Lab6_多周期 mips-cpu 设计

金泽文 PB15111604

实验目的：

设计多周期 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。
- 实验设计中可以不使用给定的数据通路和状态机，但仅允许使用一个存储器。
- 对指令/数据存储器的附加要求：
    - 使用异步存储器，最高评分为√√
    - 使用同步存储器，最高评分为√√√，使用同步存储器时，需要对数据通路和状态机进行适当修改。

实验分析与设计：

我采用了同步存储器。针对 ppt 中给出的状态机，出于对时延的考虑，在 S3 与 S4 之间，S5 之后，S8 之后，分别加了一个等待状态。

Top 和 control 基本按照 ppt 中数据通路实现。需要指出的是，我省略了 2 位的 AluOp，直接使用了 3 位的 AluControl，除了 S8 对应的 beq 需要“-”运算，其他都是“+”运算。

对于 ALU 模块，不同于以往，需要设置 zero 输出变量。

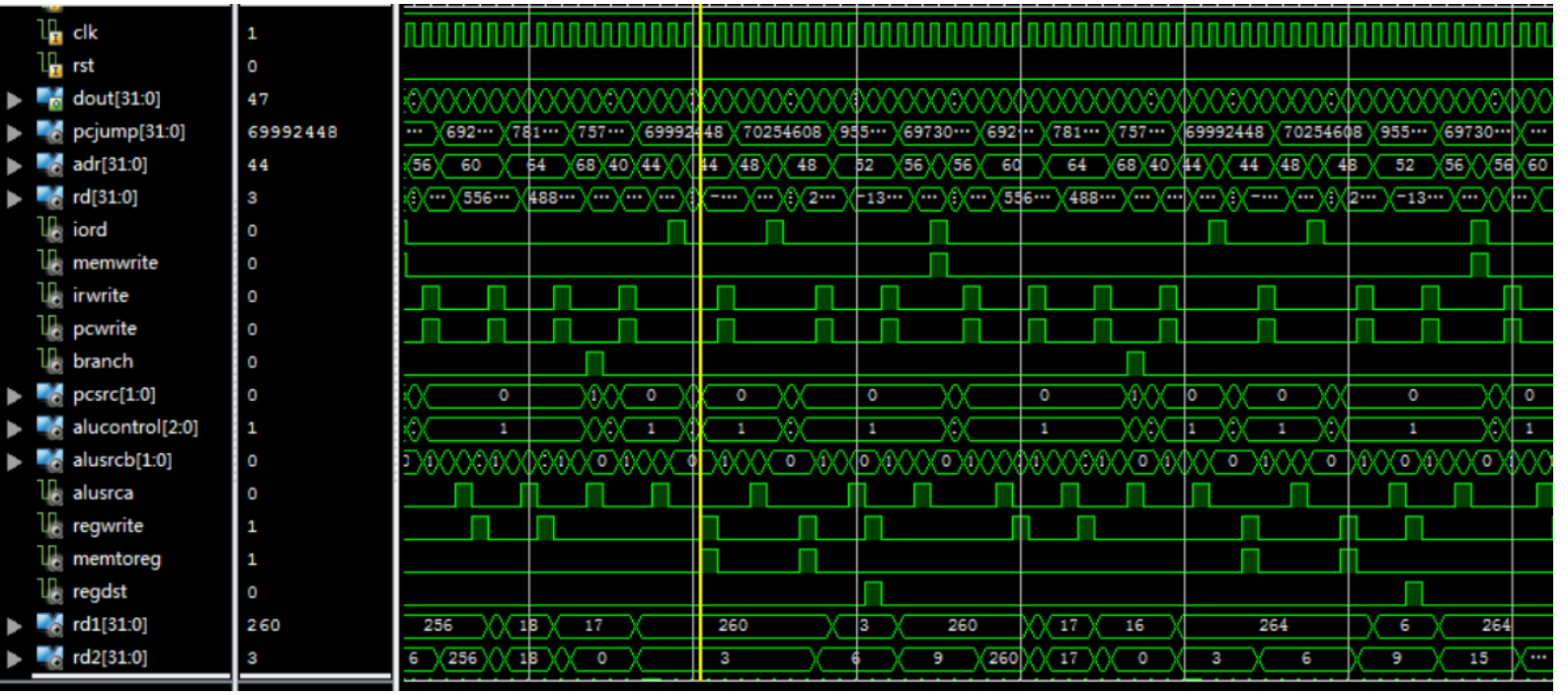
对于 regfile，沿用之前。

除此之外，需要注意的是，由于 mem 一个地址对应的是 4 个字节，所以在计算对应 mem 地址的时候需要考虑截断后两位。同时，为了方便仿真时的观察，将数据输出首地址改为 0x40，对应的 coe 需要修改四处：

20080100，200d0150，200b0154，200c0154

实验结果：

仿真波形：



内存：

0x34	0	0	0	0
0x38	0	0	0	0
0x3C	0	0	0	0
0x40	3	3	6	9
0x44	15	24	39	63
0x48	102	165	267	432
0x4C	699	1131	1830	2961
0x50	4791	7752	12543	20295
0x54	20	3	3	0

实验代码：

Top：

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

reg     [31:0]      pc1;
reg     [31:0]      pc;
wire    [31:0]      pcjump;
wire    [31:0]      adr;
wire    [31:0]      rd;
reg     [31:0]      instr;
reg     [31:0]      data;
wire                iord;
wire                memwrite;
wire                irwrite;
wire                pcwrite;
wire                branch;
wire    [1:0]       pcsrc;
wire    [2:0]       alucontrol;
wire    [1:0]       alusrcb;
wire                alusrca;
wire                regwrite;
wire                memtoreg;
wire                regdst;
wire    [31:0]      rd1;
wire    [31:0]      rd2;
reg     [31:0]      a;
reg     [31:0]      b;
wire    [31:0]      result;
wire    [4:0]       writereg;
reg     [31:0]      signimm;
wire    [31:0]      signimml2;
wire    [31:0]      srca;
reg     [31:0]      srcb;
wire                zero;
wire    [31:0]      aluresult;
reg     [31:0]      aluout;

always@(posedge clk or posedge rst)
begin
    if(rst)
        pc <= 32'b0;
    else if(pcen)
        pc <= pc1;
end

assign adr = iord ? aluout : pc;

always@(posedge clk)
begin
    if(irwrite)
        instr <= rd;
end

always@(posedge clk)
begin
    data <= rd;
end

assign writereg = regdst ? instr[15:11] : instr[20:16];

assign result = memtoreg ? data : aluout;

always@(*)
begin
    if(instr[15])
        signimm = 32'hffff0000 + instr[15:0];
    else
        signimm = 32'h00000000 + instr[15:0];
end

assign signimml2[31:2]= signimm[29:0];
assign signimml2[1:0] = 2'h0;

always@(posedge clk)
begin
    a <= rd1;
end

always@(posedge clk)
begin
    b <= rd2;
end

assign srca = alusrca ? a : pc;

always@(*)
begin
    case(alusrcb)
        2'b00: srcb = b;
        2'b01: srcb = 32'h4;
        2'b10: srcb = signimm;
        2'b11: srcb = signimml2;
    endcase
end

assign pcjump[31:28] = pc[31:28];
assign pcjump[27:2] = instr[25:0];
assign pcjump[1:0] = 2'h0;

assign pcen = pcwrite | (branch & zero);

always@(posedge clk)
begin
    aluout <= aluresult;
end

always@(*)
begin
    case(pcsrc)
        2'b00: pc1 = aluresult;
        2'b01: pc1 = aluout;
        2'b10: pc1 = pcjump;
        default: pc1 = 32'h0;
    endcase
end

assign dout = aluresult;

control     u_control(
.opcode     (instr[31:26]       ),
.funct      (instr[5:0]         ),
.clk        (clk                ),
.rst        (rst                ),
.iord       (iord               ),
.memwrite   (memwrite           ),
.irwrite    (irwrite            ),
.pcwrite    (pcwrite            ),
.branch     (branch             ),
.pcsrc      (pcsrc              ),
.alucontrol (alucontrol[2:0]    ),
.alusrcb    (alusrcb[1:0]       ),
.alusrca    (alusrca            ),
.regwrite   (regwrite           ),
.memtoreg   (memtoreg           ),
.regdst     (regdst             )
);

mem         u_mem(
.clka       (clk                ),
.wea        (memwrite           ),
.addra      (adr[8:2]           ),
.dina       (b[31:0]            ),
.douta      (rd                 )
);

REG_FILE    u_REG_FILE(
.clk        (clk                ),
.r1_addr    (instr[25:21]       ),
.r2_addr    (instr[20:16]       ),
.r3_addr    (writereg           ),
.r3_din     (result             ),
.r3_wr      (regwrite           ),
.r1_dout    (rd1                ),
.r2_dout    (rd2                )
);

alu         u_alu(
.alu_a      (srca               ),
.alu_b      (srcb               ),
.alu_op     (alucontrol         ),
.alu_out    (aluresult          ),
.alu_zero   (zero               )
);

endmodule
```

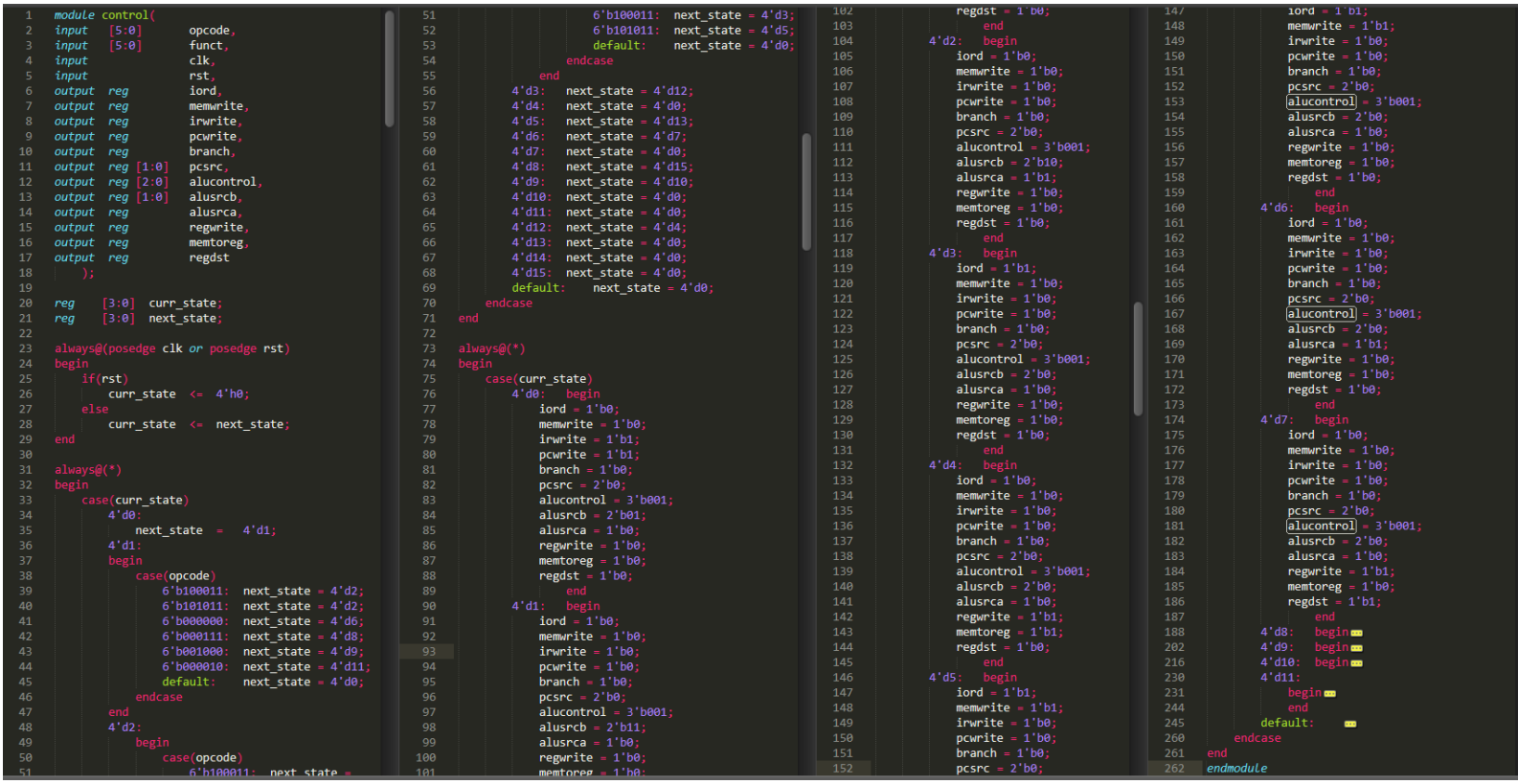
Reg_file

```
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_zero
);

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@(*)
begin
    if(alu_out > 0)
        alu_zero = 1'h1;
    else
        alu_zero = 1'h0;
end

endmodule

```

control



```
module control(
input   [5:0]       opcode,
input   [5:0]       funct,
input               clk,
input               rst,
output  reg         iord,
output  reg         memwrite,
output  reg         irwrite,
output  reg         pcwrite,
output  reg         branch,
output  reg [1:0]   pcsrc,
output  reg [2:0]   alucontrol,
output  reg [1:0]   alusrcb,
output  reg         alusrca,
output  reg         regwrite,
output  reg         memtoreg,
output  reg         regdst
    );

reg    [3:0]  curr_state;
reg    [3:0]  next_state;

always@(posedge clk or posedge rst)
begin
    if(rst)
        curr_state  <=  4'h0;
    else
        curr_state  <=  next_state;
end

always@(*)
begin
    case(curr_state)
        4'd0:
            next_state  =   4'd1;
        4'd1:
        begin
            case(opcode)
                6'b100011:  next_state = 4'd2;
                6'b101011:  next_state = 4'd2;
                6'b000000:  next_state = 4'd6;
                6'b000111:  next_state = 4'd8;
                6'b001000:  next_state = 4'd9;
                6'b000010:  next_state = 4'd11;
                default:    next_state = 4'd0;
            endcase
        end
        4'd2:
            begin
                case(opcode)
                    6'b100011:  next_state =
                    6'b101011:  next_state = 4'd5;
                    default:    next_state = 4'd0;
                endcase
            end
        4'd3:   next_state = 4'd12;
        4'd4:   next_state = 4'd0;
        4'd5:   next_state = 4'd13;
        4'd6:   next_state = 4'd7;
        4'd7:   next_state = 4'd0;
        4'd8:   next_state = 4'd15;
        4'd9:   next_state = 4'd10;
        4'd10:  next_state = 4'd0;
        4'd11:  next_state = 4'd0;
        4'd12:  next_state = 4'd4;
        4'd13:  next_state = 4'd0;
        4'd14:  next_state = 4'd0;
        4'd15:  next_state = 4'd0;
        default:    next_state = 4'd0;
    endcase
end

always@(*)
begin
    case(curr_state)
        4'd0:   begin
            iord = 1'b0;
            memwrite = 1'b0;
            irwrite = 1'b1;
            pcwrite = 1'b1;
            branch = 1'b0;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b01;
            alusrca = 1'b0;
            regwrite = 1'b0;
            memtoreg = 1'b0;
            regdst = 1'b0;
                end
        4'd1:   begin
            iord = 1'b0;
            memwrite = 1'b0;
            irwrite = 1'b0;
            pcwrite = 1'b0;
            branch = 1'b1;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b11;
            alusrca = 1'b0;
            regwrite = 1'b0;
            memtoreg = 1'b0;
            regdst = 1'b0;
                end
        4'd2:   begin
            iord = 1'b0;
            memwrite = 1'b0;
            irwrite = 1'b0;
            pcwrite = 1'b0;
            branch = 1'b0;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b10;
            alusrca = 1'b1;
            regwrite = 1'b0;
            memtoreg = 1'b0;
            regdst = 1'b0;
                end
        4'd3:   begin
            iord = 1'b1;
            memwrite = 1'b0;
            irwrite = 1'b0;
            pcwrite = 1'b0;
            branch = 1'b0;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b0;
            alusrca = 1'b0;
            regwrite = 1'b0;
            memtoreg = 1'b0;
            regdst = 1'b0;
                end
        4'd4:   begin
            iord = 1'b0;
            memwrite = 1'b0;
            irwrite = 1'b0;
            pcwrite = 1'b0;
            branch = 1'b0;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b0;
            alusrca = 1'b0;
            regwrite = 1'b1;
            memtoreg = 1'b1;
            regdst = 1'b0;
                end
        4'd5:   begin
            iord = 1'b1;
            memwrite = 1'b1;
            irwrite = 1'b0;
            pcwrite = 1'b0;
            branch = 1'b0;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b0;
            alusrca = 1'b1;
            regwrite = 1'b0;
            memtoreg = 1'b0;
            regdst = 1'b0;
                end
        4'd6:   begin
            iord = 1'b0;
            memwrite = 1'b0;
            irwrite = 1'b0;
            pcwrite = 1'b0;
            branch = 1'b0;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b0;
            alusrca = 1'b0;
            regwrite = 1'b0;
            memtoreg = 1'b0;
            regdst = 1'b0;
                end
        4'd7:   begin
            iord = 1'b0;
            memwrite = 1'b0;
            irwrite = 1'b0;
            pcwrite = 1'b0;
            branch = 1'b0;
            pcsrc = 2'b0;
            alucontrol = 3'b001;
            alusrcb = 2'b0;
            alusrca = 1'b0;
            regwrite = 1'b1;
            memtoreg = 1'b0;
            regdst = 1'b1;
                end
        4'd8:   begin...
        4'd9:   begin...
        4'd10:  begin...
        4'd11:
            begin...
            end
        default:    ...
    endcase
end
endmodule
```