COD_Lab4 多周期 CPU 设计

一、 实验目的

- 理解计算机硬件的基本组成、结构和工作原理
- 掌握数字系统的设计和调试方法
- 熟练掌握数据通路和控制器的设计和描述方法

二、 实验内容

1. 多周期 CPU 设计主要思路

（1）指令

1. 设计实现多周期 CPU，可执行如下六条指令：

• **add**: rd <- rs + rt; op = 000000, funct = 100000

R类型：

31-26	25-21	20-16	15-11	10-6	5-0
op	rs	rt	rd	shamt	func
6位	5位	5位	5位	5位	6位

• **addi**: rt <- rs + imm; op = 001000

lw: rt <- M(rs + addr); op = 100011

sw: M(rs + addr) <- rt; op = 101011

beq: if (rs = rt) then pc <- pc + 4 + addr << 2

else pc <- pc + 4; op = 000100

I类型：

31-26	25-21	20-16	15-0
op	rs	rt	immediate/addr
6位	5位	5位	16位

• **j**: pc <- (pc+4)[31:28] | (add<<2)[27:0]; op = 000010

J类型：

31-26	25-0
op	address
6位	26位

2. 分析指令功能，设计数据通路和控制器。
3. 指令和数据存储共用一个 RAM 存储器，采用 IP 例化实现，容量为 512x32 位分布式存储器。

（2）ALU 编码表及其功能表：

ALUOP	funct（6bits）/funct_imm(3bits)	ALUControl	ind
10	100000	000	add
10	-	001	sub
10	-	010	and
10	-	011	or
10	-	100	xor
00	-	000	lw,sw
01	-	001	beq
11	-	000	addi

ALU 模块功能表

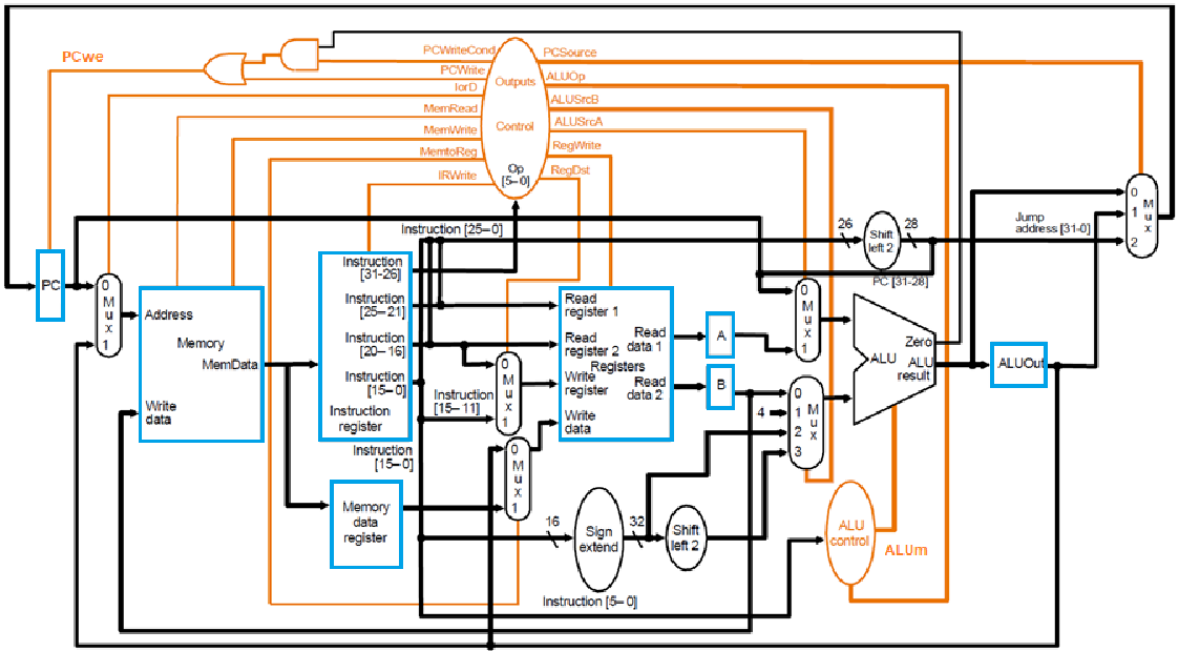
m	y
000	a + b
001	a - b
010	a & b
011	a | b
100	a ^ b
其他	x

ALU 控制单元

```
module ALUcontrol(
    input [5:0]funct,
    input [1:0]ALUOp,
    output reg [2:0]ALUm
    );

always@(*)
begin
    case(ALUOp)
        2'b10: // R-type
        begin
```

```
            if(funct == 6'b100000) ALUm = 3'b000; // add
            else ALUm = 0;
        end
        2'b00:begin ALUm = 3'b000; end // I-type lw/sw
        2'b01:begin ALUm = 3'b001; end // I-type beq
        2'b11:begin ALUm = 3'b000; end // I-type addi
        default: ALUm = 0;
    endcase
end
endmodule
```

(3) 数据通路:



(4) 控制信号含义:

PCWriteCond：分支信号 (== Branch)

PCWrite：PC 写入信号

IorD：选择输入 MemData 的地址（0 - 指令地址；1 - 数据地址）

MemRead：MemData 读使能信号（可有可无）

MemWrite：MemData 写使能信号

MemtoReg：选择写入RegFile 的数据来源（0 - ALUOut;1 - MDR）

IRWrite：Instruction register 的写使能信号，控制当前指令的更新

PCSource：PC 更新选择信号

ALUOp：ALU 操作信号，和 funct 一起控制 ALUcontrol

ALUSrcA：ALU 的 a 端口写入数据来源

ALUSrcB：ALU 的 b 端口写入数据来源

RegWrite：RegFile 寄存器堆写入使能信号

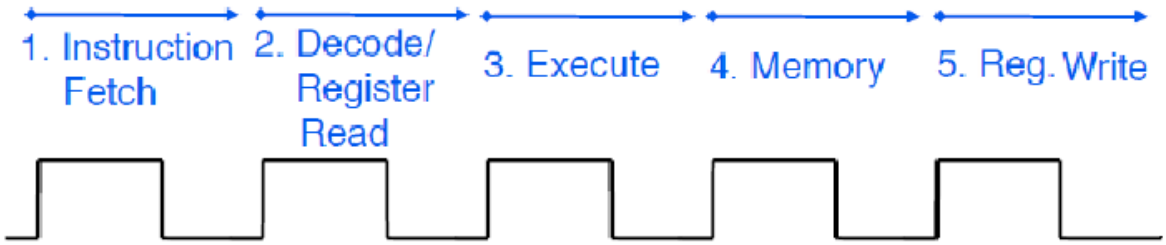
RegDst：寄存器堆写入的寄存器编号选择

数据通路链接 -- CPU 顶层模块

```
module m_cpu(
    input clk,
    input rst
    );
    wire PCWriteCond,
PCWrite,lorD,MemRead,MemWrite,MemtoReg,IRWrite,ALUSrcA,RegWrite,RegDst;
    wire PCwe,zero;
    wire [1:0]ALUOp,PCSource,ALUSrcB;
    wire [31:0]
pcin,pcout,MemD,extend,Jump_addr,ALU_out,ALU_result,Rd1,Rd2,RA,RB,mdr,sw_da
ta;
    wire [31:0] m0_out,m2_out,m3_out,m4_out,m5_out,ex_sl;
    wire [4:0] m1_out,ins_20_16,ins_25_21;
    wire [5:0] ins_31_26;
    wire [15:0] ins_15_0;
    wire [27:0] shift_out;
    wire [25:0] ins_25_0;
    wire [2:0] ALUm;
    wire [10:0] sw_addr;

    assign Jump_addr = pcout[31:28]|shift_out[27:0];
    assign PCwe = PCWrite | (zero & PCWriteCond);
    assign ex_sl = extend << 2;
    assign ins_25_0 = {ins_25_21,ins_20_16,ins_15_0};
    assign sw_addr = 11'b00000001000;
    dist_mem_gen_0 mem_data
(.a(m0_out[10:2]),.d(RB),.clk(clk),.dpra(sw_addr[10:2]),.we(MemWrite),.spo(
MemD),.dpo(sw_data));
    Ins_Reg insreg
(.clk(clk),.rst(rst),.IRWrite(IRWrite),.instruct(MemD),.ins_15_0(ins_15_0),
.ins_20_16(ins_20_16),.ins_25_21(ins_25_21),.ins_31_26(ins_31_26));
    Reg_file regfile
(.clk(clk),.ra0(ins_25_21),.rd0(Rd1),.ra1(ins_20_16),.rd1(Rd2),.we(RegWrite
),.wa(m1_out),.wd(m2_out));
    Register MemDR (.clk(clk),.rst(rst),.data(MemD),.exout(mdr));
    Register RegA (.clk(clk),.rst(rst),.data(Rd1),.exout(RA));
    Register RegB (.clk(clk),.rst(rst),.data(Rd2),.exout(RB));
    Register ALUOut
(.clk(clk),.rst(rst),.data(ALU_result),.exout(ALU_out));

    PC pc (.clk(clk),.rst(rst),.PCen(PCwe),.pcin(pcin),.ExtOut(pcout));

    mux2to1_d32 m0 (.a(pcout),.b(ALU_out),.sel(lorD),.out(m0_out));
    mux2to1_d5 m1
(.a(ins_20_16),.b(ins_15_0[15:11]),.sel(RegDst),.out(m1_out));
    mux2to1_d32 m2 (.a(ALU_out),.b(mdr),.sel(MemtoReg),.out(m2_out));
    mux2to1_d32 m3 (.a(pcout),.b(RA),.sel(ALUSrcA),.out(m3_out));
    mux4to1_d32 m4
(.a(RB),.b(4),.c(extend),.d(ex_sl),.sel(ALUSrcB),.out(m4_out));
    mux3to1_d32 m5
(.a(ALU_result),.b(ALU_out),.c(Jump_addr),.sel(PCSource),.out(pcin));

    sign_extend SE (.imm(ins_15_0),.ExtOut(extend));
    shift_left_d25 SL (.addr(ins_25_0),.new_addr(shift_out));

    ALU alu (.a(m3_out),.b(m4_out),.y(ALU_result),.m(ALUm),.zf(zero));
    ALUcontrol alu_con (.funct(ins_15_0[5:0]),.ALUOp(ALUOp),.ALUm(ALUm));

```

```
46      ControlUnit CU
(.Op(ins_31_26),.clk(clk),.rst(rst),.ALUOp(ALUOp),.PCSource(PCSource),.ALUS
rcB(ALUSrcB),.PCWriteCond(PCWriteCond),.PCWrite(PCWrite),.IorD(IorD),.MemRe
ad(MemRead),.MemWrite(MemWrite),.MemtoReg(MemtoReg),.IRWrite(IRWrite),.ALUS
rcA(ALUSrcA),.RegWrite(RegWrite),.RegDst(RegDst));
47
48
49  endmodule
```

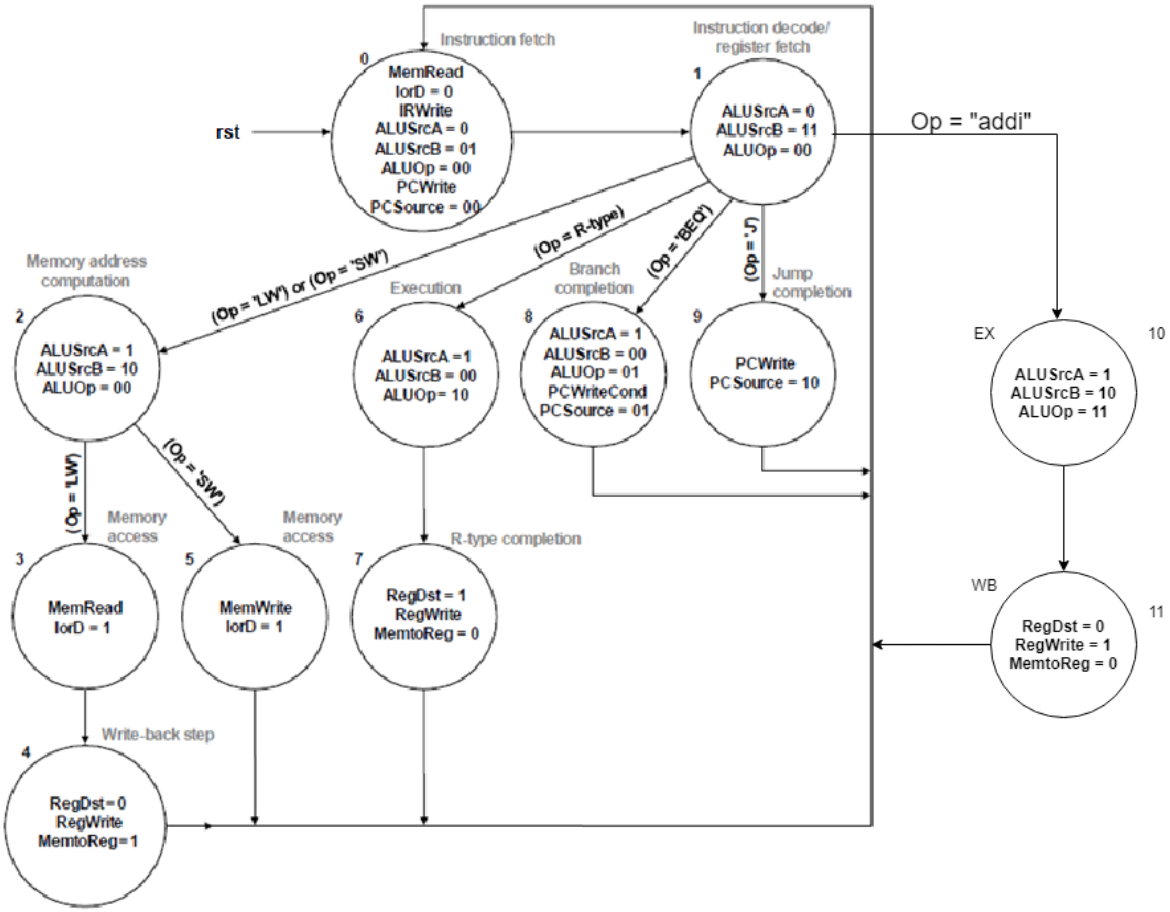
(5) 多周期操作：



Step	R-Type	lw/sw	beq/bne	j
IF	IR = Mem[PC] PC = PC + 4			
ID	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (SE(IR[15-0]) << 2)			
EX	ALUOut = A op B	ALUOut = A +SE(IR[15-0])	If (A==B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0]<<2
MEM	Reg[IR[15-11]] = ALUOut	MDR=Mem[ALUOut] Mem[ALUOut] = B		
WB		Reg[IR[20-16]] = MDR		

(6) 状态图：

> 根据多周期对应操作的控制信号实现



控制单元 ControlUnit

```
module ControlUnit(
    input [5:0]Op,
    input clk,
    input rst,
    output reg PCWriteCond,
    output reg PCWrite,
    output reg IorD,
    output reg MemRead,
    output reg MemWrite,
    output reg MemtoReg,
    output reg IRWrite,
    output reg [1:0]PCSource,
    output reg [1:0]ALUOp,
    output reg ALUSrcA,
    output reg [1:0]ALUSrcB,
    output reg RegWrite,
    output reg RegDst
    );
    parameter BEGIN = 3'b000,
              IF = 3'b001,
              ID = 3'b010,
              EX = 3'b011,
              MEM = 3'b100,
              WB = 3'b101;

    parameter op_add = 6'b000000,
              op_addi = 6'b001000,
              op_lw = 6'b100011,
              op_sw = 6'b101011,
```

```
30              op_beq = 6'b000100,
31              op_j = 6'b000010;
32
33      parameter aluop_R = 2'b10,
34              aluop_lw = 2'b00,
35              aluop_sw = 2'b00,
36              aluop_addi = 2'b11,
37              aluop_beq = 2'b01;
38
39      reg [2:0] curr_state,next_state;
40
41      always@(posedge clk or posedge rst)
42      begin
43          if(rst) curr_state <= BEGIN;
44          else curr_state <= next_state;
45      end
46
47      always@(*)
48      begin
49          case(curr_state)
50              BEGIN:begin next_state = IF; end
51              IF:begin next_state = ID; end
52              ID:begin next_state = EX; end
53              EX:
54              begin
55                  if(Op == op_add || Op == op_addi) next_state = WB;
56                  else if (Op == op_j || Op == op_beq) next_state = IF;
57                  else next_state = MEM;
58              end
59              MEM:
60              begin
61                  if(Op == op_sw) next_state = IF;
62                  else next_state = WB;
63              end
64              WB:begin next_state = IF; end
65              default next_state = BEGIN;
66          endcase
67      end
68
69      always@(*)
70      begin
71          PCWrite = 0;
72          PCWriteCond = 0;
73          IRWrite = 0;
74          RegWrite = 0;
75          MemRead = 0;
76          MemWrite = 0;
77          MemtoReg = 0;
78          ALUSrcA = 0;
79          ALUSrcB = 2'b01;
80          ALUOp = 2'b00;
81          IorD = 0;
82          PCSource = 2'b00;
83          RegDst = 0;
84
85          case(curr_state)
86              BEGIN:;
87              IF:
```

```
            begin
                PCWrite = 1;
                IRWrite = 1;
            end
            ID:
            begin
                ALUSrcB = 2'b11;
            end
            EX:
            begin
                case(Op)
                    op_add:
                    begin
                        ALUSrcA = 1;
                        ALUSrcB = 2'b00;
                        ALUOp = aluop_R;
                    end
                    op_j:
                    begin
                        PCSource = 2'b10;
                        PCWrite = 1;
                    end
                    op_beq:
                    begin
                        ALUSrcA = 1;
                        ALUSrcB = 2'b00;
                        ALUOp = aluop_beq;
                        PCSource = 1;
                        PCWriteCond = 1;
                        PCWrite = 0;
                    end
                    op_addi:
                    begin
                        ALUSrcA = 1;
                        ALUSrcB = 2'b10;
                        ALUOp = aluop_addi;
                    end
                    op_lw,op_sw:
                    begin
                        ALUSrcA = 1;
                        ALUSrcB = 2'b10;
                        ALUOp = aluop_lw;
                    end
                    default:begin ALUOp = 00; end
                endcase
            end
            MEM:
            begin
                case(Op)
                    op_lw:
                    begin
                        MemRead = 1;
                        lorD = 1;
                    end
                    op_sw:
                    begin
                        MemWrite = 1;
                        lorD = 1;
```

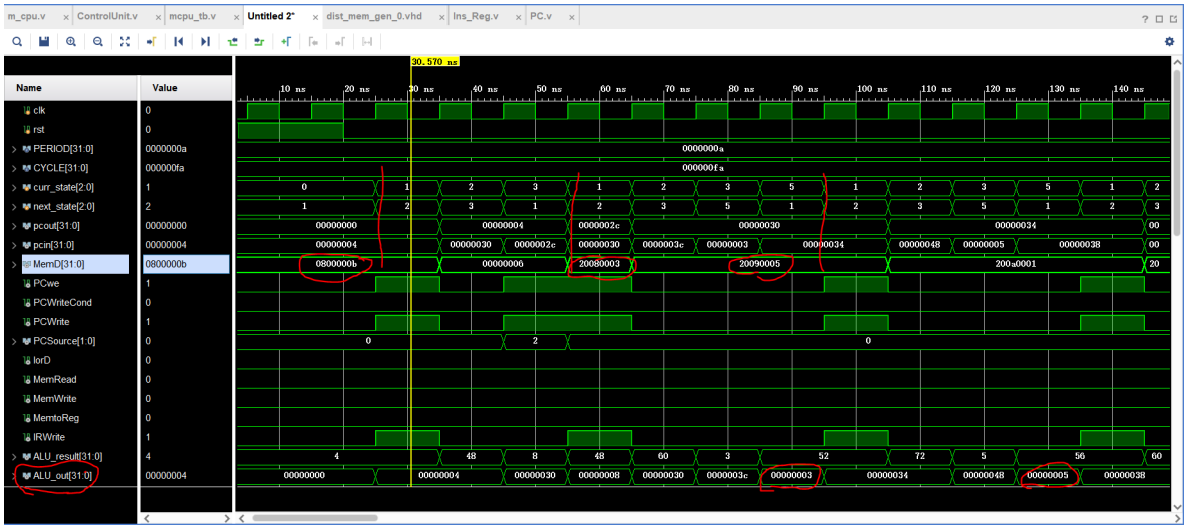
```
                    end
                    default:begin MemRead = 0; MemWrite = 0; end
                endcase
            end
            WB:
            begin
                case(Op)
                    op_add,op_addi:
                    begin
                        MemtoReg = 0;
                        RegWrite = 1;
                        RegDst = (Op == op_add)?1:0;
                    end
                    op_lw:
                    begin
                        RegWrite = 1;
                        RegDst = 0;
                        MemtoReg = 1;
                    end
                    default:begin RegWrite = 0; end
                endcase
            end
            default:begin MemRead = 0; MemWrite = 0; RegWrite = 0;end
        endcase

    end
endmodule
```

2. 测试指令及其仿真结果

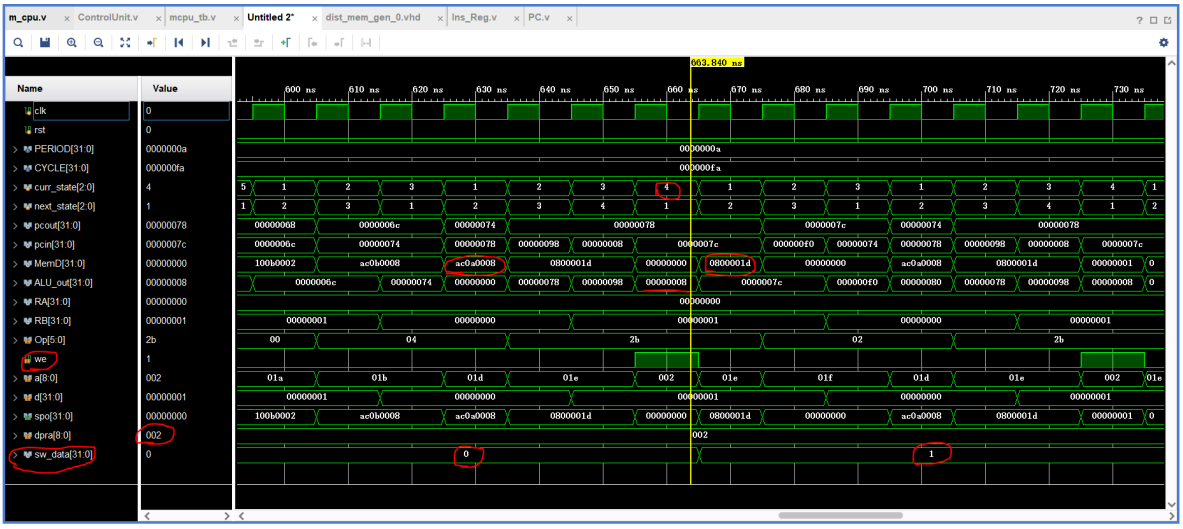
测试指令：

```
# 本文档存储器以字编址
# 本文档存储器以字编址
# 本文档存储器以字编址
# 初始PC = 0x00000000

    j _start    # 0

.data
    .word 0,6,0,8,0x80000000,0x80000100,0x100,5,0,0,0   #编译成机器码时，编译器会在前面多加个0，所以后面lw指令地址会多加4

_start:
        addi $t0,$0,3           #t0=3   44
        addi $t1,$0,5           #t1=5   48
        addi $t2,$0,1           #t2=1   52
        addi $t3,$0,0           #t3=0   56

        add  $s0,$t1,$t0        #s0=t1+t0=8  测试add指令     60
        lw   $s1,12($0)         #                             64
        beq  $s1,$s0,_next1     #正确跳到_next                68

        j _fail

_next1:
        lw $t0, 16($0)          #t0 = 0x80000000    76
```

```
25          lw $t1, 20($0)          #t1 = 0x80000100    80
26
27          add  $s0,$t1,$t0        #s0 = 0x00000100 = 256  84
28          lw $s1, 24($0)          #                       88
29          beq  $s1,$s0,_next2     #正确跳到_success       92
30
31          j _fail
32
33  _next2:
34          add $0, $0, $t2         #$0应该一直为0          100
35          beq $0,$t3,_success     #                       104
36
37
38  _fail:
39          sw   $t3,8($0) #失败通过看存储器地址0x08里值，若为0则测试不通过，最初地址
    0x08里值为0 108
40          j    _fail
41
42  _success:
43          sw   $t2,8($0)    #全部测试通过，存储器地址0x08里值为1 116
44          j   _success
45
46                      #判断测试通过的条件是最后存储器地址0x08里值为1，说明全部通过测
    试
```



每个 curr_state = 1 时MemD 的输出数据，即为取指操作中取出的指令，addi 指令的写回状态（5）中 ALU_out 的值即为写回值。



全部测试通过，存储器地址 0x08 里值为1

三、实验总结

本次实验主要有以下收获：

1. 熟练掌握多周期 CPU 的数据通路和各个功能块的设计与实现，能够通过控制单元来实现一些简单的指令。
2. 能够熟练使用仿真波形调试代码。
3. 对 mips 指令执行的各个过程有了更加详细的了解和认识。
4. 提高了自己的逻辑思维能力和工程能力。