



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](mailto:fengjianwen@hdu.edu.cn)

# 实验九 R-I 型指令的 CPU 设计

## ❖ 1、实验目的

- 掌握 MIPS R 型和 I 型指令的综合数据通路设计
- 掌握数据流的多路选通控制方法
- 掌握取数指令 lw 和存数指令 sw 指令的寻址方式及其有效地址产生方法
- 实现 MIPS 的部分 I 型和 R 型指令的功能

# 实验九 R-I 型指令的 CPU 设计

## ❖ 2、实验内容与原理

- 实验八的基础上，再行实现 MIPS 的 6 条 I 型指令：
  - 4 条立即数寻址的运算和传送指令
  - 2 条相对寄存器寻址的存数和取数指令。
  - 与原理课相比，多了 4 条立即数运算指令。

# 实验九 R-I 型指令的 CPU 设计

## ❖ (1) MIPS 的 I 型立即数寻址指令及数据通

字段	OP	rs	rt	imm	功能描述
位数	6	5	5	16	
汇编助记符	编码				
addi rt, rs, imm	00100 0	rs	rt	imm	算术加： rs + imm→rt
andi rt, rs, imm	00110 0	rs	rt	imm	逻辑与： rs & imm→rt
xori rt, rs, imm	00111 0	rs	rt	imm	逻辑异或： rs⊕imm→rt
sltiu rt, rs, imm	00101 1	rs	rt	imm	无符号数小于则置位： if (rs < imm) rt=1 else rt=0

## 实验九 R-I 型指令的 CPU 设计

### ❖ I 型与 R 型指令有明显不同：

- 没有 rd 寄存器，使用 rt 作为目的寄存器；
- 源操作数有一个为立即数，位于指令的低 16 位。

# 实验九 R-I 型指令的 CPU 设计

## ❖ 解决目的寄存器的可选性：

- 设置一个二选一数据选择器，控制信号为 **rd\_rt\_s**：
  - 当 **rd\_rt\_s=0**，将指令的 **rd 字段**送写地址 W\_Addr；
  - 当 **rd\_rt\_s=1**，将指令的 **rt 字段**送写地址 W\_Addr。
- Verilog 语句如下：
  - `assign W_Addr = (rd_rt_s) ? rt : rd;`

# 实验九 R-I 型指令的 CPU 设计

## ❖ 扩展 16 位的立即数 imm

- 设置一位 imm\_s 来控制这两种扩展：
- **imm\_s=1**，符号扩展；
- **imm\_s=0**，则 0 扩展。

- Verilog 语句如下：

```
assign imm_data=(imm_s) ?{16{imm[15]},imm} :  
    {16{1'b0},imm};
```

# 实验九 R-I 型指令的 CPU 设计

## ❖ ALU 的输入数据 B 端的数据选择

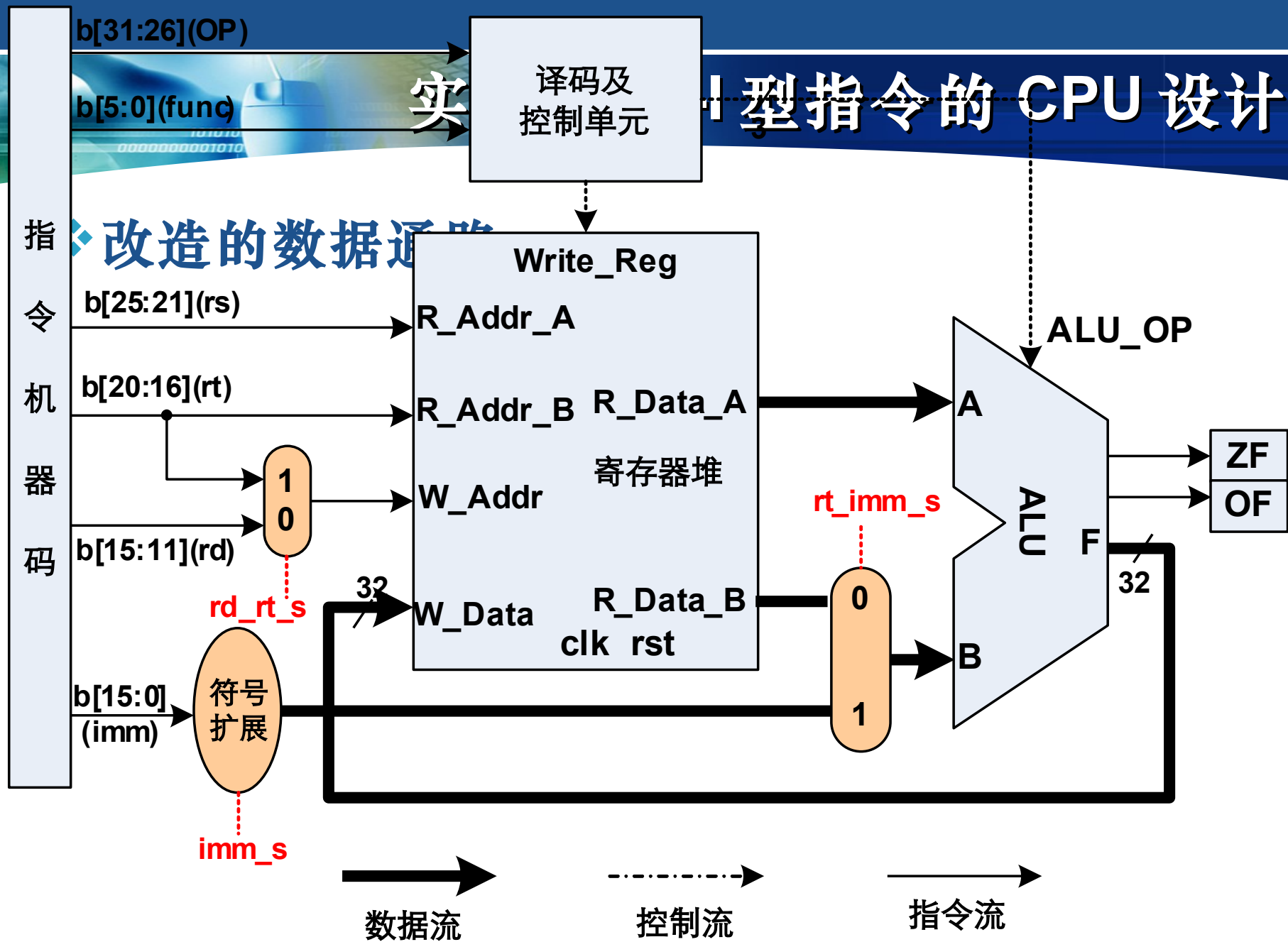
- 方法：设置二选一数据选择器（控制信号为 `rt_imm_s`）
- 当 `rt_imm_s=0`，将寄存器堆的 B 端口读出数据 `R_Data_B` 送 ALU 的 B 端
- 当 `rt_imm_s=1`，将扩展好的立即数 `imm_data` 送 ALU 的 B 输入端

- Verilog 语句如下：

```
assign ALU_B = (rt_imm_s) ? imm_data : R_Data_B;
```



# I型指令的 CPU 设计



# 实验九 R-I 型指令的 CPU 设计

## ❖ 2.1 型取数 / 存数指令及其数据通路

### • MIPS I 型存储器访问指令格式及编码

字段	OP	rs	rt	offset	功能描述
位数	6	5	5	16	
汇编助记符	编码				
lw rt, offset(rs)	100011	rs	rt	offset	取数 : (rs+offset)→rt
sw rt, offset(rs)	101011	rs	rt	offset	存 数 :rt→(rs+offset)

## 实验九 R-I 型指令的 CPU 设计

### ❖ 改进数据通路，实现两条访存指令

- 添加一个数据存储器 **RAM**，存放指令访问的数据
  - 必须添加吗？
- 有效地址 **EA** 的计算：ALU 来实现，置  $rt\_imm\_s=1$ ， $imm\_s=1$ 。
  - 为何是带符号扩展？
- 将 **ALU** 的输出直接送存储器地址端口
  - Verilog 描述：  
`assign Mem_Addr = ALU_F`

## 实验九 R-I 型指令的 CPU 设计

### ❖ 改进数据通路，实现两条访存指令

#### ■ 存储器读出的数据：

- `alu_mem_s=0`，则将 ALU 的输出送寄存器堆的写数据端口
- `alu_mem_s=1`，则将存储器的读出数据送寄存器堆的写数据端口。

#### ■ Verilog 描述如下

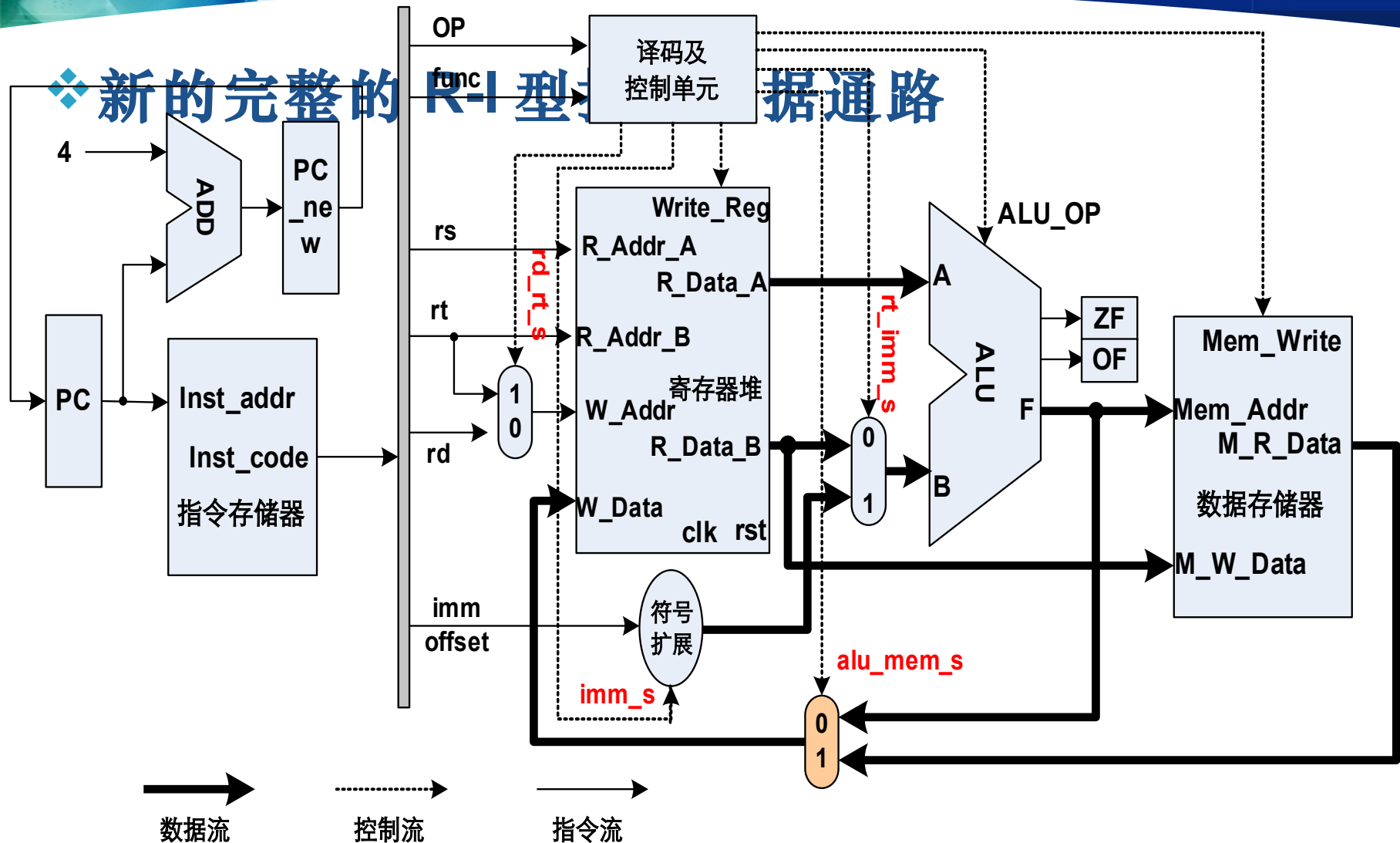
```
assign W_Data=alu_mem_s ?M_R_Data :ALU_F;
```

# 实验九 R-I 型指令的 CPU 设计

## ❖ 存储器的写入数据

- 将寄存器堆的 B 端口数据直接送至存储器的写数据端口
- Verilog 描述：
  - `assign M_W_Data = R_Data_B;`

# 实验九 R-I 型指令的 CPU 设计



# 实验九 R-I 型指令的 CPU 设计

指令	rd_rt_s	imm_s	rt_imm_s	alu_me m_s	ALU_OP	Write_Reg	Mem_Write
add rd,rs,rt	0	———	0	0	100	1	0
sub rd,rs,rt	0	———	0	0	101	1	0
and rd,rs,rt	0	———	0	0	000	1	0
or rd,rs,rt	0	———	0	0	001	1	0
xor rd,rs,rt	0	———	0	0	010	1	0
nor rd,rs,rt	0	———	0	0	011	1	0
sltu rd,rs,rt	0	———	0	0	110	1	0
sllv rd,rs,rt	0	———	0	0	111	1	0
addi rt,rs,imm	1	1	1	0	100	1	0
andi rt, rs, imm	1	0	1	0	000	1	0
xori rt, rs, imm	1	0	1	0	010	1	0
sltiu rt, rs, imm	1	0	1	0	110	1	0
lw rt, offset(rs)	1	1	1	1	100	1	0
sw rt, offset(rs)	———	1	1	———	100	0	1

# 实验九 R-I 型指令的 CPU 设计

## ❖ 3. I 型指令的时序

■ 立即数寻址的 I 型指令，执行的时序同 R 型指令：

- 在 **clk 的上跳沿**，指令存储器执行读操作
- 在 **clk 正脉冲内**，读出的指令经过译码、执行运算
- 在 **clk 的下跳沿**，将运算结果打入目的寄存器 **rd** 或者 **rt**

■ 对于取数 / 存数指令，对数据存储器的读和写访问都要与 **clk 脉冲同步**。



# 实验九 R-I 型指令的 CPU 设计

## ❖ 4. 指令测试

### ■ 测试代码

### ■ 汇编后机器码

- 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。

## ❖ 在和数据存储器模块 RAM\_B 相关联的 \*.coe 文件中，可以随意填入一些数据

#baseAddr 0000

xori	\$1,	\$0,	0x1234;	#\$1=0000_1234
addi	\$2,	\$0,	0x6789;	#\$2=0000_6789
addi	\$3,	\$0,	-0x7000;	#\$3=FFFF_9000
xori	\$4,	\$0,	0x0010;	#\$4=0000_0010
sllv	\$5,	\$2,	\$4;	#\$5=6789_0000
or	\$6,	\$1,	\$5;	#\$6=6789_1234
sllv	\$7,	\$3,	\$4;	#\$7=9000_0000
add	\$8,	\$2,	\$6;	#\$8=6789_79BD
sub	\$9,	\$2,	\$1;	#\$9=0000_5555
sub	\$10,	\$1,	\$2;	#\$10=FFFF_AAAB
addi	\$11,	\$3,	0x7FFF;	#\$11=0000_0FFF
addi	\$12,	\$3,	-0x8000;	#\$12=FFFF_1000
andi	\$13,	\$10,	0xFFFF;	#\$13=0000_AAAB
sltiu	\$14,	\$2,	0x6788;	#\$14=0000_0000
sltiu	\$15,	\$2,	0x678A;	#\$15=0000_0001

# 实验九 R-I 型指令的 CPU 设计

```
sw    $11,    0($4); #mem(0000_0010)=0000_0FFF
sw    $12,    20($0);#mem(0000_0014)=FFFF_1000
sw    $13,    16($4);#mem(0000_0020)=0000_AAAB
sw    $14,    20($4);#mem(0000_0024)=0000_0000
lw    $16,    16($0); #$16=mem(0000_0010)=0000_0FFF
lw    $17,    4($4);  #$17=mem(0000_0014)=FFFF_1000
or    $18,    $16,    $17;  #$18=FFFF_1FFF
lw    $19,    16($4); #$19=mem(0000_0020)=0000_AAAB
lw    $20,    20($4);#$20=mem(0000_0024)=0000_0000
nor   $21,    $19,    $20;  #$21=FFFF_5554
lw    $22,    -0x10($4); #$22=mem(0000_0000), 譬如
8888_8888
lw    $23,    -0x0C($4); #$23=mem(0000_0004), 譬如
9999_9999
sltu  $24,    $22,    $23 #$24=? , 譬如 =0000_0001
```

返回

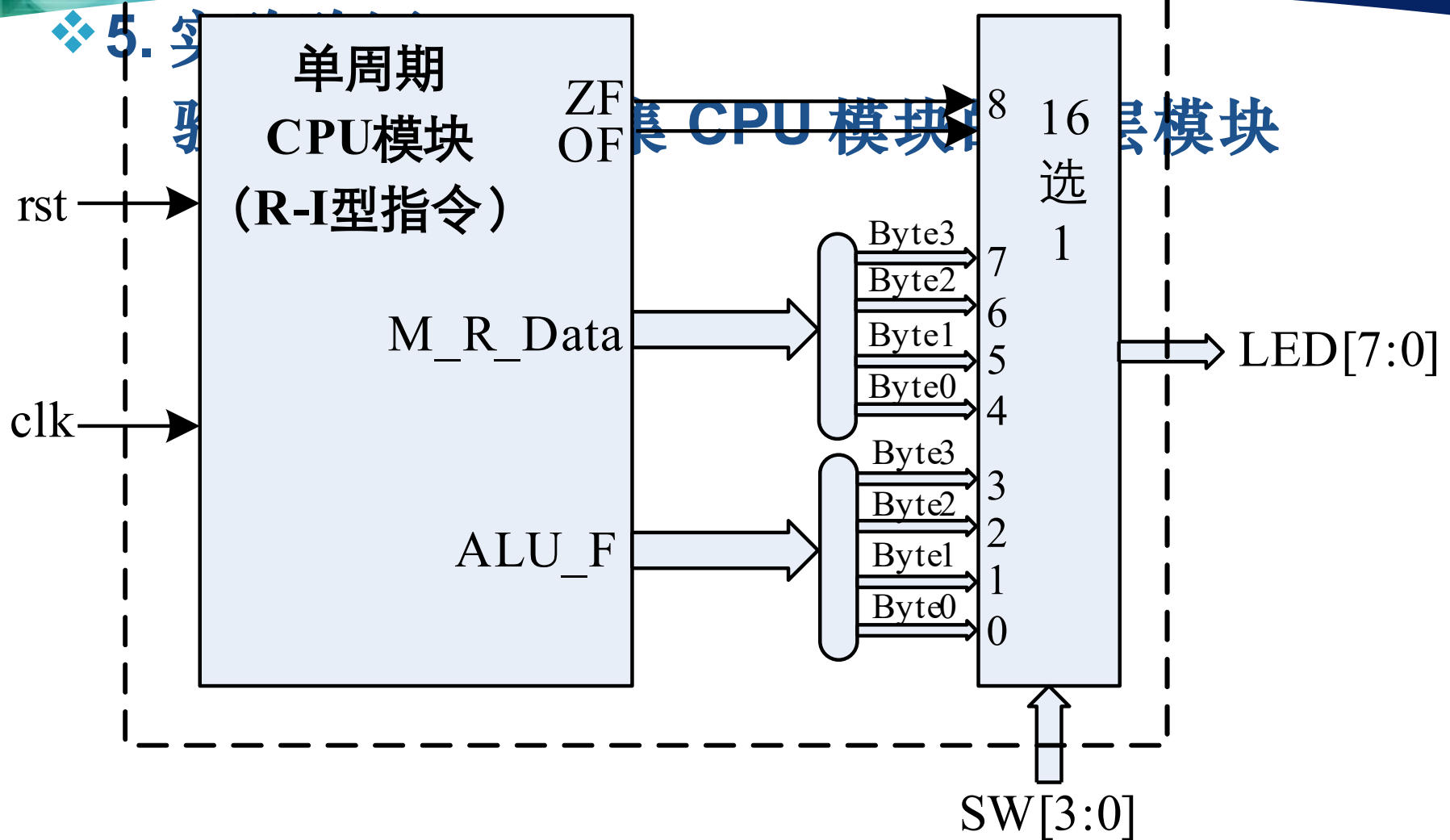
# 实验九 R-I 型指令的 CPU 设计

38011234,	20026789,
20039000,	38040010,
00822804,	00253025,
00833804,	00464020,
00414822,	00225022,
206b7fff,	206c8000,
314dffff,	2c4e6788,
2c4f678a,	ac8b0000,
ac0c0014,	ac8d0010,
ac8e0014,	8c100010,
8c910004,	02119025,
8c930010,	8c940014,
0274a827,	8c96fff0,
8c97fff4,	02d7c02b

返回

# 实验九 R-I 型指令的 CPU 设计

顶层测试模块



# 实验九 R-I 型指令的 CPU 设计

## ❖ 实验要求

- 在实验八的基础上，编写一个 CPU 模块
  - 实现实验八的 8 条 R 型指令
  - 实现新的 6 条 I 型指令
- 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。

# 实验九 R-I 型指令的 CPU 设计

	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 OE、ZF.
输出 信号	LED[7:0]	8 个 LED 灯	显示字节数据或标志

# 实验九 R-I 型指令的 CPU 设计

## ❖ 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。**




# 实验九 R-I 型指令的 CPU 设计

## ❖ 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 6.24 中，分析结果正确与否？如果不正确，请分析原因。
- （2）I 型指令 `lui rt, imm` 将立即数 `imm` 装入 `rt` 寄存器的高 16 位，低 16 位清零。它的 OP 编码为 `6'b001111`，`rs` 字段为 `5'b00000`，试着实现该指令
- （3）说说你在实验中碰到了哪些问题，你是如何解决的？





The End!