MIPS Register & Memory

조교 서유권

Register

```
module regm(
    input wire              clk,
    input wire  [4:0]       read1, read2,   // READ ADDRESS
    output wire [31:0]      data1, data2,
    input wire              regwrite,
    input wire  [4:0]       wrreg,          // WRITE ADDRESS
    input wire  [31:0]      wrdata
    );

    // 32 bit register X 32
    reg [31:0]  mem [0:31];

    reg [31:0]  reg_data1, reg_data2;

    // READ First ADDRESS
    always @(*)
    begin
        if (read1 == 5'd0)
            reg_data1 <= 32'd0; // Register #0에는 항상 0 저장
        else if ((read1 == wrreg) && regwrite)
            reg_data1 <= wrdata;
        else
            reg_data1 <= mem[read1][31:0];
    end

    // READ Second ADDRESS
    always @(*)
    begin
        if (read2 == 5'd0)
            reg_data2 <= 32'd0; // Register #0에는 항상 0 저장
        else if ((read2 == wrreg) && regwrite)
            reg_data2 <= wrdata;
        else
            reg_data2 <= mem[read2][31:0];
    end

    assign data1 = reg_data1;
    assign data2 = reg_data2;

    // Write
    always @(posedge clk)
    begin
        if (regwrite && wrreg != 5'd0)
            mem[wrreg] <= wrdata;
    end

endmodule
```

Data memory

```
module dm(
    input wire              clk,
    input wire  [6:0]       addr,
    input wire              rd, wr,
    input wire  [31:0]      wdata,
    output wire [31:0]      rdata
    );

    // 32 bit memory with 128 address
    reg [31:0]  mem [0:127];

    always @(posedge clk)
    begin
        if (wr)
            mem[addr] <= wdata;
    end

    assign  rdata = rd ? mem[addr][31:0] : (wr ? wdata : 32'dX);

endmodule
```

ID → EX → MEM → WB

```
module id_ex_mem_wb(
    input wire          clk,
    input wire  [31:0]  instruction
    );

    wire    [5:0]   opcode;
    wire    [4:0]   rs, rt;

    // R-Format
    wire    [4:0]   rd;
//    wire    [4:0]   shamt;

    // I-Format
    wire    [15:0]  imm;
    wire    [31:0]  seimm;  // Sign-extend imm

    assign  opcode = instruction[31:26];
    assign  rs = instruction[25:21];
    assign  rt = instruction[20:16];
    assign  rd = instruction[15:11];
    assign  imm = instruction[15:0];
//    assign  shamt = instruction[10:6];
    assign  seimm = {{16{instruction[15]}}, instruction[15:0]};
```

ID → EX → MEM → WB

```
// Control
wire    branch_eq, branch_ne;   // 이번 시간에는 사용하지 않는 신호
wire    [1:0]   aluop;
wire    memread, memwrite, memtoreg;
wire    regdst, regwrite, alusrc;
wire    jump;   // 이번 시간에는 사용하지 않는 신호

control    ctl_unit (opcode, branch_eq, branch_ne, aluop, memread, memwrite, memtoreg, regdst, regwrite, alusrc, jump);

// Register
wire [31:0]    data1, data2;
wire [4:0]     wrreg;
wire [31:0]    wrdata;

assign wrreg = (regdst)? rd : rt;

regm   register (clk, rs, rt, data1, data2, regwrite, wrreg, wrdata);

// ALU (EX)
wire [31:0] alu_data2;
assign  alu_data2 = (alusrc)? seimm : data2;

wire [31:0] alu_out;
wire        zero;

alu_alu_control_integration    alu_unit (seimm[5:0], aluop, data1, alu_data2, alu_out, zero);
```

ID → EX → MEM → WB

```
    // Data Memory (MEM)
    wire [31:0] rdata;

    dm data_memory (clk, alu_out[8:2], memread, memwrite, data2, rdata);

    // Write Back (WB)
    assign wrdata = (memtoreg)? rdata : alu_out;

endmodule
```