

计算机组成原理实验六_多周期 CPU

一、实验目的：

了解多周期的工作原理，基于 MIPS 架构设计一个多周期 CPU，正确执行已给程序

二、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。

```
* .data
* fibs: .word 0:20 # "array" of 20 words to contain fib values
* size: .word 20 # size of "array"
* temp: .word 33
*
* .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
* bgtz $t1, loop # repeat if not finished yet
*
* out:
* j out
```

- CPU 为多周期
- 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。

- 实验设计的指令：

addi add lw sw bgtz j

三、实验分析：

1.实验原理

多周期 CPU：根据指令执行所使用的功能部件，将执行过程分成多个阶

段，每个阶段一个周期

--在一个周期内各个部件并行工作

--功能部件可以在不同阶段（周期）复用，有利于降低硬件实现复杂度
周期时间确定

--每个周期的工作尽量平衡

--假设一个周期内可以完成：

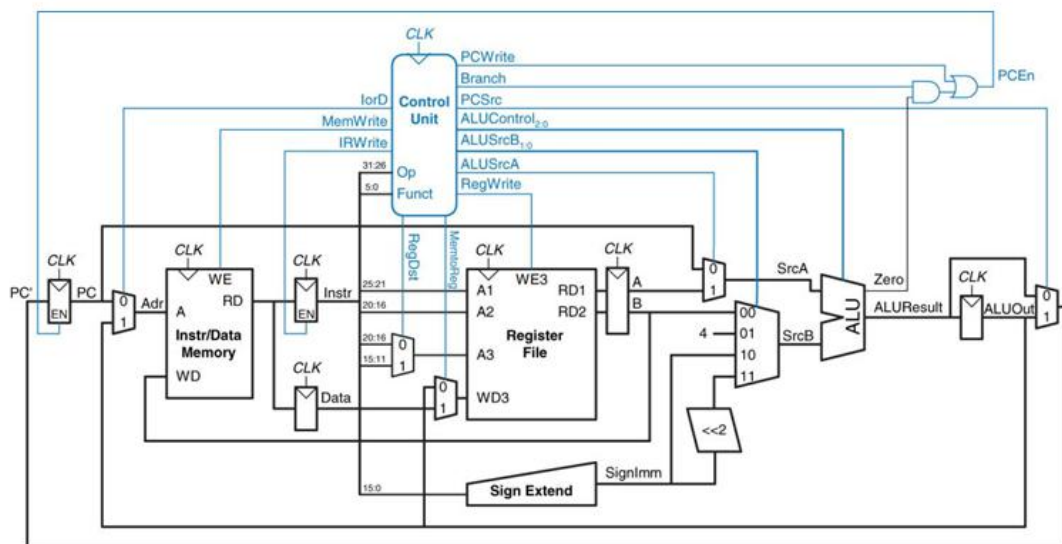
一次 mem 访问，or

一次寄存器访问，or

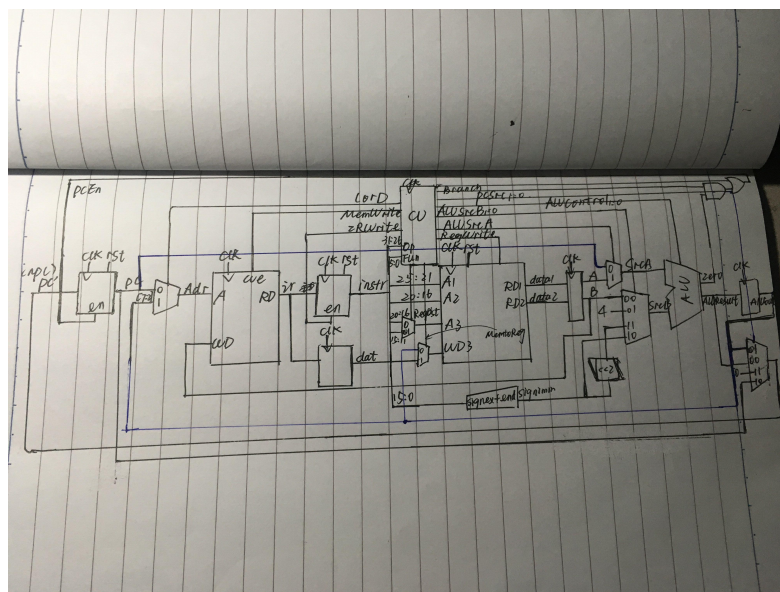
一次 ALU 操作

指令共分为五个阶段：取址、译码、执行（分支指令完成）、访存（store 和 R-type 指令完成）、load 完成

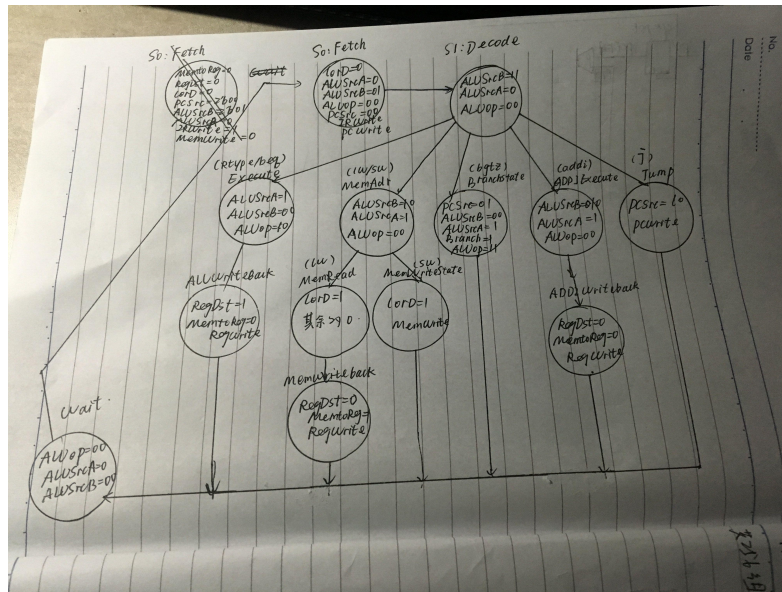
2.数据通路



根据本实验修改后的数据通路：



以及对应的状态图：



3. 相关指令

op

(1) add 000000 rs rt rd shamt funct=100000

$rs + rt \rightarrow rd$

(2) addi 001000 rs rt immediate

$rs + (\text{signextend})\text{immediate} \rightarrow rt$

(3) sw 101011 rs rt immediate

$rt \rightarrow \text{memory}[rs + (\text{signextend})\text{immediate}]$

(4) lw 100011 rs rt immediate

$\text{memory}[rs + (\text{signextend})\text{immediate}] \rightarrow rt$

(5) bgtz 000111 rs rt offset

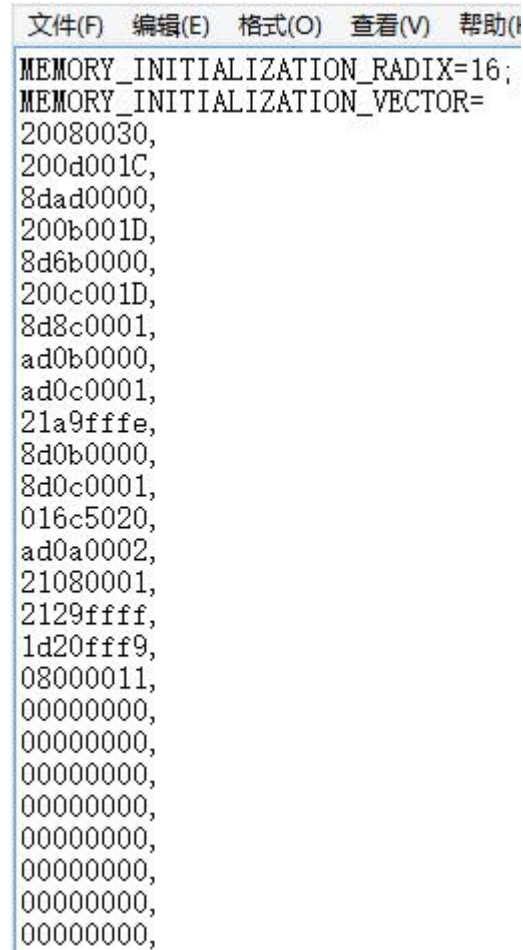
rs 中的值大于 0 时跳转，跳转值为 offset

(6) j 000010 target address

$(PC+4)[31:28], \text{address}, 0, 0 \rightarrow PC$ (26 位扩展为 32 位)

4.实验过程分析

a.同单周期一样用 Mars 编译好程序，到处代码，指令和数据写在一个 coe 文件里，其中数据段的初始地址设置为 0030。



```
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
20080030,
200d001C,
8dad0000,
200b001D,
8d6b0000,
200c001D,
8d8c0001,
ad0b0000,
ad0c0001,
21a9fffe,
8d0b0000,
8d0c0001,
016c5020,
ad0a0002,
21080001,
2129ffff,
1d20fff9,
08000011,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
```

b.例化一个 IP 核，建立一个 memory，并初始化

c.根据数据通路和状态图完成各个模块的编写

d.进行连线

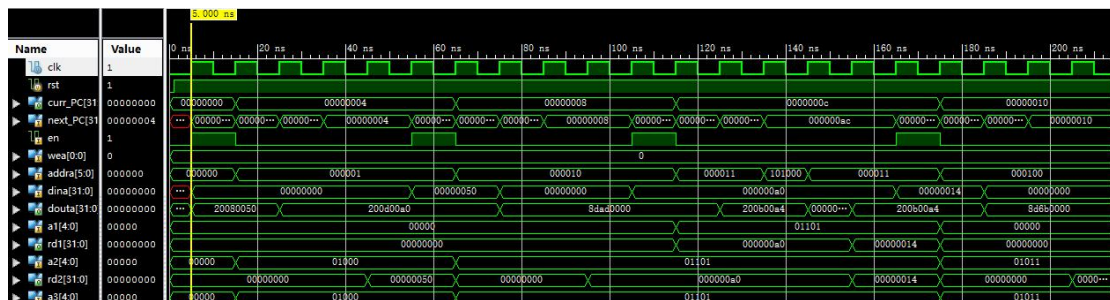
```

59 PC PC(clk,rst,npc,PCEn,pc);
60 Mux6bit Mux1(pc[7:2],ALUOut[7:2],lorD,Adr);//?b:a
61 Memory Memory(clk,MemWrite,Adr,B,ir);
62 IR IR(clk,rst,IRWrite,ir,instr);
63 RegisterFile RegisterFile(clk,rst,instr[25:21],instr[20:16],A3,WD3,RegWrite,data1,data2);
64 Mux5bit Mux2(instr[20:16],instr[15:11],RegDst,A3);//?b:a
65 AB AB(clk,data1,data2,A,B);
66 data data(clk,ir,dat);
67 Mux32bit Mux3(ALUOut,dat,MemtoReg,WD3);//?b:a
68 MainDecoder MainDecoder(clk,rst,instr[31:26],MemtoReg,RegDst,lorD,PCSrc,ALUSrcB,ALUSrcA,IRWrite,MemWrite,PC
69 ALUDecoder ALUDecoder(instr[5:0],ALUOp,ALUControl);
70 Mux32bit Mux4(pc,A,ALUSrcA,SrcA);
71 SignExtend SignExtend(instr[15:0],SignImm);
72 Mux4to1 Mux5(B,32'h4,SignImm,SignImm<<2,ALUSrcB,SrcB);
73
74 ALU ALU(SrcA,SrcB,ALUControl,Zero,ALUResult);
75 aluout aluout(clk,ALUResult,ALUOut);
76 Mux4to1 Mux6(ALUResult,ALUOut,pc,32'b0,PCSrc,npc);
77
78 assign PCEn=PCWrite|(Branch&Zero);

```

四、实验结果：

成功完成代码的编写，仿真成功，ram 里数据计算正确。



	0	1	2	3	4	5	6	7
0x0	537395280	537723040	2376925184	537591972	2372599808	537657508	2374762500	2903179264
0x8	2903244804	564789246	2366308352	2366373892	23875616	2903113736	554172420	556400639
0x10	488701945	134217745	0	0	3	3	6	9
0x18	15	24	39	63	102	165	267	432
0x20	699	1131	1830	2961	4791	7752	12543	20295
0x28	20	3	3	0	0	0	0	0
0x30	0	0	0	0	0	0	0	0
0x38	0	0	0	0	0	0	0	0

五、附录：

1.各模块代码

(a) ALU:

```

module ALU(
    input [31:0]SrcA,
    input [31:0]SrcB,
    input [2:0]ALUControl,
    output reg Zero,
    output reg [31:0]ALUResult
);
parameter A_AND = 3'b000;

```



```

        6'b100100: ALUControl<=3'b000;
        6'b100101: ALUControl<=3'b001;
        6'b101010: ALUControl<=3'b111;
        default:   ALUControl<=3'b000;
    endcase
end
2'b11: ALUControl<=3'b111;
default: ALUControl<=3'b000;
endcase
end
Endmodule

```

(c) aluout

```

module aluout(
    input clk,
    input [31:0]ALUResult,
    output reg[31:0]ALUOut
);
always@(posedge clk)
begin
    ALUOut<=ALUResult;
end

endmodule

```

(d) PC

```

module PC(
    input clk,
    input rst,
    input [31:0]next_PC,
    input en,
    output reg [31:0]curr_PC
);

always@(posedge clk or negedge rst)
begin
    if(~rst)curr_PC<=0;
    else if(en)
        curr_PC<=next_PC;
    else curr_PC<=curr_PC;
end

endmodule

```

(e) Mux6bit

```

module Mux6bit(
    input [5:0]a,
    input [5:0]b,
    input sw,
    output [5:0]out
);

assign out=sw?b:a;      //1:b    0:a
Endmodule

```

(f) Mux5bit

```

module Mux5bit(
    input [4:0]a,
    input [4:0]b,
    input sw,
    output [4:0]out
);

```

```

assign out=sw?b:a;

```

```

Endmodule

```

(g) Mux32bit

```

module Mux32bit(
    input [31:0]a,
    input [31:0]b,
    input sw,
    output [31:0]out
);

```

```

assign out=sw?b:a;      //1:b    0:a
endmodule

```

(h) Mux4to1

```

module Mux4to1(
    input [31:0]a,
    input [31:0]b,
    input [31:0]c,
    input [31:0]d,
    input [1:0]sw,
    output reg [31:0]out
);

```

```

always@(*)

```

```

begin

```



```

    case(sw)
        2'b00: out<=a;
        2'b01: out<=b;
        2'b10: out<=c;
        2'b11: out<=d;
        default: out<=32'b0;
    endcase
end
Endmodule

```

(i) IR

```

module IR(
    input clk,
    input rst,
    input en,
    input [31:0]next_instr,
    output reg [31:0]instr
);

always@(posedge clk or negedge rst)
begin
    if(~rst)instr<=0;
    else if(en)
        instr<=next_instr;
    else instr<=instr;
end

endmodule

```

(j) data

```

module data(
    input clk,
    input [31:0]instr,
    output reg [31:0]data
);

always@(negedge clk)
begin
    data<=instr;
end

endmodule

```

(k) regfile

```

module RegisterFile(

```

```

        input clk,
        input rst,
        input [4:0]a1,
        input [4:0]a2,
        input [4:0]a3,
        input [31:0]wd,
        input we,
        output [31:0]rd1,
        output [31:0]rd2
    );

    reg [31:0]r[31:0];

    assign rd1=r[a1];
    assign rd2=r[a2];
    integer i;
    always@(posedge clk or negedge rst)
    begin
        if(~rst)
            begin
                for(i=0;i<32;i=i+1)
                    r[i]<=0;
            end
        else if(we)
            r[a3]<=wd;
    end

endmodule

```

(I) AB

```

module AB(
    input clk,
    input [31:0]data1,
    input [31:0]data2,
    output reg[31:0]A,
    output reg[31:0]B
);
    always@(posedge clk)
    begin
        A<=data1;
        B<=data2;
    end
endmodule

```

(m) signextend

```

module SignExtend(
    input [15:0]din,
    output [31:0]dout
);
assign dout[31:0]={16{din[15]}},din[15:0];
endmodule

```

(n) mainDecoder

```

module MainDecoder(//control_unit
    input clk,
    input rst,
    input [5:0]opcode,
    output reg MemtoReg,
    output reg RegDst,
    output reg lorD,
    output reg [1:0]PCSrc,
    output reg [1:0]ALUSrcB,
    output reg ALUSrcA,
    output reg IRWrite,
    output reg MemWrite,
    output reg PCWrite,
    output reg Branch,
    output reg RegWrite,
    output reg[1:0]ALUOp
);

reg [3:0]next_state;
reg [3:0]curr_state;
parameter Fetch          = 4'h0;//取址
parameter Decode         = 4'h1;//译码
parameter MemAdr         = 4'h2;//memory 取址
parameter MemRead        = 4'h3;//memory 读
parameter MemWriteback   = 4'h4;//memory 写回
parameter MemWritestate  = 4'h5;//memory 写
parameter Execute        = 4'h6;//执行
parameter ALUWriteback   = 4'h7;//ALU 写回
parameter Branchstate    = 4'h8;//分支
parameter ADDIExecute    = 4'h9;//执行 addi
parameter ADDIWriteback  = 4'hA;//addi 写回
parameter Jump           = 4'hB;//无条件跳转
parameter Wait           = 4'hF;//等待

parameter Rtype          = 6'b0000000;//本实验中为 add

```

```

parameter lw      = 6'b100011;
parameter sw      = 6'b101011;
parameter beq     = 6'b000100;
parameter bgtz    = 6'b000111;
parameter addi    = 6'b001000;
parameter j       = 6'b000010;

```

```

always@(*)
begin
    if(~rst)
        next_state <= Wait;
    else case(curr_state)
        Fetch      :next_state<=Decode;
        Decode     :
            begin
                case(opcode)
                    Rtype :next_state<=Execute;
                    lw    :next_state<=MemAdr;
                    sw    :next_state<=MemAdr;
                    beq    :next_state<=Execute;
                    bgtz   :next_state<=Branchstate;
                    addi   :next_state<=ADDIExecute;
                    j      :next_state<=Jump;
                    default:next_state<=Wait;
                endcase
            end
        MemAdr     :
            begin
                case(opcode)
                    lw      :next_state<=MemRead;
                    sw      :next_state<=MemWritestate;
                    default:next_state<=Wait;
                endcase
            end
        MemRead    :next_state<=MemWriteback;
        MemWriteback :next_state<=Wait;
        MemWritestate :next_state<=Wait;
        Execute     :next_state<=ALUWriteback;
        ALUWriteback :next_state<=Wait;
        Branchstate  :next_state<=Wait;
        ADDIExecute  :next_state<=ADDIWriteback;
        ADDIWriteback :next_state<=Wait;
        Jump         :next_state<=Wait;
        Wait         :next_state<=Fetch;
    endcase
end

```

```

        default      :next_state<=Fetch;
    endcase
end

```

```

always@(*)

```

```

begin

```

```

    case(curr_state)

```

```

        Fetch      :

```

```

            begin

```

```

                MemtoReg<=0;

```

```

                RegDst<=0;

```

```

                lorD<=0;

```

```

                PCSrc<=2'b00;

```

```

                ALUSrcB<=2'b01;

```

```

                ALUSrcA<=0;

```

```

                IRWrite<=1;

```

```

                MemWrite<=0;

```

```

                PCWrite<=1;

```

```

                Branch<=0;

```

```

                RegWrite<=0;

```

```

                ALUOp<=2'b00;

```

```

            end

```

```

        Decode      :

```

```

            begin

```

```

                MemtoReg<=0;

```

```

                RegDst<=0;

```

```

                lorD<=0;

```

```

                PCSrc<=2'b00;

```

```

                ALUSrcB<=2'b11;

```

```

                ALUSrcA<=0;

```

```

                IRWrite<=0;

```

```

                MemWrite<=0;

```

```

                PCWrite<=0;

```

```

                Branch<=0;

```

```

                RegWrite<=0;

```

```

                ALUOp<=2'b00;

```

```

            end

```

```

        MemAdr      :

```

```

            begin

```

```

                MemtoReg<=0;

```

```

                RegDst<=0;

```

```

                lorD<=0;

```

```

                PCSrc<=2'b00;

```

```

                ALUSrcB<=2'b10;

```

```

        ALUSrcA<=1;
        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=0;
        Branch<=0;
        RegWrite<=0;
        ALUOp<=2'b00;
    end
MemRead      :
    begin
        MemtoReg<=0;
        RegDst<=0;
        lorD<=1;
        PCSrc<=2'b00;
        ALUSrcB<=2'b00;
        ALUSrcA<=0;
        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=0;
        Branch<=0;
        RegWrite<=0;
        ALUOp<=2'b00;
    end
MemWriteback :
    begin
        MemtoReg<=1;
        RegDst<=0;
        lorD<=0;
        PCSrc<=2'b00;
        ALUSrcB<=2'b00;
        ALUSrcA<=0;
        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=0;
        Branch<=0;
        RegWrite<=1;
        ALUOp<=2'b00;
    end
MemWritestate :
    begin
        MemtoReg<=0;
        RegDst<=0;
        lorD<=1;
        PCSrc<=2'b00;

```

```

        ALUSrcB<=2'b00;
        ALUSrcA<=0;
        IRWrite<=0;
        MemWrite<=1;
        PCWrite<=0;
        Branch<=0;
        RegWrite<=0;
        ALUOp<=2'b00;
    end
Execute      :
    begin
        ALUSrcA<=1;
        ALUSrcB<=2'b00;
        ALUOp<=2'b10;
    end
ALUWriteback :
    begin
        RegDst<=1;
        MemtoReg<=0;
        RegWrite<=1;
    end
Branchstate  :
    begin
        MemtoReg<=0;
        RegDst<=0;
        lorD<=0;
        PCSrc<=2'b01;
        ALUSrcB<=2'b00;
        ALUSrcA<=1;
        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=0;
        Branch<=1;
        RegWrite<=0;
        ALUOp<=2'b11;
    end
ADDIExecute  :
    begin
        MemtoReg<=0;
        RegDst<=0;
        lorD<=0;
        PCSrc<=2'b00;
        ALUSrcB<=2'b10;
        ALUSrcA<=1;

```



```

        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=0;
        Branch<=0;
        RegWrite<=0;
        ALUOp<=2'b00;
    end
    ADDIWriteback :
    begin
        MemtoReg<=0;
        RegDst<=0;
        lorD<=0;
        PCSrc<=2'b00;
        ALUSrcB<=2'b00;
        ALUSrcA<=0;
        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=0;
        Branch<=0;
        RegWrite<=1;
        ALUOp<=2'b00;
    end
    Jump :
    begin
        MemtoReg<=0;
        RegDst<=0;
        lorD<=0;
        PCSrc<=2'b10;
        ALUSrcB<=2'b00;
        ALUSrcA<=0;
        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=1;
        Branch<=0;
        RegWrite<=0;
        ALUOp<=2'b00;
    end
    Wait :
    begin
        MemtoReg<=0;
        RegDst<=0;
        lorD<=0;
        PCSrc<=2'b00;
        ALUSrcB<=2'b00;

```

```

        ALUSrcA<=0;
        IRWrite<=0;
        MemWrite<=0;
        PCWrite<=0;
        Branch<=0;
        RegWrite<=0;
        ALUOp<=2'b00;
    end
endcase
end

always@(posedge clk or negedge rst)
begin
    if(~rst)
        curr_state<=Wait;
    else
        curr_state<=next_state;
    end
endmodule

```

(o) Main

```

module main(
    input clk,
    input rst
);
wire [31:0]pc;
wire [31:0]npc;
wire [5:0]opcode;
wire MemtoReg;
wire RegDst;
wire lorD;
wire [1:0]PCSrc;
wire [1:0]ALUSrcB;
wire ALUSrcA;
wire IRWrite;
wire MemWrite;
wire PCWrite;
wire Branch;
wire RegWrite;
wire [1:0]ALUOp;
wire [2:0]ALUControl;
wire [5:0]Adr;
wire [31:0]A;

```

```

wire [31:0]B;
wire [31:0]ir;
wire [31:0]instr;
wire [4:0]A3;
wire [31:0]data1;
wire [31:0]data2;
wire [31:0]dat;
wire [31:0]SrcA;
wire [31:0]SrcB;
wire [31:0]ALUOut;
wire Zero;
wire [31:0]ALUResult;
wire PCEn;
wire [31:0]WD3;
wire [31:0]SignImm;

PC PC(clk,rst,npc,PCEn,pc);
Mux6bit Mux1(pc[7:2],ALUOut[7:2],lorD,Adr);//? b:a
Memory Memory(clk,MemWrite,Adr,B,ir);
IR IR(clk,rst,IRWrite,ir,instr);
RegisterFile
RegisterFile(clk,rst,instr[25:21],instr[20:16],A3,WD3,RegWrite,data1,data2);
Mux5bit Mux2(instr[20:16],instr[15:11],RegDst,A3);//?b:a
AB AB(clk,data1,data2,A,B);
data data(clk,ir,dat);
Mux32bit Mux3(ALUOut,dat,MemtoReg,WD3);//?b:a
MainDecoder
MainDecoder(clk,rst,instr[31:26],MemtoReg,RegDst,lorD,PCSrc,ALUSrcB,ALUSrcA,IR
Write,MemWrite,PCWrite,Branch,RegWrite,ALUOp);
ALUDecoder ALUDecoder(instr[5:0],ALUOp,ALUControl);
Mux32bit Mux4(pc,A,ALUSrcA,SrcA);
SignExtend SignExtend(instr[15:0],SignImm);
Mux4to1 Mux5(B,32'h4,SignImm,SignImm<<2,ALUSrcB,SrcB);

ALU ALU(SrcA,SrcB,ALUControl,Zero,ALUResult);
aluout aluout(clk,ALUResult,ALUOut);
Mux4to1 Mux6(ALUResult,ALUOut,pc,32'b0,PCSrc,npc);

assign PCEn=PCWrite|(Branch&Zero);
endmodule

```

```
initial
begin
    clk = 0;
    forever
    begin
        #5;
        clk=~clk;
    end
end

initial begin
    // Initialize :
    clk = 0;
    rst = 0;

    // Wait 100 ns
    #1;
    rst=1;
```

2.仿真: