



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

实验项目

A composite image showing a computer keyboard and mouse in a blue and green color scheme, overlaid with binary code (0s and 1s) and a faint grid pattern.

主讲教师：冯建文
fengjianwen@hdu.edu.cn

实验十 R-I-J 型指令的 CPU

❖ 1、实验目的

- 掌握 **MIPS R 型、I 型和 J 型指令** 的综合数据通路设计
- 掌握各种**转移类指令的控制流和指令流的多路选通控制**方法；
- 掌握 J 型、I 型和 R 型转移指令的**指令格式和寻址方式**，学习转移地址的产生方法
- 掌握**无条件转移指令和条件转移指令**的实现方法；
- 编程实现 MIPS 的部分 J 型、I 型和 R 型转移指令的功能

实验十 R-I-J 型指令的 CPU

❖ 2、实验内容与原理

- 实验九的基础上，预备实现 1 条 R 型转移指令、2 条 I 型条件转移指令和 2 条 J 型转移指令。
- 与原理课相比，多了 3 条转移指令。

R 型指令	字段	OP	rs	rt	rd	shamt	func	功能描述
	位数	6	5	5	5	5	6	
jr rs		000000	rs	5'b0	5'b0	5'b0	001000	无条件跳转： rs → PC
I 型指令	字段	OP	rs	rt	offset		功能描述	
	位数	6	5	5	16			
beq rs, rt, label		000100	rs	rt	offset		相等转移： if(rs=rt) PC+4+offset*4→PC	
bne rs, rt, label		000101	rs	rt	offset		不相等转移： if(rs≠rt) PC+4+offset*4→PC	
J 型指令	字段	OP	address				功能描述	
	位数	6	26					
j label		000010	address				无条件跳转： {(PC+4) 高 4 位 ,address,0,0} →PC	
jal label		000011	address				无条件跳转并链接： (PC+4)→\$31, 页面转 移地址→ PC	

实验十 R-I-J 型指令的 CPU

❖ 从上表可知：

- J 型格式的指令

- 6 位的 OP 字段和 26 位的 address 字段构成
- 26 位的 address 不是直接转移地址，需要和 (PC+4) 的高 4 位并位处理

- 2 条 J 型格式的指令 j 和 jal 是无条件必转指令

- 相比 j 指令，jal 指令不仅转移，且在转移前，将 jal 指令的下一条指令的地址保存到编号为 31 的 \$ra (\$31) 寄存器。

- R 型格式必转指令 jr

- 将 rs 寄存器中的 32 位数据作指令地址，直接置入 PC; 常和 jal 搭配使用，用于子程序的调用与返回

实验十 R-I-J 型指令的 CPU

- jal 指令：相当于 call（子程序调用）指令，jr 指令相当于 ret（子程序返回）指令 ----- 查看举例代码
- 2 条条件转移指令是 I 型指令格式
 - **offset 是带符号数**，需做符号扩展（为 32 位）后，再左移两位，与新的 PC 值（PC+4）相加，得到转移地址

实验十 R-I-J 型指令的 CPU

主存地址：指令

.....

0x0040_0000:jal subroutine1;# 子程序调用,
\$ra=0x0040_0004

0x0040_0004:

.....

0x0040_0038: subroutine1:

.....

0x0040_0040:jr \$31; # 子程序返回

0x0040_0044:

返回

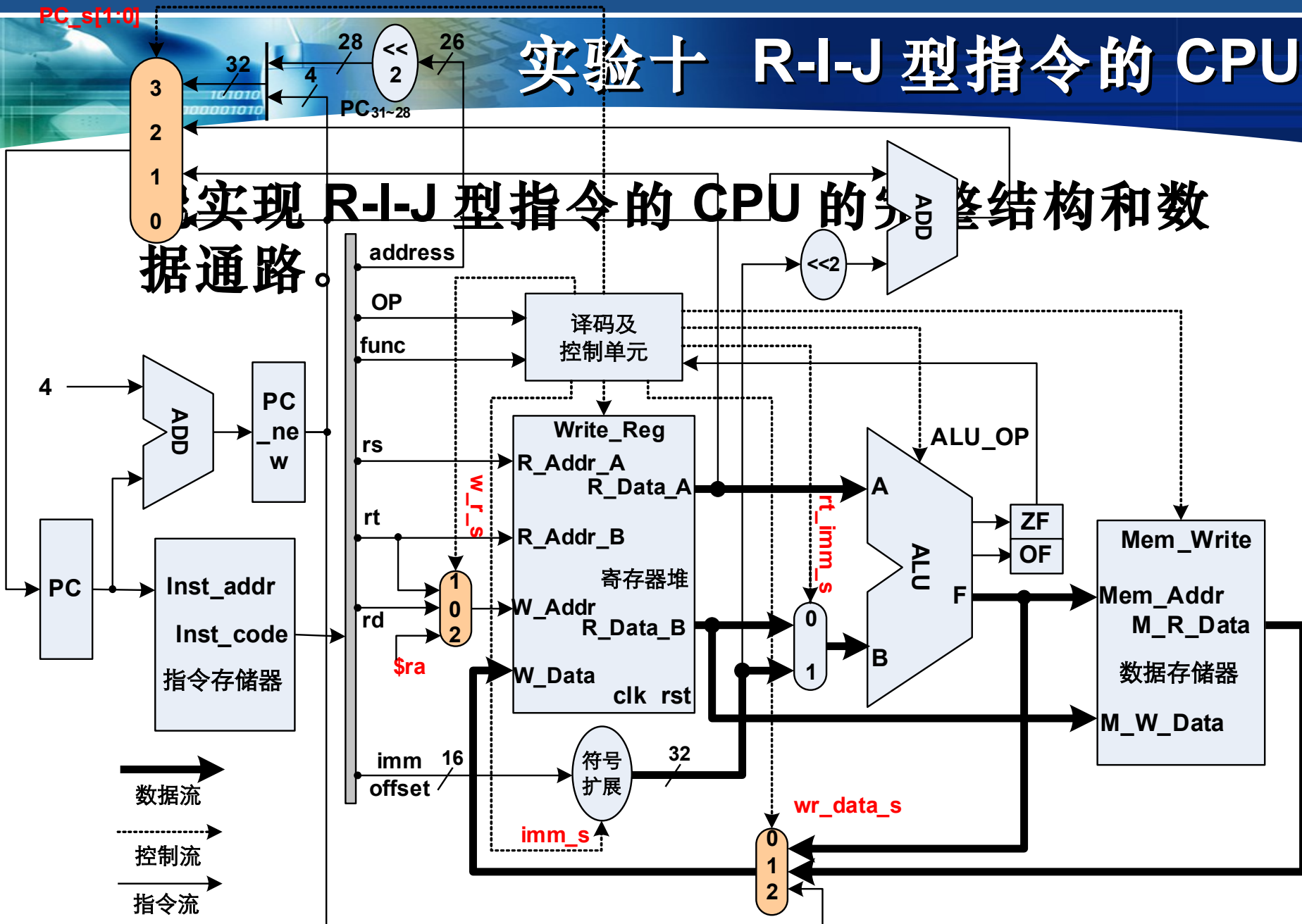
实验十 R-I-J 型指令的 CPU

❖ (2) 转移指令的数据通路

- 分析 5 条转移指令，**转移地址的产生方法**有 3 种：
 - (1) rs ;
 - (2) $PC+4+offset*4$;
 - (3) $\{(PC+4) \text{ 高 } 4 \text{ 位}, address, 0, 0\}$;
- 如何产生 PC 后继地址
 - 对于 **PC 自增**，可以使用 $PC_new (=PC+4)$
 - 对于 **rs** ，直接使用寄存器堆的读出 A 数据端口
 - 对于**相对转移**，添加一个地址加法器，将 PC_new 和符号扩展和左移 2 位后的 $offset$ 相加
 - 对于**页面寻址的转移地址**，需简单的左移和拼接操作

实验十 R-I-J 型指令的 CPU

实现 R-I-J 型指令的 CPU 的 CPU 内部结构和数据通路。



实验十 R-I-J 型指令的 CPU

- **wr_data_s** 信号用于选择写入寄存器的数据来源
 - 用 Verilog HDL 描述如下：

```
assign W_Addr = (w_r_s[1]) ? 5'b11111 : ((w_r_s[0]) ? rt : rd);  
assign W_Data = (wr_data_s[1]) ? PC_new : ((wr_data_s[0]) ?  
M_R_Data : ALU_F);
```

- **PC 的四选一数据选择器及转移地址的计算**

```
always @(negedge clk)  
case (PC_s)  
    2'b00: PC <= PC_new;  
    2'b01: PC <= R_Data_A;  
    2'b10: PC <= PC_new + (imm_data<<2);  
    2'b11: PC <= {PC_new[31:28], address, 2'b00};  
endcase
```

指令	w_r_s	imm_s	rt_im m_s	wr_da ta_s	ALU_ OP	Write_Re g	Mem_Wr ite	PC_s
add rd,rs,rt	00	——	0	00	100	1	0	00
sub rd,rs,rt	00	——	0	00	101	1	0	00
and rd,rs,rt	00	——	0	00	000	1	0	00
or rd,rs,rt	00	——	0	00	001	1	0	00
xor rd,rs,rt	00	——	0	00	010	1	0	00
nor rd,rs,rt	00	——	0	00	011	1	0	00
sltu rd,rs,rt	00	——	0	00	110	1	0	00
sllv rd,rs,rt	00	——	0	00	111	1	0	00
addi rt,rs,imm	01	1	1	00	100	1	0	00
andi rt, rs, imm	01	0	1	00	000	1	0	00
xori rt, rs, imm	01	0	1	00	010	1	0	00
sltiu rt, rs, imm	01	0	1	00	110	1	0	00

实验十 R-I-J 型指令的 CPU

■ R-I-J 型指令的控制流

指令	w_r_s	imm_s	rt_im m_s	wr_da ta_s	ALU_ OP	Write_Re g	Mem_Wr ite	PC_s
lw rt, offset(rs)	01	1	1	01	—	1	0	00
sw rt, offset(rs)	—	1	1	—	—	0	1	00
jr rs	—	—	—	—	—	0	0	01
beq rs, rt, label	—	—	0	—	101	0	0	00/10
bne rs, rt, label	—	—	0	—	101	0	0	00/10
j label	—	—	—	—	—	0	0	11
jal label	1X	—	—	1X	—	1	0	11

实验十 R-I-J 型指令的 CPU

❖ (3) 指令测试

- 在 MIPS 模拟器上用实现的 8 条 R 型指令、6 条 I 型指令和 5 条 J 型指令，编写一段用于测试的汇编程序
- 测试程序 1（将内存单元 10H 开始的 20 个数据进行累加，累加和送至内存单元 30H 单元）
- 程序 1 汇编后机器码
 - 00004020, 00004820, 200a0014, 8d2b0010, 010b4020, 21290004, 214affff, 11400001, 08000003, ac0b0030
- 程序 2（将内存 0 号单元开始的 10 个数据复制 20 号单元开始的数据区，其中使用了 BankMove 子程序）

实验十 R-I-J 型指令的 CPU

#baseAddr 0000

```
add    $t0,    $zero, $zero;  #$8=0000_0000 , 累加器
add    $t1,    $zero, $zero;  #$9=0000_0000 , 变址指针
addi   $t2,    $zero, 20;     #$10=0000_0014 , 计数器
Loop1: lw    $t3,    0x10($t1);  #$11=mem(0000_0010+$9)
add    $t0,$t0, $t3           #$8= 累加和
addi   $t1,$t1, 4             # 指针 +4
addi   $t2,$t2, -1            #$10= 计数器 -1
beq    $t2,$zero,Loop2        #$10 等于 0 , 则跳出循环
j      Loop1
Loop2: sw $t3, 0x30($zero)      # 存数到 0x30H 单元
```

返回

实验十 R-I-J 型指令的 CPU

主程序：

#baseAddr 0000

```
add $a0, $zero, $zero;    #$a0=0000_0000 源数据区域首址
addi $a1,$zero,20;        #$a1=0000_0014 , 目的数据区域首址
addi $a2, $zero,10;       #$a2=0000_000a, 复制的数据个数
jal BankMove              # 子程序调用
```

#BankMove 子程序：

```
add $t0, $a0, $zero;      #$t0= 源数据区域首址
add $t1, $a1, $zero;      #$t1= 目的数据区域首址
add $t2, $a2, $zero;      #$t2= 数据块长度
Loop1: lw $t3, 0($t0);     #$t3= 取出数据
sw $t3, 0($t1);           # 存数据
addi $t2, $t2, -1;        # 计数值 -1
bne $t2, $zero, Loop1;
                                # 计数值 ≠ 0 , 则没有复制完, 转循环体首部
jr $ra                    # 复制完成, 则子程序返回
                        返回
```

实验十 R-I-J 型指令的 CPU

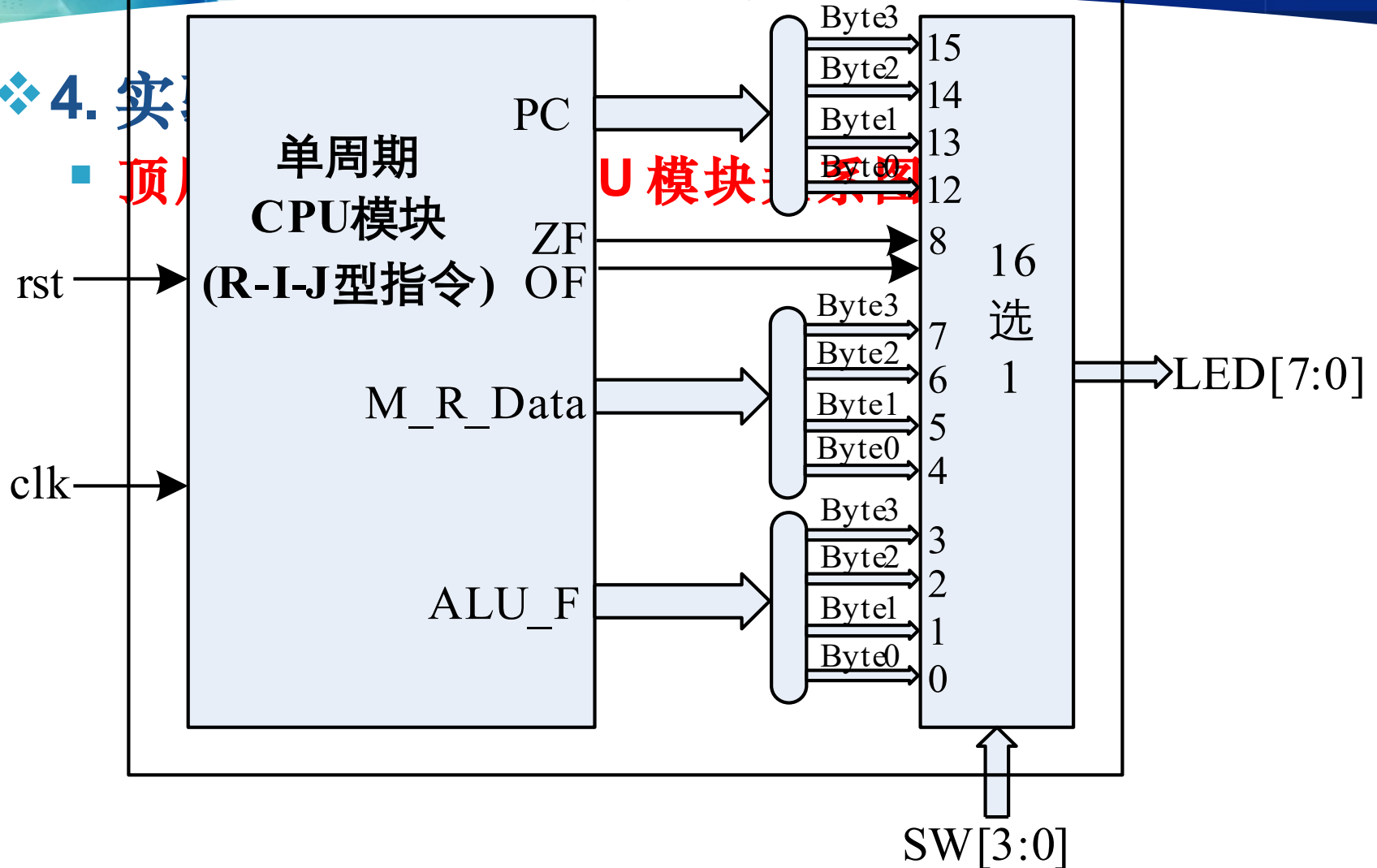
- 程序 2 汇编后，机器码如下：
 - 00002020, 20050014, 2006000a, 0c000004,
00804020, 00a04820, 00c05020, 8d0b0000,
ad2b0000, 214affff , 1540fffc, 03e00008
- 将上述机器指令码填入到和指令存储器模块 **ROM_B 相关联的 *.coe 文件中，也可以调用 *.coe 的生成软件来完成。**
- 在 RAM_B 相关联的 *.coe 文件中，可以随意填入一些数据。
- 最后执行指令存储器（ROM_B 的实例）和数据存储器（RAM_B 的实例）的 Regenerate Core 操作，更新指令存储器和数据存储器的初始化操作。

实验十 R-I-J 型指令的 CPU

顶层测试模块

❖ 4. 实验

■ 顶层



实验十 R-I-J 型指令的 CPU

❖ 若按上图所示的多路选择器来控制输出显示：

- 当执行 R 型指令和 I 型的立即数寻址指令时，可以将 4 位的开关 SW[3:0] 置于 00XX 状态，从 LED 灯观察 ALU 的运算结果（最低 2 位选择字节）
- 在执行 I 型的访存指令 lw 和 sw 指令时
 - 如果将开关 SW[3:0] 置于 00XX 状态，则从 LED 灯观察到的是数据存储器地址（也就是从 ALU 输出的地址加法运算结果）
 - 如果将开关 SW[3:0] 置于 01XX 状态，则从 LED 灯观察到的是数据存储器的读出数据（最低 2 位选择字节）

实验十 R-I-J 型指令的 CPU

- 开关 SW[3:0] 置于 1000 状态时，则显示 ZF 和 OF 标志。
- 如果在执行 J 型指令时，可以将 4 位的开关 SW[3:0] 置于 11XX 状态，从 LED 灯观察 PC 的 32 位值（是转移后的指令地址）。

实验十 R-I-J 型指令的 CPU

❖ 3、实验要求

- 实验九的基础上，编写一个 CPU 模块，除了能够实现实验九的 8 条 R 型指令、6 条 I 型指令外，还要求能够实现新的 5 条 J 型指令
 - 将实验九的工程拷贝至新目录下，成为一个新工程；修改 ROM_B 和 RAM_B 的初始化关联文件为新工程下的 *.coe 文件。
 - 定义一些控制和数据信号，添加 PC 的四选一数据通道和移位部件、地址加法器部件，重新对各模块进行逻辑连接。
 - 修改和扩充 CPU 模块中指令译码、指令执行控制部分的代码，完善 CPU 模块

实验十 R-I-J 型指令的 CPU

- 编写一个实验验证的顶层模块
- 实验室任务：
 - 配置管脚：见下表
 - 生成 *.bit 文件，下载到 Nexys3 实验板中。
 - 完成板级验证。
- 撰写实验报告。

实验十 R-I-J 型指令的 CPU

❖ 信号配置表

	信号	配置设备管脚	功能说明
输入信号	rst	1 个按钮	清零
	clk	1 个按钮	时钟引脚（BTND 或者 BTNR）
	选择信号	4 个逻辑开关	选择显示的 ALU 运算结果或存储器读出数据字节或者标志 OF、ZF：
输出信号	LED[7:0]	8 个 LED 灯	显示字节数据或标志

实验十 R-I-J 型指令的 CPU

❖ 4、实验步骤


- 在 Xilinx ISE 中创建工程，编源码，然后编译、综合
- 编写激励代码，观察仿真波形，直至验证正确
- **实验准备：**
 - 设置 N3 板卡电源开关跳线 J1，选择从 USB 取电；
 - 用 USB 电缆连接 PC 机和 N3 板卡；
 - 开 N3 实验板的电源开关；
- 在 PC 机上打开工程文件，进行**管脚配置**。
- **生成编程文件 *.bit，下载到板卡中。**
- **实验。**



实验十 R-I-J 型指令的 CPU

❖ 5、思考与探索：必做（1）

- （1）将各条指令执行的结果和标志记录到表 6.28 中，分析结果正确与否？如果不正确，请分析原因。
- （2）转移指令的 offset 字段和 address 字段的编码，计算出转移地址，观察是否和你的转移目标地址一致
- （3）I 型指令 bltzal rs, label （branch if less than zero, and link）的功能是：
 - 若寄存器 rs 小于 0，则转移并链接，相对当前指令（PC+4）转移的指令数由 offset 来决定。它的 OP 编码为 6'b000001，rt 字段为 5'b10000



The End!