Lab5_单周期 mips-cpu 设计

金泽文 PB15111604

实验目的：

设计单周期 mips-cpu。

实验内容：

- 设计 CPU ,完成以下程序代码的执行 ,其功能是起始数为 3 和 3 的斐波拉契数列的计算。只计算 20 个数。
- 本次实验要求设计为单周期 CPU，基本思路是依据给定过的指令集（6 条），设计核心的控制信号。依据前面给定的数据通路和控制单元信号进行设计。
- 注意现在涉及到两个 ram，一个 regfile，现在均要求是异步读，同步写。

实验分析与设计：

首先要考虑的是 6 条指令的解码，通过 control 模块解码对应 opcode，由于 add 与 addi 的前 opcode 不同，所以考虑的 case 情况为 7+1 个（还有一个 default）。

根据不同的情况，我们需要设置以下变量，方便 top 模

```
3  output reg memtoreg,
4  output reg memwrite,
5  output reg branch,
6  output reg [2:0] alucontrol,
7  output reg alusrc,
8  output reg regdst,
9  output reg regwrite,
10 output reg jump
11 );
```

块使用。大多数用来作为 mux 的输入。

解码 opcode 之后在 top 模块中需要构造数据通路。

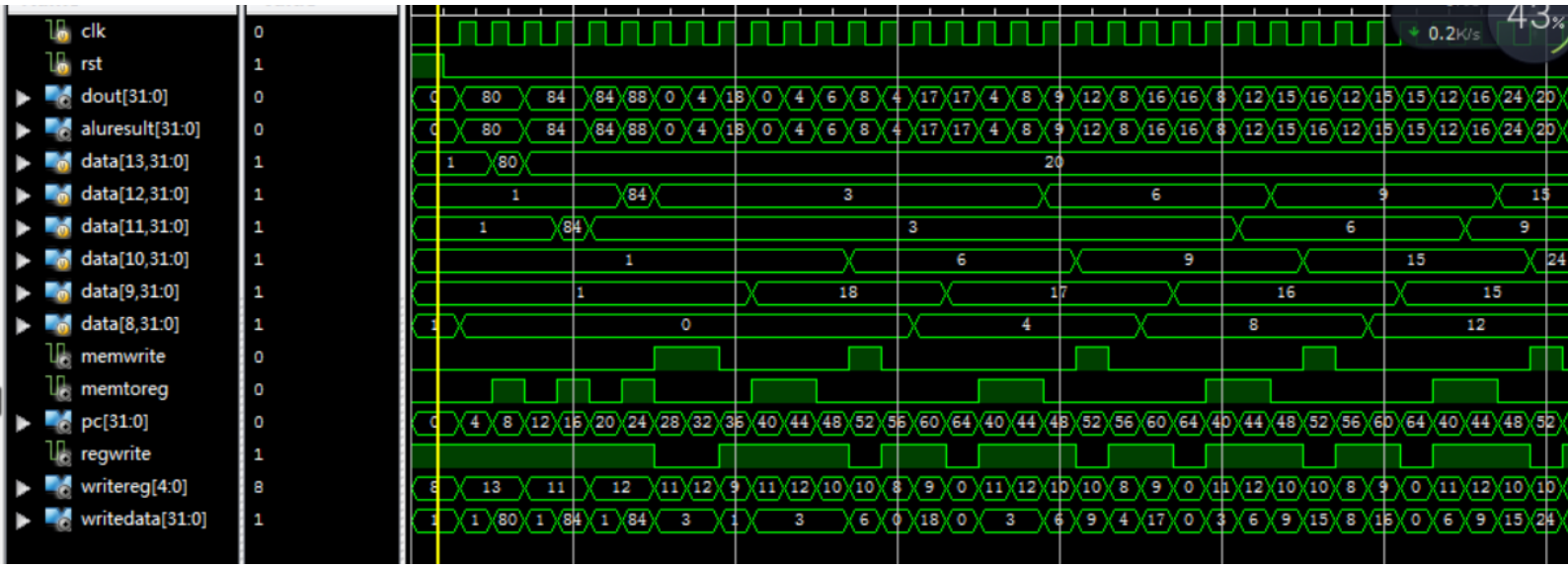
对于 ALU 模块，不同于以往，需要设置 bgtz 输出变量。

对于 regfile，沿用之前的。

除此之外，需要注意的是，由于 data_mem 一个地址对应的是 4 个字节，所以在计算对应 data_mem 地址的时候需要考虑截断后两位。

实验结果：

仿真波形：



数据对应内存：

	0	1	2	3
0x0	3	3	6	9
0x4	15	24	39	63
0x8	102	165	267	432
0xC	699	1131	1830	2961
0x10	4791	7752	12543	20295
0x14	20	3	3	0
0x18	0	0	0	0
0x1C	0	0	0	0
0x20	0	0	0	0

实验代码：

Top

```
module top(
input clk,
input rst,
output[31:0] dout
);

wire [31:0] pcjump;
wire [31:0] pc2;
wire [31:0] pc1;
wire [31:0] pc;
wire [31:0] pcplus4;
wire [31:0] instr;
wire memtoreg;
wire memwrite;
wire branch;
wire [2:0] alucontrol;
wire alusrc;
wire regdst;
wire regwrite;
wire jump;
wire pcsrc;
wire [4:0] writereg;
reg [31:0] signimm;
wire [31:0] signimml2;
wire [31:0] srca;
wire [31:0] srcb;
wire bgtz;
wire [31:0] aluresult;
wire [31:0] writedata;
wire [31:0] pcbranch;
wire [31:0] readdata;
wire [31:0] result;
```

```
assign pcjump[31:28]=
    pcplus4[31:28];
assign pcjump[27:2] = instr
    [25:0];
assign pcjump[1:0] = 2'h0;
assign signimml2[31:2]=
    signimm[29:0];
assign signimml2[1:0] =
    2'h0;
assign pcplus4 = pc + 32'h4
    ;
assign pcbranch =
    signimml2 + pcplus4;
assign pcsrc = branch &
    bgtz;
assign pc2 = pcsrc ?
    pcbranch : pcplus4;
assign pc1 = jump ? pcjump
    :pc2;
assign writereg = regdst ?
    instr[15:11] : instr[20
    :16];
assign srcb = alusrc ?
    signimm :writedata;
assign result = memtoreg ?
    readdata: aluresult;
assign dout = aluresult;

always@(*)
begin
    if(instr[15])
        signimm =
```

```
                32'hffff0000
                + instr[15:0]
                ;
    else
        signimm =
                32'h00000000
                + instr[15:0]
                ;
end

pc_reg u_pc_reg(
.clk (clk ),
.rst (rst ),
.pc1 (pc1 ),
.pc (pc )
);

control u_control(
.opcode (instr[31:26]),
.memtoreg (memtoreg ),
.memwrite (memwrite ),
.branch (branch ),
.alucontrol (alucontrol )
    ,
.alusrc (alusrc ),
.regdst (regdst ),
.regwrite (regwrite ),
.jump (jump )
);

ins_mem u_ins_mem(
.a (pc[9:2] ),
.spo (instr )
);

REG_FILE u_REG_FILE(
.clk (clk ),
.rst (rst),
.r1_addr (instr[25:21]),
.r2_addr (instr[20:16]),
.r3_addr (writereg ),
.r3_din (result ),
.r3_wr (regwrite ),
.r1_dout (srca ),
.r2_dout (writedata )
);

alu u_alu(
.alu_a (srca ),
.alu_b (srcb ),
.alu_op (alucontrol ),
.alu_out (aluresult ),
.alu_bgtz (bgtz )
);

data_mem u_data_mem(
.clk (clk ),
.we (memwrite ),
.a (aluresult[11:2] ),
.d (writedata ),
.spo (readdata ));

endmodule
```

Reg_file

```
module REG_FILE(
input clk,
input rst,
input [4:0] r1_addr,
input [4:0] r2_addr,
input [4:0] r3_addr,
input [31:0] r3_din,
input r3_wr,
output reg [31:0] r1_dout,
output reg [31:0] r2_dout
);
reg [31:0] data [31:0];
always@(posedge clk or
    posedge rst)
begin
    if (rst) begin
        data[0] <= 32'b1;
        data[1] <= 32'b1;
        data[2] <= 32'b1;
        data[3] <= 32'b1;
        data[4] <= 32'b1;
        data[5] <= 32'b1;
        data[6] <= 32'b1;
        data[7] <= 32'b1;
        data[8] <= 32'b1;
        data[9] <= 32'b1;
        data[10] <= 32'b1;
        data[11] <= 32'b1;
        data[12] <= 32'b1;
        data[13] <= 32'b1;
        data[14] <= 32'b1;
        data[15] <= 32'b1;
        data[16] <= 32'b1;
        data[17] <= 32'b1;
        data[18] <= 32'b1;
        data[19] <= 32'b1;
        data[20] <= 32'b1;
        data[21] <= 32'b1;
        data[22] <= 32'b1;
        data[23] <= 32'b1;
        data[24] <= 32'b1;
        data[25] <= 32'b1;
        data[26] <= 32'b1;
        data[27] <= 32'b1;
        data[28] <= 32'b1;
        data[29] <= 32'b1;
        data[30] <= 32'b1;
        data[31] <= 32'b1;
    end
    else if(r3_wr)
        data[r3_addr]<= r3_din;
end

always@(*)
begin
    if(r1_addr)
        r1_dout = data[r1_addr];
    else
        r1_dout = 32'h0;
end

always@(*)begin
    if(r2_addr)
        r2_dout = data[r2_addr];
    else
        r2_dout = 32'h0;
end
endmodule

```

Alu

```
module alu(
input signed [31:0] alu_a,
input signed [31:0] alu_b,
input [2:0] alu_op,
output reg [31:0] alu_out,
output reg alu_bgtz
);
always@(*)
    begin
        case(alu_op)
            3'h00: alu_out = 32'h0;
            3'h01: alu_out = alu_a + alu_b;
            3'h02: alu_out = alu_a - alu_b;
            3'h03: alu_out = alu_a & alu_b;
            3'h04: alu_out = alu_a | alu_b;
            3'h05: alu_out = alu_a ^ alu_b;
            3'h06: alu_out = ~(alu_a | alu_b);
            default: alu_out = 32'h0;
        endcase
    end
always@(*) //for bgtz
begin
    if(alu_out <= 0)
        alu_bgtz = 1'h0;
    else
        alu_bgtz = 1'h1;
end
endmodule
```

pc_reg

```
module pc_reg(
input clk,
input rst,
input [31:0] pc1,
output reg [31:0] pc
);

always@(posedge clk or posedge rst)
begin
    if(rst)
        pc <= 32'h0;
    else
        pc <= pc1;
end
endmodule
```

control

```
module control(
input [5:0] opcode,
output reg memtoreg,
output reg memwrite,
output reg branch,
output reg [2:0] alucontrol,
output reg alusrc,
output reg regdst,
output reg regwrite,
output reg jump
);
always@(*)
begin
    case(opcode)
        6'b001000: //la
            begin
                memtoreg = 0;
                memwrite = 0;
                branch = 0;
                alucontrol = 3'h1;
                alusrc = 1;
                regdst = 0;
                regwrite = 1;
                jump = 0;
            end
        6'b100011: //lw
            begin
                memtoreg = 1;
                memwrite = 0;
                branch = 0;
                alucontrol = 3'h1;
                alusrc = 1;
                regdst = 0;
                regwrite = 1;
                jump = 0;
            end
        6'b101011: //sw
            begin
                memtoreg = 0;
                memwrite = 1;
                branch = 0;
                alucontrol = 3'h1;
                alusrc = 1;
                regdst = 0;
                regwrite = 0;
                jump = 0;
            end
        6'b001000: //addi
            begin
                memtoreg = 0;
                memwrite = 0;
                branch = 0;
                alucontrol = 3'h1;
                alusrc = 1;
                regdst = 0;
                regwrite = 1;
                jump = 0;
            end
        6'b000000: //add
            begin
                memtoreg = 0;
                memwrite = 0;
                branch = 0;
                alucontrol =
                    3'h0;
                alusrc = 0;
                regdst = 0;
                regwrite = 0;
                jump = 1;
            end
        default:
            begin
                memtoreg = 0;
                memwrite = 0;
                branch = 0;
                alucontrol =
                    3'h0;
                alusrc = 0;
                regdst = 0;
                regwrite = 0;
                jump = 0;
            end
    endcase
end

endmodule
```