

实验四--集成替换 CPU 核实验报告

姓名: 蒋仕彪 学号: 3170102587 专业: 求是科学班(计算机)1701

课程名称: 计算机组装实验 同组学生姓名:

实验时间: 2019-3-18 实验地点: 紫金港东 4-509 指导老师: 马德

个人形象照:



一、实验目的和要求

1. 寄存器传输控制技术
2. 掌握 CPU 的核心组成: 数据通路与控制器
3. 设计数据通路的功能部件
4. 进一步了解计算机系统的基本结构
5. 熟练掌握 IP 核的使用方法

二、实验任务和原理

2.1 实验任务

1. 用 IP 核集成 CPU 并替换实验三的 CPU 核

选用教材提供的 IP 核集成实现 CPU

此实验在 Exp03 的基础上完成

2. 设计数据通路部件并作时序仿真:

ALU

Register Files

3 熟练掌握 IP 核的使用方法

2.2 实验原理

2.2.1 Data path

□ Data_path

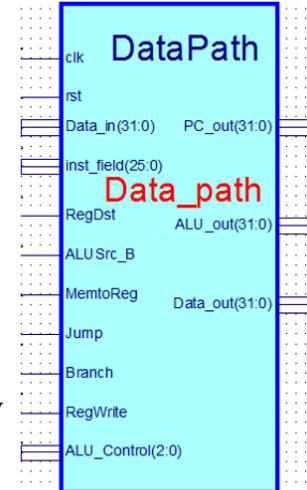
- CPU主要部件之一
- 寄存器传输控制对象：通用数据通路

□ 基本功能

- 具有通用计算功能的算术逻辑部件
- 具有通用目的寄存器
- 具有通用计数所需的尽可能的路径

□ 本实验用IP 软核- Data_path

- 核调用模块Data_path.ngc
- 核接口信号模块(空文档): Data_path.v
- 核模块符号文档: Data_path.sym



```
module Data_path( input clk, //寄存器时钟
                  input rst, //寄存器复位
                  input[25:0]inst_field, //指令数据域
                  input RegDst,
                  input ALUSrc_B,
                  input MemtoReg,
                  input Jump,
                  input Branch,
                  input RegWrite,
                  input[31:0]Data_in,
                  input[2:0]ALU_Control,
                  output[31:0]ALU_out,
                  output[31:0]Data_out,
                  output[31:0]PC_out
                );
endmodule
```

2.2.2 SCPU_ctrl

□ SCPU_ctrl

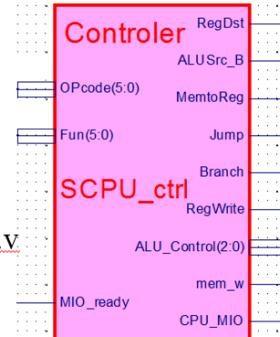
- CPU主要部件之一
- 寄存器传输控制技术中的运算和通路控制器：

□ 基本功能

- 指令译码
- 产生操作控制信号：ALU运算控制
- 产生指令所需的路径选择

□ 本实验用IP 软核- SCPU_ctrl

- 核调用模块SCPU_ctrl.ngc
- 核接口信号模块(空文档): SCPU_ctrl.v
- 核模块符号文档: SCPU_ctrl.sym



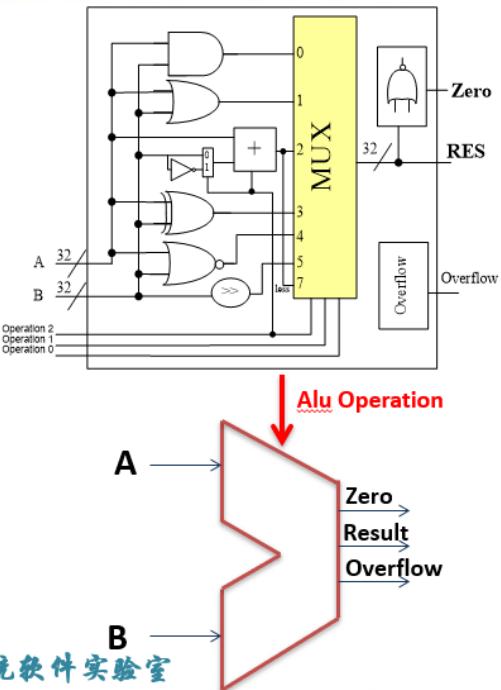
```
module      SCPU_ctrl( input[5:0]Opcode,      //OPcode
                      input[5:0]Fun,          //Function
                      input MIO_ready,        //CPU Wait

                      output reg RegDst,
                      output reg ALUSrc_B,
                      output reg MemtoReg,
                      output reg Jump,
                      output reg Branch,
                      output reg RegWrite,
                      output reg mem_w,
                      output reg [2:0]ALU_Control,
                      output reg CPU_MIO
                    );
endmodule
```

2.2.3 数据通路功能部件：SCPU

- 实现5个基本运算
 - 整理逻辑实验八的ALU
 - 逻辑图输入并仿真

ALU Control Lines	Function	note
000	And	兼容
001	Or	兼容
010	Add	兼容
110	Sub	兼容
111	Set on less than	
100	nor	扩展
101	srl	扩展
011	xor	扩展

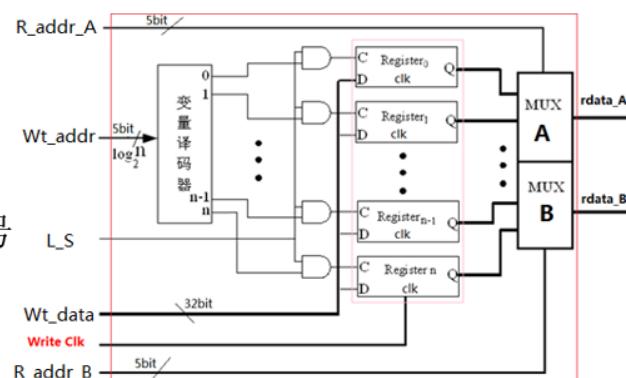


浙江大学 计算机学院 系统结构与系统软件实验室

2.2.4 数据系统功能部件：Register files

- 实现 $32 \times 32\text{bit}$ 寄存器组
 - 优化逻辑实验Regs
 - 行为描述并仿真结果

- 端口要求
 - 二个读端口：
 - R_addr_A
 - R_addr_B
 - 一个写端口，带写信号
 - Wt_addr
 - L_S



三、主要仪器设备

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

四、操作方法与实验步骤

4.1 分解 CPU

设计工程：OExp04-IP2CPU

◎ 分解CPU为二个IP核

- └ 在Exp03工基础上用二个IP核构建CPU
- └ 顶层模块延用Exp03
 - 模块名: Top_OExp04_IP2CPU.sch

◎ 逻辑实验输出模块优化

- └ ALU模块优化
- └ Register Files模块优化
- └ 优化目标: 满足MIPS处理器的要求

4.2 移植实验三至实验四

清理Exp03工程

□ 移除工程中的CPU核

- Exp03工程中移除CPU核关联

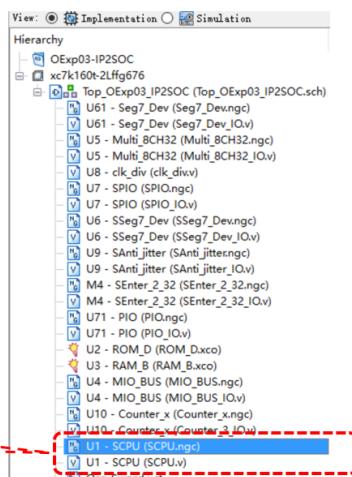
□ 删除工程中CPU核文件

- SCPU.ngc 和 SCPU.v 文件
- 在Project菜单中运行:
Cleanup Project Files ...

□ 建议用Exp03资源重建工程

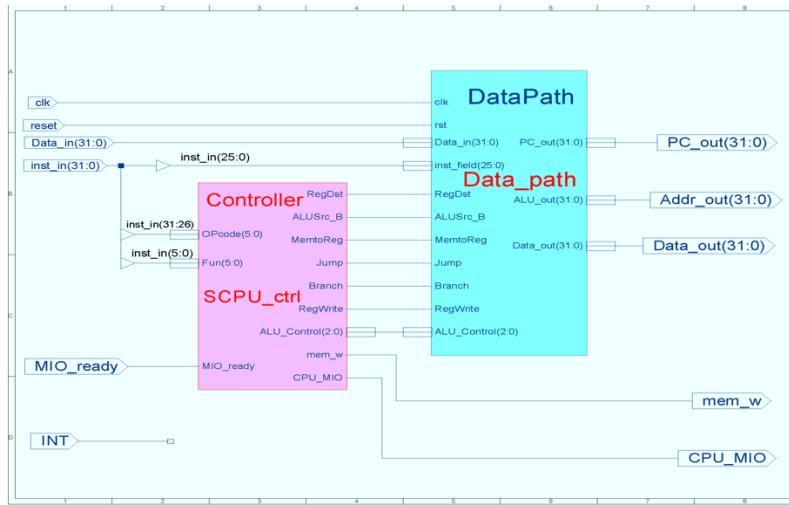
- 除CPU核
- 命名: OExp04-IP2CPU

Exp03需要清理的核

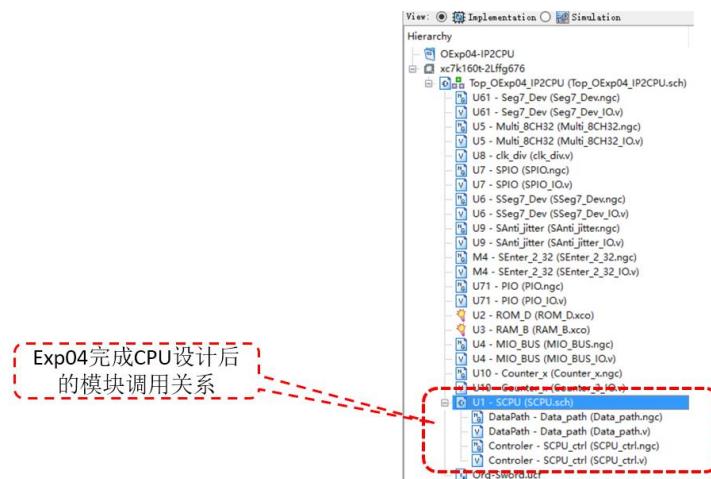


4.3 用原理图设计 CPU

用逻辑原理图输入CPU设计



4.4 最终调用模块结构

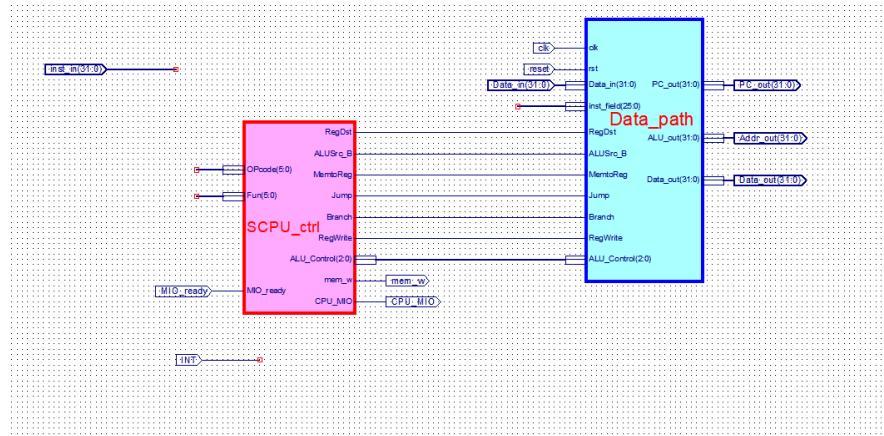


五、实验结果与分析

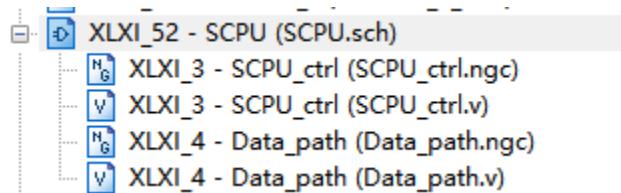
5.1 SCPU 顶层模块展示

本实验的核心是：重构 CPU，将原来的 ngc 用逻辑图再连一遍。

下面展示一下 SCPU 顶层模块的 sch 展示。



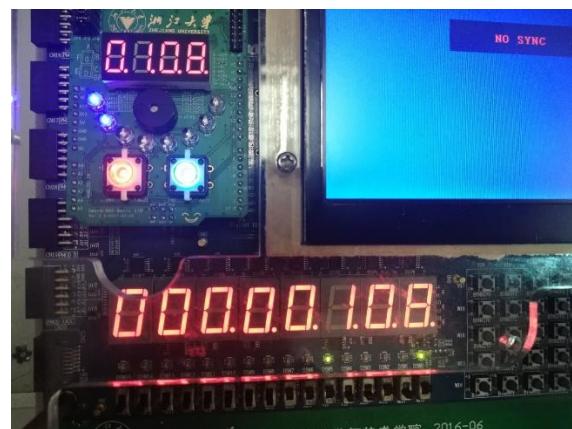
虽然顶层逻辑已经手写过，两个关键部件 SCPU_ctrl 和 Data_path 依然是调用 ngc 的。



5.2 实验现象一

SW[0]=1, SW[2]=1, SW[7:5]=111。

输出 CPU 指令字节地址 PC_out。

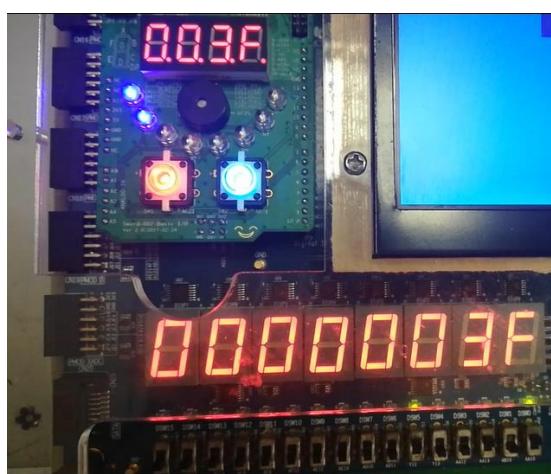
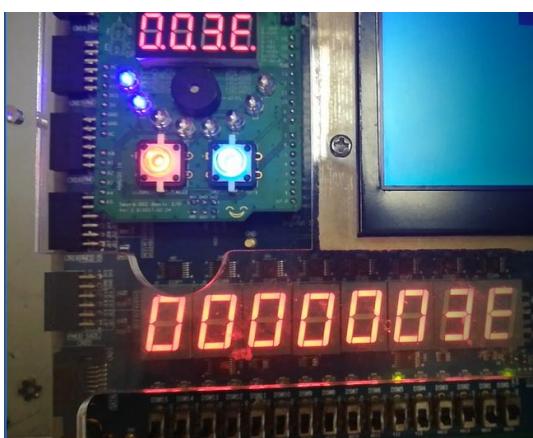
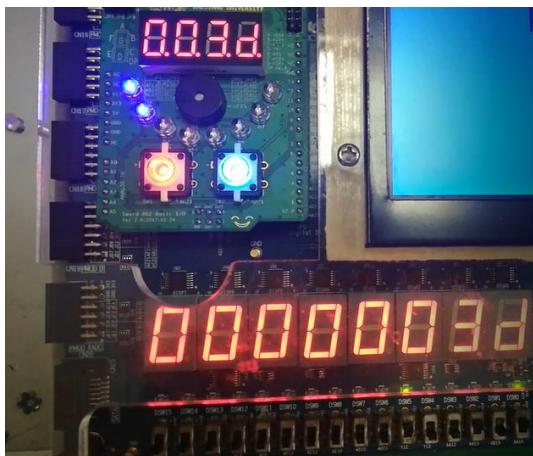


□ 实验现象 1

5.3 实验现象二

$\text{SW}[0]=1$, $\text{SW}[2]=1$, $\text{SW}[7:5]=001$ 。

输出 CPU 指令字地址 PC_out[31:2]。

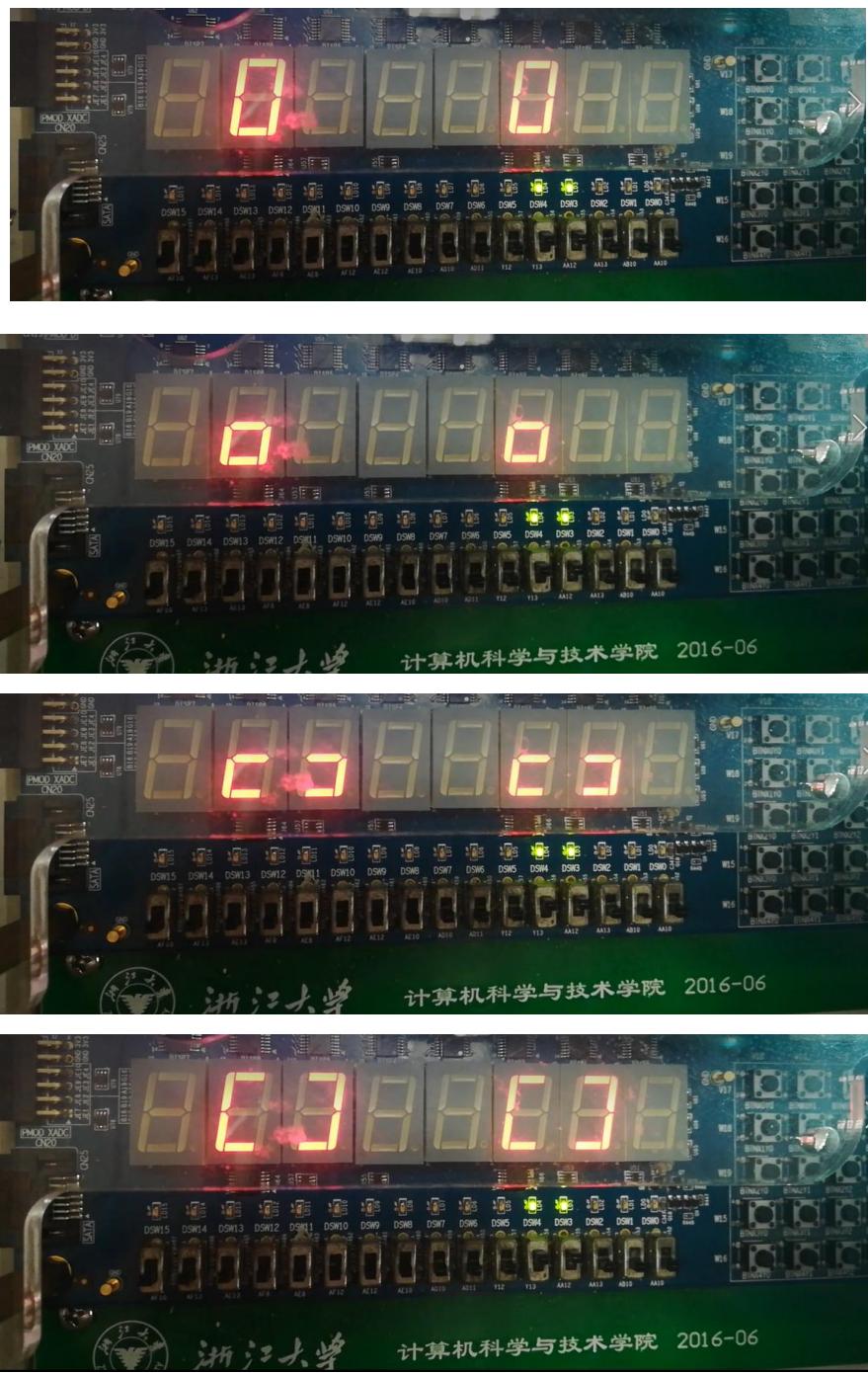


□ 实验现象 2

5.4 实验现象三

SW[0]=0, SW[3]=1, SW[4]=1。

矩形框跳舞。

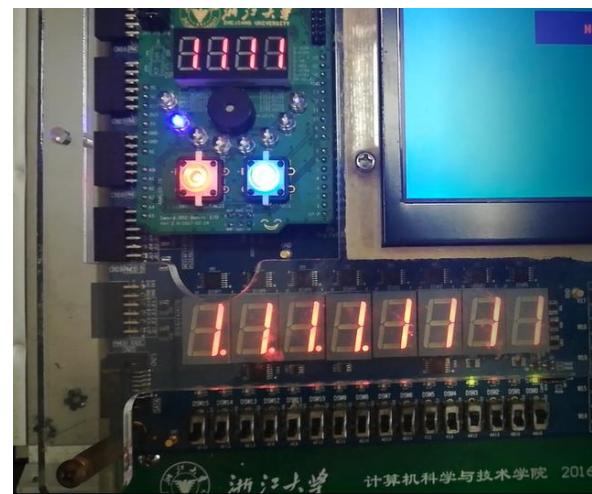
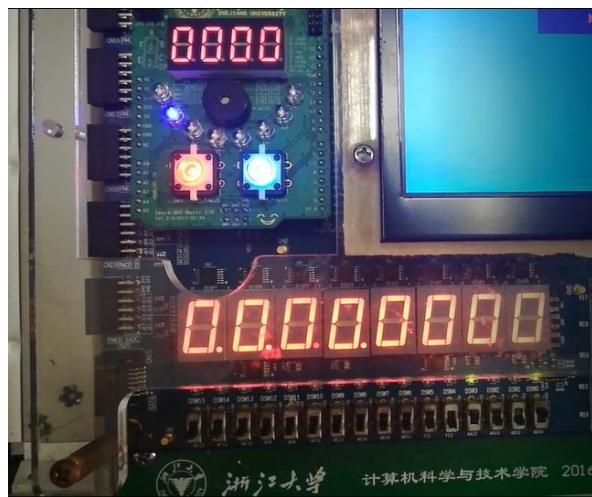


□ 实验现象 3

5.5 实验现象四

SW[0]=1, SW[3]=1。

全速时钟。



□ 实验现象 4

六、讨论、心得

这次的实验是用 SCPU_ctrl.ngc 和 Data_path.ngc 文件与 SCPU 原理图实现 SCPU 并替换之前的 SCPU 模块。

相比于以前的实验，实验 4 的操作难度不大，但是需要我们花大量的时间来思考 SCPU 的逻辑。而且此时，SCPU_ctrl 和 Data_path 都还是封装好的 ngc 文件，根本不清楚其内部实现。在这样的情况下，理解 SCPU 究竟在干什么就更抽象了。

我做实验的时候，只是朦朦胧胧地觉得，Data_path 连好了 SCPU 的主要逻辑图，而 SCPU_ctrl 相当于一个集成的控制器，用来控制控制 Data_path 里的各种二路选择。

随着计组理论课的不断学习，我对 SCPU 的理解终于越来越深。

实验五-- CPU 设计之数据通路

姓名: 蒋仕彪 学号: 3170102587 专业: 求是科学班(计算机) 1701

课程名称: 计算机组装实验 同组学生姓名:

实验时间: 2019-3-25 实验地点: 紫金港东 4-509 指导老师: 马德

一、实验目的和要求

1. 用寄存器传输控制技术
2. 握 CPU 的核心: 数据通路组成与原理
3. 设计数据通路
4. 学习测试方案的设计
5. 学习测试程序的设计

二、实验任务和原理

2.1 实验任务

1. 设计 9+ 条指令的数据通路

用逻辑原理图设计实现数据通路
ALU 和 Regs 调用 Exp04 设计的模块
替换 Exp04 的数据通路核
此实验在 Exp04 的基础上完成

2. 设计数据通路测试方案:

部件测试: ALU、Register Files
通路测试: I-格式通路、R-格式通路

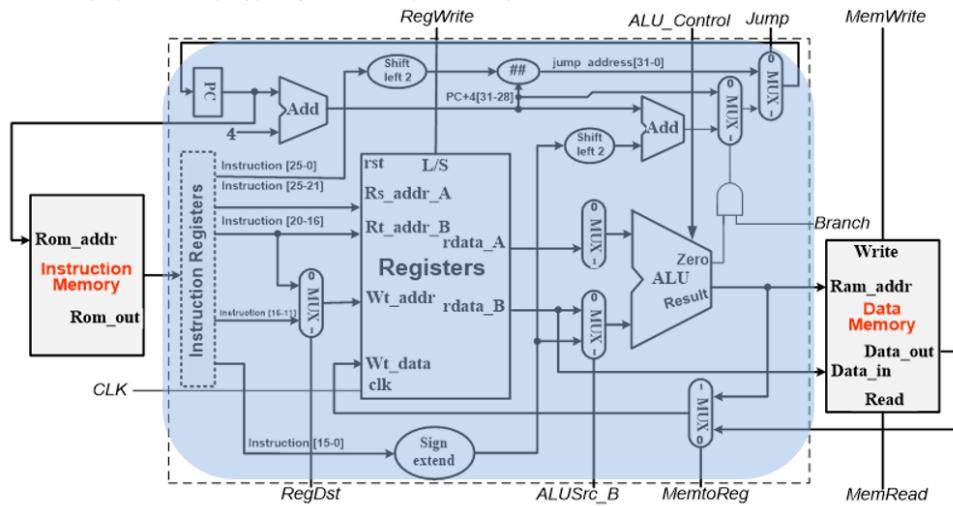
3. 设计数据通路测试

2.2 实验原理

2.2.1 单周期数据通路结构

单周期数据通路结构

- 找出九条指令的通路：5个MUX



2.2.2 控制信号定义

控制信号定义

- ## □ 通路与操作控制

信号	源数目	功能定义	赋值0时动作	赋值1时动作
<u>ALUSrc_B</u>	2	ALU端口B输入选择	选择寄存器B数据	选择32位立即数 (符号扩展后)
<u>RegDst</u>	2	寄存器写地址选择	选择指令rt域	选择指令rs域
<u>MemtoReg</u>	2	寄存器写入数据选择	选择存储器数据	选择ALU输出
<u>Branch</u>	2	<u>Beq</u> 指令目标地址选择	选择PC+4地址	选择转移地址 (Zero=1)
<u>Jump</u>	2	J指令目标地址选择	选择J目标地址	由 <u>Branch</u> 决定输出
<u>RegWrite</u>	-	寄存器写控制	禁止寄存器写	使能寄存器写
<u>MemWrite</u>	-	存储器写控制	禁止存储器写	使能存储器写
<u>MemRead</u>	-	存储器读控制	禁止存储器读	使能存储器读
<u>ALU_Control</u>	000- 111	3位ALU操作控制	参考表 Lab4	Lab4

三、主要仪器设备

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

四、操作方法与实验步骤

4.1 设计工程：OExp05-Datapath

设计工程：OExp05-Datapath

◎ 设计CPU之数据通路

- ↳ 根据理论课分析讨论设计9+条指令的数据通路
- ↳ 仿真测试DataPath模块

◎ 集成替换验证通过的数据通路模块

- ↳ 替换实验四(Exp04)中的Data_Path.ngc核
- ↳ 顶层模块延用Exp04
 - ◎ 模块名：Top_OExp05_DataPath.sch

◎ 测试数据通路模块

- ↳ 设计测试程序(MIPS汇编)测试：
- ↳ ALU功能
- ↳ I-指令通路
- ↳ R-指令通路

4.2 设计 32 位寄存器

设计要点

□ 设计32位寄存器

- 用途：PC指针、数据、地址或指令锁存
- 参考逻辑实验Exp10，用行为描述实验
- 模块名：REG32
 - 上升沿触发：clk
 - 使能信号：CE
 - 同步复位：rst=1
 - 数据输入：D(31:0)
 - 数据输出：Q(31:0)

■ 参考描述结构

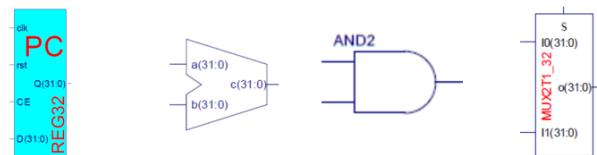
```
module REG32(input clk,  
.....  
always @(posedge clk or posedge rst)  
  if (rst==? ) Q <= ? ;  
  else if (?) Q <= ? ;  
endmodule
```



封装后的逻辑符号

4.3 设计 PC 通路

- 建立DataPath原理图输入模板
- 设计PC通路
 - 调用REG32、add_32、AND2(库)和MUX2T1_32模块

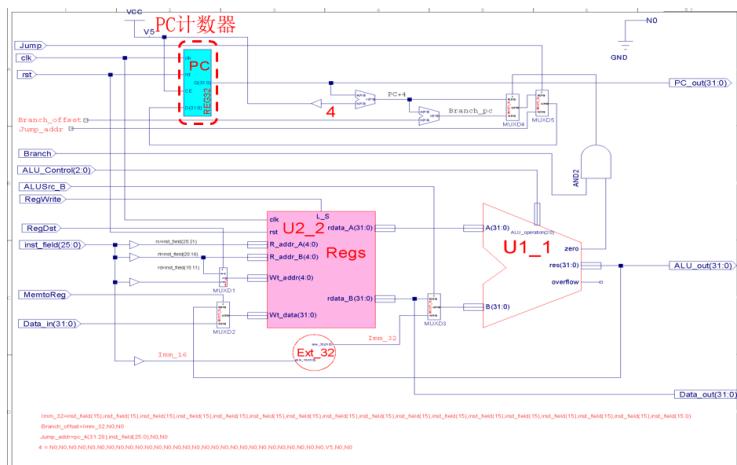


- 设计：
 - 顺序执行(PC+4)、Jump和Beq时的PC值
 - 计算和通路
- 注意常数的电路实现：
 - “4” = ? ? ? ?
 - Branch_offset = ? ? ? ?
 - Jump_addr = ? ? ? ?

4.4 设计 32 位寄存器

数据通路参考逻辑结构图

不唯一



4.5 替换 Datapath 的 ngc

DataPath替换集成

□ 集成替换

- 仿真正确后替换Exp04的数据通路IP核

□ 清理Exp04工程

- 移除工程中的数据通路核

- Exp04工程中移除数据通路核关联

- 删除工程中CPU核文件

- Data_path.ngc和 Data_path.v 文件

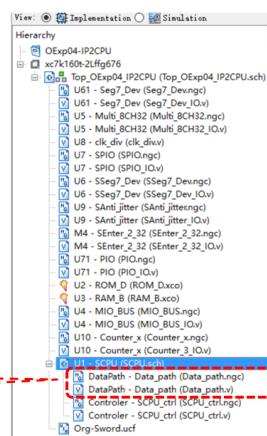
- 在Project菜单中运行:

- Cleanup Project Files ...

- 建议用Exp04资源重建工程

- 除Data_path.ngc核

Exp04需要清理的核

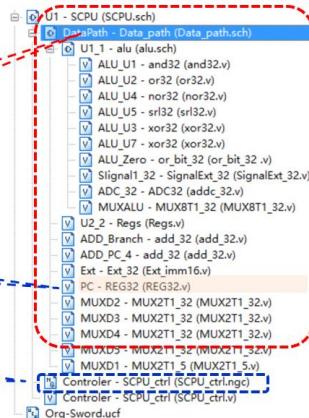


□ 集成替换DapaPath核后的模块层次结构

Exp05完成数据通路替换
后的模块调用关系

PC计数器
本实验设计

控制器不变
仍然使用IP核



五、实验结果与分析

5.1 REG32.v

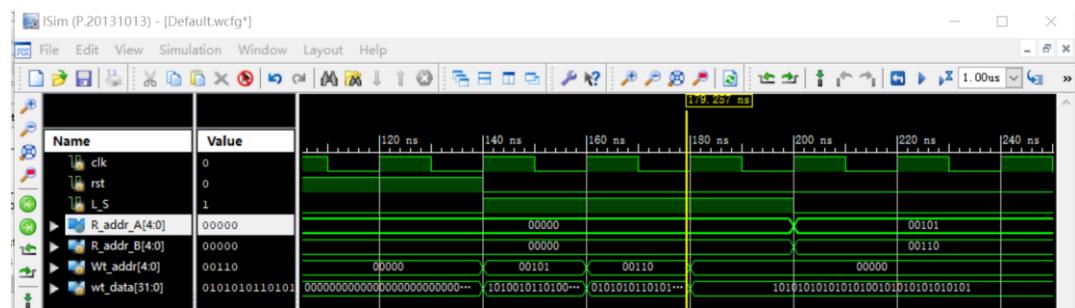
5.1.1 源码展示

```
module REG32(
    input clk, input CE, input rst, input [31:0] D,
    output reg [31:0] Q
);
    always @ (posedge clk or posedge rst)
        if(rst==1) Q<=32'b0000_0000_0000_0000_0000_0000_0000;
        else if (CE ==1) Q<=D;
endmodule
```

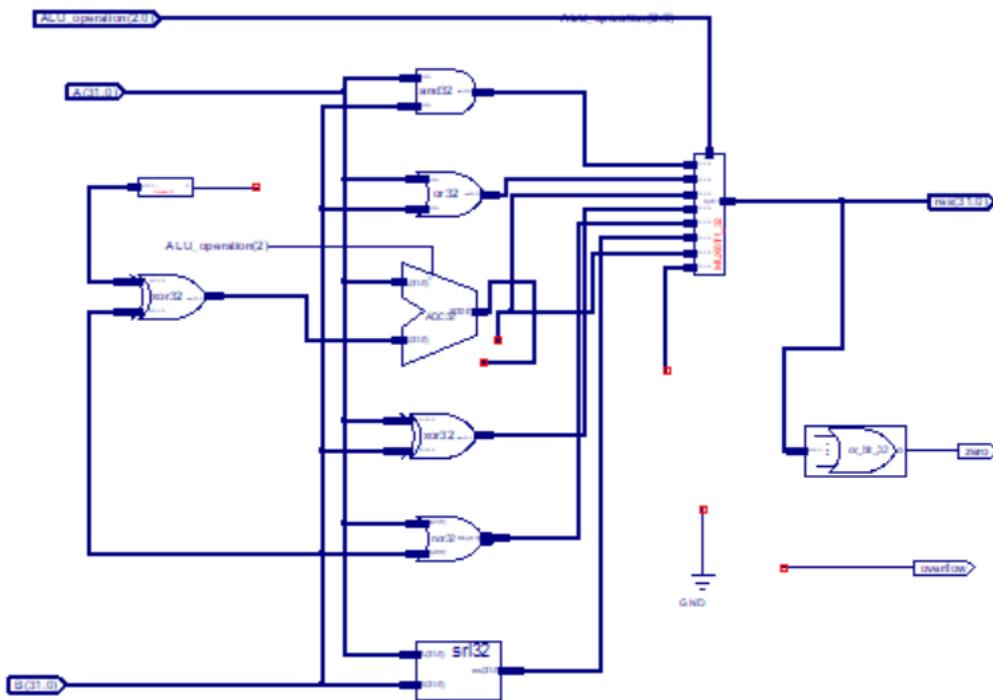
5.1.2 仿真代码

```
initial begin
    clk = 0;
    rst = 0;
    L_S = 0;
    R_addr_A = 0;
    R_addr_B = 0;
    Wt_addr = 0;
    wt_data = 0;
    #100;
    rst = 1;
    #40;
    rst = 0;
    Wt_addr = 5'b00101;
    wt_data = 32'ha5a5a5a5;
    L_S = 1;
    #20;
    Wt_addr = 5'b00110;
    wt_data = 32'h55aa55aa;
    #20;
    Wt_addr = 5'b00000;
    wt_data = 32'haaaa5555;
    #20;
    L_S = 0;
    R_addr_A = 5'b00101;
    R_addr_B = 5'b00110;
end
```

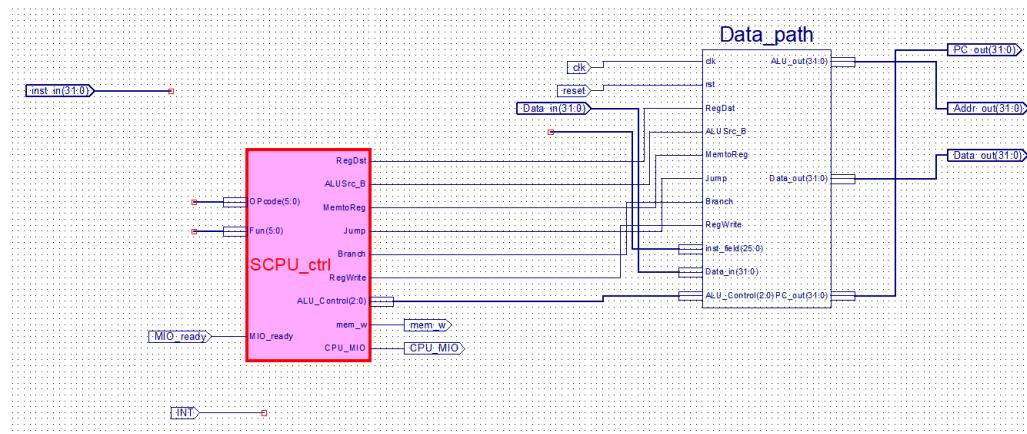
5.1.3 仿真结果



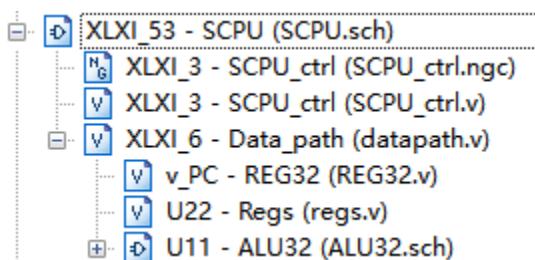
5.2 ALU32 原理图



5.3 SCPU 原理图展示



5.4 SCPU 模块结构



5.5 Datapath 源码

SCPU 顶层有两个模块，比较清晰，所以用画逻辑图的方式会比较好。但是 Datapath 模块线路复杂，画图比较繁琐且不易查错，所以我选择了写代码的方式。

而且写代码比画图稳定很多，以后我也会逐渐采用写代码的模式。

```
wire V5,N0;
assign V5=1'b1;
assign N0=1'b0;
wire [31:0] inD;
wire[31:0]Jump_addr;
wire[31:0]Branch_offset;
wire[31:0] PC_4;
wire zero;
assign PC_4=PC_out+32'b100;
assign
inD=(Jump?Jump_addr:((Branch&&zero)?PC_4+Branch_offset:PC_4));
REG32 v_PC(.clk(clk),.rst(rst),.CE(V5),.D(inD),.Q(PC_out));
wire [31:0]RdataA;
wire [31:0]RdataB;
regs
U22(.clk(clk),.rst(rst),.R_addr_A(inst_field[25:21]),.R_addr_B(inst_field[20:16]),
      .Wt_addr(RegDst?inst_field[15:11]:inst_field[20:16]),.Wt_data(MemtoReg?Data_in[31:0]:ALU_out[31:0]),
      .L_S(RegWrite),.rdata_A(RdataA),.rdata_B(RdataB));
assign Data_out=RdataB;
wire [31:0]imm32;
assign
imm32={inst_field[15],inst_field[15],inst_field[15],inst_field[15],inst_field[15],
        inst_field[15],inst_field[15],inst_field[15],inst_field[15],inst_field[15],
        inst_field[15],inst_field[15],inst_field[15],inst_field[15],inst_field[15],
        inst_field[15],inst_field[15],inst_field[15:0]};
ALU
U(.A(RdataA),.B(ALUSrc_B?imm32:RdataB),.ALU_operation(ALU_Control),.zero(zero),.res(ALU_out));

assign Jump_addr={PC_4[31:28],inst_field[25:0],N0,N0};
assign Branch_offset={imm32,N0,N0};

endmodule
```

5.6 实验现象一

SW[0]=1, SW[2]=1, SW[7:5]=111。

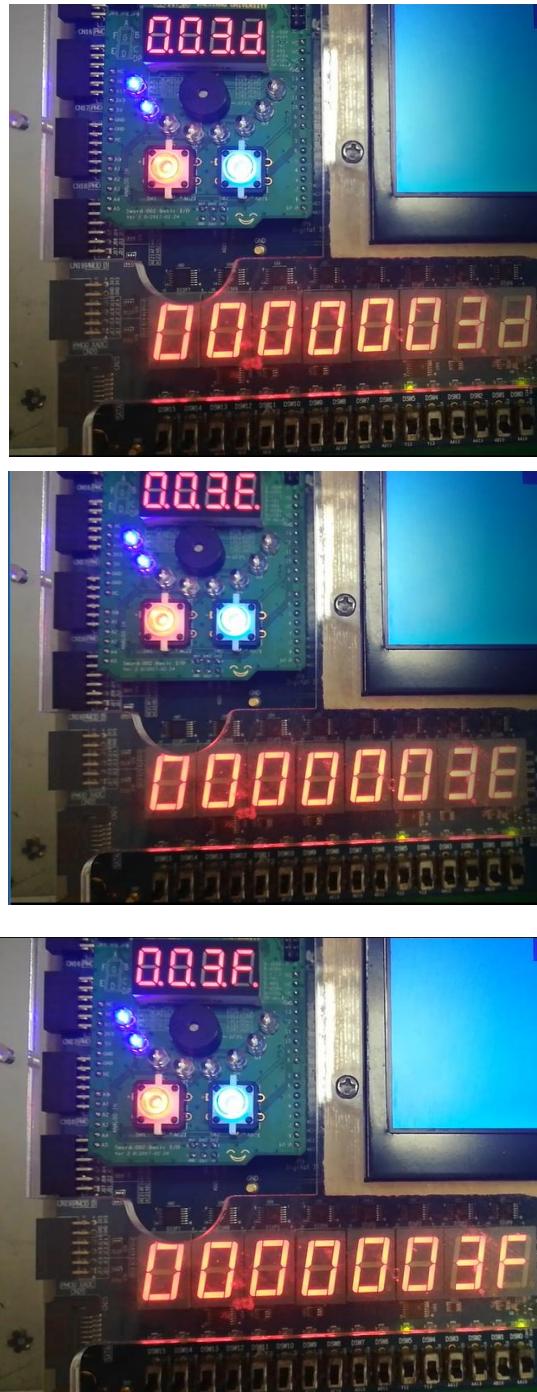
输出 CPU 指令字节地址 PC_out。



□ 实验现象 1

5.7 实验现象二

SW[0]=1, SW[2]=1, SW[7:5]=001。
输出 CPU 指令字地址 PC_out[31:2]。



□ 实验现象 2

5.8 实验现象三

$\text{SW}[0]=0$, $\text{SW}[3]=1$, $\text{SW}[4]=1$ 。

矩形框跳舞。

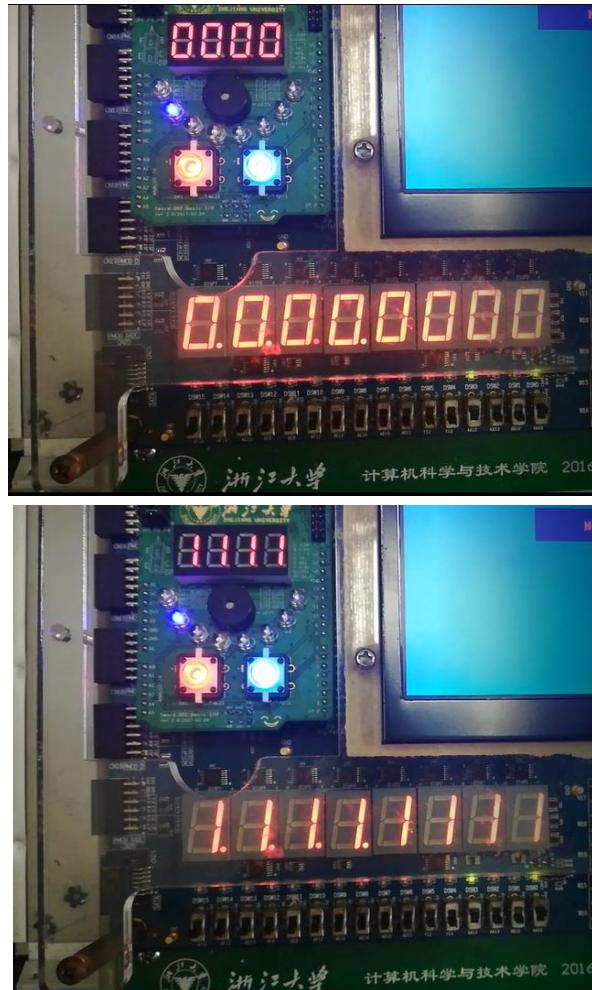


□ 实验现象 3

5.9 实验现象四

SW[0]=1, SW[3]=1。

全速时钟。



□ 实验现象 4

六、讨论、心得

上次实验是写好了 SCPU 的顶层模块，这次就是先拆解 Data_path 了。感觉实验设计的老师煞费苦心，设计的实验也蛮不错的。

Data_path 模块线路比较复杂，我就使用了写代码的方式。它里面也有一些子模块，如 ALU、regs 等，在实现这些小模块的过程中一定要仔细，不然在之后的查错过程中会造成很大的麻烦。如果不放心的话，可以一个一个做仿真测试。

实验中我还经常遇到一个问题：如果一个模块有一个小改动，返回上层的 sch，应该怎么调整呢？经过很多次的尝试，我发现要先把上层的 sch 关掉，然后重新生成这个模块的 sym（如果无法覆盖，要手动删掉原来的那一块）；再把 sch 重新打开，就会跳出一个弹窗，问你要不要 update 这个小模块。

实验六--CPU 设计之控制器

姓名: 蒋仕彪 学号: 3170102587 专业: 求是科学班(计算机) 1701

课程名称: 计算机组装实验 同组学生姓名:

实验时间: 2019-4-2 实验地点: 紫金港东 4-509 指导老师: 马德

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. 设计控制器
4. 学习测试方案的设计
5. 学习测试程序的设计

二、实验任务和原理

2.1 实验任务

1. 设计 9⁺条指令的控制器

用硬件描述语言设计实现控制器

根据 Exp05 数据通路及指令编码完成控制信号真值表

替换 Exp05 的控制器核

此实验在 Exp05 的基础上完成

2. 设计控制器测试方案:

OP 译码测试: R-格式、访存指令、分支指令, 转移指令

运算控制测试: Function 译码测试

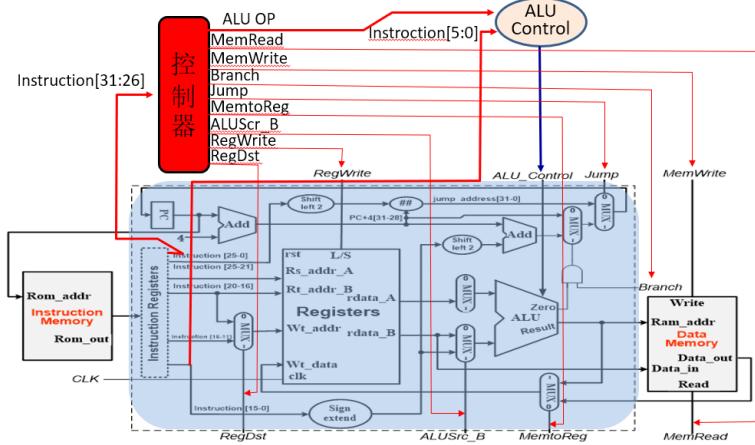
3. 设计控制器测试程序

2.2 实验原理

2.2.1 单周期数据通路结构

控制对象：数据通路结构

□ 写出九条指令控制信号真值表



2.2.2 控制信号定义

控制信号定义

□ 通路与操作控制

信号	源数目	功能定义	赋值0时动作	赋值1时动作
ALUSrc_B	2	ALU端口B输入选择	选择寄存器B数据	选择32位立即数 (符号扩展后)
RegDst	2	寄存器写地址选择	选择指令rt域	选择指令rs域
MemtoReg	2	寄存器写入数据选择	选择存储器数据	选择ALU输出
Branch	2	Beq指令目标地址选择	选择PC+4地址	选择转移地址 (Zero=1)
Jump	2	J指令目标地址选择	选择J目标地址	由Branch决定输出
RegWrite	-	寄存器写控制	禁止寄存器写	使能寄存器写
MemWrite	-	存储器写控制	禁止存储器写	使能存储器写
MemRead	-	存储器读控制	禁止存储器读	使能存储器读
ALU_Control	000-111	3位ALU操作控制	参考表Exp04	Exp04

2.2.3 ALU 操作译码

ALU操作译码

Second level

□ 参考实验四

Instruction opcode	ALUop	Instruction operation	Funct field	Desired ALU action	ALU_Control
LW	00	Load word	xxxxxx	Load word	010
SW	00	Store word	xxxxxx	Store word	010
Beq	01	branch equal	xxxxxx	branch equal	110
R-type	11	add	10 0000	add	010
R-type	11	subtract	10 0010	subtract	110
R-type	11	AND	10 0100	AND	000
R-type	11	OR	10 0101	OR	001
R-type	11	Set on less than	10 1010	Set on less than	111
R-type	11	NOR	10 0111	NOR	100

2.2.4 SCPU_ctrl

CPU部件之数据通路接口：SCPU_ctrl

□ SCPU_ctrl

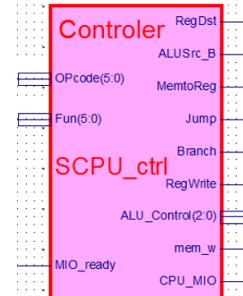
- CPU主要部件之一
- 寄存器传输控制者：编码转换成命令

□ 基本功能

- 微控制控制
- 数据传输通道控制
- 时序控制：单周期时序在那里？

□ 接口要求- SCPU_ctrl

- 控制器接口信号如右图
- 模块符号文档：SCPU_ctrl.sym



数据通路接口信号标准- SCPU_ctrl.v

```

module SCPU_ctrl(
    input [5:0]OPcode, //OPcode
    input [5:0]Fun, //Function
    input MIO_ready, //CPU Wait
    output reg RegDst,
    output reg ALUSrc_B,
    output reg MemtoReg,
    output reg Jump,
    output reg Branch,
    output reg RegWrite,
    output reg mem_w,
    output reg [2:0]ALU_Control,
    output reg CPU_MIO
);
endmodule

```

三、主要仪器设备

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

四、操作方法与实验步骤

4.1 设计工程

设计工程: OExp06-OwnSCPU

◎ 设计CPU之控制器

- └ 根据理论课分析讨论设计实验五数据通路的控制器
- └ 原理图或HDL描述均可
 - 但必须用函数表达式结构描述
- └ 仿真测试控制器模块

◎ 集成替换验证通过的数据通路模块

- └ 替换实验五(Exp05)中的SCPU_ctrl.ngc核
- └ 顶层模块延用Exp05
 - 模块名: Top_OExp06_OwnSCPU.sch

◎ 测试控制器模块

- └ 设计测试程序(MIPS汇编)测试:
- └ OP译码测试:
 - R-格式、访存指令、分支指令, 转移指令
- └ 运算控制测试: Function译码测试

4.2 设计要点

设计要点

□ 设计主控制器模块

- 写出控制器的函数表达式
- 结构描述实现电路

□ 设计ALU操作译码

- 写出ALU操作译码函数表达式
- 结构描述实现电路
- 使用DEMO作功能初步调试
 - ALU必须运算包含“nor”操作
 - 否则需要修改或重新设计调试程序

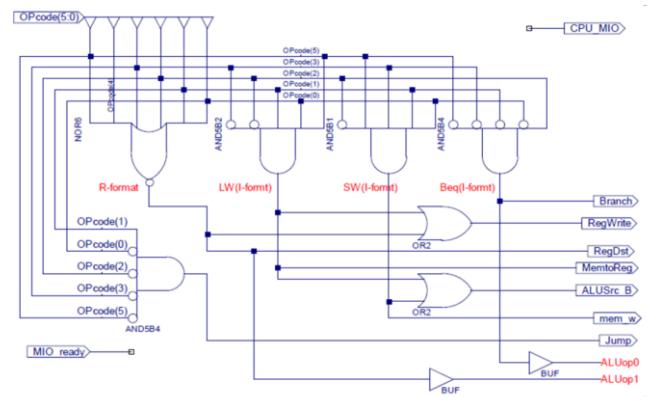
□ 仿真二个控制器电路模块

- 可以单独或合并仿真, 但最后要合并为一个控制模块

4.3 SCPU_ctrl

4.3.1 逻辑电路

指令译码-主控制器逻辑电路



4.3.1 逻辑电路

主控制器HDL描述结构

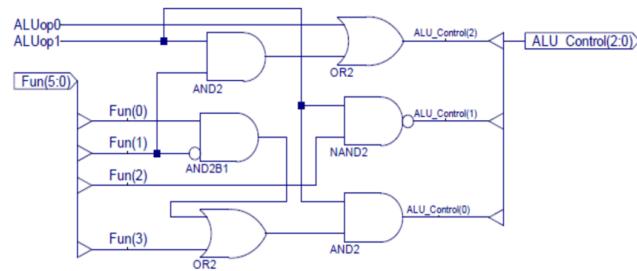
□ 指令译码器参考描述

```
'define CPU_ctrl_signals
{RegDst,ALUSrc_B,MemtoReg,RegWrite,MemRead,MemWrite,Branch,Jump,ALUOp}
    assign mem_w = MemWrite&&(~MemRead);
    always @* begin
        case(OPcode)
            6'b000000: begin CPU_ctrl_signals = ?; end
            6'b100011: begin CPU_ctrl_signals = ?; end
            6'b101011: begin CPU_ctrl_signals = ?; end
            6'b000100: begin CPU_ctrl_signals = ?; end
            6'b000010: begin CPU_ctrl_signals = ?; end
            6'h24:      begin CPU_ctrl_signals = ?; end
            .....
            default:   begin CPU_ctrl_signals = ?; end
        endcase
    end
```

4.5 ALU 操作译码

4.5.1 逻辑电路

ALU操作译码器逻辑电路



4.5.2 参考代码

ALU操作译码器HDL描述结构

□ ALU控制器参考描述

```
always @* begin
    case(ALUop)
        2'b00: ALU_Control = ?;           //add计算地址
        2'b01: ALU_Control = ?;           //sub比较条件
        2'b10:
            case(Fun)
                6'b100000: ALU_Control = 3'b010; //add
                6'b100010: ALU_Control = ?;      //sub
                6'b100100: ALU_Control = ?;      //and
                6'b100101: ALU_Control = ?;      //or
                6'b101010: ALU_Control = ?;      //slt
                6'b100111: ALU_Control = ?;      //nor:~(A | B)
                6'b000010: ALU_Control = ?;      //srl
                6'b010110: ALU_Control = ?;      //xor
                .....
            default: ALU_Control=3'bx;
        endcase
        2'b11: ALU_Control = ?;           //slti
    endcase

```

五、实验结果与分析

5.1 SCPU_ctrl

5.1.1 代码

```
module SCPU_ctrl( input[5:0]OPcode,
                   input[5:0]Fun,
                   input MIO_ready,
                   output reg RegDst,
                   output reg ALUSrc_B,
                   output reg MemtoReg,
                   output reg Jump,
                   output reg Branch,
                   output reg RegWrite,
                   output reg mem_w,
                   output reg [2:0]ALU_Control,
                   output reg CPU_MIO
);
reg [1:0] ALUop;
reg MemWrite, MemRead;
always @* begin
  `define CPU_ctrl_signals
{RegDst,ALUSrc_B,MemtoReg,RegWrite,mem_w,Branch,Jump,ALUop}
  case(OPcode)
    6'b000000: begin `CPU_ctrl_signals = 9'b100100010; end
    6'b100011: begin `CPU_ctrl_signals = 9'b011100000; end
    6'b101011: begin `CPU_ctrl_signals = 9'b010010000; end
    6'b000100: begin `CPU_ctrl_signals = 9'b000001001; end
    6'b000010: begin `CPU_ctrl_signals = 9'b000000100; end
    6'b100100: begin `CPU_ctrl_signals = 9'b010100011; end
    default: begin `CPU_ctrl_signals = 9'b100000010; end
  endcase
end
always @* begin
  case(ALUop)
    2'b00: ALU_Control = 3'b010;
    2'b01: ALU_Control = 3'b110;
    2'b10:
      case(Fun)
        6'b100000: ALU_Control = 3'b010;
        6'b100010: ALU_Control = 3'b110;
        6'b100100: ALU_Control = 3'b000;
        6'b100101: ALU_Control = 3'b001;
      endcase
  end
end
```

```

        6'b101010: ALU_Control = 3'b111;
        6'b100111: ALU_Control = 3'b100;
        6'b000010: ALU_Control = 3'b101;
        6'b010110: ALU_Control = 3'b011;
        default:   ALU_Control = 3'bxxxx;
    endcase
    2'b11: ALU_Control = 3'b111;
endcase
end

endmodule

```

5.1.2 仿真代码

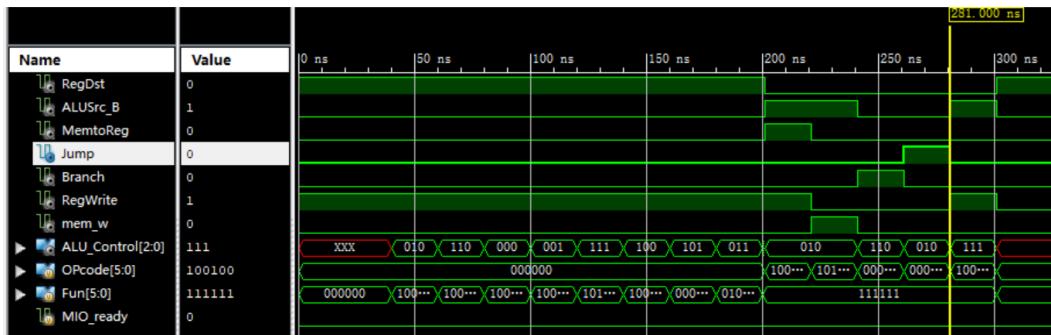
```

initial begin
    Opcode = 0;
    Fun = 0;
    MIO_ready = 0;
    #40;
    Opcode = 0;
    Fun = 6'b100000;      //add,ALU_Control=3'b010
    #20;
    Fun = 6'b100010;      //sub,ALU_Control=3'b110
    #20;
    Fun = 6'b100100;      //and,ALU_Control=3'b000
    #20;
    Fun = 6'b100101;      //or,ALU_Control=3'b001
    #20;
    Fun = 6'b101010;      //slt,ALU_Control=3'b111
    #20;
    Fun = 6'b100111;      //nor,ALU_Control=3'b100
    #20;
    Fun = 6'b000010;      //srl,ALU_Control=3'b101
    #20;
    Fun = 6'b010110;      //xor,ALU_Control=3'b011
    #20;
    Fun = 6'b111111;      //
    #1;
    Opcode = 6'b100011;//loadALUop=2'b00, RegDst=0,
    #20;                  // ALUSrc_B=1, MemtoReg=1, RegWrite=1
    Opcode = 6'b101011;
    #20; //store ALUop=2'b00, mem_w=1, ALUSrc_B=1
    Opcode = 6'b000100;//beq ALUop=2'b01, Branch=1
    #20;

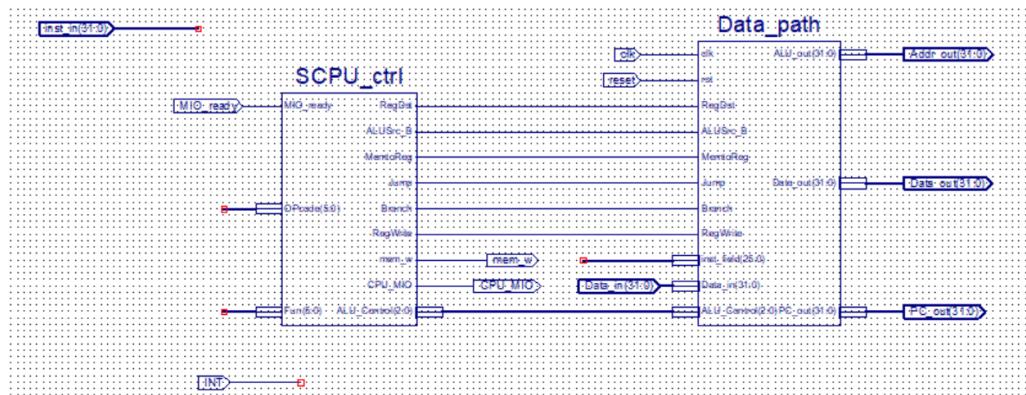
```

```
OPcode = 6'b000010; //jump Jump=1  
#20;  
OPcode = 6'h24; //slti ALUop=2'b11, RegDst=0,  
#20; //ALUSrc_B=1, RegWrite=1  
OPcode = 6'h3f;  
Fun = 6'b000000;  
end  
endmodule
```

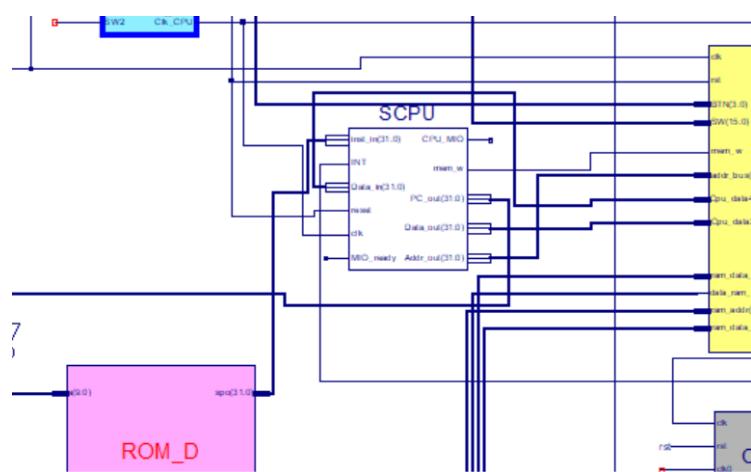
5.1.3 仿真结果



5.2 SCPU 顶层图



5.3 top 连线



5.4 实验现象一

SW[0]=1, SW[2]=1, SW[7:5]=111。

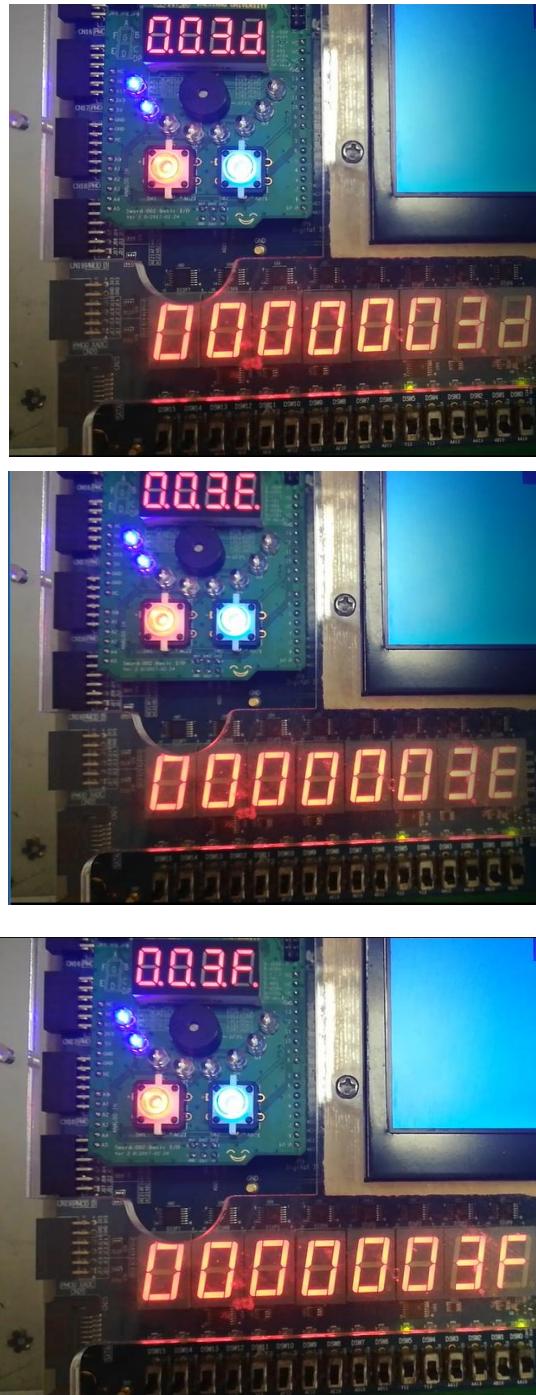
输出 CPU 指令字节地址 PC_out。



□ 实验现象 1

5.5 实验现象二

SW[0]=1, SW[2]=1, SW[7:5]=001。
输出 CPU 指令字地址 PC_out[31:2]。

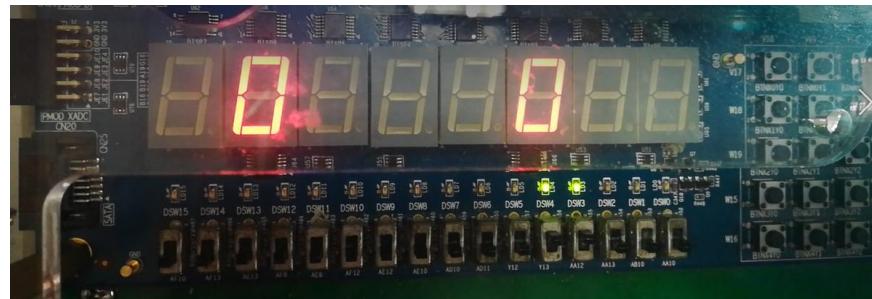


□ 实验现象 2

5.6 实验现象三

SW[0]=0, SW[3]=1, SW[4]=1。

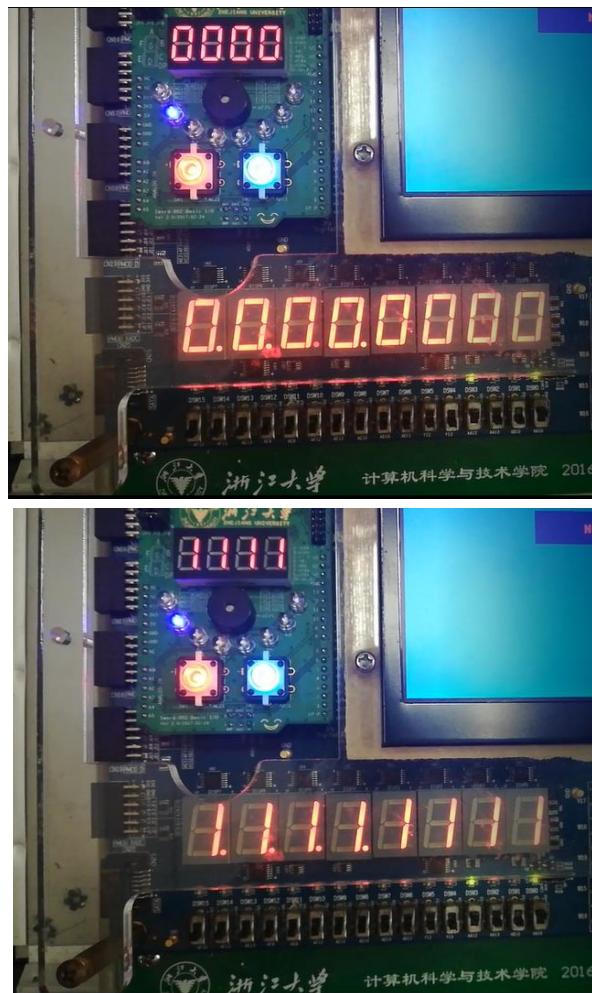
矩形框跳舞。



5.7 实验现象四

SW[0]=1, SW[3]=1。

全速时钟。



□ 实验现象 4

六、讨论、心得

实验 4-6 果然是紧密的一个系列的。到了现在这个实验，SCPU_ctrl 也被“解体”手写了。我觉得这次的 SCPU_ctrl 比上次的额 Data_path 要好玩多了。因为上次主要是连线，又单调又繁琐。这次不一样，最终代码并不是很长，而是着重思考：我们要根据 Datapath，以及每种指令的具体作用，来推断这些控制指令的 01 取值。

一些关键指令，如 lw, sw，如果稍有 01 取错，下载到板子上后就无法得到想要的效果。因此，这个实验的精度要求还是挺高的，要求我们能做到滴水不漏地写出控制变量。