实验六 CPU 综合设计

一、实验目的

1. 掌握复杂系统设计方法。
2. 深刻理解计算机系统硬件原理。

二、实验内容

1. 设计一个基于 MIPS 指令集的 CPU，支持以下指令：{addu, subu, ori, lw, sw, beq, lui, nop}（及格）
2. CPU 需要包含寄存器组、RAM 模块、ALU 模块、指令译码模块。
3. 该 CPU 能运行基本的汇编指令（编写测试程序完成所有指令测试，要求与 MARS 模拟器运行结果一致）。
4. 在 1 基础上，扩展指令集，实现 MIPS-Lite 指令，见下页。（A-~A，编写测试程序完成所有指令测试，要求与 MARS 模拟器运行结果一致）
5. 在 4 基础上，实现 5 级流水线 CPU。（A+，编写测试程序完成所有指令测试，要求与 MARS 模拟器运行结果一致）
6. 如发现代码为网上下载代码，成绩一律按不及格处理。

三、实验要求

1. 编写相应测试程序，完成所有指令测试。

四、实验代码及结果

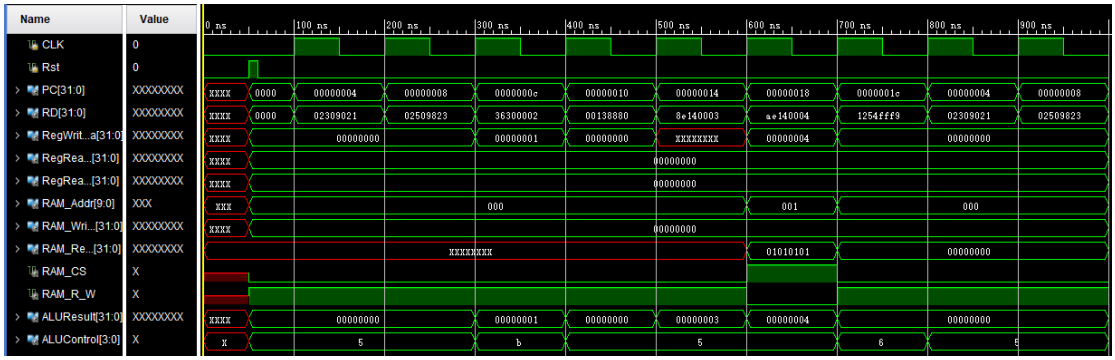
本实验设计的能实现 MIPS-Lite 指令的 CPU，采用哈佛结构，即数据和指令分开存储，主要模块包括：程序计数器 PC，能实现地址的自增，使得 CPU 能按照指令存储器中的指令顺序执行，并使指令存在 RD 中；控制器 Controller，能将读取到的指令译码并发出各个对应的模块的控制信号；寄存器堆 RegFile，包括各个指令中的用到的寄存器；数据存储器 RAM，负责存储数据，包括初始数据与保存的运算结果，实验中，RAM 的初始数据是随机设定了一组数据存在其中，通过 ram.txt 文件引入模块中；算术逻辑运算单元 ALU，负责完成指令中的各种算数逻辑运算，并输出结果；若干的多路选择器 mux，负责通过不同信号线进行输入输出的选择。

汇编代码：

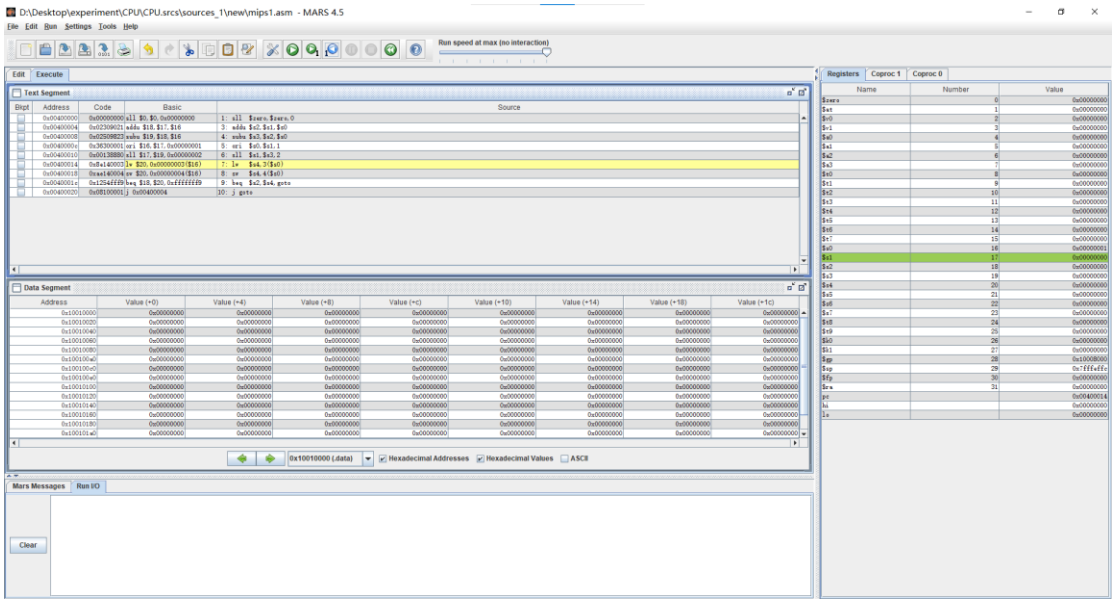
```
sll  $zero,$zero,0
goto:
addu $s2,$s1,$s0
subu $s3,$s2,$s0
ori  $s0,$s1,1
sll  $s1,$s3,2
lw   $s4,3($s0)
sw   $s4,4($s0)
beq  $s2,$s4,goto
j goto
```

对应十六进制保存在 mips1.txt 中

仿真波形：



mars 模拟会报错，只能执行到 lw 命令前，结果为：



对结果的分析：

以 addu $s2,$s1,$s0 指令为例：将寄存器 s1s0 数据相加并将结果写入 s2。对应十六进制指令为 02309021。

其指令运行过程为，pc 对应地址在指令存储器中取出指令后，交由 Controller 进行译码，产生的控制信号：由于 addu 为 R 型指令，其特征为十六进制指令中 op 部分对应为 000000，产生信号为 RegWrite,RegDst 置为高电位，ALUOP 信号使 ALU 进行加法运算。其中 Reg Write 为 RegFile 的写信号，置高即允许寄存器堆进行写操作，RegDst 则是控制 mux 使另一个操作数类型为寄存器。各个控制信号控制从 RegFile 中读出数据后，将数据输入 ALU 进行运算，并将结果写回目标寄存器。

从波形来看，由于 s1 和 s0 寄存器均初始化为 0，所以读出的数据、ALU 运算结果、写回的数据均为 0。从 mars 模拟结果 s2 结果为 0。

以 ori $s0,$s1,1 指令为例：将寄存器 s1 数据与立即数 1 求或并将结果写入 s0。对应十六进制指令为 36300002。

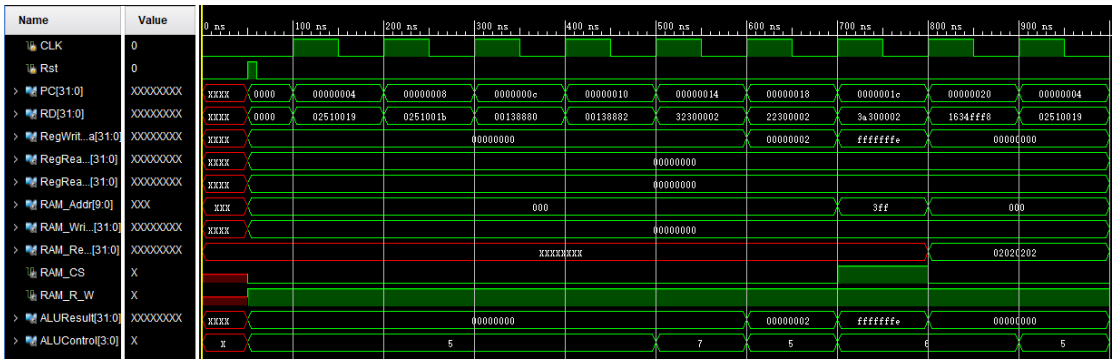
其指令运行过程为，pc 对应地址在指令存储器中取出指令后，交由 Controller 进行译码，产生的控制信号：由于 ori 为 I 型指令，其十六进制指令中 op 部分对应为 001101，产生信号为 RegWrite， ALUSrc 置为高电位，ALUOP 信号使 ALU 进行或运算。其中 Reg Write 为 RegFile 的写信号，置高即允许寄存器堆进行写操作，ALUSrc 则是控制 mux 使另一个操作数类型为立即数。各个控制信号控制从 RegFile 中读出数据，从指令中读出立即数后，将数据

输入 ALU 进行运算，并将结果写回目标寄存器。
从波形来看，由于 s1 寄存器均初始化为 0，立即数为 0000 0000 0000 0001，故或结果为 0000 0000 0000 0001，所以 ALU 运算结果、写回的数据均为 1。从 mars 模拟结果 s0 结果为 1。

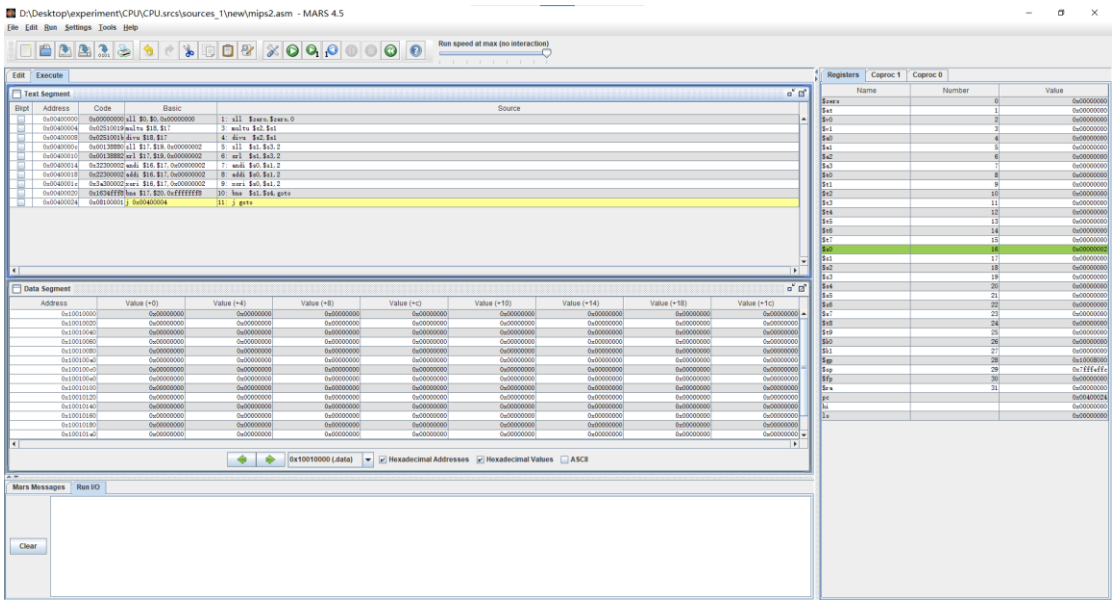
下图对应汇编代码：

```
sll  $zero,$zero,0
goto:
multu $s2,$s1
divu  $s2,$s1
sll  $s1,$s3,2
srl  $s1,$s3,2
andi $s0,$s1,2
addi $s0,$s1,2
xori $s0,$s1,2
bne  $s1,$s4,goto
j goto
```

对应的十六进制保存在 mips2.txt 中。
仿真波形：

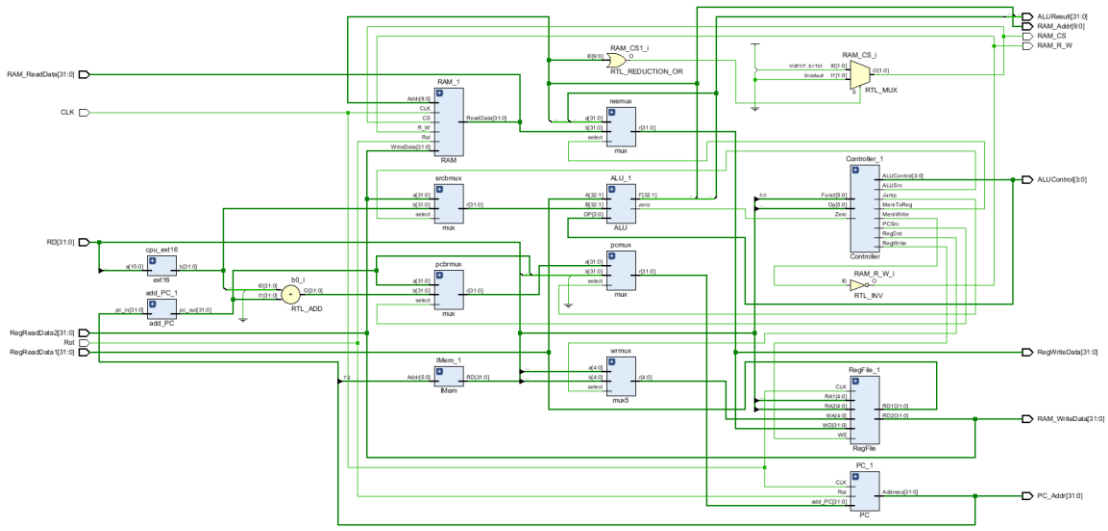


mars 模拟结果：
由于没有设置终止指令，j 跳转指令会产生循环，故模拟不会结束，所以手动定位到 j 指令。



以 j goto 指令为例，无条件跳转，对应十六进制指令为 08100001。
其指令运行过程为，pc 对应地址在指令存储器中取出指令后，交由 Controller 进行译码，产生的控制信号：由于 j 为 J 型指令，其十六进制指令中 op 部分对应为 000010，产生信号为 Jump 置为高电位表示为跳转型指令，控制 mux 使 pc 输入不为自增结果，而转为指令中的偏移地址。
从波形来看，看最后两个取值周期，前一个为执行 bne $s1,$s4,goto 指令，而后一个为执行 multu $s2,$s1 指令，可见正常跳转。

线路图：



代码汇总：
CPU.v

```
`define DATA_WIDTH 32
`define instruction_width 32
`define RAM_Addr_Width 10
module CPU(
    input CLK,
    input Rst,
    input wire [`instruction_width-1:0] RD, //指令
    output [`DATA_WIDTH-1:0] PC_Addr,
    output wire [`DATA_WIDTH-1:0] RegWriteData,
    input wire [`DATA_WIDTH-1:0] RegReadData1, RegReadData2,
    output wire [`DATA_WIDTH-1:0] RAM_WriteData,
    input wire [`DATA_WIDTH-1:0] RAM_ReadData,
    output wire [`RAM_Addr_Width-1:0] RAM_Addr,
    output RAM_CS,
    output RAM_R_W,
    output wire [`DATA_WIDTH-1:0] ALUResult,
    output wire [3:0] ALUControl);

    wire PCSrc,Zero,MemToReg, MemWrite, ALUSrc, RegDst, RegWrite, Jump, C1;
```

```
    wire [`DATA_WIDTH-1:0] ext16_out;
    wire [4:0] RegWriteAddr;
    //wire [3:0] ALUControl;
    wire [`DATA_WIDTH-1:0] ALU_A, ALU_B;
    wire [`DATA_WIDTH-1:0] add_PC;
    wire [`DATA_WIDTH-1:0] pcbranch, pcnextbr, pcnext;

    mux pcbrmux(add_PC, ({ext16_out[29:0], 2'b00} + add_PC), PCSrc, pcnextbr);
    mux pcmux(pcnextbr, {add_PC[31:28], RD[25:0], 2'b00}, Jump, pcnext);
    PC PC_1(CLK, Rst, pcnext, PC_Addr);
    add_PC add_PC_1(PC_Addr, add_PC);
    IMem IMem_1(PC_Addr[7:2], RD);
    //Controller([5:0] Op, Funct,Zero,MemToReg, MemWrite,PCSrc, ALUSrc,RegDst,
RegWrite,Jump,[3:0] ALUControl);
    Controller      Controller_1(RD[`instruction_width-1:`instruction_width-6],
RD[5:0], Zero,
    MemToReg, MemWrite, PCSrc, ALUSrc, RegDst, RegWrite, Jump, ALUControl);
    //RegFile(CLK,    Reg_WE,    RegReadAddr1,    RegReadAddr2,    RegWriteAddr,
RegWriteData, RegReadData1, RegReadData2);
    RegFile  RegFile_1(CLK,  RegWrite,  RD[25:21],  RD[20:16],  RegWriteAddr,
RegWriteData, RegReadData1, RegReadData2);
    mux5 wrmux(RD[20:16], RD[15:11], RegDst, RegWriteAddr);
    ext16 cpu_ext16(RD[15:0], ext16_out);
    ALU ALU_1(ALUControl, RegReadData1, ALU_B, ALUResult, Zero);
    mux srcbmux(RegReadData2, ext16_out, ALUSrc, ALU_B);
    //addr_32bit addr_32bit_1(pcbranch, C1, (ext16_out<<2), add_PC, 0);
    //RAM(WriteData, ReadData,  Addr,  Rst,  R_W,  CS,  CLK) R_W 1 read
    RAM RAM_1(RAM_WriteData, RAM_ReadData, RAM_Addr, Rst, RAM_R_W, RAM_CS, CLK);
    assign RAM_WriteData = RegReadData2;
    assign RAM_Addr = ALUResult[11:2];
    assign RAM_R_W = ~MemWrite;
    assign RAM_CS = (RAM_Addr) ? 1 : 0;
    mux resmux(ALUResult, RAM_ReadData, MemToReg, RegWriteData);
endmodule
```

PC.v

```
module PC(
    input CLK, Rst,
    input [31:0] add_PC,
    output reg [31:0] Address);

    always @(posedge CLK, posedge Rst)
    begin
        if (Rst)
```

```
            Address <= 32'b0;
        else
            Address <= add_PC;  // 跳转到下一指令
    end
endmodule
```

add_PC.v

```
module add_PC(
    input [31:0] pc_in,
    output [31:0] pc_out);
    assign pc_out = pc_in + 32'd4;    //pc=pc+4
endmodule
```

ALU.v

```
module ALU(OP, A, B, F, zero);
    parameter SIZE = 32;
    input [3:0] OP;
    input [SIZE:1] A;
    input [SIZE:1] B;
    output reg [SIZE:1] F;
    output reg zero;

    always @(*)
    begin
        case(OP)
            4'b0000: begin F = A << B; end
            4'b0001: begin F = A >>> B; end
            4'b0010: begin F = A >> B; end
            4'b0011: begin F = A * B; end
            4'b0100: begin F = A / B; end
            4'b0101: begin F = A + B; end
            4'b0110: begin F = A - B; end
            4'b0111: begin F = A & B; end
            4'b1000: begin F = A | B; end
            4'b1001: begin F = A^B; end //??
            4'b1010: begin F = ~(A | B); end
            4'b1011: begin
            if(A[SIZE-1]==0 && B[SIZE-1]==1) F = 0;
            else if(A[SIZE-1]==1 && B[SIZE-1]==0) F = 1;
            else if(A[SIZE-1]==0 && B[SIZE-1]==0) F = A < B;
            else  F = ~(A < B);
            end
            4'b1100: begin F = A < B; end
        endcase
```

```
        zero = F ? 0 : 1;
    end
endmodule

Controller.v
module Controller(
    input [5:0] Op, Funct,
    input Zero,
    output MemToReg, MemWrite,
    output PCSrc, ALUSrc,
    output RegDst, RegWrite,
    output Jump,
    output [3:0] ALUControl);

    wire [1:0] ALUOp;
    wire Branch;

    MainDec MainDec_1(Op, MemToReg, MemWrite, Branch, ALUSrc, RegDst, RegWrite,
Jump, ALUOp);
    ALUDec ALUDec_1(Funct, ALUOp, ALUControl);
    assign PCSrc = Branch & Zero;
endmodule

ALUDec.v
module ALUDec(
    input [5:0] Funct,
    input [1:0] ALUOp,
    output reg [3:0] ALUControl);

    always@(*)
        case(ALUOp)
            3'b000: ALUControl <= 4'b0101;//add (for lw/sw/addi)
            3'b010: ALUControl <= 4'b0110;//sub (for beq)
            3'b011: ALUControl <= 4'b1011;//liu/slt/sltu
            3'b111: ALUControl <= 4'b1000;//ori
            3'b001: ALUControl <= 4'b0111;//andi
            3'b110: ALUControl <= 4'b1001;//xori
            default :case(Funct)        //R-type Instructions
                6'b100000: ALUControl <= 4'b0101;//add
                6'b100001: ALUControl <= 4'b0101;//addu
                6'b100010: ALUControl <= 4'b0110;//sub
                6'b100011: ALUControl <= 4'b0110;//subu
                6'b100100: ALUControl <= 4'b0111;//and
                6'b100101: ALUControl <= 4'b1000;//or
```

```
                6'b100110: ALUControl <= 4'b1001;//xor
                6'b100111: ALUControl <= 4'b1010;//nor
                6'b101010: ALUControl <= 4'b1011;//slt
                6'b101011: ALUControl <= 4'b1100;//sltu
                6'b000000: ALUControl <= 4'b0000;//sll
                6'b000010: ALUControl <= 4'b0010;//srl
                6'b000011: ALUControl <= 4'b0001;//sra
                6'b000100: ALUControl <= 4'b0000;//sllv
                6'b000110: ALUControl <= 4'b0010;//srlv
                6'b000111: ALUControl <= 4'b0001;//srav
                6'b011000: ALUControl <= 4'b0011;//mult
                6'b011001: ALUControl <= 4'b0011;//multu
                6'b011010: ALUControl <= 4'b0100;//div
                6'b011011: ALUControl <= 4'b0100;//divu
                6'b001000: ALUControl <= 4'b0101;//jr
                default:   ALUControl <= 4'bxxxx;//???
            endcase
        endcase
endmodule

MainDec.v
module MainDec(
    input [5:0] Op,
    output MemToReg, MemWrite,
    output Branch, ALUSrc,
    output RegDst, RegWrite,
    output Jump,
    output [2:0] ALUOp);

    reg [8:0] Controls;

    assign {RegWrite, RegDst, ALUSrc, Branch, MemWrite, MemToReg, Jump, ALUOp}
= Controls;
    always@(*)
        case(Op)
            6'b000000: Controls <= 9'b1100000100;//register-type
            6'b001000: Controls <= 9'b1010000000;//addi
            6'b001001: Controls <= 9'b1010000000;//addiu
            6'b001100: Controls <= 9'b1010000001;//andi
            6'b001101: Controls <= 9'b1010000111;//ori
            6'b100000: Controls <= 9'b1010010000;//lb
            6'b100001: Controls <= 9'b1010010000;//lbu
            6'b100001: Controls <= 9'b1010010000;//lh
            6'b100101: Controls <= 9'b1010010000;//lhu
```

```
            6'b100011: Controls <= 9'b1010010000;//lw
            6'b101011: Controls <= 9'b0010100000;//sw
            6'b000100: Controls <= 9'b0001000010;//beq
            6'b000101: Controls <= 9'b0001000010;//bne
            6'b001110: Controls <= 9'b1010000110;//xori
            6'b001111: Controls <= 9'b1010000011;//liu
            6'b001010: Controls <= 9'b1010000011;//slti
            6'b001011: Controls <= 9'b1010000011;//sltiu
            6'b000010: Controls <= 9'b0000001000;//j
            6'b000011: Controls <= 9'b0000001000;//jal
            6'b101000: Controls <= 9'b0110101000;//sb
            6'b101001: Controls <= 9'b0110101000;//sh
            default:   Controls <= 9'bxxxxxxxxx;//illegal Op
        endcase
endmodule
```

ext16.v

```
module ext16
    #(parameter DEPTH=16)
    (input [DEPTH-1:0] a,
    output reg [31:0] b);

    always@(a) begin
        if(a[DEPTH-1]==1)
        begin
           b[31:0]=32'hffffffff;
           b[DEPTH-1:0]=a[DEPTH-1:0];
        end
        else
        begin
            b[31:0]=32'h00000000;
            b[DEPTH-1:0]=a[DEPTH-1:0];
        end
    end
endmodule
```

IMem.v

```
`define DATA_WIDTH 32
module IMem(
    input [5:0] Addr,
    output [`DATA_WIDTH-1:0] RD);//指令

    parameter IMEM_SIZE = 64;  //一次从文件中可读取的指令总条数，与 Addr^2 相等
    reg [`DATA_WIDTH-1:0] RAM [IMEM_SIZE-1:0];
```

```
    initial begin

$readmemh("D:/Desktop/experiment/CPU/CPU.srcs/sources_1/new/mips2.txt",RAM);
    end
    assign RD = RAM[Addr];
endmodule
```

mux.v

```
module mux
    #(parameter WIDTH=32)
    (input [WIDTH-1:0] a,
    input [WIDTH-1:0] b,
    input select,
    output reg [WIDTH-1:0] r);

    always @(*)begin
        case(select)
            1'b0:r = a;
            1'b1:r = b;
        endcase
    end
endmodule
```

mux5.v

```
module mux5
    #(parameter WIDTH=5)
    (input [WIDTH-1:0] a,
    input [WIDTH-1:0] b,
    input select,
    output reg [WIDTH-1:0] r);

    always @(*)begin
        case(select)
            1'b0:r = a;
            1'b1:r = b;
        endcase
    end
endmodule
```

RAM.v

```
module RAM(WriteData, ReadData, Addr, Rst, R_W, CS, CLK);
    parameter Addr_Width = 10;
    parameter Data_Width = 32;
    parameter SIZE = 2 ** Addr_Width;
```

```
    input [Data_Width-1:0] WriteData;
    output reg [Data_Width-1:0] ReadData;
    input [Addr_Width-1:0] Addr;
    input Rst, R_W, CS, CLK;
    integer i;
    reg [Data_Width-1:0] Data_i;
    reg [Data_Width-1:0] RAM [SIZE-1:0];

    initial
    begin

$readmemh("D:/Desktop/experiment/CPU/CPU.srcs/sources_1/new/ram.txt",RAM);
    end

    //assign Data = (R_W) ? Data_i : 32'bz;
    always@(*)begin
        casex({CS, Rst, R_W})
            //3'bx1x: for(i = 0;i <= SIZE-1;i = i+1) RAM[i] = 0;
            3'b101: ReadData <= RAM[Addr];//¶Á
            3'b100: RAM[Addr] <= WriteData;//Ð´
            //default: Data_i = 32'bz;
        endcase
    end
endmodule

RegFile.v
`define DATA_WIDTH 32
module RegFile
    #(parameter ADDR_SIZE = 5)
    (input CLK, WE,
    input [ADDR_SIZE-1:0] RA1, RA2, WA,
    input [`DATA_WIDTH-1:0] WD,
    output [`DATA_WIDTH-1:0] RD1, RD2);

    integer i;
    reg [`DATA_WIDTH-1:0] rf [2 ** ADDR_SIZE-1:0];
    initial begin
        for(i = 0;i <= 2 ** ADDR_SIZE-1;i = i+1) rf[i] = 0;
        rf[28] = 32'h10008000;
        rf[29] = 32'h7fffeffc;
    end

    always@(posedge CLK)
        if(WE) rf[WA] <= WD;
```

```
    assign RD1 = (RA1 != 0) ? rf[RA1] : 0;
    assign RD2 = (RA2 != 0) ? rf[RA2] : 0;
endmodule
```

sim_CPU.v

```
module sim_CPU();
    reg CLK;
    reg Rst;
    wire [31:0]PC;
    wire [31:0]RD;
    wire [`DATA_WIDTH-1:0] RegWriteData;
    wire [`DATA_WIDTH-1:0] RegReadData1, RegReadData2;
    wire [9:0] RAM_Addr;
    wire [31:0] RAM_WriteData;
    wire [31:0] RAM_ReadData;
    wire RAM_CS;
    wire RAM_R_W;
    wire [31:0] ALUResult;
    wire [3:0] ALUControl;

    initial
    begin
        CLK = 0;#50
        forever #50 CLK = ~CLK;
        #2000 $stop(0);
    end
    initial
    begin
        Rst = 0;
        #50 Rst=1;
        #10 Rst=0;
    end
    CPU CPU_1(CLK, Rst, RD, PC, RegWriteData, RegReadData1, RegReadData2,
RAM_WriteData, RAM_ReadData, RAM_Addr, RAM_CS, RAM_R_W, ALUResult, ALUControl);
endmodule
```

mips1.txt

```
00000000
02309021
02509823
36300001
00138880
8e140003
ae140004
```

1254fff9
08100001
mips2.txt
00000000
02510019
0251001b
00138880
00138882
32300002
22300002
3a300002
1634fff8
08100001

ram.txt
00000000
01010101
02020202
03030303
04040404
05050505
06060606
07070707
08080808
09090909
00000000
01010101
02020202
03030303
04040404
05050505
06060606
07070707
08080808
09090909
00000000
01010101
02020202
03030303
04040404
05050505
06060606
07070707
08080808

09090909
00000000
01010101
02020202
03030303
04040404
05050505
06060606
07070707
08080808
09090909
00000000
01010101
02020202
03030303
04040404
05050505
06060606
07070707
08080808
09090909

五、调试和心得体会

在实际设计过程中，发现自己对书上这部分理论内容还是不太熟悉，最开始只是一头雾水照线路图连线，在实际的调整错误中才逐步熟练了 cpu 单周期指令的全过程，实验内容和课本内容相对照，对实际能力的提升很有帮助。