2022 Spring COMP311 : Logic Circuit Design

Final Project

Student ID / Name : 2019112130 / 최부광

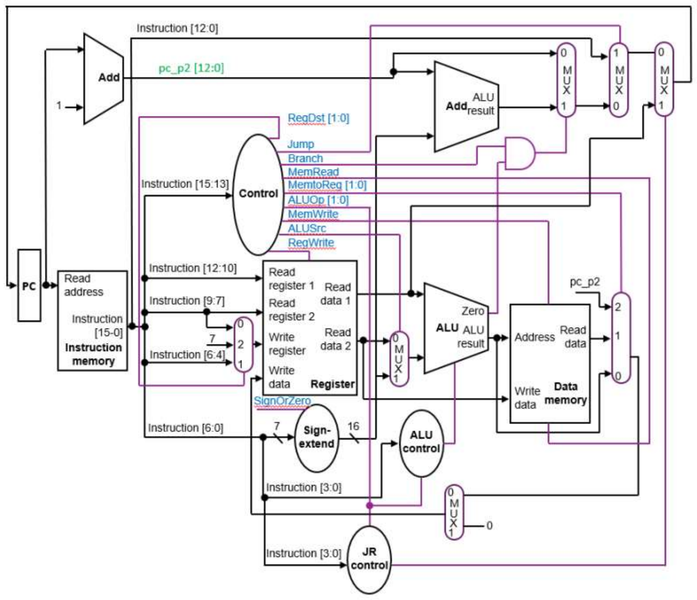
Final Project check list

	Answers
1. First value of $s1, $s2	35, 9
2. First value of $s3	44
3. Final $s3 value	5
4. What are the instructions implemented, but not used?	Sub, or, mul, slti
5. What is the final PC # processed?	19
6. What are the PC #s that have not been processed?	7, 20
7. Parameterized MUX design	Y
8. load memory with instructions	Y
9. Implemented result stored in the register after PC 0000	Y
10. Implemented result after beq	Y
11. Implemented result after first sw is done	Y
12. When run is complete: (1) final $s3 value,	Y
13. When run is complete: (2) registers have intended results	Y

1. Code

Submodule은 다음의 그림과 표를 기반으로 구성하였습니다.

▶ Architecture

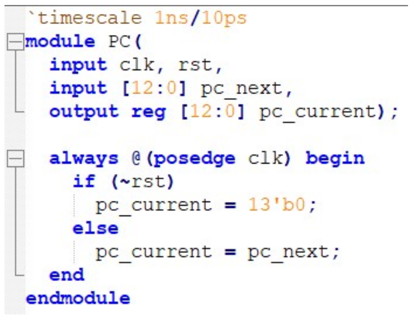


▶ Main Control Table

	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp	Jump
R-type	1	0	0	1	0	0	0	00	0
Slti	0	1	0	1	0	0	0	10	0
J	0	0	0	0	0	0	0	00	1
Jal	2	0	2	1	0	0	0	00	1
Lw	0	1	1	1	1	0	0	11	0
Sw	0	1	0	0	0	1	0	11	0
Beq	0	0	0	0	0	0	1	01	0
addi	0	1	0	1	0	0	0	11	0

1. Code

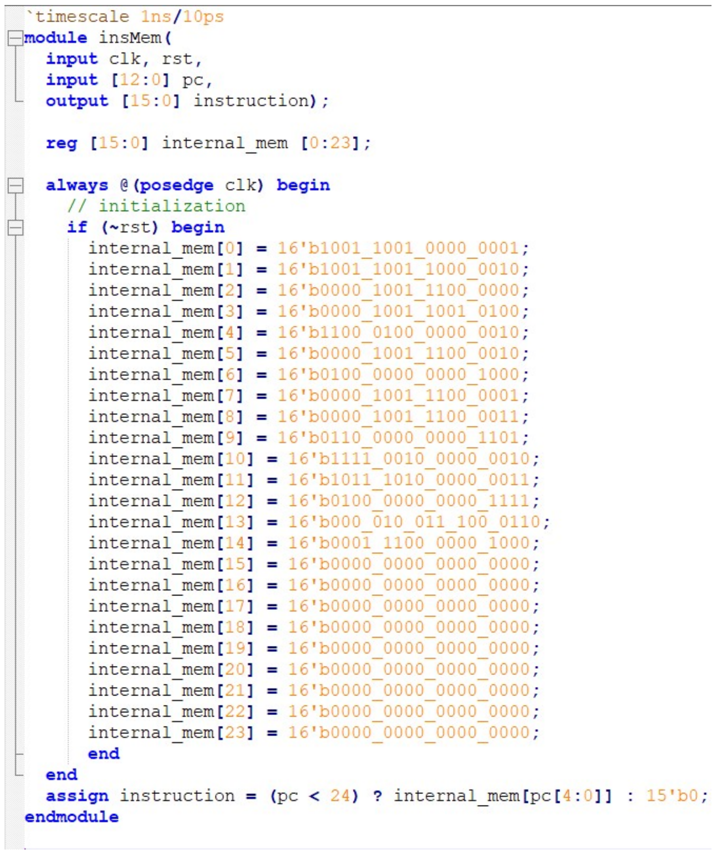
1) Program Counter



```
`timescale 1ns/10ps
module PC(
  input clk, rst,
  input [12:0] pc_next,
  output reg [12:0] pc_current);

  always @(posedge clk) begin
    if (~rst)
      pc_current = 13'b0;
    else
      pc_current = pc_next;
  end
endmodule
```

- Posedge clk마다 다음 명령어 주소를 내보낸다

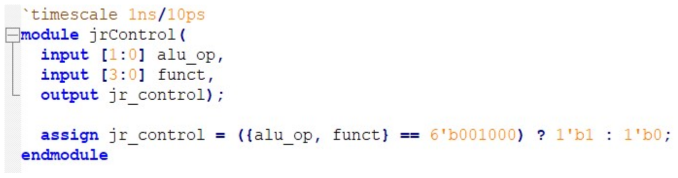
2) Instruction Memory



```
`timescale 1ns/10ps
module insMem(
  input clk, rst,
  input [12:0] pc,
  output [15:0] instruction);

  reg [15:0] internal_mem [0:23];

  always @(posedge clk) begin
    // initialization
    if (~rst) begin
      internal_mem[0] = 16'b1001_1001_0000_0001;
      internal_mem[1] = 16'b1001_1001_1000_0010;
      internal_mem[2] = 16'b0000_1001_1100_0000;
      internal_mem[3] = 16'b0000_1001_1001_0100;
      internal_mem[4] = 16'b1100_0100_0000_0010;
      internal_mem[5] = 16'b0000_1001_1100_0010;
      internal_mem[6] = 16'b0100_0000_0000_1000;
      internal_mem[7] = 16'b0000_1001_1100_0001;
      internal_mem[8] = 16'b0000_1001_1100_0011;
      internal_mem[9] = 16'b0110_0000_0000_1101;
      internal_mem[10] = 16'b1111_0010_0000_0010;
      internal_mem[11] = 16'b1011_1010_0000_0011;
      internal_mem[12] = 16'b0100_0000_0000_1111;
      internal_mem[13] = 16'b000_010_011_100_0110;
      internal_mem[14] = 16'b0001_1100_0000_1000;
      internal_mem[15] = 16'b0000_0000_0000_0000;
      internal_mem[16] = 16'b0000_0000_0000_0000;
      internal_mem[17] = 16'b0000_0000_0000_0000;
      internal_mem[18] = 16'b0000_0000_0000_0000;
      internal_mem[19] = 16'b0000_0000_0000_0000;
      internal_mem[20] = 16'b0000_0000_0000_0000;
      internal_mem[21] = 16'b0000_0000_0000_0000;
      internal_mem[22] = 16'b0000_0000_0000_0000;
      internal_mem[23] = 16'b0000_0000_0000_0000;
      end
  end
  assign instruction = (pc < 24) ? internal_mem[pc[4:0]] : 15'b0;
endmodule
```

- rst = 0 일 때 메모리 초기화
- 입력된 PC값에 해당하는 주소의 메모리값을 내보낸다

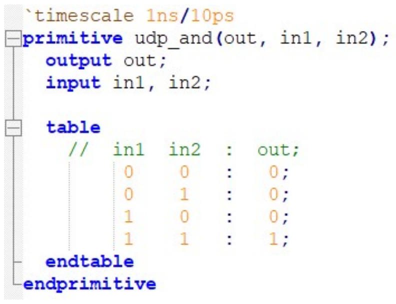
3) JR Controller



```
`timescale 1ns/10ps
module jrControl(
  input [1:0] alu_op,
  input [3:0] funct,
  output jr_control);

  assign jr_control = ({alu_op, funct} == 6'b001000) ? 1'b1 : 1'b0;
endmodule
```

- JR(Jump register) 명령어의 경우,
  {Opcode = 00, funct = 1000} 이 되어야 한다
- 위의 조건을 만족하면 컨트롤러가 1을 내보낸다

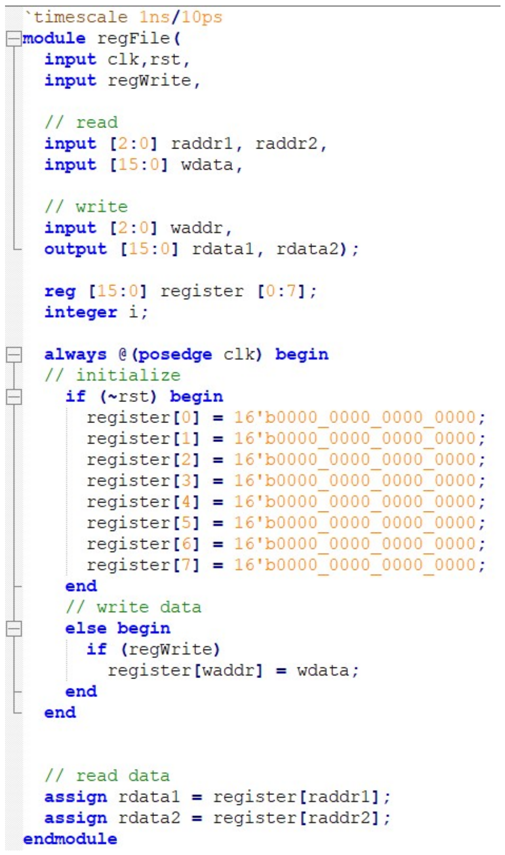
4) And Gate



```
`timescale 1ns/10ps
primitive udp_and(out, in1, in2);
  output out;
  input in1, in2;

  table
    //  in1  in2  :  out;
         0    0   :   0;
         0    1   :   0;
         1    0   :   0;
         1    1   :   1;
  endtable
endprimitive
```

1. Code

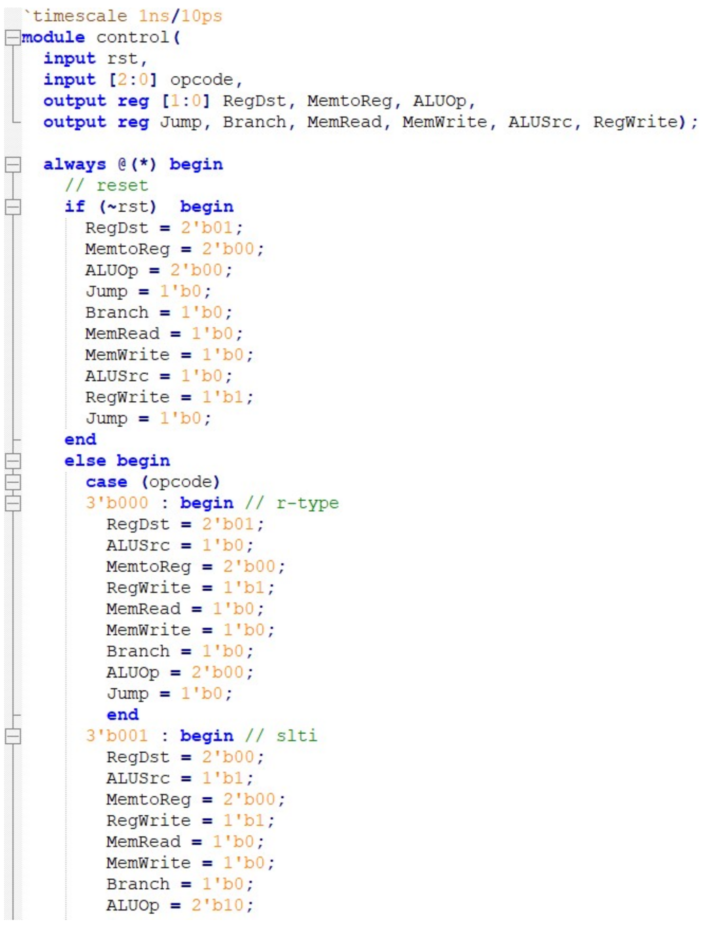
5) Register File



```
`timescale 1ns/10ps
module regFile(
  input clk,rst,
  input regWrite,

  // read
  input [2:0] raddr1, raddr2,
  input [15:0] wdata,

  // write
  input [2:0] waddr,
  output [15:0] rdata1, rdata2);

  reg [15:0] register [0:7];
  integer i;

  always @(posedge clk) begin
  // initialize
    if (~rst) begin
      register[0] = 16'b0000_0000_0000_0000;
      register[1] = 16'b0000_0000_0000_0000;
      register[2] = 16'b0000_0000_0000_0000;
      register[3] = 16'b0000_0000_0000_0000;
      register[4] = 16'b0000_0000_0000_0000;
      register[5] = 16'b0000_0000_0000_0000;
      register[6] = 16'b0000_0000_0000_0000;
      register[7] = 16'b0000_0000_0000_0000;
    end
    // write data
    else begin
      if (regWrite)
        register[waddr] = wdata;
    end
  end


  // read data
  assign rdata1 = register[raddr1];
  assign rdata2 = register[raddr2];
endmodule
```

- rst = 0 일 때 레지스터 초기화
- Write Register는 regWrite 컨트롤 신호가 들어올 때만 수행한다
- Read Register는 항상 수행

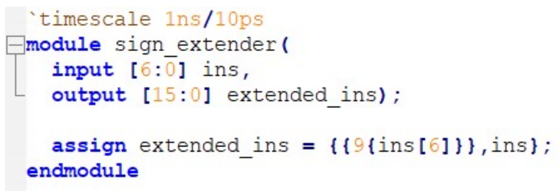
6) Main Control



```
`timescale 1ns/10ps
module control(
  input rst,
  input [2:0] opcode,
  output reg [1:0] RegDst, MemtoReg, ALUOp,
  output reg Jump, Branch, MemRead, MemWrite, ALUSrc, RegWrite);

  always @(*) begin
    // reset
    if (~rst)  begin
      RegDst = 2'b01;
      MemtoReg = 2'b00;
      ALUOp = 2'b00;
      Jump = 1'b0;
      Branch = 1'b0;
      MemRead = 1'b0;
      MemWrite = 1'b0;
      ALUSrc = 1'b0;
      RegWrite = 1'b1;
      Jump = 1'b0;
    end
    else begin
      case (opcode)
      3'b000 : begin // r-type
        RegDst = 2'b01;
        ALUSrc = 1'b0;
        MemtoReg = 2'b00;
        RegWrite = 1'b1;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        Branch = 1'b0;
        ALUOp = 2'b00;
        Jump = 1'b0;
        end
      3'b001 : begin // slti
        RegDst = 2'b00;
        ALUSrc = 1'b1;
        MemtoReg = 2'b00;
        RegWrite = 1'b1;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        Branch = 1'b0;
        ALUOp = 2'b10;
```

- 각 명령어의 opcode를 입력받아 해당하는 컨트롤 신호를 내보낸다
- 나머지 명령에 대한 코드는 길어서 **2페이지의 표**로 대체

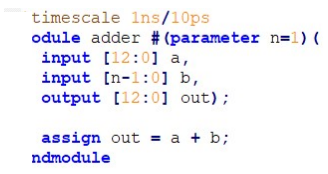
7) Sign_extender



```
`timescale 1ns/10ps
module sign_extender(
  input [6:0] ins,
  output [15:0] extended_ins);

  assign extended_ins = {{9{ins[6]}},ins};
endmodule
```

- I-type 명령어의 7-bit 값을 PC 값과 더해주기 위해 길이를 늘려준다. 양수(MSB[0])의 경우 0으로, 음수(MSG[1])의 경우 1로 상위 9-bit를 채워준다

8) Parameterized - Adder

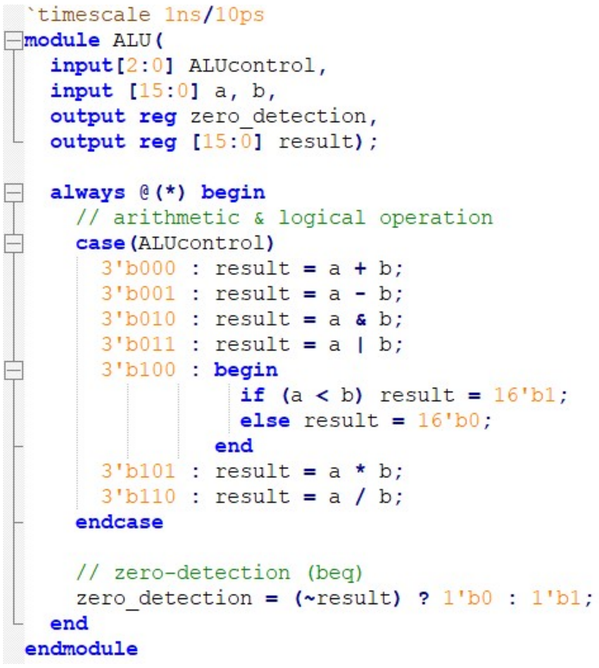


```
timescale 1ns/10ps
odule adder #(parameter n=1)(
 input [12:0] a,
 input [n-1:0] b,
 output [12:0] out);

 assign out = a + b;
ndmodule
```

- PC 값 계산을 위한 Adder

1. Code

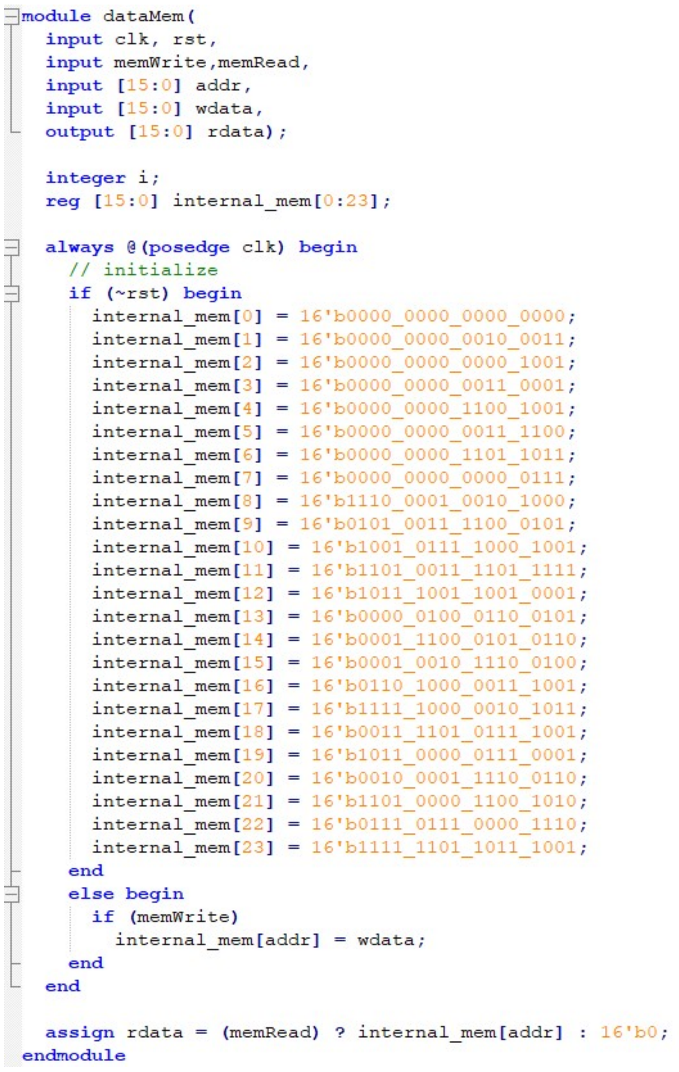
9) ALU



```verilog
`timescale 1ns/10ps
module ALU(
  input[2:0] ALUcontrol,
  input [15:0] a, b,
  output reg zero_detection,
  output reg [15:0] result);

  always @(*) begin
    // arithmetic & logical operation
    case(ALUcontrol)
      3'b000 : result = a + b;
      3'b001 : result = a - b;
      3'b010 : result = a & b;
      3'b011 : result = a | b;
      3'b100 : begin
                 if (a < b) result = 16'b1;
                 else result = 16'b0;
               end
      3'b101 : result = a * b;
      3'b110 : result = a / b;
    endcase

    // zero-detection (beq)
    zero_detection = (~result) ? 1'b0 : 1'b1;
  end
endmodule
```

- Arithmetic & Logic Operation 수행
- 입력으로 두 개의 operand를 받아 ALUcontrol 신호에 따른 연산 결과(result), zero-detection 결과 반환

10) Data Memory

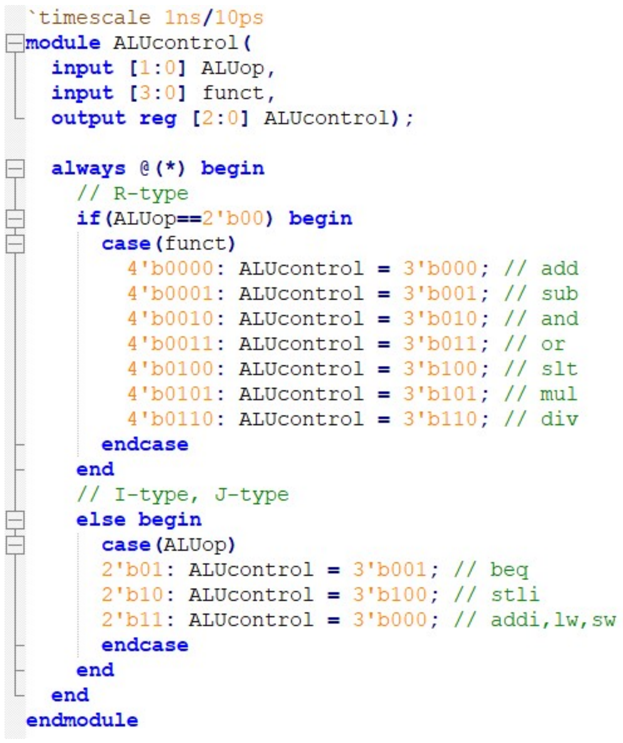


```verilog
module dataMem(
  input clk, rst,
  input memWrite,memRead,
  input [15:0] addr,
  input [15:0] wdata,
  output [15:0] rdata);

  integer i;
  reg [15:0] internal_mem[0:23];

  always @(posedge clk) begin
    // initialize
    if (~rst) begin
      internal_mem[0] = 16'b0000_0000_0000_0000;
      internal_mem[1] = 16'b0000_0000_0010_0011;
      internal_mem[2] = 16'b0000_0000_0000_1001;
      internal_mem[3] = 16'b0000_0000_0011_0001;
      internal_mem[4] = 16'b0000_0000_1100_1001;
      internal_mem[5] = 16'b0000_0000_0011_1100;
      internal_mem[6] = 16'b0000_0000_1101_1011;
      internal_mem[7] = 16'b0000_0000_0000_0111;
      internal_mem[8] = 16'b1110_0001_0010_1000;
      internal_mem[9] = 16'b0101_0011_1100_0101;
      internal_mem[10] = 16'b1001_0111_1000_1001;
      internal_mem[11] = 16'b1101_0011_1101_1111;
      internal_mem[12] = 16'b1011_1001_1001_0001;
      internal_mem[13] = 16'b0000_0100_0110_0101;
      internal_mem[14] = 16'b0001_1100_0101_0110;
      internal_mem[15] = 16'b0001_0010_1110_0100;
      internal_mem[16] = 16'b0110_1000_0011_1001;
      internal_mem[17] = 16'b1111_1000_0010_1011;
      internal_mem[18] = 16'b0011_1101_0111_1001;
      internal_mem[19] = 16'b1011_0000_0111_0001;
      internal_mem[20] = 16'b0010_0001_1110_0110;
      internal_mem[21] = 16'b1101_0000_1100_1010;
      internal_mem[22] = 16'b0111_0111_0000_1110;
      internal_mem[23] = 16'b1111_1101_1011_1001;
    end
    else begin
      if (memWrite)
        internal_mem[addr] = wdata;
    end
  end

  assign rdata = (memRead) ? internal_mem[addr] : 16'b0;
endmodule
```

- rst = 0 일 때 메모리 초기화
- Read Memory : memRead 컨트롤 신호가 1이 될 때만 해당하는 주소의 메모리값을 내보낸다
- Write Memory : memWrite 컨트롤 신호가 1이 될 때만 해당하는 주소에 값을 저장한다

1. Code

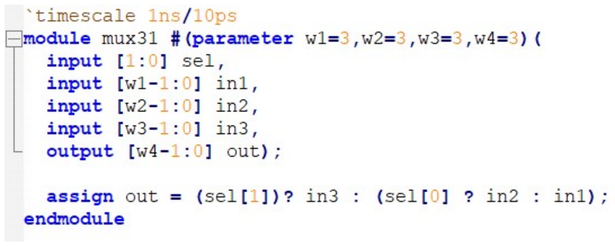
11) ALUcontrol



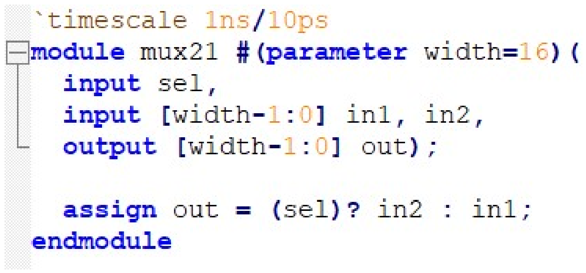
```
`timescale 1ns/10ps
module ALUcontrol(
  input [1:0] ALUop,
  input [3:0] funct,
  output reg [2:0] ALUcontrol);

  always @(*) begin
    // R-type
    if(ALUop==2'b00) begin
      case(funct)
        4'b0000: ALUcontrol = 3'b000; // add
        4'b0001: ALUcontrol = 3'b001; // sub
        4'b0010: ALUcontrol = 3'b010; // and
        4'b0011: ALUcontrol = 3'b011; // or
        4'b0100: ALUcontrol = 3'b100; // slt
        4'b0101: ALUcontrol = 3'b101; // mul
        4'b0110: ALUcontrol = 3'b110; // div
      endcase
    end
    // I-type, J-type
    else begin
      case(ALUop)
      2'b01: ALUcontrol = 3'b001; // beq
      2'b10: ALUcontrol = 3'b100; // stli
      2'b11: ALUcontrol = 3'b000; // addi,lw,sw
      endcase
    end
  end
endmodule
```

- 각 명령어에 따라 적합한 연산을 ALU에서 수행하기 위해 적절한 컨트롤 신호 생성

12) Parameterized - MUX



```
`timescale 1ns/10ps
module mux31 #(parameter w1=3,w2=3,w3=3,w4=3)(
  input [1:0] sel,
  input [w1-1:0] in1,
  input [w2-1:0] in2,
  input [w3-1:0] in3,
  output [w4-1:0] out);

  assign out = (sel[1])? in3 : (sel[0] ? in2 : in1);
endmodule
```

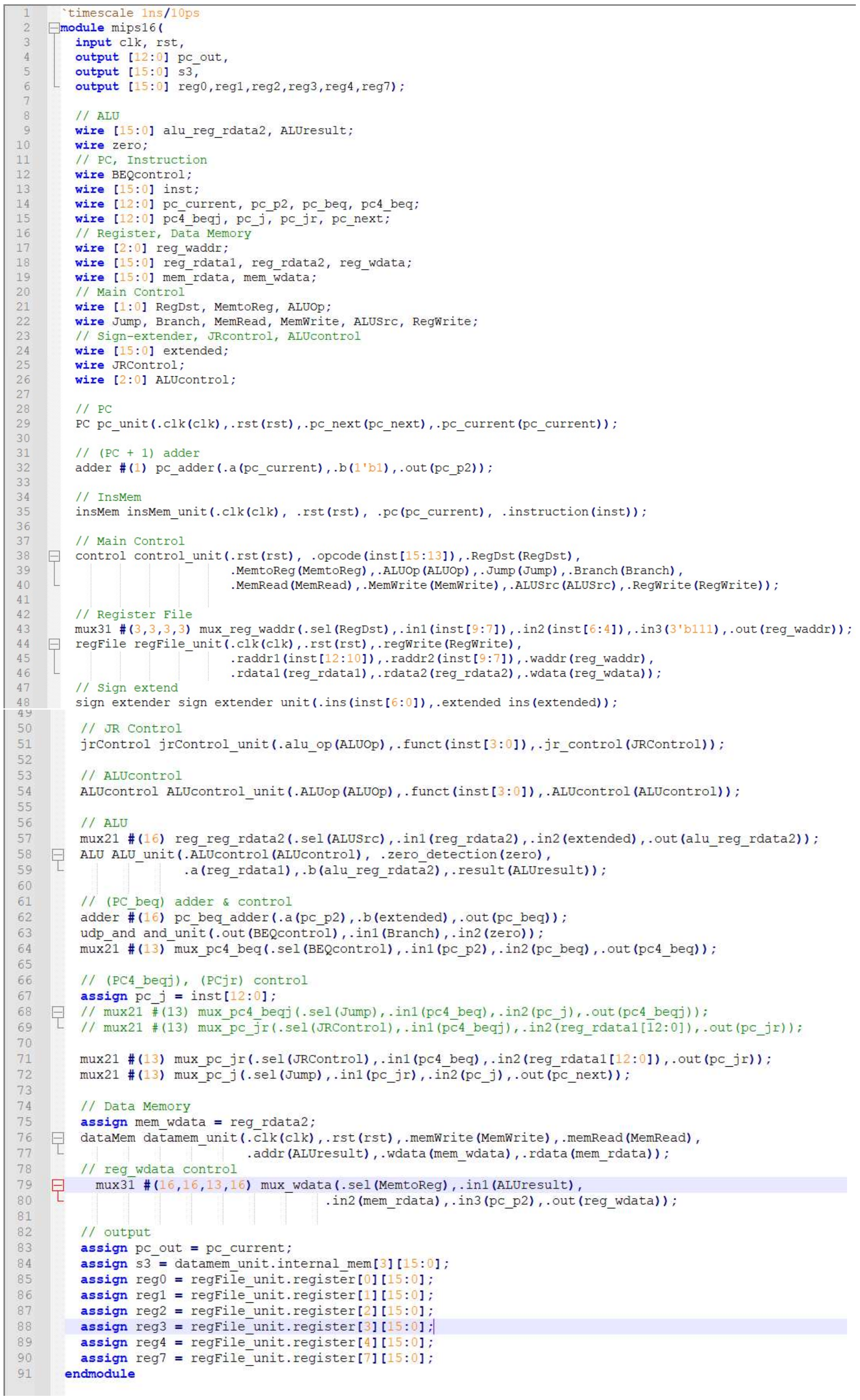


```
`timescale 1ns/10ps
module mux21 #(parameter width=16)(
  input sel,
  input [width-1:0] in1, in2,
  output [width-1:0] out);

  assign out = (sel)? in2 : in1;
endmodule
```

- 각 명령어에 따라 차별되는 동작, 또는 데이터 입력을 위한 MUX

1. Code

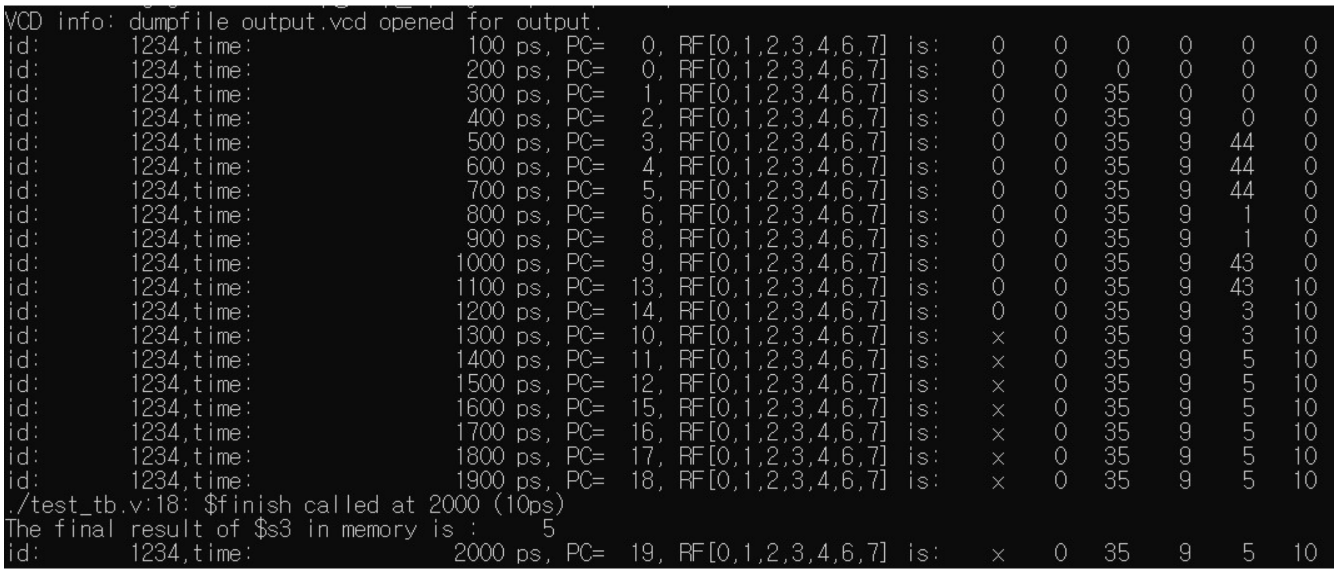
13) MIPS16 (Top Module)



```verilog
`timescale 1ns/10ps
module mips16(
  input clk, rst,
  output [12:0] pc_out,
  output [15:0] s3,
  output [15:0] reg0,reg1,reg2,reg3,reg4,reg7);

  // ALU
  wire [15:0] alu_reg_rdata2, ALUresult;
  wire zero;
  // PC, Instruction
  wire BEQcontrol;
  wire [15:0] inst;
  wire [12:0] pc_current, pc_p2, pc_beq, pc4_beq;
  wire [12:0] pc4_beqj, pc_j, pc_jr, pc_next;
  // Register, Data Memory
  wire [2:0] reg_waddr;
  wire [15:0] reg_rdata1, reg_rdata2, reg_wdata;
  wire [15:0] mem_rdata, mem_wdata;
  // Main Control
  wire [1:0] RegDst, MemtoReg, ALUOp;
  wire Jump, Branch, MemRead, MemWrite, ALUSrc, RegWrite;
  // Sign-extender, JRcontrol, ALUcontrol
  wire [15:0] extended;
  wire JRControl;
  wire [2:0] ALUcontrol;

  // PC
  PC pc_unit(.clk(clk),.rst(rst),.pc_next(pc_next),.pc_current(pc_current));

  // (PC + 1) adder
  adder #(1) pc_adder(.a(pc_current),.b(1'b1),.out(pc_p2));

  // InsMem
  insMem insMem_unit(.clk(clk), .rst(rst), .pc(pc_current), .instruction(inst));

  // Main Control
  control control_unit(.rst(rst), .opcode(inst[15:13]),.RegDst(RegDst),
                      .MemtoReg(MemtoReg),.ALUOp(ALUOp),.Jump(Jump),.Branch(Branch),
                      .MemRead(MemRead),.MemWrite(MemWrite),.ALUSrc(ALUSrc),.RegWrite(RegWrite));

  // Register File
  mux31 #(3,3,3,3) mux_reg_waddr(.sel(RegDst),.in1(inst[9:7]),.in2(inst[6:4]),.in3(3'b111),.out(reg_waddr));
  regFile regFile_unit(.clk(clk),.rst(rst),.regWrite(RegWrite),
                      .raddr1(inst[12:10]),.raddr2(inst[9:7]),.waddr(reg_waddr),
                      .rdata1(reg_rdata1),.rdata2(reg_rdata2),.wdata(reg_wdata));
  // Sign extend
  sign_extender sign_extender_unit(.ins(inst[6:0]),.extended_ins(extended));

  // JR Control
  jrControl jrControl_unit(.alu_op(ALUOp),.funct(inst[3:0]),.jr_control(JRControl));

  // ALUcontrol
  ALUcontrol ALUcontrol_unit(.ALUop(ALUOp),.funct(inst[3:0]),.ALUcontrol(ALUcontrol));

  // ALU
  mux21 #(16) reg_reg_rdata2(.sel(ALUSrc),.in1(reg_rdata2),.in2(extended),.out(alu_reg_rdata2));
  ALU ALU_unit(.ALUcontrol(ALUcontrol), .zero_detection(zero),
              .a(reg_rdata1),.b(alu_reg_rdata2),.result(ALUresult));

  // (PC_beq) adder & control
  adder #(16) pc_beq_adder(.a(pc_p2),.b(extended),.out(pc_beq));
  udp_and and_unit(.out(BEQcontrol),.in1(Branch),.in2(zero));
  mux21 #(13) mux_pc4_beq(.sel(BEQcontrol),.in1(pc_p2),.in2(pc_beq),.out(pc4_beq));

  // (PC4_beqj), (PCjr) control
  assign pc_j = inst[12:0];
  // mux21 #(13) mux_pc4_beqj(.sel(Jump),.in1(pc4_beq),.in2(pc_j),.out(pc4_beqj));
  // mux21 #(13) mux_pc_jr(.sel(JRControl),.in1(pc4_beqj),.in2(reg_rdata1[12:0]),.out(pc_jr));

  mux21 #(13) mux_pc_jr(.sel(JRControl),.in1(pc4_beq),.in2(reg_rdata1[12:0]),.out(pc_jr));
  mux21 #(13) mux_pc_j(.sel(Jump),.in1(pc_jr),.in2(pc_j),.out(pc_next));

  // Data Memory
  assign mem_wdata = reg_rdata2;
  dataMem datamem_unit(.clk(clk),.rst(rst),.memWrite(MemWrite),.memRead(MemRead),
                      .addr(ALUresult),.wdata(mem_wdata),.rdata(mem_rdata));
  // reg_wdata control
    mux31 #(16,16,13,16) mux_wdata(.sel(MemtoReg),.in1(ALUresult),
                                  .in2(mem_rdata),.in3(pc_p2),.out(reg_wdata));

  // output
  assign pc_out = pc_current;
  assign s3 = datamem_unit.internal_mem[3][15:0];
  assign reg0 = regFile_unit.register[0][15:0];
  assign reg1 = regFile_unit.register[1][15:0];
  assign reg2 = regFile_unit.register[2][15:0];
  assign reg3 = regFile_unit.register[3][15:0];
  assign reg4 = regFile_unit.register[4][15:0];
  assign reg7 = regFile_unit.register[7][15:0];
endmodule
```

- 2페이지의 Architecture를 참고하여
- 다만 JR 명령어의 경우 {ALUOp, funct} = 6'b001000이 되는데, 기존 구조에서는 J 8 명령어가 이와 같은 값을 가지게 되어 JR Controller에서 1을 내보내게 된다. 이 상황을 방지하기 위해 PC값 계산 시 사용되는 J-MUX와 JR-MUX 위치를 바꾸었다.

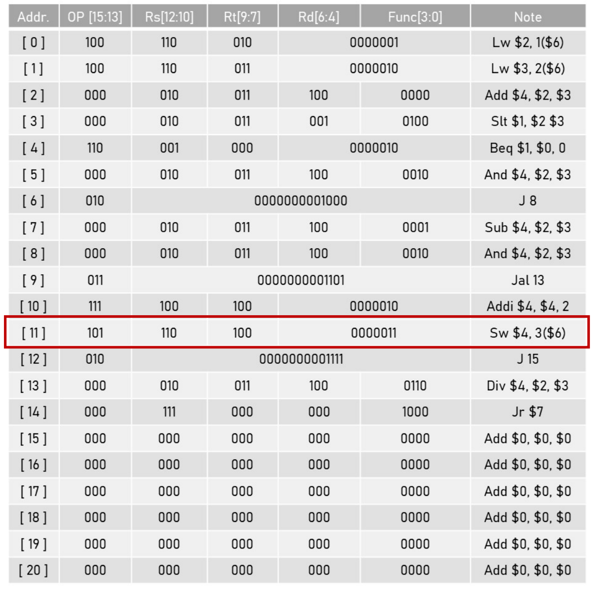
2. 결과



```
VCD info: dumpfile output.vcd opened for output.
id:       1234,time:                  100 ps, PC=   0, RF[0,1,2,3,4,6,7] is:   0    0    0    0    0    0
id:       1234,time:                  200 ps, PC=   0, RF[0,1,2,3,4,6,7] is:   0    0    0    0    0    0
id:       1234,time:                  300 ps, PC=   1, RF[0,1,2,3,4,6,7] is:   0    0   35    0    0    0
id:       1234,time:                  400 ps, PC=   2, RF[0,1,2,3,4,6,7] is:   0    0   35    9    0    0
id:       1234,time:                  500 ps, PC=   3, RF[0,1,2,3,4,6,7] is:   0    0   35    9   44    0
id:       1234,time:                  600 ps, PC=   4, RF[0,1,2,3,4,6,7] is:   0    0   35    9   44    0
id:       1234,time:                  700 ps, PC=   5, RF[0,1,2,3,4,6,7] is:   0    0   35    9   44    0
id:       1234,time:                  800 ps, PC=   6, RF[0,1,2,3,4,6,7] is:   0    0   35    9    1    0
id:       1234,time:                  900 ps, PC=   8, RF[0,1,2,3,4,6,7] is:   0    0   35    9    1    0
id:       1234,time:                 1000 ps, PC=   9, RF[0,1,2,3,4,6,7] is:   0    0   35    9   43    0
id:       1234,time:                 1100 ps, PC=  13, RF[0,1,2,3,4,6,7] is:   0    0   35    9   43   10
id:       1234,time:                 1200 ps, PC=  14, RF[0,1,2,3,4,6,7] is:   0    0   35    9    3   10
id:       1234,time:                 1300 ps, PC=  10, RF[0,1,2,3,4,6,7] is:   x    0   35    9    3   10
id:       1234,time:                 1400 ps, PC=  11, RF[0,1,2,3,4,6,7] is:   x    0   35    9    5   10
id:       1234,time:                 1500 ps, PC=  12, RF[0,1,2,3,4,6,7] is:   x    0   35    9    5   10
id:       1234,time:                 1600 ps, PC=  15, RF[0,1,2,3,4,6,7] is:   x    0   35    9    5   10
id:       1234,time:                 1700 ps, PC=  16, RF[0,1,2,3,4,6,7] is:   x    0   35    9    5   10
id:       1234,time:                 1800 ps, PC=  17, RF[0,1,2,3,4,6,7] is:   x    0   35    9    5   10
id:       1234,time:                 1900 ps, PC=  18, RF[0,1,2,3,4,6,7] is:   x    0   35    9    5   10
./test_tb.v:18: $finish called at 2000 (10ps)
The final result of $s3 in memory is :      5
id:       1234,time:                 2000 ps, PC=  19, RF[0,1,2,3,4,6,7] is:   x    0   35    9    5   10
```

※ internal_mem[3] 값을 가져오게 된 이유

→ 오른쪽은 Instruction Memory 초기화 값을 해석한 것이다. [11] 명령어에 의해 $s3값이 Mem[3($6)]에 저장되게 된다. $6에 저장된 값이 0이므로, Mem[3($6)]은 Mem[3]과 같은 의미가 된다



Addr.	OP [15:13]	Rs[12:10]	Rt[9:7]	Rd[6:4]	Func[3:0]	Note
[0]	100	110	010	0000001		Lw $2, 1($6)
[1]	100	110	011	0000010		Lw $3, 2($6)
[2]	000	010	011	100	0000	Add $4, $2, $3
[3]	000	010	011	001	0100	Slt $1, $2 $3
[4]	110	001	000	0000010		Beq $1, $0, 0
[5]	000	010	011	100	0010	And $4, $2, $3
[6]	010	0000000001000				J 8
[7]	000	010	011	100	0001	Sub $4, $2, $3
[8]	000	010	011	100	0010	And $4, $2, $3
[9]	011	0000000001101				Jal 13
[10]	111	100	100	0000010		Addi $4, $4, 2
[11]	101	110	100	0000011		Sw $4, 3($6)
[12]	010	0000000001111				J 15
[13]	000	010	011	100	0110	Div $4, $2, $3
[14]	000	111	000	000	1000	Jr $7
[15]	000	000	000	000	0000	Add $0, $0, $0
[16]	000	000	000	000	0000	Add $0, $0, $0
[17]	000	000	000	000	0000	Add $0, $0, $0
[18]	000	000	000	000	0000	Add $0, $0, $0
[19]	000	000	000	000	0000	Add $0, $0, $0
[20]	000	000	000	000	0000	Add $0, $0, $0