# 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

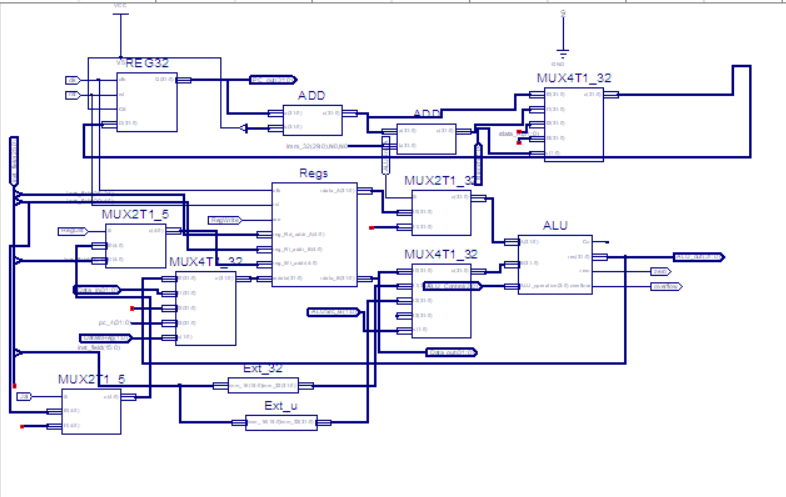
3. Xilinx ISE14.4及以上开发工具

3.2 材料

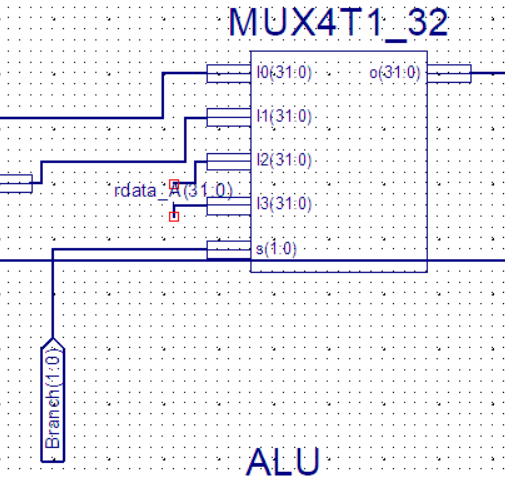
无

# 四、实验实现方法、步骤与调试

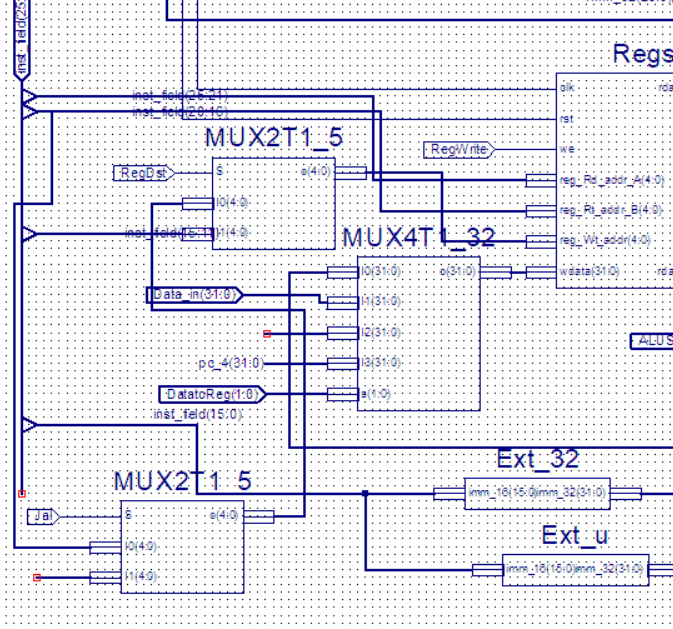
4.1 设计扩展DataPath结构



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。

![C:\Users\asus\AppData\Roaming\Tencent\Users\897735626\TIM\WinTemp\RichOle\])WI)YLVN)](KGPAQ`M[659.png](data:image/png;base64,)

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

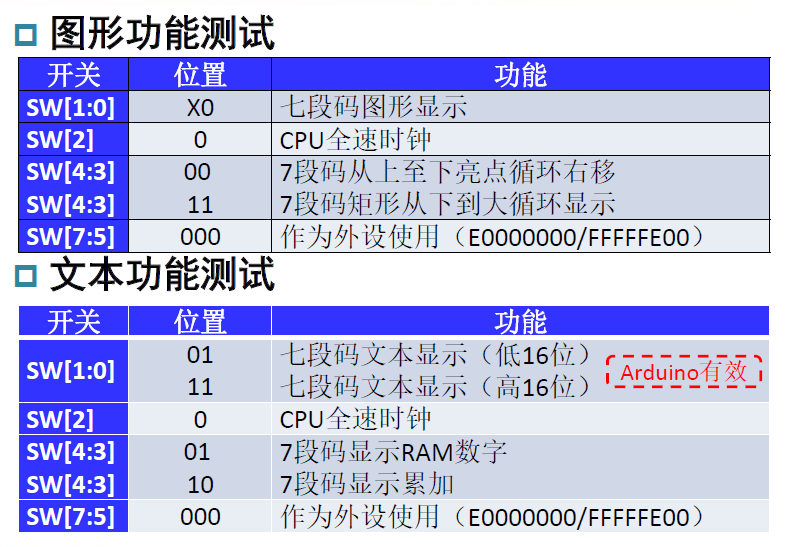
代码如下：

|  |
| --- |
| `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。



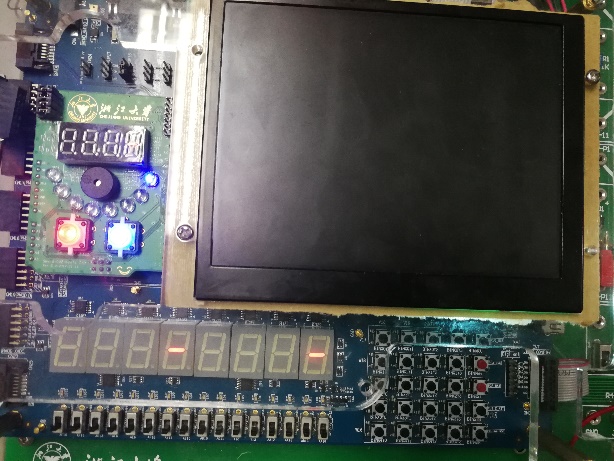


Figure 1 实验结果(1) 跑马灯

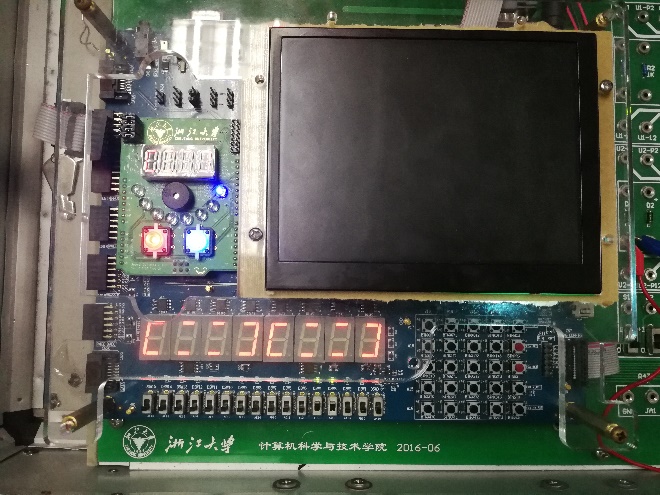


Figure 2 实验结果(2) 矩形

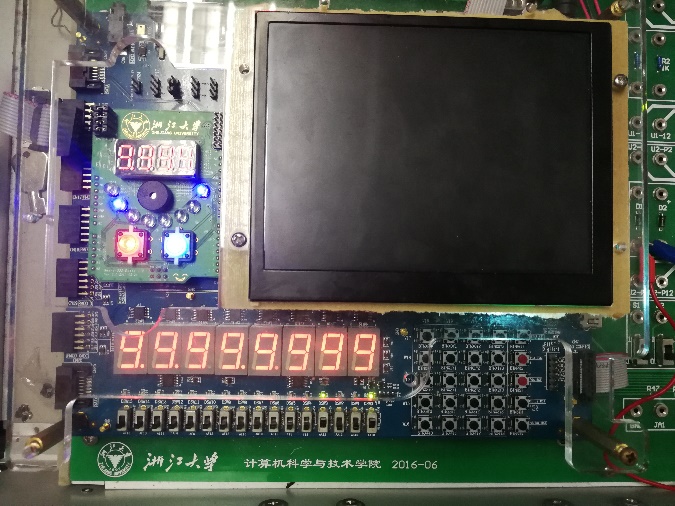


Figure 3 实验结果(3) RAM

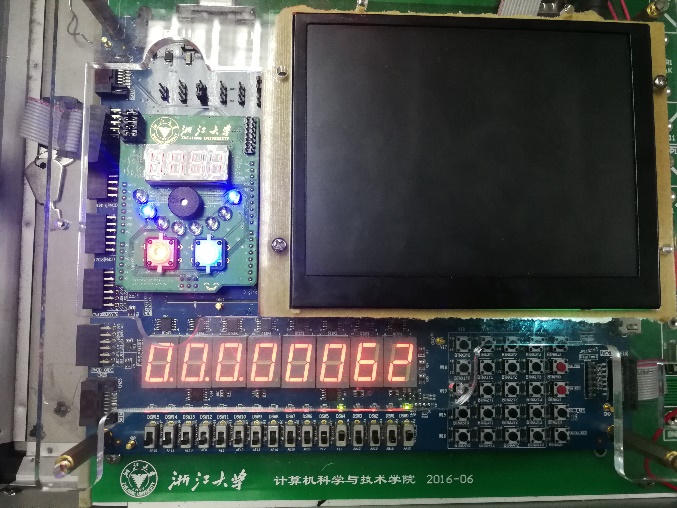


Figure 4 实验结果(4) 累加

# 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。