Lab4
- 逻辑设计
  - cpu
  - DBU
- 核心代码
  - control
  - ALU & regfile
  - top
  - DBU
- 仿真结果
- 结果分析
- 实验总结
- 意见建议
- 思考题(以accm指令为例)

Lab4

逻辑设计

cpu

相比于`单周期cpu`, 主要的不同点在于:

- `data`和`instr`存在一个`dist_mem`中
- 新增了`IR`, `MEMDATAREAD`两个寄存器存储`mem`读结果
- 新增了`A`, `B`, `ALUOut`存储`regfile`的`rd1`, `rd2`和`ALU`的运算结果
- `PC`的自增运算和跳转不需要单独的模块(因此只需要一个`ALU`)
- `control`模块的变化

DBU

只是增加了几个读端口

核心代码

control

这分出了若干个状态, 采用三段式描述

```
 //------n_state---------//
    always @(*) begin
        case (c_state)
            4'd0: n_state=1;
            4'd1: // decode
                case (opcode)
                    6'b000000: n_state = 4'd6; //R-ex
                    6'b100011: n_state = 4'd2;//lw
                    6'b101011: n_state = 4'd2;//sw
                    6'b000100: n_state = 4'd8;//beq
                    6'b000010: n_state = 4'd9;//jump
                    6'b001000: n_state = 4'd10;//addi
                    default: ;
                endcase
            4'd2: //MEMaddr
                case (opcode)
                    6'b100011: n_state=4'd3;
                    6'b101011: n_state=4'd5;
                    default: ;
                endcase
            4'd3: //memread
                n_state = 4'd4;
```

```
            4'd4: //memwriteback
                n_state = 4'd0;
            4'd5: //memwrite
                n_state = 4'd0;
            4'd6: //execute
                n_state = 4'd7;
            4'd7: //aluwriteback
                n_state = 4'd0;
            4'd8: //branch
                n_state = 4'd0;
            4'd9: //jump
                n_state = 4'd0;
            4'd10: //addi
                n_state = 4'd11;
            4'd11: //addi writeback
                n_state = 4'd0;
            4'd15: // rst
                n_state = 4'd0;
            default: ;
        endcase
    end
```

```
//------signal----------//
    always @(*)begin
        if(rst) begin
            IorD = 1'b0;
            PCSource = 2'b11; //nextPC=0;
            PCWrite = 1'b1;
        end
        else
            case(c_state)
                4'd0://fetch
                begin
                    PCWriteCond = 0;
                    IorD = 1'b0; // Memaddr: PC
                    MemRead = 1'b1; // Mem read
                    MemWrite = 1'b0;
```

```
16                      IRWrite = 1'b1; // save
Instr
17                      RegDst = 1'b0;
18                      MemtoReg = 1'b0;
19                      RegWrite = 1'b0;
20                      ALUSrcA = 1'b0; // srcA: PC
21                      ALUSrcB = 2'b01; // srcB: 4
22                      ALUcontrol = 3'b000; //
ALU's func: add
23                      Branch = 1'b0;
24                      PCWrite = 1'b1; // enable
update PC
25                      PCSource = 2'b00; // select
neOtPC=PC+4
26                  end
27                  4'd1://decode
28                  begin
29                      PCWriteCond = 0;
30                      IorD = 1'b0;
31                      MemRead = 1'b0;
32                      MemWrite = 1'b0;
33                      IRWrite = 1'b0;
34                      RegDst = 1'b0;
35                      MemtoReg = 1'b0;
36                      RegWrite = 1'b0;
37                      ALUSrcA = 1'b0; // srcA: PC
38                      ALUSrcB = 2'b11; // srcB:
SignEOtended<<2
39                      ALUcontrol = 3'b000; //
ALU's func: add
40                      Branch = 1'b0;
41                      PCWrite = 1'b0;
42                      PCSource = 2'b00;
43                  end
44                  4'd2: //memaddr
45                  begin
46                      PCWriteCond = 0;
```

```
47                      IorD = 1'b0;
48                      MemRead = 1'b0;
49                      MemWrite = 1'b0;
50                      IRWrite = 1'b0;
51                      RegDst = 1'b0;
52                      MemtoReg = 1'b0;
53                      RegWrite = 1'b0;
54                      ALUSrcA = 1'b1; // srcA:
RegRdout1
55                      ALUSrcB = 2'b10; // srcB:
SignExtended
56                      ALUcontrol = 3'b000; // add
57                      Branch = 1'b0;
58                      PCWrite = 1'b0;
59                      PCSource = 2'b00;
60                  end
61                  4'd3: //memread
62                  begin
63                      PCWriteCond = 0;
64                      IorD = 1'b1; //
ALUResult_DFF
65                      MemRead = 1'b1; //  Mem
read
66                      MemWrite = 1'b0;
67                      IRWrite = 1'b0;
68                      RegDst = 1'b0;
69                      MemtoReg = 1'b0;
70                      RegWrite = 1'b0;
71                      ALUSrcA = 1'b0;
72                      ALUSrcB = 2'b00;
73                      ALUcontrol = 3'b000;
74                      Branch = 1'b0;
75                      PCWrite = 1'b0;
76                      PCSource = 2'b00;
77                  end
78                  4'd4: //memwriteback
79                  begin
```

```
80                      PCWriteCond = 0;
81                      IorD = 1'b0;
82                      MemRead = 1'b0;
83                      MemWrite = 1'b0;
84                      IRWrite = 1'b0;
85                      RegDst = 1'b0; //
    RegWdaddr: Rt
86                      MemtoReg = 1'b1; //
    RegWdin: Memout
87                      RegWrite = 1'b1; // enable
    Reg write
88                      ALUSrcA = 1'b0;
89                      ALUSrcB = 2'b00;
90                      ALUcontrol = 3'd6;
91                      Branch = 1'b0;
92                      PCWrite = 1'b0;
93                      PCSource = 2'b00;
94                  end
95                  4'd5: //memwrite
96                  begin
97                      PCWriteCond = 0;
98                      IorD = 1'b1; // Memaddr:
    ALUResult_DFF
99                      MemRead = 1'b0;
100                     MemWrite = 1'b1; // enable
    Mem write
101                     IRWrite = 1'b0;
102                     RegDst = 1'b0;
103                     MemtoReg = 1'b0;
104                     RegWrite = 1'b0;
105                     ALUSrcA = 1'b0;
106                     ALUSrcB = 2'b00;
107                     ALUcontrol = 3'd6;
108                     Branch = 1'b0;
109                     PCWrite = 1'b0;
110                     PCSource = 2'b00;
111                 end
```

```
112                 4'd6: //R type eOecute
113                 begin
114                     PCWriteCond = 0;
115                     IorD = 1'b0;
116                     MemRead = 1'b0;
117                     MemWrite = 1'b0;
118                     IRWrite = 1'b0;
119                     RegDst = 1'b0;
120                     MemtoReg = 1'b0;
121                     RegWrite = 1'b0;
122                     ALUSrcA = 1'b1; // srcA:
RegRdout1_DFF
123                     ALUSrcB = 2'b00; // srcB:
RegRdout2_DFF
124                     case(FUNC) // ALU's func:
decided by 'Funct'
125                         6'b100000: ALUcontrol =
5'h00;//add
126                         6'b100010: ALUcontrol =
5'h01;//sub
127                         6'b100100: ALUcontrol =
5'h02;//and
128                         6'b100101: ALUcontrol =
5'h03;//or
129                         6'b100110: ALUcontrol =
5'h04;//xor
130                         default:;
131                     endcase
132                     Branch = 1'b0;
133                     PCWrite = 1'b0;
134                     PCSource = 2'b00;
135                 end
136                 4'd7: //aluwriteback
137                 begin
138                     PCWriteCond = 0;
139                     IorD = 1'b0;
140                     MemRead = 1'b0;
```

```
                    MemWrite = 1'b0;
                    IRWrite = 1'b0;
                    RegDst = 1'b1; //
RegWdaddr: Rd
                    MemtoReg = 1'b0; //
RegWdin: ALUResult_DFF
                    RegWrite = 1'b1; // enable
Reg write
                    ALUSrcA = 1'b0;
                    ALUSrcB = 2'b00;
                    ALUcontrol = 3'd6;
                    Branch = 1'b0;
                    PCWrite = 1'b0;
                    PCSource = 2'b00;
                end
                4'd8: //branch begin
                begin
                    IorD = 0;
                    PCWriteCond = 1;
                    MemRead = 1'b0;
                    MemWrite = 1'b0;
                    IRWrite = 1'b0;
                    RegDst = 1'b0;
                    MemtoReg = 1'b0;
                    RegWrite = 1'b0;
                    ALUSrcA = 1'b1;
                    ALUSrcB = 2'b00;
                    ALUcontrol = 3'b001;
                    Branch = 1'b1;
                    PCWrite = 1'b0;
                    PCSource = 2'b01;
                end
                4'd9: //jump
                begin
                    PCWriteCond = 0;
                    IorD = 1'b0;
                    MemRead = 1'b0;
```

```
175                     MemWrite = 1'b0;
176                     IRWrite = 1'b0;
177                     RegDst = 1'b0;
178                     MemtoReg = 1'b0;
179                     RegWrite = 1'b0;
180                     ALUSrcA = 1'b0;
181                     ALUSrcB = 2'b00;
182                     ALUcontrol = 3'd6;
183                     Branch = 1'b0;
184                     PCWrite = 1'b1; // enable
    update PC
185                     PCSource = 2'b10; // select
    neOtPC = PCJump
186                 end
187                 4'd10: //addi execute
188                 begin
189                     PCWriteCond = 0;
190                     IorD = 1'b0;
191                     MemRead = 1'b0;
192                     MemWrite = 1'b0;
193                     IRWrite = 1'b0;
194                     RegDst = 1'b0;
195                     MemtoReg = 1'b0;
196                     RegWrite = 1'b0;
197                     ALUSrcA = 1'b1;
198                     ALUSrcB = 2'b10;
199                     ALUcontrol = 3'b000;
200                     Branch = 1'b0;
201                     PCWrite = 1'b0;
202                     PCSource = 2'b00;
203                 end
204                 4'd11: //addi regwriteback
205                 begin
206                     PCWriteCond = 0;
207                     IorD = 1'b0;
208                     MemRead = 1'b0;
209                     MemWrite = 1'b0;
```

```
                    IRWrite = 1'b0;
                    RegDst = 1'b0; // RegWdaddr: Rt
                    MemtoReg = 1'b0; // RegWdin: ALUResult_DFF
                    RegWrite = 1'b1; // enable Reg write
                    ALUSrcA = 1'b0;
                    ALUSrcB = 2'b00;
                    ALUcontrol = 3'd6;
                    Branch = 1'b0;
                    PCWrite = 1'b0;
                    PCSource = 2'b00;
                end
                default:;
            endcase
    end
```

ALU & regfile

和单周期相仿

top

按照电路图连线

```
module top(
    input clk,
    input rst
);
//------------------variable----------------//
    reg [31:0] IR, MDR; // mem data register
    reg [31:0] PC;
    reg [31:0] A, B, ALUOut;
    wire [4:0] Rd, Rt, Rs;
    wire [25:0] JumpIMM;
```

```
    wire [15:0] IMM16;
    wire [5:0] Op, Funct;
    wire [31:0] SignExtented, shleft, Memdata;
    wire [31:0] ALUresult;

    wire Zero;

    wire IorD;
    wire MemRead;
    wire MemWrite;
    wire IRWrite;
    wire RegDst;
    wire MemtoReg;
    wire RegWrite;
    wire ALUSrcA;
    wire [1:0] ALUSrcB;
    wire [2:0] ALUControl;
    wire Branch;
    wire PCWrite;
    wire PCWriteCond;
    wire [1:0] PCSource;
    wire PCwe;

    wire k;
    assign k = 1;
//-----------------Control------------------
-//
    control control1(.clk(clk),
                     .rst(rst),
                     .zero(Zero),
                     .IorD(IorD),
                     .MemRead(MemRead),
                     .MemWrite(MemWrite),
                     .RegWrite(RegWrite),
                     .IRWrite(IRWrite),
                     .RegDst(RegDst),
                     .MemtoReg(MemtoReg),
```

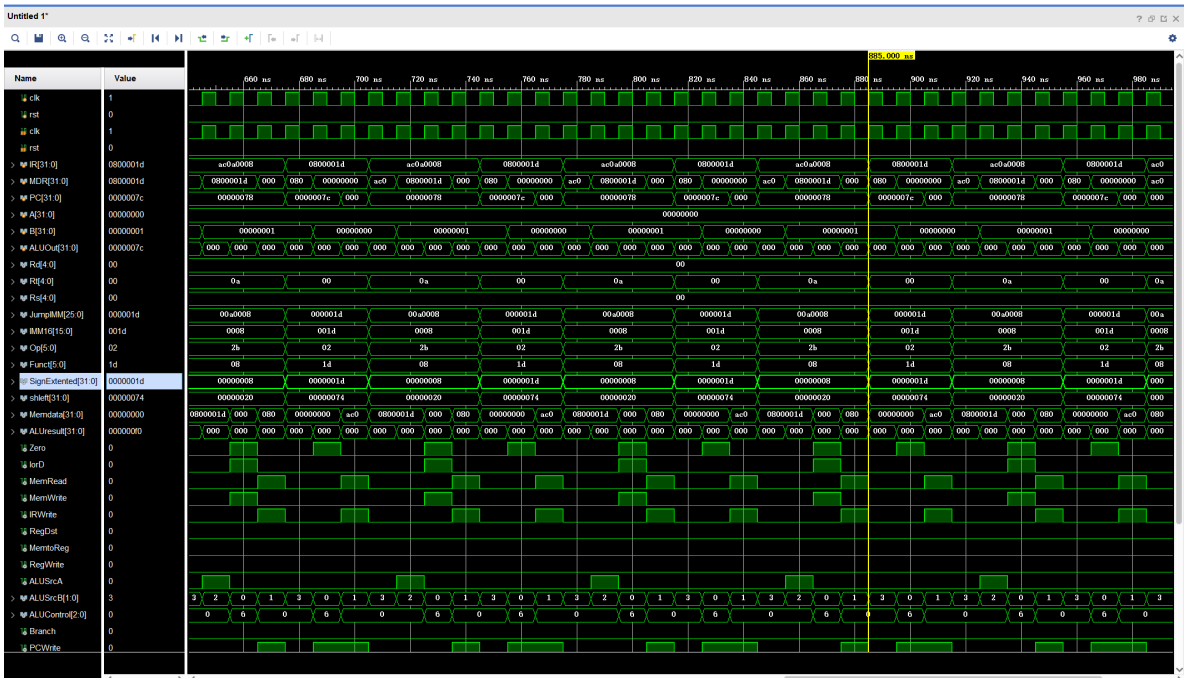
```
                        .ALUSrcA(ALUSrcA),
                        .ALUSrcB(ALUSrcB),
                        .ALUcontrol(ALUControl),
                        .Branch(Branch),
                        .PCWrite(PCWrite),
                        .PCWriteCond(PCWriteCond),
                        .PCSource(PCSource),
                        .PCwe(PCwe),
                        .FUNC(Funct),
                        .opcode(Op));
//-----------------ASSIGN-------------------
-//
    assign JumpIMM = IR[25:0];
    assign Funct = IR[5:0];
    assign IMM16 = IR[15:0];
    assign Rd = IR[15:11];
    assign Rt = IR[20:16];
    assign Rs = IR[25:21];
    assign Op = IR[31:26];
    assign SignExtented = IMM16[15]?
{16'hffff,IMM16}:{16'h0,IMM16};
    assign shleft = SignExtented<<2;
//-----------------PC----------------------//
    wire [31:0] PCin;
    mux4 PCmux(.a(ALUresult),
               .b(ALUOut),
               .c({PC[31:28],{2'b00,JumpIMM}
<<2}),
               .d(0),
               .sel(PCSource),
               .y(PCin));
    always @(posedge clk or posedge rst) begin
        if(rst) PC = 0;
        else if(PCwe) PC=PCin;
    end
//----------------Memory-------------------//
```

```
80      MEM1 MEM(.clk(clk), .spo(Memdata),
    .we(MemWrite),.a(IorD?
    ALUOut[10:2]:PC[10:2]),.d(B));
81  //----------------IR MDR-------------------//
82      always @(posedge clk)
83          MDR <= Memdata;
84      always @(posedge clk)
85          if(IRWrite) IR <= Memdata;
86  //----------------REGFILE------------------//
87      wire [31:0] rd1, rd2;
88      always @(posedge clk) A <= rd1;
89      always @(posedge clk) B <= rd2;
90      reg_file REG(.clk(clk),
91                   .rst(rst),
92                   .ra1(Rs),
93                   .ra2(Rt),
94                   .rd1(rd1),
95                   .rd2(rd2),
96                   .we(RegWrite),
97                   .wd(MemtoReg ? MDR : ALUOut),
98                   .wa(RegDst?Rd:Rt));
99  //----------------ALU----------------------//
100     wire [31:0] ALUB;
101     mux4
    alub(.a(B),.b(4),.c(SignExtented),.d(shleft),.s
    el(ALUSrcB),.y(ALUB));
102     ALU alu(.a(ALUSrcA?A:PC),
103             .b(ALUB),
104             .m(ALUControl),
105             .y(ALUresult),
106             .zf(Zero));
107     always @(posedge clk) ALUOut = ALUresult;
108 endmodule // top
```

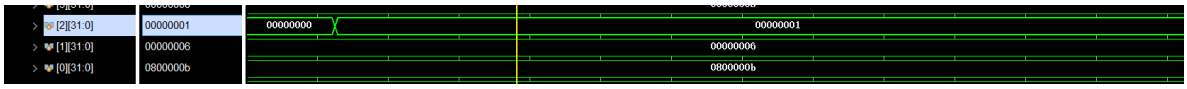
DBU

这部分代码和单周期几乎相同, 具体可见附件

仿真结果



结果



结果分析

cpu可以稳定运行, 在多个周期后得到`success`的结果并储存在了`mem`的适当位置

实验总结

1. 不必刻意地抽象, 分割功能到更多子模块, 但是每个模块的代码应该条理清晰(注释完备)
2. 状态机还是应该每个变量都赋值(而不是赋高阻态)

意见建议

多周期还是比较简单, 可以时间缩减到1周

思考题(以accm指令为例)

> 这部分没有仿真过

- 修改`IorD`和`ALUSrcA`的位宽到2位, 修改对应的2位mux变成4位mux
  <<<<<<< Updated upstream
- 4位mux对应的位置连上`rd1`和`memdataread`
- 控制器在`R-ex`的状态时增加对`accm`的处理

具体可以看我的accm分支