实验 7 –CPU 设计-指令集扩展实验报告

姓名：林逸竹　学号：3160104229　专业：计算机科学与技术

课程名称：计算机组成与设计实验　同组学生姓名：无

实验时间：2018-4-23　实验地点：紫金港东 4-509　指导老师：　施青松，黎金洪

一、实验目的和要求

1. 运用寄存器传输控制技术
2. 掌握 CPU 的核心：指令执行过程与控制流关系
3. 设计数据通路和控制器
4. 设计测试程序

二、实验内容和原理

2.1 实验任务

1. 扩展实验六 CPU 指令集

- **重新设计数据通路和控制器**
- **兼容 Exp05 的数据通路和控制器**
- **替换 Exp05 的数据通路控制器核**
- **扩展不少于下列指令**

R-Type：add, sub, and, or, xor, nor, slt, srl*, jr, jalr, eret；

I-Type：addi, andi, ori, xori, lui, lw, sw, beq, bne, slti

J-Type：J, Jal*；

- **此实验在 Exp06 的基础上完成**

2. 设计指令集测试方案

3. 设计指令集测试程序序

三、主要仪器设备

3.1 实验设备

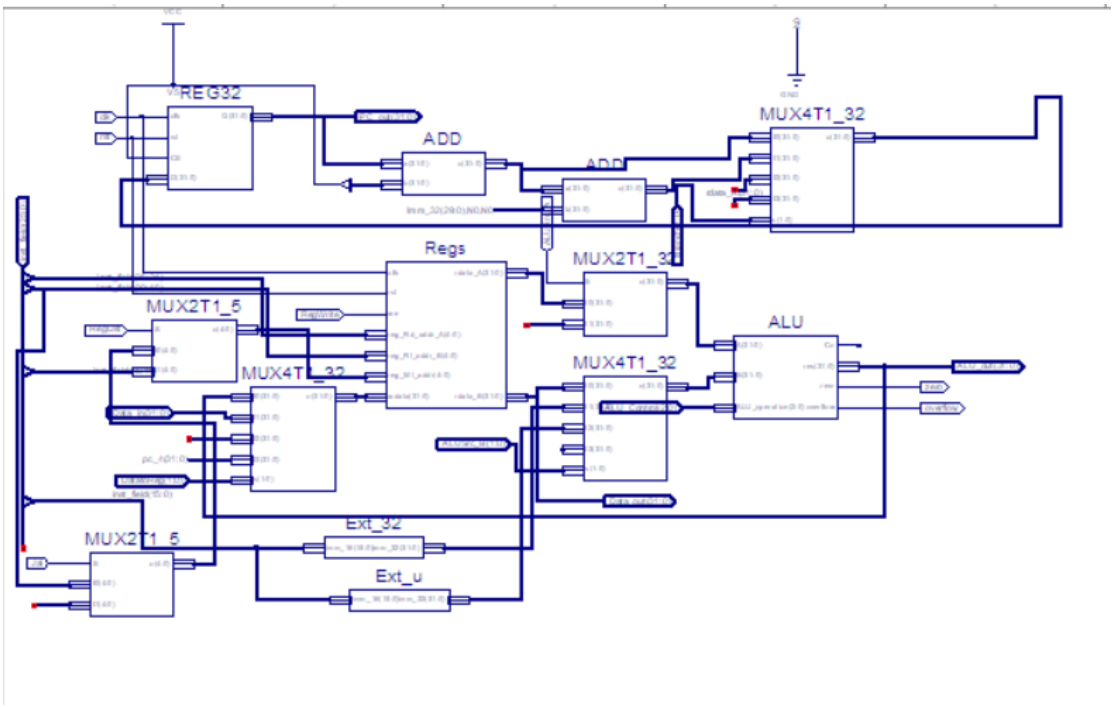
1. 计算机（Intel Core i5 以上，4GB 内存以上）系统
2. 计算机软硬件课程贯通教学实验系统
3. Xilinx ISE14.4 及以上开发工具

3.2 材料

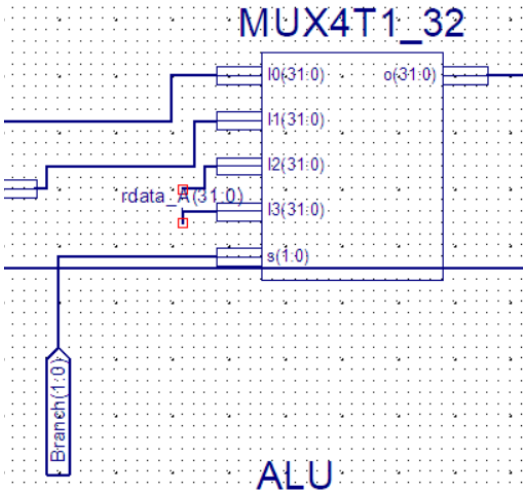
无

四、实验实现方法、步骤与调试

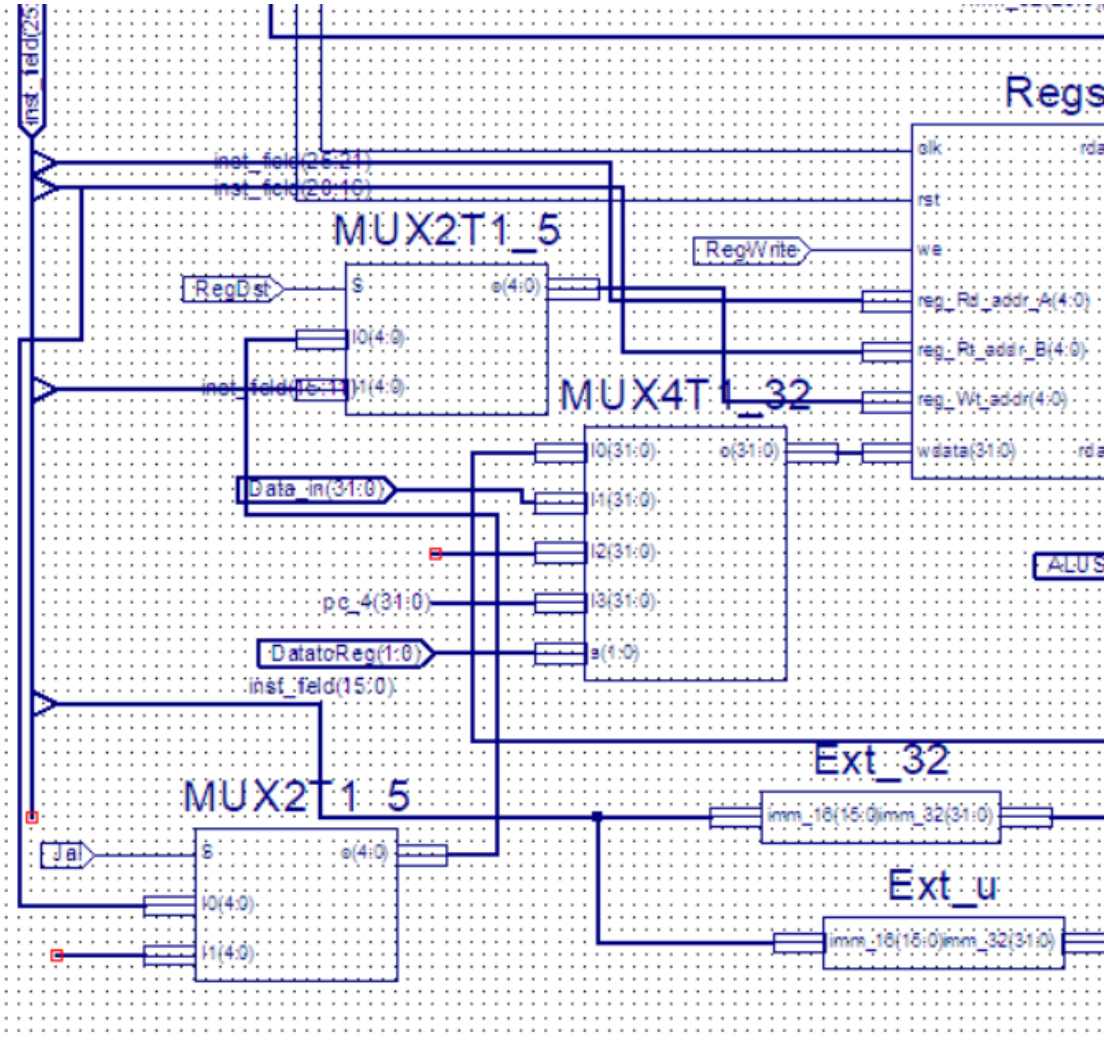
4.1 设计扩展 DataPath 结构



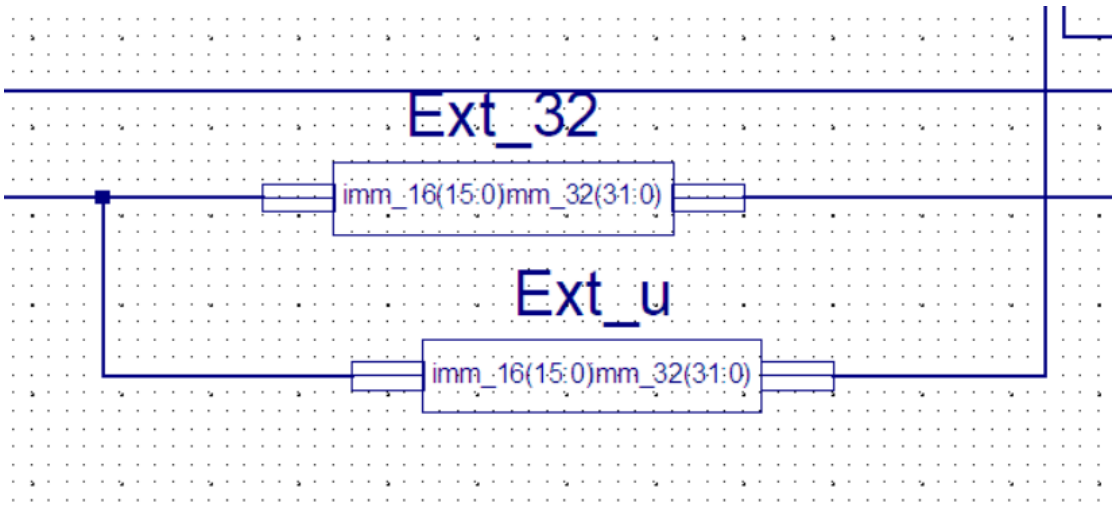
其中增加了以下部分，首先是 PC 跳转处增加了一个 4 选 1 模块。



控制信号为扩展为 2bits 的 Branch 信号，I0 为 PC+4, I1 为 PC+4+offset，I2 为 j 指令跳转的地址，I3 为地址 A 中的数据，用于 jr 跳转。



在 Regs 前也增设了一些多路选择器，用于 jal、jalr 跳转的 2 选 1 选择器，$ra 默认地址为 11111，同时需要把 PC+4 写入。



增设了无符号数的扩展，用于立即数指令中的逻辑操作，如 andi，ori 等等。其控制信号是由扩展成 2bit 的 ALU_Src_B 来控制的。

为了实现 srl 指令，传入的操作数 A 设置为 shamt。增加了一个 2 选一选择器，并增加信号 ALUSrc_A。

4.2 根据新的 DataPath 结构设计控制器

代码如下：

```
`timescale 1ns / 1ps
`define CPU_ctrl_signals {RegDst, ALUSrc1, ALUSrc0, DatatoReg1,
DatatoReg0, RegWrite, MemRead, MemWrite, Branch1, Branch0, ALUop2,
ALUop1, ALUop0, Jal}

module SCPU_ctrl( input[5:0]OPcode,                    //OPcode
                  input[5:0]Fun,                       //Function

                   input MIO_ready,                    //CPU Wait
                   input zero,
                   output reg Jal,
                   output reg RegDst,
                   output [1:0]ALUSrc_B,
                   output [1:0]DatatoReg,
                   output [1:0]Branch,
                   output reg RegWrite,
                   output mem_w,
                   output reg [2:0]ALU_Control,
                   output reg ALUSrc_A,
                   output CPU_MIO
                 );
reg MemRead, MemWrite;
reg ALUop2, ALUop1, ALUop0;
reg Branch1, Branch0;
reg DatatoReg1, DatatoReg0;
reg ALUSrc1, ALUSrc0;
assign ALUSrc_B = {ALUSrc1, ALUSrc0};
assign DatatoReg = {DatatoReg1, DatatoReg0};
assign Branch = {Branch1, Branch0};
assign mem_w = MemWrite && (~MemRead);
always@*begin
    case(OPcode)
    6'b000000:begin
       if(Fun==6'h8)        `CPU_ctrl_signals = 14'b10000100110100;
```

```
// jr
       else  if(Fun==6'h9)  `CPU_ctrl_signals  =  14'b10011100110101;
// jalr
       else                 `CPU_ctrl_signals  =  14'b10000100000100;
     // ALU
    end

    6'b100011:begin  `CPU_ctrl_signals  =  14'b00101110000000;end  //
load
    6'b101011:begin  `CPU_ctrl_signals  =  14'b00100001000000;end  //
store
    6'b000100:begin
       if(zero==1'b1) `CPU_ctrl_signals = 14'b00000000010010;   //
beq
       else              `CPU_ctrl_signals = 14'b00000000000010;
    end
    6'b000101:begin
       if(zero==1'b0) `CPU_ctrl_signals = 14'b00000000010010;   //
bne
       else              `CPU_ctrl_signals = 14'b00000000000010;
    end
    6'b000010:begin  `CPU_ctrl_signals  =  14'b00000000100100;end  //
jump
    6'h000011:begin `CPU_ctrl_signals = 14'b00000000100101;end // jal
    6'h24:begin       `CPU_ctrl_signals  =  14'b00100100000110;end  //
slti
    6'hf: begin       `CPU_ctrl_signals = 14'b00010100000000;end //
lui
    6'h8: begin       `CPU_ctrl_signals = 14'b00100100000000;end //
addi
    6'hc: begin       `CPU_ctrl_signals = 14'b01000100001000;end //
andi
    6'hd: begin       `CPU_ctrl_signals = 14'b01000100001010;end //
ori
    6'he: begin       `CPU_ctrl_signals = 14'b01000100001100;end //
xori
    default:begin   `CPU_ctrl_signals = 14'b00000000000000;end
    endcase
end

always@*begin
    case( {ALUop2, ALUop1, ALUop0} )
    3'b000: begin ALU_Control = 3'b010;    ALUSrc_A = 1'b0; end
    // add
    3'b001: begin ALU_Control = 3'b110;    ALUSrc_A = 1'b0; end
    // sub
    3'b010:
       case(Fun)
       6'b001000: begin ALU_Control = 3'b010; ALUSrc_A  =  1'b0;  end
    // jr
       6'b100000: begin ALU_Control = 3'b010; ALUSrc_A  =  1'b0;  end
// add
       6'b100010: begin ALU_Control = 3'b110; ALUSrc_A  =  1'b0;  end
    // sub
       6'b100100: begin ALU_Control = 3'b000; ALUSrc_A  =  1'b0;  end
    // and
       6'b100101: begin ALU_Control = 3'b001; ALUSrc_A  =  1'b0;  end
    // or
       6'b101010: begin ALU_Control = 3'b111; ALUSrc_A = 1'b1; end//
slt
       6'b100111: begin ALU_Control = 3'b100; ALUSrc_A  =  1'b0;  end
```

```
    // nor
        6'b000010: begin ALU_Control = 3'b101; ALUSrc_A  =  1'b0;  end
    // srl
        6'b010110: begin ALU_Control = 3'b011; ALUSrc_A  =  1'b0;  end
    // xor
        default:   begin ALU_Control = 3'bx;ALUSrc_A = 1'b0; end
        endcase
    3'b011: begin ALU_Control = 3'b111;    ALUSrc_A  =  1'b1;  end//
slti
    3'b100: begin ALU_Control = 3'b000;    ALUSrc_A = 1'b0; end
    // andi
    3'b101: begin ALU_Control = 3'b001;    ALUSrc_A = 1'b0; end
    // ori
    3'b110: begin ALU_Control = 3'b011;    ALUSrc_A = 1'b0; end
    // xori
    endcase
end


endmodule
```

由于是在实验 6 的基础上进行修改，且为了满足附录中对操作数的一些设定，对 ALUop 进行了扩充为了满足立即数的操作，同时被扩充的还有 DatatoReg, Branch 和 ALU_Src_B 等指令。

五、实验结果与分析

能够实现下表功能。

图形功能测试

开关	位置	功能
SW[1:0]	X0	七段码图形显示
SW[2]	0	CPU全速时钟
SW[4:3]	00	7段码从上至下亮点循环右移
SW[4:3]	11	7段码矩形从下到大循环显示
SW[7:5]	000	作为外设使用（E0000000/FFFFFE00）

文本功能测试

开关	位置	功能
SW[1:0]	01	七段码文本显示（低16位）
	11	七段码文本显示（高16位） Arduino有效
SW[2]	0	CPU全速时钟
SW[4:3]	01	7段码显示RAM数字
SW[4:3]	10	7段码显示累加
SW[7:5]	000	作为外设使用（E0000000/FFFFFE00）

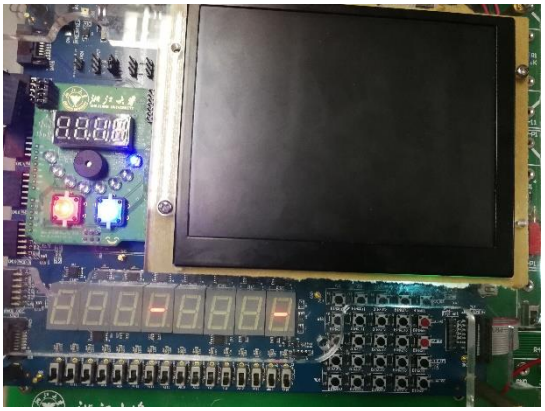


Figure 1 实验结果(1) 跑马灯

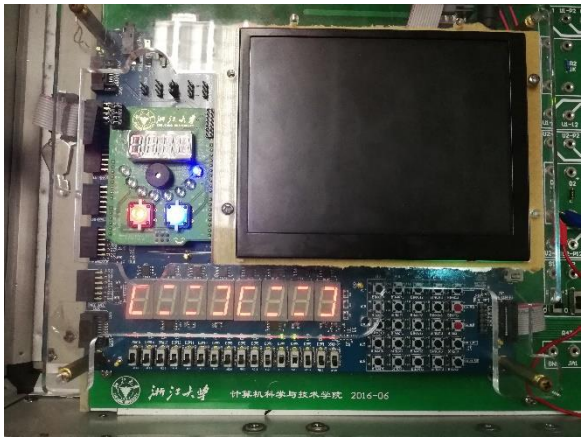


Figure 2 实验结果(2) 矩形

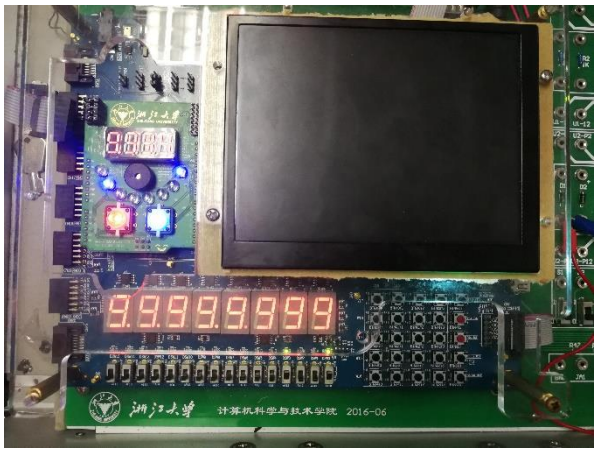


Figure 3 实验结果(3) RAM

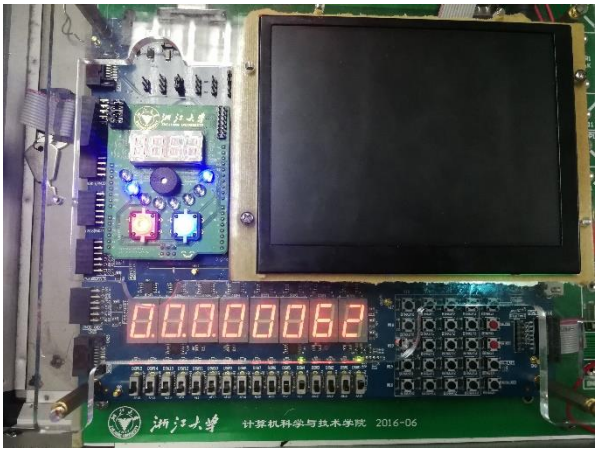


Figure 4 实验结果(4) 累加

六、讨论、心得

6.1 思考题

6.1.1 指令扩展时控制器用二级比译码设计存在什么问题？

答: 在本实验中我就使用了二级译码，这样使得必须扩展 ALUop 信号，且其中有许多的重复操作，降低了代码的可读性，增加了代码的复杂性，当然同样能够达到预期的结果。

6.1.2 设计 bne 指令需要增加控制信号吗？

答：如果根据本设计是不需要增加的，但如果使用增加控制信号同样能够达到预期的效果。

6.1.3 设计 andi 时需要增加新的数据通道吗?

答：需要，因为 andi 进行的是无符号数扩展。

6.2 心得

这次实验是从数逻课以来最多自己思考内容的一次，增加了很多都是靠自己去思考而得出的结果，虽然做完后跟其他人的想法一比对发现还是可能会存在过度复杂等问题，但因为都是自己想得反而更加熟悉。但这次实验存在比较大的不足是仿真做得少了，因为时间关系下板验证的时间也少，跟其他同学学习了如何自己编码进行物理验证却没有很好的运用。在提交完实验报告后还会抽时间去进行这一步工作。