

浙江大学

本科实验报告

课程名称：计算机体系结构

姓 名：郑昱笙

学 院：计算机科学与技术学院

系：计算机科学与技术系

专 业：地理信息科学

学 号：3180102760

指导教师：常瑞

2020 年 4 月 5 日

浙江大学实验报告

课程名称： 计算机体系结构 实验类型： 综合

实验项目名称： Project 2 多周期 CPU 设计

学生姓名： 郑昱笙 专业： 地理信息科学 学号： 3180102760

同组学生姓名： 无 指导老师： 常瑞

实验地点： 家里 实验日期： 20202 年 4 月 5 日

一、 实验目的和要求

1. Design the CPU Controller, Datapath, bring together the basic units into Multiple-cycle CPU;
2. Verify the MC CPU with program and observe the execution of program.

二、 实验内容和原理

- 1、 本实验实现了 18 条指令的多周期 CPU，最长周期数为 5 个周期：

R 型指令：

ADD、SUB、SLT、AND、OR、XOR、NOR

I 型指令：

ADDi、SLTi、ANDi、ORi、XORi

SW、LW

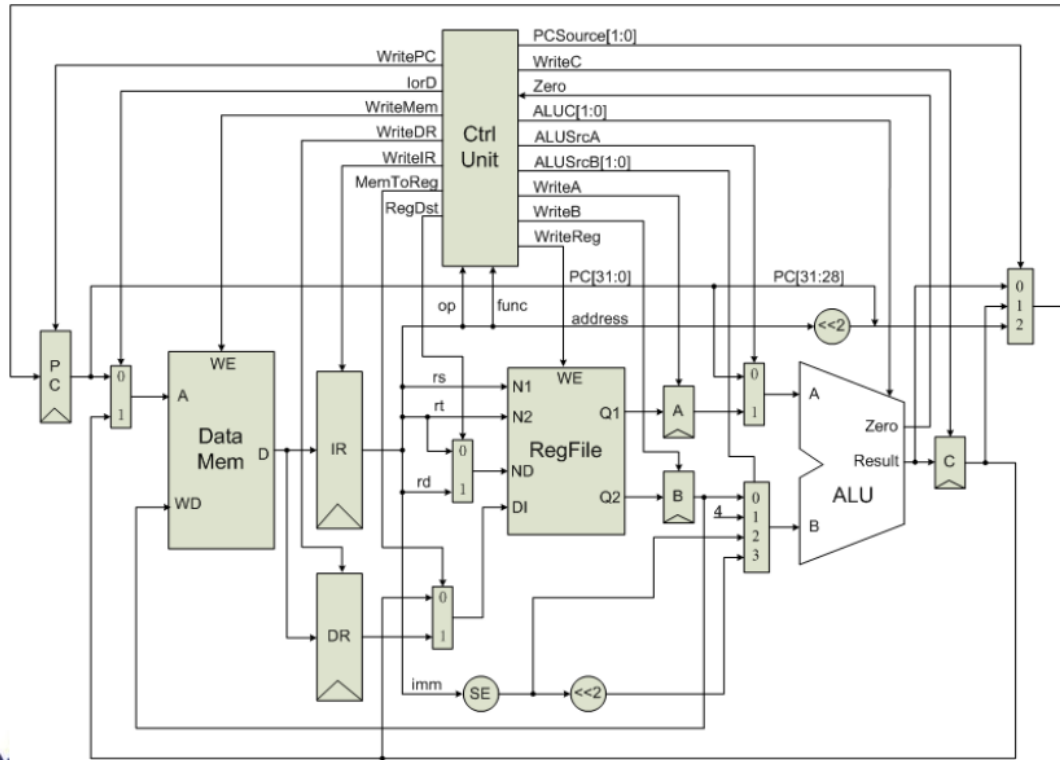
BEQ、BNE

J 类型指令：

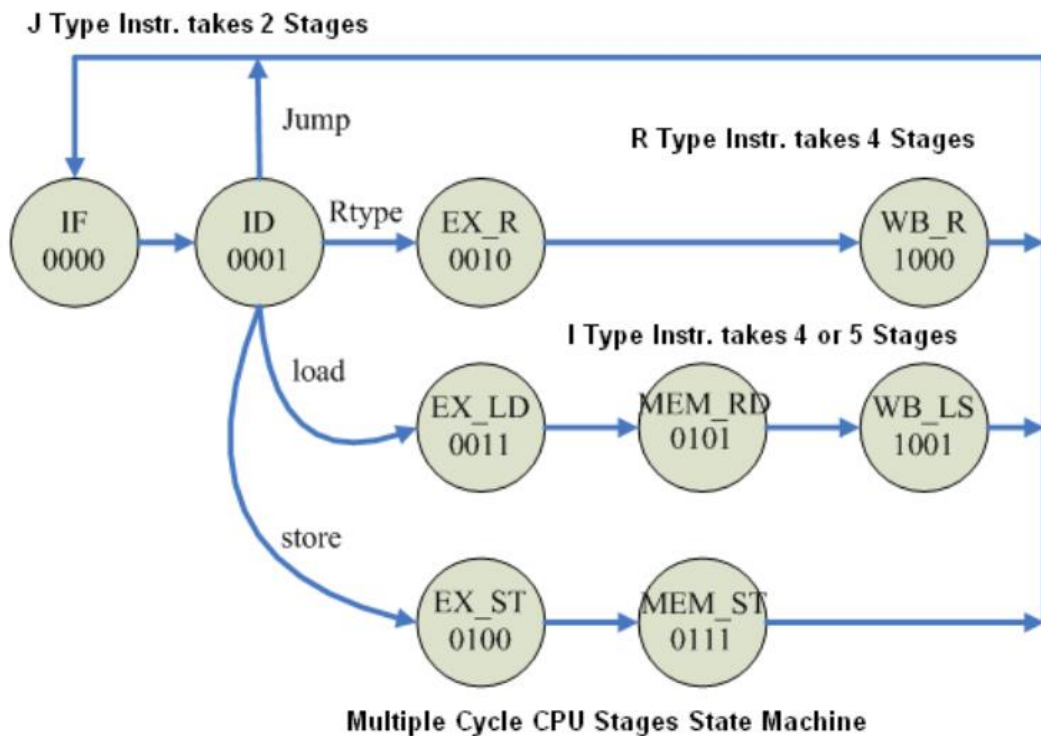
J

以 32 位为最小操作、编址单位

- 2、 本实验所设计的多周期 CPU 数据通路图：
其中 ALU 操作码改为 3 位，实现八种运算；



3、 本实验所设计的 CPU 控制器状态机图：



4、 输入输出：

BTN2 为 reset

显示屏第一行显示指令和当前控制器状态，

第二行显示 ALU 输出和存储器访问地址；

5、 CPU 控制器采用 HDL 直接描述实现状态机：大致代码结构如下：

```

always @(posedge clk or posedge rst) begin
    if(rst) begin
        startIF;
    end
    else begin
        case(state)
            IF:begin
                .....
            end
            ID:begin
                case(op)
                    J:begin
                        .....
                    end
                    R:begin
                        case(func)
                            ADD: ALUC <=3'b000;
                            SUB: ALUC <=3'b001;
                            SLT: ALUC <=3'b111;
                            AND: ALUC <=3'b010;
                            OR:  ALUC <=3'b011;
                            XOR: ALUC <=3'b100;
                            NOR: ALUC <=3'b101;
                        Endcase
                    end
                    ....
                end
                ....
            endcase
        end
        EX_R:begin
            .....
        endcase
    end
    ....
endcase
end
end

```

6、 存储器实现:

```

module ram(
    input [31:0] datain,
    input [31:0] addr,
    input WriteMem,
    output [31:0] out
);
    reg [31:0] data[255:0];
    assign out = data[addr[9:2]];
    always@(posedge WriteMem) begin
        data[addr[9:2]] <= datain;
    end
endmodule

```

测试存储器内存最大深度设置为 256，取输出地址码的 2-9 位作为地址码；后续考虑采用 ip 核生成方式设计存储器。

三、 实验过程和数据记录

1、 ALU 设计：

采用纯逻辑方式实现：

```
module ALU(  
    input [31:0] A,  
    input [31:0] B,  
    input [2:0] ALUC,  
    output [31:0] result,  
    output zero  
);  
assign result =  
    ALUC[2]?  
    (ALUC[1]?(  
        ALUC[0]? {31'b0, $signed(A) < $signed(B)} : {31'b0, A < B})  
    :(  
        ALUC[0]? ~(A|B) : A^B )):  
    (ALUC[1]?(  
        ALUC[0]? A|B : A&B):(  
        ALUC[0]? A-B : A+B));  
assign zero = ~|result;  
  
endmodule
```

ALU 操作码对应的运算分别为：

- 000: add
- 001: sub
- 010: and
- 011: or
- 100: xor
- 101: nor
- 110: slt (无符号数)
- 111: slt (有符号数)

ALU 测试代码：

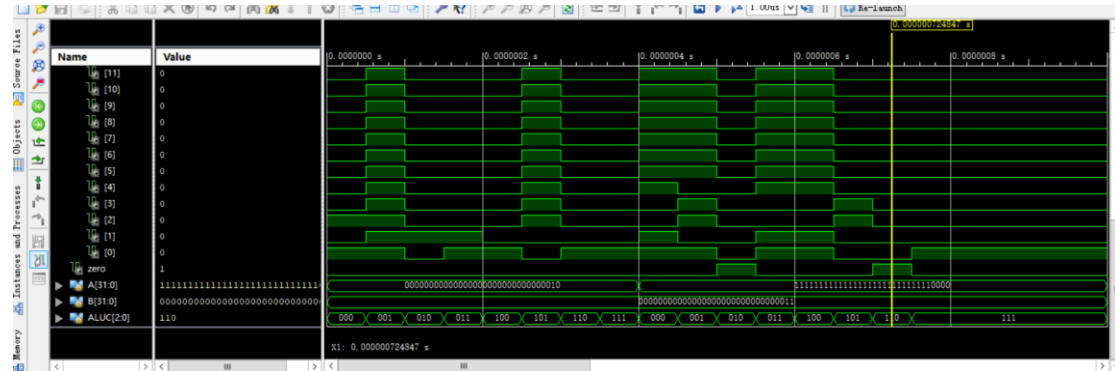
```
initial begin  
    // Initialize Inputs  
    A = 32'h00000002;  
    B = 32'h00000003;  
    for(ALUC=0;ALUC<7;ALUC=ALUC+1) begin  
        #50;  
    end  
    #50;  
    A = 32'hffffffff0;
```

```

B = 32'h00000003;
for(ALUC=0;ALUC<7;ALUC=ALUC+1) begin
    #50;
end
end
end

```

ALU 测试波形：（显示结果的后几位波形）



2、 CPU 测试：

汇编语言测试代码：从内存中读取连续的五个数值的值，并计算它们的和，将和存入内存中：

.text

```

main:    addi $5, $0, 5      # set loop
         addi $4, $0, 48    # load data address
         add $8, $0, $0     # sum = 0
loop:    lw $9, 0($4)       # load data
         add $8, $8, $9     # sum
         addi $5, $5, -1    # counter - 1
         addi $4, $4, 4     # address + 4
         slt $3, $0, $5     # finish?
         bne $3, $0, loop   # finish?
         or $2, $8, $0      # move result to $v0
         sw $2, 0($4)       # store sum
         j main             # jump to the begining

```

.data

```
data: .word 0x00000001, 0x00000002, 0x00000003, 0x00000004, 0x00000005
```

结果应为 0x0000000f

编译为十六进制机器码，存入 memory.list 中，通过使用

```
$readmemh("memory.list",data);
```

系统命令进行加载：

verilog 测试代码：

```

initial begin
    clk = 0;
    rst = 1;
    #10;

```

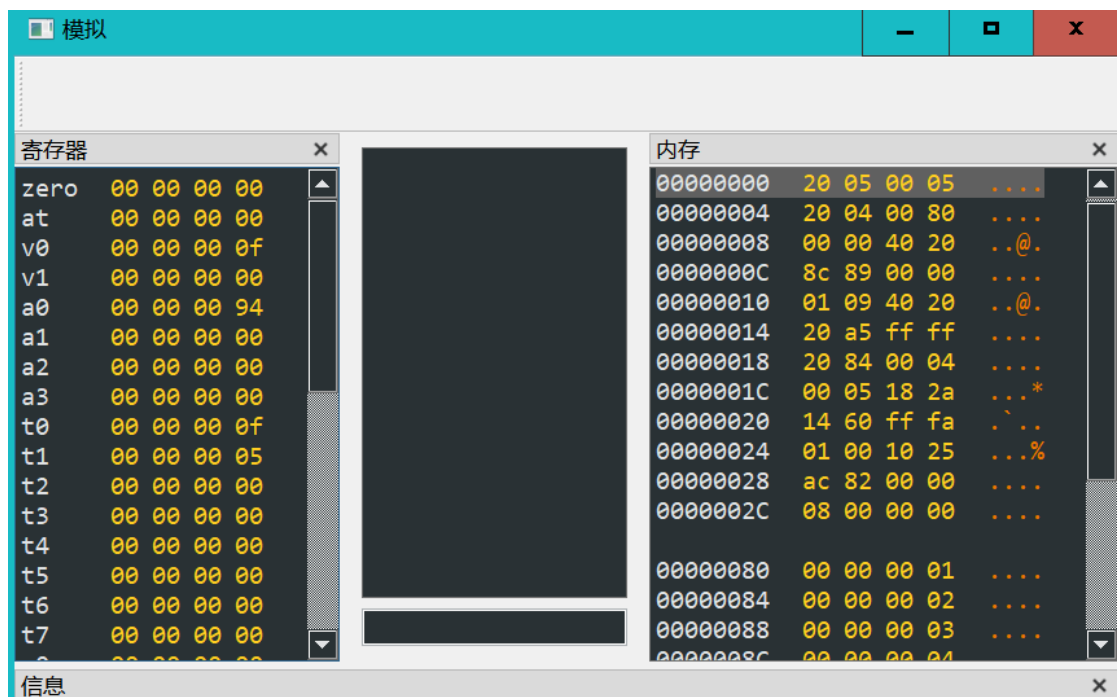
```
rst = 0;
end
```

测试结果

在上述代码刚运行结束跳转回 PC=0 时：



与模拟器进行对照，寄存器数据相同



内存数据：可见 sum 值为 0x0000000f

Name	Value
[20,31:0]	00000000000000000000000000000000
[19,31:0]	00000000000000000000000000000000
[18,31:0]	00000000000000000000000000000000
[17,31:0]	00000000000000000000000000001111
[16,31:0]	00000000000000000000000000000101
[15,31:0]	00000000000000000000000000000100
[14,31:0]	00000000000000000000000000000011
[13,31:0]	00000000000000000000000000000010
[12,31:0]	00000000000000000000000000000001
[11,31:0]	00001000000000000000000000000000
[10,31:0]	10101100100000100000000000000000
[9,31:0]	00000001000000000001000000100101
[8,31:0]	0001010001100000111111111111010
[7,31:0]	00000000000001010001100000101010
[6,31:0]	00100000100001000000000000000100
[5,31:0]	00100000101001011111111111111111
[4,31:0]	000000010000100101000000010000

可见程序运行正确。

四、 讨论与心得

在实现这个CPU的过程中,对CPU的运行原理和设计方法有了一个直观的了解;不过这两天实现的这个还是一个非常简陋的多周期CPU,测试也不是很全面,希望能在之后的组成和体系结构课程中慢慢改进吧。