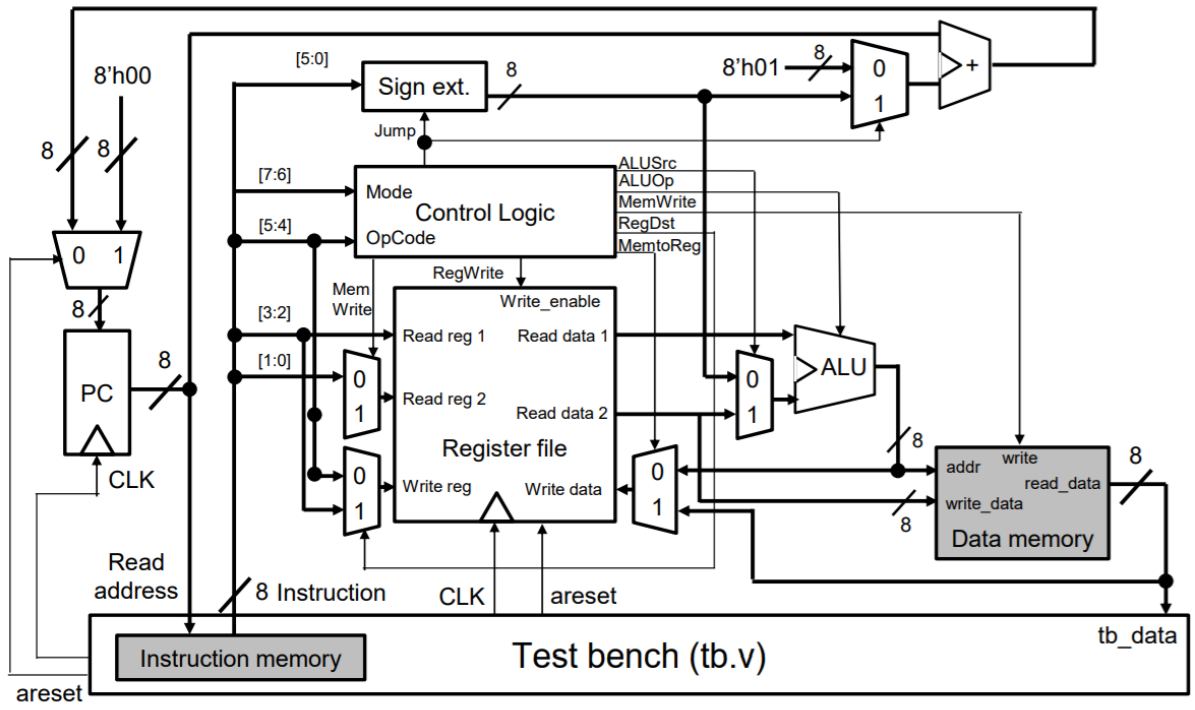
논리설계 FINAL PROJECT

2017-18538
컴퓨터공학부
황선영

1. Overall Design and Structure in Detail



위의 data path를 토대로 8-bit microprocesser를 구현하였다. 레지스터와 데이터의 크기도 8-bit이다. 레지스터의 개수는 r0, r1, r2, r3으로 4개이다. instruction의 수는 6개로 add, sub, load, store, jump, nop의 기능을 수행한다. fpga와 연결하여 32개의 값을 가지고 있는 데이터 메모리(총 3개)와 비교하여 bcd display판에 몇 개의 오답이 있는지 확인한다.

```
module cpu      //Do not change top module name or ports.
(
        input   clk,
        input   areset,
        output  [7:0] imem_addr,        //Request instruction memory
        input   [7:0] imem_data,        //Returns
        output  [7:0] tb_data           //Testbench wiring.
);

        //Data memory and testbench wiring. you may rename them if you like.
        wire dmem_write;
        wire [7:0] dmem_addr, dmem_write_data, dmem_read_data;
        wire Jump, ALUSrc, RegDst, MemtoReg, RegWrite;
        wire [7:0] MUXtoPC, PC_out, Sign_out;
        wire [7:0] MUXtoAdder, AddertoMUX;
        wire [7:0] RFtoALU, MUXtoALU;
        wire [7:0] MUXtoRF;
        wire [1:0] MUX2toRF, MUX3toRF, ALUOp;

        //Data memory module in tb.v.
        memory dmem(    .clk(clk), .areset(areset),
                        .write(dmem_write), .addr(dmem_addr),
                        .write_data(dmem_write_data), .read_data(dmem_read_data));

        assign tb_data = dmem_read_data;
        //Testbench wiring end.

        //Write your code here.
        ProgramCounter PC(.CLK(clk), .in(MUXtoPC), .out(PC_out));
        RegisterFile RF(.CLK(clk), .ReadReg1(imem_data[3:2]), .ReadReg2(MUX2toRF), .Write_enable(RegWrite),
                        .WriteReg(MUX3toRF), .WriteData(MUXtoRF), .areset(areset), .ReadData1(RFtoALU), .ReadData2(dmem_write_data));
        ControlLogic CL(.Mode(imem_data[7:6]), .OpCode(imem_data[5:4]), .Jump(Jump), .ALUSrc(ALUSrc),
                        .ALUOp(ALUOp), .MemWrite(dmem_write), .RegDst(RegDst), .MemtoReg(MemtoReg), .RegWrite(RegWrite));
        SignExtensionUnit SEU(.in(imem_data[5:0]), .Jump(Jump), .out(Sign_out));
        ALU alu(.opcode(ALUOp), .ain(RFtoALU), .bin(MUXtoALU), .result(dmem_addr));
        EightBitMUX mux1(.control(areset), .in0(AddertoMUX), .in1(8'h00), .out(MUXtoPC));
        TwoBitMUX mux2(.control(dmem_write), .in0(imem_data[1:0]), .in1(imem_data[5:4]), .out(MUX2toRF));
        TwoBitMUX mux3(.control(RegDst), .in0(imem_data[5:4]), .in1(imem_data[3:2]), .out(MUX3toRF));
        EightBitMUX mux4(.control(MemtoReg), .in0(dmem_addr), .in1(dmem_read_data), .out(MUXtoRF));
        EightBitMUX mux5(.control(ALUSrc), .in0(Sign_out), .in1(dmem_write_data), .out(MUXtoALU));
        EightBitMUX mux6(.control(Jump), .in0(8'h01), .in1(Sign_out), .out(MUXtoAdder));
        EightBitAdder adder(.a(PC_out), .b(MUXtoAdder), .s(AddertoMUX));

        assign imem_addr = PC_out;
endmodule
```

cpu 모듈은 하위 모듈을 연결하는 모듈이다. 적절한 wire을 통해 주어진 data path와 동일하게 각 모듈을 연결하였다.

2. Specify the Funtionality of Each Module

```
module ProgramCounter
(
        input CLK,
    input [7:0] in,
    output reg [7:0] out
);

                always@(posedge CLK)
                        begin
                                out <= in;
                        end
endmodule
```

1) program counter

clock의 posedge마다 input을 output에 넣어주는 기능을 한다. Instruction memory로부터 address를 받는다.

```
module RegisterFile
(
        input CLK,
        input [1:0] ReadReg1,
        input [1:0] ReadReg2,
        input Write_enable,
        input [1:0] WriteReg,
        input [7:0] WriteData,
        input areset,
        output [7:0] ReadData1,
        output [7:0] ReadData2
);

        reg [7:0] registers [3:0];

        initial
                begin
                        registers[0] <= 8'b00000000;
                        registers[1] <= 8'b00000000;
                        registers[2] <= 8'b00000000;
                        registers[3] <= 8'b00000000;
                end

        assign ReadData1 = registers[ReadReg1];
        assign ReadData2 = registers[ReadReg2];

        always@(posedge areset or posedge CLK)
                begin
                if(areset==1'b1)
                        begin
                                registers[0] = 8'b00000000;
                                registers[1] = 8'b00000000;
                                registers[2] = 8'b00000000;
                                registers[3] = 8'b00000000;
                        end

                else if(Write_enable==1'b1)
                        begin
                                registers[WriteReg] = WriteData;
                        end
                end
endmodule
```

2)register file

레지스터 파일은 네개의 레지스터를 가지고 있다. read/write reg를 이용하여 레지스터의 값을 읽거나 쓴다. 레지스터의 값은 write될 때에만 변한다.

```
module ControlLogic
(
        input [1:0] Mode,
        input [1:0] OpCode,
        output reg Jump,
        output reg ALUSrc,
        output reg [1:0] ALUOp,
        output reg MemWrite,
        output reg RegDst,
        output reg MemtoReg,
        output reg RegWrite
);

        initial
                begin
                        {Jump, ALUSrc, ALUOp, MemWrite, RegDst, MemtoReg, RegWrite} = 8'b00000000;
                end
        always@(Mode or OpCode)
                begin
                        case({Mode, OpCode})
                                4'b1101: //add
                                        begin
                                                Jump <= 1'b0;
                                                ALUSrc <= 1'b0;
                                                ALUOp <= 2'b11;
                                                MemWrite <= 1'b0;
                                                RegDst <= 1'b1;
                                                MemtoReg <= 1'b0;
                                                RegWrite <= 1'b1;
                                        end
                                4'b1110: //sub
                                        begin
                                                Jump <= 1'b0;
                                           ALUSrc <= 1'b0;
                                                ALUOp <= 2'b10;
                                                MemWrite <= 1'b0;
                                                RegDst <= 1'b1;
                                                MemtoReg <= 1'b0;
                                                RegWrite <= 1'b1;
                                        end
                                4'b01xx: //load
                                        begin
                                                Jump <= 1'b0;
                                                ALUSrc <= 1'b1;
                                                ALUOp <= 2'b01;
                                                MemWrite <= 1'b0;
                                                RegDst <= 1'b0;
                                                MemtoReg <= 1'b1;
                                                RegWrite <= 1'b1;
                                        end
                                 4'b10xx: //store
                                         begin
                                                 Jump <= 1'b0;
                                                 ALUSrc <= 1'b1;
                                                 ALUOp <= 2'b01;
                                                 MemWrite <= 1'b1;
                                                 RegDst <= 1'b1;
                                                 MemtoReg <= 1'b1;
                                                 RegWrite <= 1'b0;
                                         end
            4'b00xx: //jump
                                         begin
                                                 Jump <= 1'b1;
                                                 ALUSrc <= 1'b0;
                                                 ALUOp <= 2'b00;
                                                 MemWrite <= 1'b0;
                                                 RegDst <= 1'b0;
                                                 MemtoReg <= 1'b0;
                                                 RegWrite <= 1'b0;
                                         end
                        endcase
                end
endmodule

module SignExtensionUnit
(
        input [5:0] in,
        input Jump,
        output reg [7:0] out
);

        always@(*)
                begin
                        if(Jump==1'b1)
                                begin
                                        out[5:0] <= in[5:0];
                                        out[7:6] <= {2{in[5]}};
                        end
                        else
                                begin
                                        out[1:0] <= in[1:0];
                                        out[7:2] <= {6{in[1]}};
                                end
                end
endmodule
```

Type	Opcode	Description	Note
Add	11 01 [rd] [rs]	$rd = $rd +$ rs	
Sub	11 10 [rd] [rs]	$rd = $rd – $rs	
Load	01 [rt] [rs] [off]	$rt = mem[$rs+off]	‘off’ is sign extended
Store	10 [rs] [rd] [off]	mem[$rd+off] = $rs	‘off’ is sign extended
Jump*	00 [off (6 bits)]*	$PC = $PC + off	‘off’ is sign extended
NOP	11 00 XX XX	Do nothing	

3)control logic

컨트롤 로직은 각 기능에 따라 output을 달리하여 기능에 맞는 수행을 하도록 한다. 위의 표처럼 인스트럭션의 7~4번째 인덱스에 따라 기능이 나뉜다.

4)sign extension unit

6-bit input을 8-bit output으로 바꾸어주는 기능을 하는 unit이다. jump가 1일 때는 5번째 인덱스를 늘려 출력하고, 0일 때는 1번째 인덱스를 늘려 출력한다(offset)

```
module ALU
(
        input [1:0] opcode,
        input [7:0] ain, bin,
        output [7:0] result
);

        wire b_l;
        assign b_l = ~(bin) + 1;

        assign result = (opcode==2'b01) ? (ain+bin) :
                                                                (opcode==2'b10) ? (ain-bin) :
                                                                (bin[7]==1) ? (ain - (b_l)) :
                                                                                                (ain+bin);
endmodule
```

5) ALU

opcode의 값에 따라 다른 연산을 수행하는 unit이다.

```
module TwoBitMUX
(
        input control,
        input [1:0] in0,
        input [1:0] in1,
        output reg [1:0] out
);

                always@(control or in0 or in1)
                        begin
                                if(control==0)
                                        begin
                                                out <= in0;
                                        end
                                else
                                        begin
                                                out <= in1;
                                        end
                        end
endmodule

module EightBitMUX
(
        input control,
        input [7:0] in0,
        input [7:0] in1,
        output reg [7:0] out
);

                always@(control or in0 or in1)
                        begin
                                if(control==0)
                                        begin
                                                out <= in0;
                                        end
                                else
                                        begin
                                                out <= in1;
                                        end
                        end
endmodule
```

6) TwoBitMux, EightBitMux

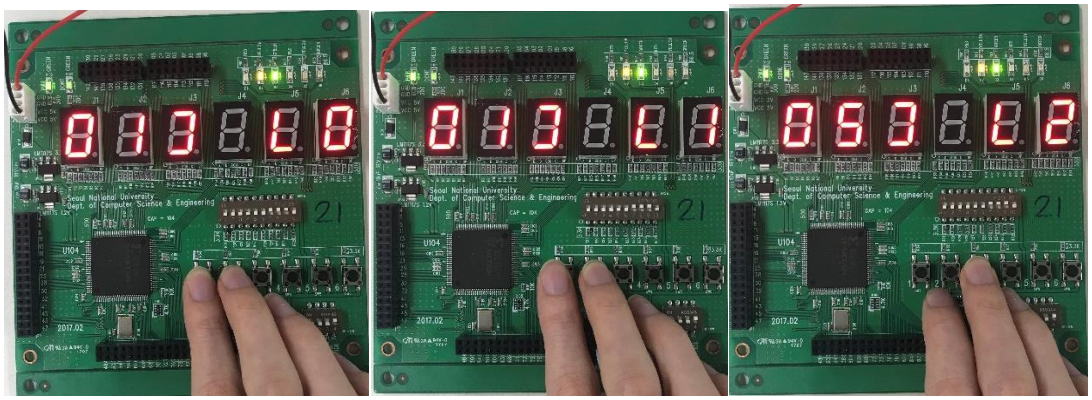
control의 값에 따라 out의 assign을 달리한다.

```
module EightBitAdder
(
        input [7:0]a,b,
        output [7:0]s
);
        assign s = a+b;
endmodule
```

7) EightBitAdder

8-bit의 두 input을 더하는 adder이다.

3. Verify Implementation



level 0은 1개, level 1은 1개, level 2는 5개를 틀렸다. tb.v를 살펴본 결과 store 연산과 오답의 개수가 같은 것으로 미루어보아 store 연산이 제대로 되지 않는다고 판단했다. 그래서 control logic의 output을 재점검했는데, 어떤 값으로 바꾸든 다 같은 개수의 오답이 나왔다. register file의 문제인가 하여 점검해보았는데 이상을 찾지 못했다.