组成原理实验课程第 六 次实验报告

实验名称	单周期 CPU 实现			班级	李涛老师
学生姓名	辛浩然	学号	2112514	指导老师	董前琨
实验地点	实验楼 A 区 304			实验时间	2023/5/30

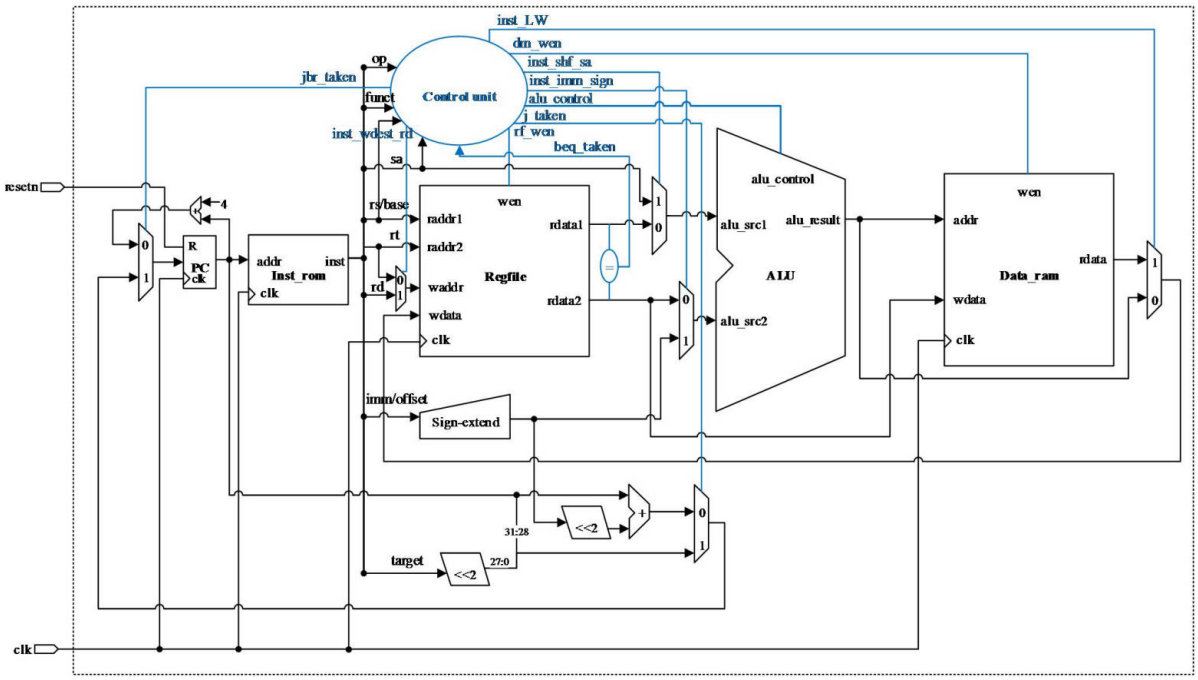
1 实验目的

1. 理解 MIPS 指令结构，理解 MIPS 指令集中常用指令的功能和编码，学会对这些指令进行归纳分类。
2. 了解熟悉 MIPS 体系的处理器结构，如延迟槽，哈佛结构的概念。
3. 熟悉并掌握单周期 CPU 的原理和设计。
4. 进一步加强运用 verilog 语言进行电路设计的能力。
5. 为后续设计多周期 cpu 的实验打下基础

2 实验内容说明

针对目前 CPU 可运行的 R 型和 I 型 MIPS 指令，各补充一条新的指令。

3 实验原理图



4 实验步骤

新增 R 型指令：算数右移；I 型指令：与立即数。以下是 single_cycle_cpu.v 文件中的修改：

```
// 译码阶段
// 加入指令列表
wire inst_SRA ;
wire inst_ANDI;
assign inst_SRA  = op_zero & (rs==5'd0) & (funct == 6'b000011);
// 算数右移
assign inst_ANDI = (op == 6'b001100);
// 与立即数

// 传递到执行模块的ALU源操作数和操作码
wire inst_sra ;
wire inst_andi;
assign inst_sra = inst_SRA;  // 算数右移
assign inst_andi= inst_ANDI; // 与立即数

wire [31:0] sext_imm;
wire   inst_shf_sa;    // 使用sa域作为偏移量的指令
wire   inst_imm_sign;  // 对立即数作符号扩展的指令
```

```
assign sext_imm      = {{16{imm[15]}}, imm};  // 立即数符号扩展
assign inst_shf_sa   = inst_SLL | inst_SRL | inst_SRA;
// sra是使用sa域作为偏移量的指令
assign inst_imm_sign = inst_ADDIU | inst_LUI | inst_LW | inst_SW |
   inst_ANDI;
// andi是对立即数作符号扩展的指令

wire [31:0] alu_operand1;
wire [31:0] alu_operand2;
wire [12:0] alu_control; // ALU操作码, 独热编码, 修改为13位
assign alu_operand1 = inst_shf_sa ? {27'd0,sa} : rs_value;
assign alu_operand2 = inst_imm_sign ? sext_imm : rt_value;
assign alu_control = {inst_add,inst_sub,inst_slt,inst_sltu,inst_and,
   inst_nor,inst_or, inst_xor,inst_sll,inst_srl,inst_lui,inst_sra,
   inst_andi};

// 写回阶段
wire inst_wdest_rt;   // 寄存器堆写入地址为rt的指令
wire inst_wdest_rd;   // 寄存器堆写入地址为rd的指令
assign inst_wdest_rt = inst_ADDIU | inst_LW | inst_LUI | inst_ANDI;
// andi指令写入地址为rt
assign inst_wdest_rd = inst_ADDU | inst_SUBU | inst_SLT | inst_AND |
   inst_NOR | inst_OR   | inst_XOR  | inst_SLL | inst_SRL | inst_SRA;
// sra指令写入地址为rd
```

对应修改 alu.v 文件，修改 alu 控制信号，新增运算。

```
module alu(
    input  [12:0] alu_control,  // ALU控制信号
    input  [31:0] alu_src1,     // ALU操作数1,为补码
    input  [31:0] alu_src2,     // ALU操作数2, 为补码
    output [31:0] alu_result    // ALU结果
    );

    // ALU控制信号, 独热码
    wire alu_sra;   //算术右移
    wire alu_andi;  //与立即数
```

```
    assign alu_sra  = alu_control[ 1];
    assign alu_andi = alu_control[ 0];

    wire [31:0] sra_result;
    wire [31:0] andi_result;

    // 与立即数
    assign andi_result= alu_src1 & alu_src2;

    // 算术右移
    wire [31:0] sra_step1;
    wire [31:0] sra_step2;
    assign sra_step1 = {32{shf_1_0 == 2'b00}} & alu_src2 | {32{shf_1_0
   == 2'b01}} & {alu_src2[31], alu_src2[31:1]} | {32{shf_1_0 == 2'b10}}
    & {{2{alu_src2[31]}}, alu_src2[31:2]} | {32{shf_1_0 == 2'b11}} &
   {{3{alu_src2[31]}}, alu_src2[31:3]};

    assign sra_step2 = {32{shf_3_2 == 2'b00}} & sra_step1  | {32{
   shf_3_2 == 2'b01}} & {{4{sra_step1[31]}}, sra_step1[31:4]} | {32{
   shf_3_2 == 2'b10}} & {{8{sra_step1[31]}}, sra_step1[31:8]} | {32{
   shf_3_2 == 2'b11}} & {{12{sra_step1[31]}},sra_step1[31:12]};

    assign sra_result = shf[4] ? {{16{sra_step2[31]}}, sra_step2
   [31:16]} : sra_step2;
```

修改 rom 文件，增加指令数量。写入新增运算 andi 和 sra 的新指令。

以下是新增指令的具体分析：

第 19 条指令：地址为 4CH，指令编码为 306D5555h，根据 op=001100，说明为 andi 指令。该指令将 $3 寄存器的值与立即数 5555h 求与后存至 $13 寄存器中。

```
assign inst_rom[19] = 32'h306D5555;
// 4CH: andi  $13,$3,5555 | $13 = 00000011H
```

第 20 条指令：地址为 50H，指令编码为 00038083h，根据 op 与 rs 全为 0 以及 funct=000011 可知，其为 sra 指令。该指令表示将 $3 寄存器的值算术右移 2 位后存至 $16 寄存器中。

```
assign inst_rom[20] = 32'h00038083;
// 50H: sra   $16,$3,2    | $16 = 00000004H
```

第 22 条指令：地址为 58H，指令编码为 00068903h，同样为 sra 指令。该指令表示将 $6 寄存器的值算术右移 4 位后存至 $17 寄存器中。

```
assign inst_rom[22] = 32'h00068903;
// 58H: sra   $17,$6,4    | $17 = FFFFFFFEH
```

以下是完整的所有指令:

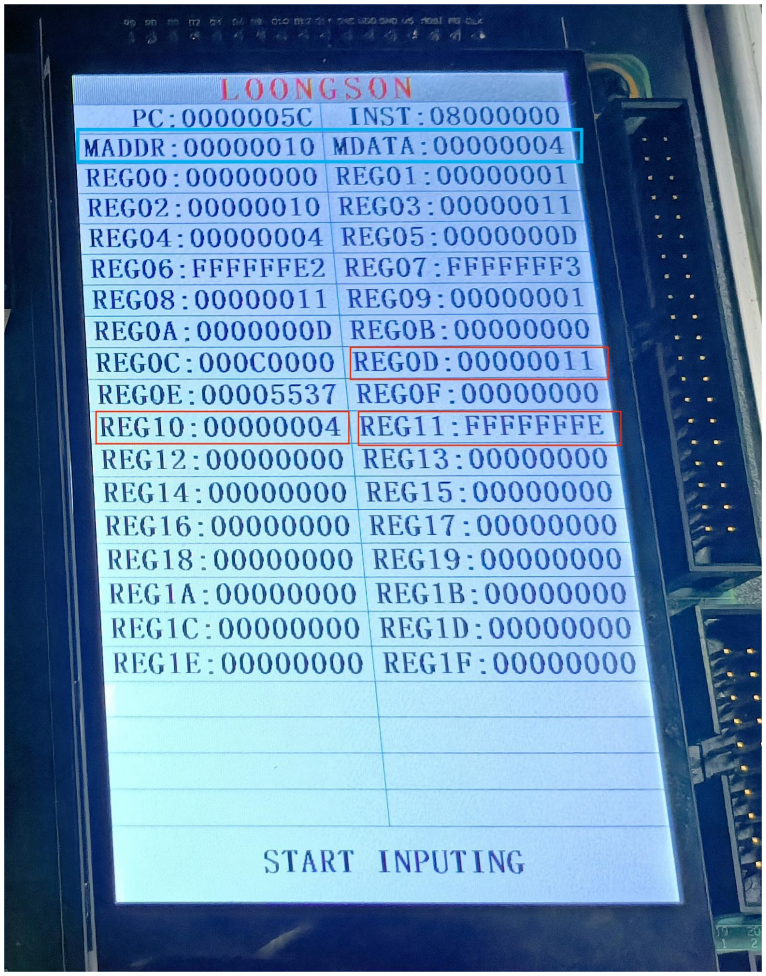
```
module inst_rom(
    input       [4 :0] addr, // 指令地址
    output reg [31:0] inst       // 指令
    );

    wire [31:0] inst_rom[23:0];  // 指令存储器, 字节地址
    assign inst_rom[ 0] = 32'h24010001;
    // 00H: addiu $1 ,$0,#1   | $1 = 0000_0001H
    assign inst_rom[ 1] = 32'h00011100;
    // 04H: sll   $2 ,$1,#4   | $2 = 0000_0010H
    assign inst_rom[ 2] = 32'h00411821;
    // 08H: addu  $3 ,$2,$1   | $3 = 0000_0011H
    assign inst_rom[ 3] = 32'h00022082;
    // 0CH: srl   $4 ,$2,#2   | $4 = 0000_0004H
    assign inst_rom[ 4] = 32'h00642823;
    // 10H: subu  $5 ,$3,$4   | $5 = 0000_000DH
    assign inst_rom[ 5] = 32'hAC250013;
    // 14H: sw    $5 ,#19($1) | Mem[0000_0014H] = 0000_000DH
    assign inst_rom[ 6] = 32'h00A23027;
    // 18H: nor   $6 ,$5,$2   | $6 = FFFF_FFE2H
    assign inst_rom[ 7] = 32'h00C33825;
    // 1CH: or    $7 ,$6,$3   | $7 = FFFF_FFF3H
    assign inst_rom[ 8] = 32'h00E64026;
    // 20H: xor   $8 ,$7,$6   | $8 = 0000_0011H
    assign inst_rom[ 9] = 32'hAC08001C;
    // 24H: sw    $8 ,#28($0) | Mem[0000_001CH] = 0000_0011H
    assign inst_rom[10] = 32'h00C7482A;
    // 28H: slt   $9 ,$6,$7   | $9 = 0000_0001H
    assign inst_rom[11] = 32'h11210002;
    // 2CH: beq   $9 ,$1,#2   | 跳转到指令34H
    assign inst_rom[12] = 32'h24010004;
    // 30H: addiu $1 ,$0,#4   | 不执行
    assign inst_rom[13] = 32'h8C2A0013;
```

```
// 34H: lw    $10,#19($1) | $10 = 0000_000DH
assign inst_rom[14] = 32'h15450003;
// 38H: bne   $10,$5,#3   | 不跳转
assign inst_rom[15] = 32'h00415824;
// 3CH: and   $11,$2,$1   | $11 = 0000_0000H
assign inst_rom[16] = 32'hAC0B001C;
// 40H: sw    $11,#28($0) | Men[0000_001CH] = 0000_0000H
assign inst_rom[17] = 32'hAC040010;
// 44H: sw    $4 ,#16($0) | Mem[0000_0010H] = 0000_0004H
assign inst_rom[18] = 32'h3C0C000C;
// 48H: lui   $12,#12     | [R12] = 000C_0000H
assign inst_rom[19] = 32'h306D5555;
// 4CH: andi  $13,$3,5555 | $13 = 00000011H
assign inst_rom[20] = 32'h00038083;
// 50H: sra   $16,$3,2    | $16 = 00000004H
assign inst_rom[21] = 32'h24CE5555;
// 54H: addi  $14,$6,5555 | $14 = 00005537H
assign inst_rom[22] = 32'h00068903;
// 58H: sra   $17,$6,4    | $17 = FFFFFFFEH
assign inst_rom[23] = 32'h08000000;
// 5CH: j     00H         | 跳转指令00H
```

5 上箱验证

上箱验证结果如图所示。



对于原有的指令，比较图中寄存器的值与前面 rom.v 代码块的预期值可以验证其正确性。任选一指令作为例子：

对于 17 号指令，sw $4 ,#16($0)，其将 4 号寄存器的值写入 Mem[00000010H]，输入 MADDR 为 10，可以成功查到存储器中对应地址存放的值为 00000004H（如上图蓝框）。

对于新增指令的验证：

如前所述，新增第 19 条 andi 指令：将 $3 寄存器的值与立即数 5555h 求与后存至 $13 寄存器中。理论上 $13 寄存器的值应为 00000011H，对比上箱结果，确实如此。

新增第 20 条 sra 指令：表示将 $3 寄存器的值算术右移 2 位后存至 $16 寄存器中。理论上 $16 的值为 00000004H，对比上箱结果，确实如此。

新增第 22 条 sra 指令：表示将 $6 寄存器的值算术右移 4 位后存至 $17 寄存器中。理论上 $17 的值为 FFFFFFFEH，对比上箱结果，确实如此。

结合以上结果，可以验证正确性。

6 总结感想

通过本次实验：

1. 理解了 MIPS 指令结构，理解了 MIPS 指令集中常用指令的功能和编码，学会了对这些指令

进行归纳分类。

2. 了解熟悉了 MIPS 体系的处理器结构。

3. 熟悉并掌握了单周期 CPU 的原理和设计。

4. 进一步加强了运用 verilog 语言进行电路设计的能力。