计算机组成原理实验报告

LAB06

题目：多周期 CPU

姓名：林祥

学号：PB16020923

实验目的

1、 实现一个多周期 MIPS 指令集的 CPU，支持包括 lw、sw、R-type、addi、beq、bgtz、j 等指令。

2、 运行一个计算斐波那契数列的程序，初始为 3，3。

实验平台

ISE 14.7

实验过程（分析）

1、 模块化设计，主要有以下几个模块

a) alu 模块——算术逻辑单元

b) regfile 模块——寄存器文件

c) mux 模块和 mux4 模块——2 路和 4 路选择器

d) IP 核生成的 Mem 模块——存放数据段和代码段

e) nextpclogic 模块——计算下一个 PC

f) dff 模块——D 触发器，用于多周期各段寄存

g) control 模块——FSM 控制 regfile、IMem、DMem、nextpclogic 和 alu

h) top 模块——实例化前几个模块，连接各个信号

2、alu 模块使用 case 语句判断 8 种操作类型。

3、regfile 模块用组合逻辑读，时序逻辑写。

4、Mem 是同步读，同步写，且由指定 coe 文件初始化，coe 文件的内容是 16

进制文本，由 Mars 编译一个汇编代码生成。

5、nextpclogic 模块通过组合逻辑计算出 nextPC 的值。

6、dff 模块用一个 always 在时钟沿触发实现非阻塞赋值。

7、control 模块内实现一个有限状态机，三段式实现：

（1）时钟上升沿，改变当前状态，cstate<=nstate

（2）组合逻辑根据当前状态和指令计算次态 nstate

（3）时钟下降沿，根据次态 nstate，更新各控制信号（将在下一个上升沿起作用），下一周期需要使用的信号置为对应值，不使用的选择信号置 x，不使用的使能置 0。

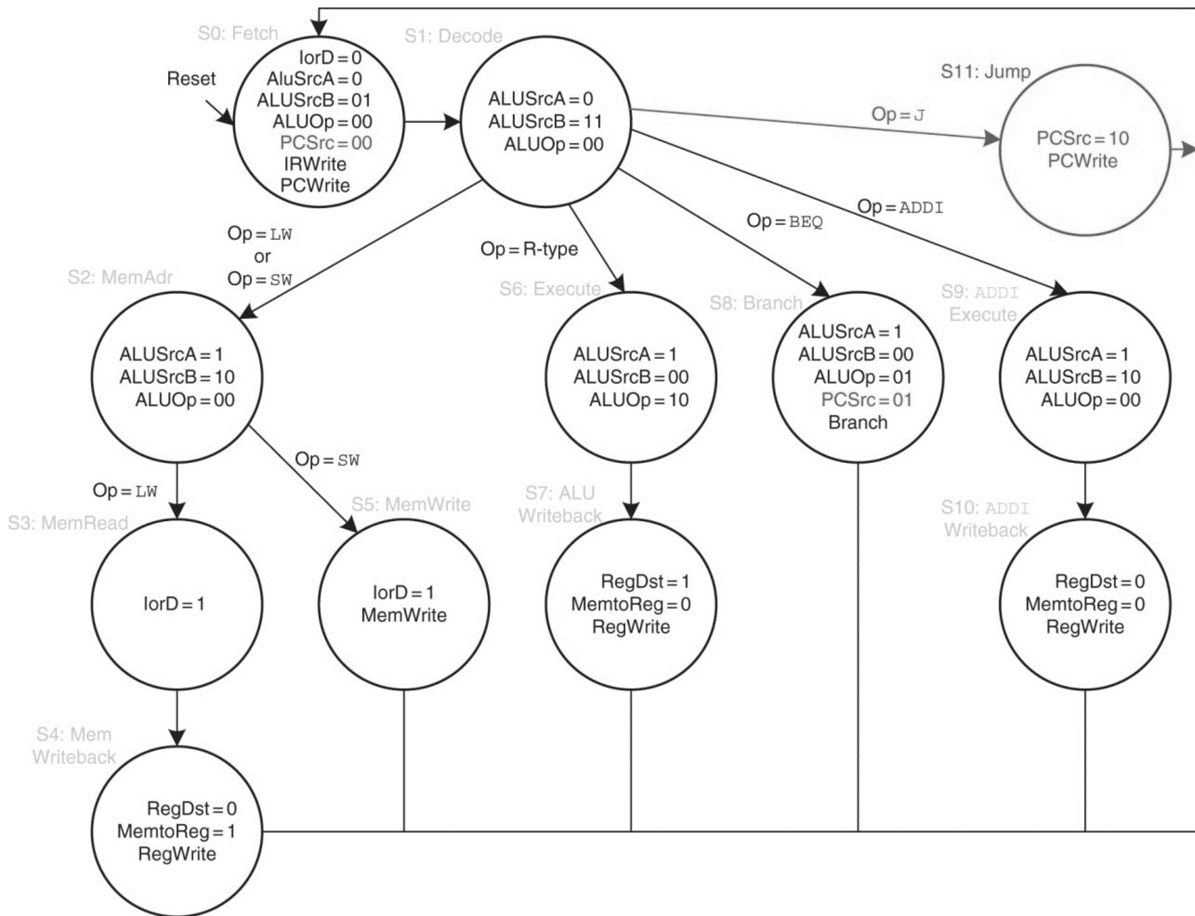
控制信号如下表：

状态	S15	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
功能 信号	Rst	Start	Instr decode	MM addr	MM read	read WB	MM write	R-type EXE	R-type WB	Beq	Addi EXE	Addi WB	Jump
LorD	0	0	x	x	1	x	1	x	x	x	x	x	x
MemRead		1	0	0	1	0	0	0	0	0	0	0	0
MemWrite		0	0	0	0	0	1	0	0	0	0	0	0
IRWrite		1	0	0	0	0	0	0	0	0	0	0	0
RegDst		x	x	x	x	0	x	x	1	x	x	0	x
MemtoReg		x	x	x	x	1	x	x	0	x	x	0	x
RegWrite		0	0	0	0	1	0	0	1	0	0	1	0
ALUSrcASel		0	0	1	x	x	x	1	x	1	1	x	x
ALUSrcBSel		01	11	10	xx	xx	xx	00	xx	00	10	xx	xx
ALUControl		001	001	001	xxx	xxx	xxx	②	xxx	010	001	xxx	xxx
Branch		0	0	0	0	0	0	0	0	1	0	0	0
PCWrite	1	1	0	0	0	0	0	0	0	0	0	0	1
PCSrc	11①	00	xx	xx	xx	xx	xx	xx	xx	01	xx	xx	10

注：①rst 状态下 PCSrc=11，利用 4 路选择器的剩余 1 路，置初始 nextPC=0.

②根据 Funct 具体设置不同的 ALUControl

状态机如下：



8、top 模块实例化其他所有模块，连接各个信号。

9、 bgtz 的实现：bgtz 是伪指令（类似地有 la 和 li），不是 MIPS 指令集的指令，编译时会进行处理，转换为几条指令实现。

这里我用以下三条指令

```
slt $t6,$zero,$t1      # if $t1 > 0, that $t6 = 1, else $t6 = 0
addi $t7,$zero,1       # set $t7 = 1
beq $t6,$t7,loop       # jump if $t6 = 1, that is to say, $t1 > 0
```

实现

```
bgtz $t1, loop         # repeat if not finished yet.
```

10、ram 模块的初始化

IP 核选择 Block Memory，设置为 Single Port RAM，

宽度为 32，深度为 256

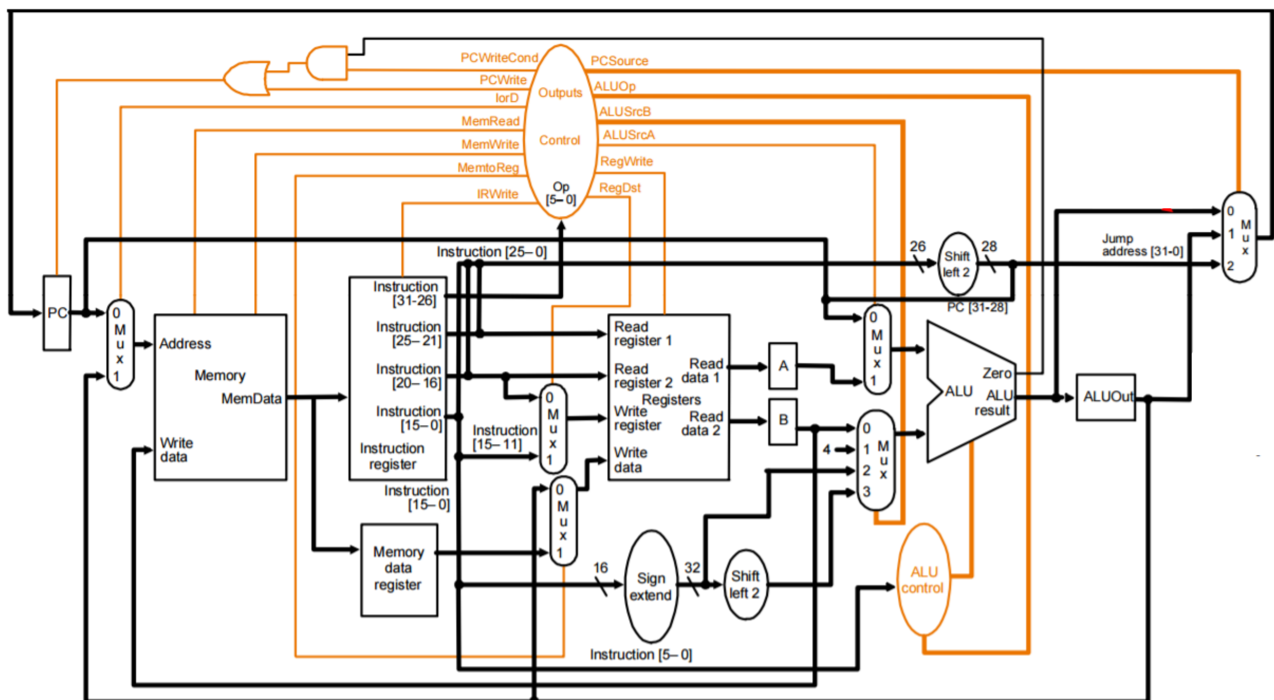
选中 Load Init File 设置，选择初始化 coe 文件的路径，

其中 coe 文件内容为（Text 段为 0-63，Data 段为 64-255）

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
20080100
200d0150
8dad0000
200b0154
8d6b0000
200c0154
8d8c0004
ad0b0000
ad0c0004
21a9fffe
8d0b0000
8d0c0004
016c5020
ad0a0008
21080004
2129ffff
0009702a
200f0001
11cffff7
08000013
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000014
00000003
00000003
…
```

11、完整的数据通路



12、分析结果

```
.data
fibs: .word   0 : 20        # "array" of  20 words to contain fib values
size: .word  20            # size of "array"
temp: .word 3  3
.text
     la    $t0, fibs        # load address of array
     la    $t5, size        # load address of size variable
     lw    $t5, 0($t5)      # load array size
     la    $t3, temp        # load
     lw    $t3, 0($t3)
     la    $t4, temp
     lw    $t4, 4($t4)

     sw    $t3, 0($t0)      # F[0] = $t3
     sw    $t4, 4($t0)      # F[1] = $t4
     addi  $t1, $t5, -2     # Counter for loop, will execute (size-2) times
loop: lw   $t3, 0($t0)      # Get value from array F[n]
     lw    $t4, 4($t0)      # Get value from array F[n+1]
     add   $t2, $t3, $t4    # $t2 = F[n] + F[n+1]
     sw    $t2, 8($t0)      # Store F[n+2] = F[n] + F[n+1] in array
     addi  $t0, $t0, 4      # increment address of Fib. number source
     addi  $t1, $t1, -1     # decrement loop counter
```

```
    slt $t6,$zero,$t1     # if $t1 > 0, that $t6 = 1, else $t6 = 0
    addi $t7,$zero,1      # set $t7 = 1
    beq $t6,$t7,loop      # jump if $t6 = 1, that is to say, $t1 > 0
out:
  j out
```

完整汇编代码如上，可知程序先向内存数据段读取数组首地址（$t0）和大小（$t5，循环次数），读取并写入 f[0]=3, f[1]=3，然后循环 18 次，执行 f[i]=f[i-1]+f[i-2]计算斐波那契数列。

如果 CPU 编写正确，Mem [0-19]（由于 Data 和 Text 在同一个 Ram，故实际上是在 64-83）内容将是 3、3、6、9、15、24、39、63、102、165、267、432、699、1131、1830、2961、4791、7752、12543、20295

Reg 内容将是：

$t0：最后一次循环后数组基址——256+4*18=328

$t1：剩余循环次数——0

$t2：最后一次的计算结果——20295

$t3：最后一次的计算数 1——7752

$t4：最后一次的计算数 2——12543

$t5：数组大小——20

$t6：最后一次不跳转——0

$t7：1

实验结果

仿真结果：

仿真得 Mem 和 reg 数据为

Mem

	0	1	2	3
0x0	537395456	537723216	2376925184	537592148
0x4	2372599808	537657684	2374762500	2903179264
0x8	2903244804	564789246	2366308352	2366373892
0xC	23875616	2903113736	554172420	556400639
0x10	618538	537853953	298844151	134217747
0x14	0	0	0	0
0x18	0	0	0	0
0x1C	0	0	0	0
0x20	0	0	0	0
0x24	0	0	0	0
0x28	0	0	0	0
0x2C	0	0	0	0
0x30	0	0	0	0
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
0x58	0	0	0	0
0x5C	0	0	0	0
0x60	0	0	0	0
0x64	0	0	0	0
0x68	0	0	0	0
0x6C	0	0	0	0

Regfile

	0	1	2	3
0x0	0	0	0	0
0x4	0	0	0	0
0x8	328	0	20295	7752
0xC	12543	20	0	1
0x10	0	0	0	0
0x14	0	0	0	0
0x18	0	0	0	0
0x1C	0	0	0	0
0x20	0	0	0	0
0x24	0	0	0	0
0x28	0	0	0	0
0x2C	0	0	0	0
0x30	0	0	0	0
0x34	0	0	0	0
0x38	0	0	0	0
0x3C	0	0	0	0

可见运行结果符合分析。

附录:

一、 编写过程中遇到问题和一些发现

1、如果位拼接作为左值，单个变量也要花括号，如

{{RegDst},{ALUSrc},{MemtoReg},{RegWrite},{MemWrite},{Branch},{ALUOp[1:0]},{Jump}}=9'b100100100;

当然，本次实验中，我没有用位拼接赋值，这样写一行太长了，也不够清晰。

2、"=="会严格匹配每一位，比如 module 内 5 位数字比较（一个常量，一个 input），但外部传入的是 3 位的，高位自动拓展好像是 x，就导致“==”不成立，这是一个不好发现的问题。

3、仿真遇到高阻态，多半是这个信号不存在，经常是拼写错了，有时候 ise 检查不出来。

4、发现如果 Mars 的 memory configuration 设置为默认，则编译后导出代码里面的所有 la（给某个寄存器赋值一个 32 位地址）被翻译为连续两句 lui 和 ori；若 memory configuration 设置为 text at address 0 则编译后导出代码里面的所有 la 被翻译为一句 addi。我觉得原因是 text 地址从 0 开始的话，如果 text 部分较短，地址非 0 部分未超过 16 位，那么 la 只需要给低 16 位赋值就行了，就可以直接用 addi 实现（addi 指令末 16 位为立即数）。如果地址是实打实的 32 位那 addi 就无能为力了，这时候需要 lui 给高 16 位赋值，低 16 位用 ori（ori 的 rs 取$0，作用就和 addi 一样）。

5、在 top 里的 wire 信号，如果通过 control 模块（或其他模块）进行控制，那么这个信号不能直接在 top 里赋初值，要到仿真模块里赋值，不然 control 里改变这个信号的值时，会变成不确定。

6、在 clk 边缘用<=对某信号（如 state）进行赋值时，如果同时刻需要对该信号进行判断（如状态机，根据状态以完成对应操作），则判断时信号的值是本次改变前的。（非阻塞赋值的特性体现）

二、 模块源代码

top.v

```
module top( //顶层模块
    input clk,
    input rst_n,
    );

    wire ALUSrcASel;
    wire [1:0] ALUSrcBSel;
    wire [31:0] ALUSrcA;
    wire [31:0] ALUSrcB;
    wire [2:0] ALUControl;
    wire [31:0] ALUResult;
    wire [31:0] ALUResult_DFF; //加_DFF 表示对应 D 触发器的输出，下同
    wire Zero;

    wire [31:0] SignExtented;


    wire [5:0] RegRdaddr1;
    wire [31:0] RegRdout1;
    wire [31:0] RegRdout1_DFF;
    wire [5:0] RegRdaddr2;
    wire [31:0] RegRdout2;
    wire [31:0] RegRdout2_DFF;
    wire [5:0] RegWdaddr;
    wire [31:0] RegWdin;
    wire RegWrite;
    wire RegDst;

    wire [31:0] Memaddr_src; //这是计算出来的访存地址
    wire [31:0] Memaddr; //这是计算出来的访存地址右移了两位，原因见下面
    wire [31:0] Memout;
    wire [31:0] Memin;
    wire MemRead;
    wire MemWrite;
    wire MemtoReg;

    wire [31:0] Instr;
    wire [31:0] Data;

    wire [31:0] PC;
    wire [31:0] nextPC;
```

```
wire PCEn;
wire PCWrite;
wire LorD;
wire IRWrite;


wire [5:0] Funct;
wire [15:0] IMM16;
wire [4:0] Rd;
wire [4:0] Rt;
wire [4:0] Rs;
wire [5:0] Op;

wire [25:0] JumpIMM;
wire [1:0] PCSrc;

wire Branch;
//以上信号基本按数据通路图的命名，意义很明显，不多做注释了

//=======================PC========================

nextpclogic nextpclogic(PC,PCWrite,PCSrc,JumpIMM,Branch,Zero,ALUResult,
                         ALUResult_DFF,nextPC,PCEn);
dff DFFPC(~clk,nextPC,PC,PCEn);

mux MUXPCALUout(LorD,PC,ALUResult_DFF,Memaddr_src);

//=======================Memory=======================
assign Memaddr = Memaddr_src >> 2;
//>>2 是因为这里 Mem 是每个地址存储 4 字节，和实际上的（一地址一字节）不一样
Mem Mem(clk,MemWrite,Memaddr,Memin,Memout);

dff DFFInstr(~clk,Memout,Instr,IRWrite);
dff DFFData(~clk,Memout,Data,1'b1);

assign JumpIMM = Instr[25:0];
assign Funct = Instr[5:0];
assign IMM16 = Instr[15:0];
assign Rd = Instr[15:11];
assign Rt = Instr[20:16];
assign Rs = Instr[25:21];
assign Op = Instr[31:26];

assign Memin = RegRdout2_DFF;
```

```
    //======================Regfile=======================
    assign RegRdaddr1 = Rs;
    assign RegRdaddr2 = Rt;
    mux #(6) MUXRegWdaddr(RegDst,{1'b0,Rt},{1'b0,Rd},RegWdaddr);
    mux MUXMemtoReg(MemtoReg,ALUResult_DFF,Data,RegWdin);
    regfile regfile(clk,rst_n,RegRdaddr1,RegRdout1,RegRdaddr2,
                    RegRdout2,RegWdaddr,RegWdin,RegWrite);
    dff DFFRegRdout1(~clk,RegRdout1,RegRdout1_DFF,1'b1);
    dff DFFRegRdout2(~clk,RegRdout2,RegRdout2_DFF,1'b1);

    //=========================ALU=========================
    assign SignExtented = {{16{IMM16[15]}},IMM16};
    mux MUXALUSrcA(ALUSrcASel,PC,RegRdout1_DFF,ALUSrcA);
    mux4 MUXALUSrcB(ALUSrcBSel,RegRdout2_DFF,4,SignExtented,
                    SignExtented << 2,ALUSrcB);
    alu alu(ALUSrcA,ALUSrcB,{2'b00,{ALUControl}},ALUResult,Zero,Sign);
    dff DFFALUResult(~clk,ALUResult,ALUResult_DFF,1'b1);

    //======================Control=======================
    control control(clk,rst_n,Op,Funct,LorD,MemRead,MemWrite,IRWrite,RegDst,MemtoReg,
          RegWrite,ALUSrcASel,ALUSrcBSel,ALUControl,Branch,PCWrite,PCSrc,tcstate);

endmodule
```

alu.v

```
parameter A_NOP =5'h00; //nop
parameter A_ADD =5'h01; //sign_add
parameter A_SUB =5'h02; //sign_sub
parameter A_AND =5'h03; //and
parameter A_OR  =5'h04; //or
parameter A_XOR =5'h05; //xor
parameter A_NOR =5'h06; //nor
parameter A_SLT =5'h07; //slt

module alu(
    input [31:0] alu_a,
    input [31:0] alu_b,
    input [4:0] alu_op,
    output reg [31:0] alu_out,
     output zero
    );
    assign zero = (alu_out == 32'b0)?1:0;
    always@(*)
        case (alu_op)
```

```
            A_NOP: alu_out = 0;
            A_ADD: alu_out = alu_a + alu_b;
            A_SUB: alu_out = alu_a - alu_b;
            A_AND: alu_out = alu_a & alu_b;
            A_OR : alu_out = alu_a | alu_b;
            A_XOR: alu_out = alu_a ^ alu_b;
            A_NOR: alu_out = ~(alu_a | alu_b);
            A_SLT: //if a<b(signed) return 1 else return 0;
                begin
                    if(alu_a[31] == alu_b[31]) alu_out = (alu_a < alu_b) ? 32'b1 : 32'b0;
                    //同号情况，后面的小于是视为无符号的比较
                    else alu_out = (alu_a[31] < alu_b[31]) ? 32'b0 : 32'b1;
                    //有符号比较符号
                end
            default: ;
        endcase
endmodule
```

regfile.v

```
module regfile( //寄存器文件
    input  clk,
    input rst_n,
    input [5:0] rAddr1,//读地址 1
    output [31:0] rDout1,//读数据 1
    input [5:0] rAddr2,//读地址 2
    output [31:0] rDout2,//读数据 2
    input [5:0] wAddr,//写地址
    input [31:0] wDin,//写数据
    input wEna//写使能

);
    reg [31:0] data [0:63];
    integer i;
    assign rDout1=data[rAddr1];//读 1
    assign rDout2=data[rAddr2];//读 2
    always@(posedge clk or negedge rst_n)//写和复位
        if(~rst_n)
        begin
            for(i=0; i<64; i=i+1) data[i]<=0;
        end
        else
        begin
            if(wEna)
                data[wAddr]<=wDin;
```

```
        end
endmodule
```

nextpclogic.v

```
module nextpclogic( //生成下一个 PC
   input [31:0] PC,
    input PCWrite,
    input [1:0] PCSrc,
    input [25:0] JumpIMM,
   input Branch,
   input Zero,
    input [31:0] ALUResult,
    input [31:0] ALUResult_DFF,
    output [31:0] nextPC,
    output PCEn
   );

    wire [31:0] ShiftLeft2;
    wire [31:0] PCJump;
    wire tmp;
    assign ShiftLeft2 = JumpIMM << 2;
    assign PCJump = {{PC[31:28]},{{2'b00,JumpIMM}<<2}};
    mux4 MUXPC(PCSrc,ALUResult,ALUResult_DFF,PCJump,0,nextPC);
    and(tmp,Branch,Zero);
    or(PCEn,tmp,PCWrite);

endmodule
```

dff.v

```
module dff #(parameter WIDTH = 32) ( //Data Flip-Flop
    input clk,
    input [WIDTH-1:0] datain,
    output reg [WIDTH-1:0] dataout,
    input en
    );
    always@(posedge clk)
    begin
        if(en)
            dataout <= datain;
    end
endmodule
```

mux.v

```
module mux #(parameter WIDTH = 32)( //2 路选择器
    input sel,
    input [WIDTH-1:0] d0,
    input [WIDTH-1:0] d1,
    output [WIDTH-1:0] out
    );
    assign out = (sel == 1'b1 ? d1 : d0);
endmodule
```

mux4.v

```
module mux4 #(parameter WIDTH = 32)( //4 路选择器
    input [1:0] sel,
    input [WIDTH-1:0] d0,
    input [WIDTH-1:0] d1,
     input [WIDTH-1:0] d2,
     input [WIDTH-1:0] d3,
    output reg [WIDTH-1:0] out
    );
    always@(*)
        case(sel)
            2'b00: out=d0;
            2'b01: out=d1;
            2'b10: out=d2;
            2'b11: out=d3;
            default:;
        endcase
endmodule
```

control.v

```
module control( //控制模块，各信号的控制
    input clk,rst_n,
    input [5:0] Op, //instr[31:26]
    input [5:0] Funct,//instr[5:0]
    output reg LorD,
    output reg MemRead,
    output reg MemWrite,
    output reg IRWrite,
    output reg RegDst,
    output reg MemtoReg,
    output reg RegWrite,
```

```
output reg ALUSrcASel,
output reg [1:0] ALUSrcBSel,
output reg [2:0] ALUControl,
output reg Branch,
output reg PCWrite,
output reg [1:0] PCSrc,
output [3:0] tcstate
);
reg [3:0] cstate;
reg [3:0] nstate;

assign tcstate=cstate;
always @(posedge clk or negedge rst_n)//上升沿改变状态
    if(~rst_n) cstate <= 4'd15;//reset 状态
    else cstate <= nstate;

always @(*)//计算次态
    case(cstate)
        4'd15://reset
            nstate = 4'd0;
        4'd0://fetch
            nstate = 4'd1;
        4'd1://decode
            case(Op)
                6'b000000: nstate = 4'd6;//R type
                6'b100011: nstate = 4'd2;//lw
                6'b101011: nstate = 4'd2;//sw
                6'b000100: nstate = 4'd8;//beq
                6'b000010: nstate = 4'd9;//jump
                6'b001000: nstate = 4'd10;//addi
            endcase
        4'd2: //memaddr
            case(Op)
                6'b100011: nstate = 4'd3;//lw
                6'b101011: nstate = 4'd5;//sw
            endcase
        4'd3: //memread
            nstate = 4'd4;
        4'd4: //memwriteback
            nstate = 4'd0;
        4'd5: //memwrite
            nstate = 4'd0;
        4'd6: //execute
            nstate = 4'd7;
        4'd7: //aluwriteback
            nstate = 4'd0;
```

```
            4'd8: //branch
                nstate = 4'd0;
            4'd9: //jump
                nstate = 4'd0;
            4'd10: //addi execute
                nstate = 4'd11;
            4'd11: //addi writeback
                nstate = 4'd0;
        endcase

    //根据下一状态更改控制信号（有使用的信号将更改并做注释，不使用的选择信号置 x,不使用的使能置 0）
    //这里虽然在下降沿更改，但信号将在下一周期（上升沿）起作用(也就是同在这个下降沿的操作读取到的信号是改变前的)
    always @(negedge clk or negedge rst_n)//下降沿时，根据次态，更改信号
    begin
    if(~rst_n)
        begin
            LorD = 1'b0;
            PCSrc = 2'b11; //nextPC=0; 利用 4 路选择器的剩余 1 路
            PCWrite = 1'b1;
        end
    else
        case(nstate)
            4'd0://fetch
                begin
                    LorD = 1'b0; //------Memaddr: PC
                    MemRead = 1'b1; //------enable Mem read
                    MemWrite = 1'b0;
                    IRWrite = 1'b1; //------enable save Instr
                    RegDst = 1'bx;
                    MemtoReg = 1'bx;
                    RegWrite = 1'b0;
                    ALUSrcASel = 1'b0; //------srcA: PC
                    ALUSrcBSel = 2'b01; //------srcB: 4
                    ALUControl = 3'b001; //------ALU's func: add
                    Branch = 1'b0;
                    PCWrite = 1'b1; //------enable update PC
                    PCSrc = 2'b00; //------select nextPC=PC+4

                end
            4'd1://decode
                begin
                    LorD = 1'bx;
                    MemRead = 1'b0;
                    MemWrite = 1'b0;
                    IRWrite = 1'b0;
```

```
            RegDst = 1'bx;
            MemtoReg = 1'bx;
            RegWrite = 1'b0;
            ALUSrcASel = 1'b0; //------srcA: PC
            ALUSrcBSel = 2'b11; //------srcB: SignExtended<<2
            ALUControl = 3'b001; //------ALU's func: add
            Branch = 1'b0;
            PCWrite = 1'b0;
            PCSrc = 2'bxx;
        end
    4'd2: //memaddr
        begin
            LorD = 1'bx;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            IRWrite = 1'b0;
            RegDst = 1'bx;
            MemtoReg = 1'bx;
            RegWrite = 1'b0;
            ALUSrcASel = 1'b1; //------srcA: RegRdout1_DFF
            ALUSrcBSel = 2'b10; //------srcB: SignExtended
            ALUControl = 3'b001; //------ALU's func: add
            Branch = 1'b0;
            PCWrite = 1'b0;
            PCSrc = 2'bxx;
        end
    4'd3: //memread
        begin
            LorD = 1'b1; //------Memaddr: ALUResult_DFF
            MemRead = 1'b1; //------enable Mem read
            MemWrite = 1'b0;
            IRWrite = 1'b0;
            RegDst = 1'bx;
            MemtoReg = 1'bx;
            RegWrite = 1'b0;
            ALUSrcASel = 1'bx;
            ALUSrcBSel = 2'bxx;
            ALUControl = 3'bxxx;
            Branch = 1'b0;
            PCWrite = 1'b0;
            PCSrc = 2'bxx;
        end
    4'd4: //memwriteback
        begin
            LorD = 1'bx;
            MemRead = 1'b0;
```

```
            MemWrite = 1'b0;
            IRWrite = 1'b0;
            RegDst = 1'b0; //------RegWdaddr: Rt
            MemtoReg = 1'b1; //------RegWdin: Memout
            RegWrite = 1'b1; //------enable Reg write
            ALUSrcASel = 1'bx;
            ALUSrcBSel = 2'bxx;
            ALUControl = 3'bxxx;
            Branch = 1'b0;
            PCWrite = 1'b0;
            PCSrc = 2'bxx;
        end
    4'd5: //memwrite
        begin
            LorD = 1'b1; //------Memaddr: ALUResult_DFF
            MemRead = 1'b0;
            MemWrite = 1'b1; //------enable Mem write
            IRWrite = 1'b0;
            RegDst = 1'bx;
            MemtoReg = 1'bx;
            RegWrite = 1'b0;
            ALUSrcASel = 1'bx;
            ALUSrcBSel = 2'bxx;
            ALUControl = 3'bxxx;
            Branch = 1'b0;
            PCWrite = 1'b0;
            PCSrc = 2'bxx;
        end
    4'd6: //R type execute
        begin
            LorD = 1'bx;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            IRWrite = 1'b0;
            RegDst = 1'bx;
            MemtoReg = 1'bx;
            RegWrite = 1'b0;
            ALUSrcASel = 1'b1; //------srcA: RegRdout1_DFF
            ALUSrcBSel = 2'b00; //------srcB: RegRdout2_DFF
            case(Funct) //------ALU's func: decided by 'Funct'
                6'b100000: ALUControl = 5'h01;//add
                6'b100010: ALUControl = 5'h02;//sub
                6'b100100: ALUControl = 5'h03;//and
                6'b100101: ALUControl = 5'h04;//or
                6'b100110: ALUControl = 5'h05;//xor
                6'b100111: ALUControl = 5'h06;//nor
```

```
            6'b101010: ALUControl = 5'h07;//slt
        endcase
        Branch = 1'b0;
        PCWrite = 1'b0;
        PCSrc = 2'bxx;
    end
4'd7: //aluwriteback
    begin
        LorD = 1'bx;
        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
        ALUSrcASel = 1'bx;
        ALUSrcBSel = 2'bxx;
        ALUControl = 3'bxxx;
        Branch = 1'b0;
        PCWrite = 1'b0;
        PCSrc = 2'bxx;
    end
4'd8: //branch
    begin
        LorD = 1'bx;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        IRWrite = 1'b0;
        RegDst = 1'bx;
        MemtoReg = 1'bx;
        RegWrite = 1'b0;
        ALUSrcASel = 1'b1; //------srcA: RegRdout1_DFF
        ALUSrcBSel = 2'b00; //------srcB: RegRdout2_DFF
        ALUControl = 3'b010; //------ALU's func: sub
        Branch = 1'b1; //------enable update PC if beq
        PCWrite = 1'b0;
        PCSrc = 2'b01;  //------select nextPC = ALUResult_DFF(PCBeq)
    end
4'd9: //jump
    begin
        LorD = 1'bx;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        IRWrite = 1'b0;
        RegDst = 1'bx;
        MemtoReg = 1'bx;
```

```
                RegWrite = 1'b0;
                ALUSrcASel = 1'bx;
                ALUSrcBSel = 2'bxx;
                ALUControl = 3'bxxx;
                Branch = 1'b0;
                PCWrite = 1'b1; //------enable update PC
                PCSrc = 2'b10;  //------select nextPC = PCJump
            end
        4'd10: //addi execute
            begin
                LorD = 1'bx;
                MemRead = 1'b0;
                MemWrite = 1'b0;
                IRWrite = 1'b0;
                RegDst = 1'bx;
                MemtoReg = 1'bx;
                RegWrite = 1'b0;
                ALUSrcASel = 1'b1; //------srcA: RegRdout1_DFF
                ALUSrcBSel = 2'b10; //------srcB: SignExtended
                ALUControl = 3'b001; //------ALU's func: add
                Branch = 1'b0;
                PCWrite = 1'b0;
                PCSrc = 2'bxx;
            end
        4'd11: //addi regwriteback
            begin
                LorD = 1'bx;
                MemRead = 1'b0;
                MemWrite = 1'b0;
                IRWrite = 1'b0;
                RegDst = 1'b0; //------RegWdaddr: Rt
                MemtoReg = 1'b0; //------RegWdin: ALUResult_DFF
                RegWrite = 1'b1; //------enable Reg write
                ALUSrcASel = 1'bx;
                ALUSrcBSel = 2'bxx;
                ALUControl = 3'bxxx;
                Branch = 1'b0;
                PCWrite = 1'b0;
                PCSrc = 2'bxx;
            end
      endcase
    end

endmodule
```

test.v

```
module test;

    // Inputs
    reg clk;
    reg rst_n;

    // Instantiate the Unit Under Test (UUT)
    top uut (
        .clk(clk),
        .rst_n(rst_n),

    );


    initial begin
        // Initialize Inputs
        clk = 1;

        //
        rst_n = 1;
        #100;

        //
        rst_n = 0;

        // Wait 100 ns for global reset to finish
        #100;
        rst_n = 1;

        forever begin
            #10;
            clk=~clk;
        end


        // Add stimulus here

    end

endmodule
```