
input [3:0] ALU_control_IN;

wire [2:0]fun3;
wire fun7;
output reg [3:0] ALU_control_OUT;

assign fun3 = ALU_control_IN[2:0];
assign fun7 = ALU_control_IN[3];

always @(*) begin
    case(ALUOp)
    2'b10: begin
        if(fun7 == 1'b1) ALU_control_OUT = 4'b0110; // SUB
        else if(fun3 == 3'b0) ALU_control_OUT = 4'b0010; // ADD
        else if(fun3 == 3'b111) ALU_control_OUT = 4'b0000; // AND
        else if(fun3 == 3'b110) ALU_control_OUT = 4'b0001; // OR
        else if (fun3 == 3'b001) ALU_control_OUT = 4'b0011; // SLLI
    end
    2'b01:begin
        case(fun3)
            3'b100: ALU_control_OUT = 4'b1100;
            3'b000: ALU_control_OUT = 4'b1000;
        endcase
    end

    2'b00:ALU_control_OUT = 4'b0010;
    endcase

end

endmodule
```

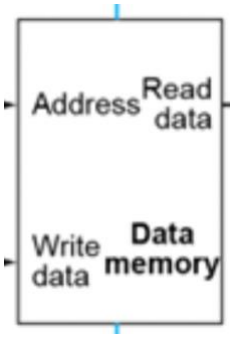
ALU Control Module은 Control Module에서 ALU의 동작을 모두 제어하기에 무리가 있어서 만들어진 Module이다. RISC-V에는 Opcode뿐 만 아니라 fun7과 fun3를 통해 명령어의 동작이 분류된다. 따라서 ALU에서 같은 동작을 하는 명령어들을 모은 후 출력을 같게 하여 ALU의 동작을 제어한다.

8. ALU

```
module ALU(ALUIN1, ALUIN2, ALUC, ALU_OUT, Zero);
    input [31:0] ALUIN1;
    input [31:0] ALUIN2;
    input [3:0] ALUC;
    output reg Zero;
    output reg [31:0] ALU_OUT;
    reg [31:0] temp;

    always @(*) begin
        case(ALUC)
            4'b0000:
            begin…
            end
            4'b0001:begin…
            end
            4'b0010:begin
                Zero = 0;
                ALU_OUT = ALUIN1 + ALUIN2;
            end
            4'b0110:begin
                Zero = 0;
                ALU_OUT = ALUIN1 - ALUIN2;
            end
            4'b1000:
            begin
                ALU_OUT = 32'bx;
                temp = ALUIN1 - ALUIN2;
                if (temp == 0) Zero = 1;
                else Zero = 0;
            end
            4'b1100: begin
                ALU_OUT = 32'bx;
                temp = ALUIN1 - ALUIN2;
                if (temp[31] == 0) Zero = 0;
                else Zero = 1;
            end
            4'b0011: begin
                Zero = 0;
                ALU_OUT = ALUIN1 << ALUIN2;
            end
            default:;
        endcase
    end

endmodule
```

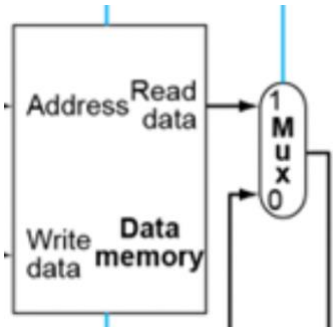
ALU Module은 연산을 한다. ALU의 첫번째 입력은 RD1으로 rs1의 주소에 있는 Register의 값이다. 두번째 입력은 rs2의 주소에 있는 Register의 값 혹은 Immediate값이며 이는 Control Module을 통해 제어된다. Opcode와 fun3, fun7을 통해 같은 연산을 하는 명령어들을 분류하여 ALU에서 연산하게 하였다. 추가로 Zero라는 변수가 있는데 이는 Branch 명령어에서 조건이 맞으면 PC를 Immediate에 따라 이동하게 되는데 조건이 맞을 경우 1이 출력되고, 조건이 맞지 않으면 0이 출력된다. Control Module에서 출력되는 Branch 변수와 AND gate로 연결되어 있어서 Branch 명령어임과 동시에 조건이 맞아야 PC가 Immediate만큼 이동하고 그렇지 않다면 PC + 4동작을 한다.

9. Data Memory



Data Memory 파일은 주어진 파일이다. Address 변수에 ALU의 결과가 입력되고, Write data 변수에는 Register에서 rs2 위치의 데이터가 입력된다. Control에서 Memory를 제어하는 변수인 RW를 통해 동작을 제어한다. 만약 RW가 0이라면 Address변수의 위치에 있는 데이터를 출력하고 RW가 1이라면 Address 위치에 Write data에 입력된 값을 저장한다.

10.MUX3



여기에서 MUX는 Register에 저장되는 값을 제어한다. Control Module에서 MemtoReg 변수를 통해 제어되며 MemtoReg가 1일 경우 Memory의 출력을, 0일 경우 ALU의 출력을 Register에 전달한다.

11. Add

```
module ALU_ADD(IN1, IN2, Counter1);

input [31:0] IN1;
input [31:0] IN2;
output [31:0] Counter1;

assign Counter1 = $unsigned(IN1) + $signed(IN2);
// always @(*) begin
//     $display("ALU_ADD: %b %b %b", IN1, IN2, Counter1);
// end


endmodule
```

Add Module은 Branch 명령어일 때 PC + Imm 을 연산한다. 이후에 MUX2를 통해 PC의 값이 결정된다. 이는 앞서 말했듯 Control의 Branch와 ALU의 Zero의 AND 연산으로 결정된다.

- Simulation 결과

```
  1 MEM: 1 9 2 3 5 10 7 6 4 8
  3 MEM: 1 9 2 3 5 10 7 6 4 8
  5 MEM: 1 9 2 3 5 10 7 6 4 8
  7 MEM: 1 9 2 3 5 10 7 6 4 8
  9 MEM: 1 9 2 3 5 10 7 6 4 8
 11 MEM: 1 9 2 3 5 10 7 6 4 8
 13 MEM: 1 9 2 3 5 10 7 6 4 8
 15 MEM: 1 9 2 3 5 10 7 6 4 8
 17 MEM: 1 9 2 3 5 10 7 6 4 8
 19 MEM: 1 9 2 3 5 10 7 6 4 8
 21 MEM: 1 9 2 3 5 10 7 6 4 8
 23 MEM: 1 9 2 3 5 10 7 6 4 8
 25 MEM: 1 9 2 3 5 10 7 6 4 8
 27 MEM: 1 9 2 3 5 10 7 6 4 8
 29 MEM: 1 9 2 3 5 10 7 6 4 8
 31 MEM: 1 9 2 3 5 10 7 6 4 8
 33 MEM: 1 9 2 3 5 10 7 6 4 8
 35 MEM: 1 9 2 3 5 10 7 6 4 8
 37 MEM: 1 9 2 3 5 10 7 6 4 8
 39 MEM: 1 9 2 3 5 10 7 6 4 8
 41 MEM: 1 9 2 3 5 10 7 6 4 8
 43 MEM: 1 9 2 3 5 10 7 6 4 8
 45 MEM: 1 9 2 3 5 10 7 6 4 8
 47 MEM: 1 9 9 3 5 10 7 6 4 8
 49 MEM: 1 9 9 3 5 10 7 6 4 8
 51 MEM: 1 2 9 3 5 10 7 6 4 8
 53 MEM: 1 2 9 3 5 10 7 6 4 8
 55 MEM: 1 2 9 3 5 10 7 6 4 8
 57 MEM: 1 2 9 3 5 10 7 6 4 8
 59 MEM: 1 2 9 3 5 10 7 6 4 8
 61 MEM: 1 2 9 3 5 10 7 6 4 8
 63 MEM: 1 2 9 3 5 10 7 6 4 8
 65 MEM: 1 2 9 3 5 10 7 6 4 8
 67 MEM: 1 2 9 9 5 10 7 6 4 8
 69 MEM: 1 2 9 9 5 10 7 6 4 8
 71 MEM: 1 2 3 9 5 10 7 6 4 8
 73 MEM: 1 2 3 9 5 10 7 6 4 8
 75 MEM: 1 2 3 9 5 10 7 6 4 8
 77 MEM: 1 2 3 9 5 10 7 6 4 8
 79 MEM: 1 2 3 9 5 10 7 6 4 8
```

```
605 MEM: 1 2 3 5 4 6 7 8 9 10
607 MEM: 1 2 3 5 4 6 7 8 9 10
609 MEM: 1 2 3 5 4 6 7 8 9 10
611 MEM: 1 2 3 5 4 6 7 8 9 10
613 MEM: 1 2 3 5 4 6 7 8 9 10
615 MEM: 1 2 3 5 4 6 7 8 9 10
617 MEM: 1 2 3 5 4 6 7 8 9 10
619 MEM: 1 2 3 5 4 6 7 8 9 10
621 MEM: 1 2 3 5 4 6 7 8 9 10
623 MEM: 1 2 3 5 4 6 7 8 9 10
625 MEM: 1 2 3 5 4 6 7 8 9 10
627 MEM: 1 2 3 5 4 6 7 8 9 10
629 MEM: 1 2 3 5 4 6 7 8 9 10
631 MEM: 1 2 3 5 4 6 7 8 9 10
633 MEM: 1 2 3 5 4 6 7 8 9 10
635 MEM: 1 2 3 5 4 6 7 8 9 10
637 MEM: 1 2 3 5 4 6 7 8 9 10
639 MEM: 1 2 3 5 4 6 7 8 9 10
641 MEM: 1 2 3 5 4 6 7 8 9 10
643 MEM: 1 2 3 5 4 6 7 8 9 10
645 MEM: 1 2 3 5 4 6 7 8 9 10
647 MEM: 1 2 3 5 4 6 7 8 9 10
649 MEM: 1 2 3 5 4 6 7 8 9 10
651 MEM: 1 2 3 5 4 6 7 8 9 10
653 MEM: 1 2 3 5 4 6 7 8 9 10
655 MEM: 1 2 3 5 4 6 7 8 9 10
657 MEM: 1 2 3 5 4 6 7 8 9 10
659 MEM: 1 2 3 5 4 6 7 8 9 10
661 MEM: 1 2 3 5 4 6 7 8 9 10
663 MEM: 1 2 3 5 4 6 7 8 9 10
665 MEM: 1 2 3 5 4 6 7 8 9 10
667 MEM: 1 2 3 5 4 6 7 8 9 10
669 MEM: 1 2 3 5 4 6 7 8 9 10
671 MEM: 1 2 3 5 5 6 7 8 9 10
673 MEM: 1 2 3 5 5 6 7 8 9 10
675 MEM: 1 2 3 4 5 6 7 8 9 10
677 MEM: 1 2 3 4 5 6 7 8 9 10
679 MEM: 1 2 3 4 5 6 7 8 9 10
681 MEM: 1 2 3 4 5 6 7 8 9 10
683 MEM: 1 2 3 4 5 6 7 8 9 10
```

- Bubble Sort 연산이 잘 이루어지는 것을 볼 수 있다.