



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)

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 1、实验目的

- 掌握 **MIPS R 型指令的数据通路设计**，掌握指令流和数据流的控制方法；
- 掌握**完整的单周期 CPU 顶层模块**的设计方法；
- **实现 MIPS R 型指令的功能**；

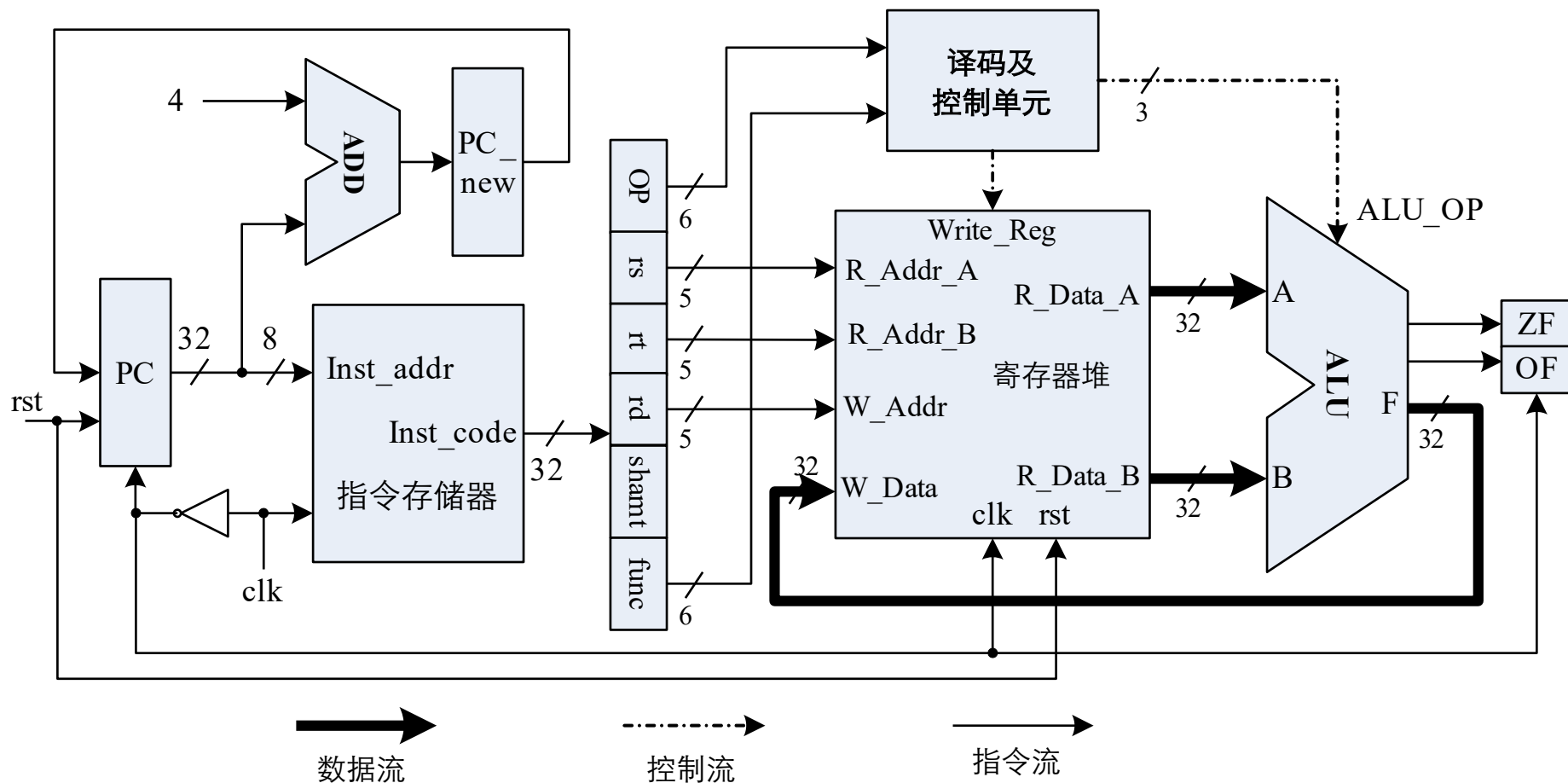
# 实验八实现 R 型指令的 CPU 设计实验

## ❖2、实验内容与原理

- 设计实现一个**单周期 CPU**，实现 8 条 R 型指令；
  - (1) 建立 R 型指令的数据通路；
  - (2) 构造顶层模块，含部件：
    - 指令存储器（实验七）
    - PC 及自增电路（实验七）
    - 寄存器堆模块（实验四）
    - ALU 模块（实验三）
    - 指令译码与控制单元：**新增**
      - 根据指令码和功能码，为数据通路上各部件发送控制信号（置位或复位）；

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ (1) R 型指令数据通路:



# 实验八实现 R 型指令的 CPU 设计实验

## ❖ (2) R 型指令集:

字段	OP	rs	rt	rd	shamt	func	功能描述
位数	6	5	5	5	5	6	
汇编助记符	编码						
add rd,rs,rt	000000	rs	rt	rd	00000	100000	算术加： $rs + rt \rightarrow rd$
sub rd,rs,rt	000000	rs	rt	rd	00000	100010	算术减： $rs - rt \rightarrow rd$
and rd,rs,rt	000000	rs	rt	rd	00000	100100	位与： $rs \& rt \rightarrow rd$
or rd,rs,rt	000000	rs	rt	rd	00000	100101	位或： $rs \mid rt \rightarrow rd$
xor rd,rs,rt	000000	rs	rt	rd	00000	100110	位异或： $rs \oplus rt \rightarrow rd$
nor rd,rs,rt	000000	rs	rt	rd	00000	100111	位或非： $\sim(rs \mid rt) \rightarrow rd$
sltu rd,rs,rt	000000	rs	rt	rd	00000	101011	无符号数小于则置位： $if(rs < rt)rd=1 \text{ else } rd=0$
sllv rd,rt,rs	000000	rs	rt	rd	00000	000100	逻辑左移： $(rt \ll rs) \rightarrow rd$

# 实验八实现 R 型指令的 CPU 设计实验

❖ 由上表可知

- **R 型指令的共同特征：**

- 操作码字段 OP=000000b

- 指令的功能则由功能码字段 func 指出

- **R 型指令的操作数有 3 个**

- 两个源操作数在 rs 和 rt 字段所指定的寄存器中

- 目的操作数是 rt 字段所指定的寄存器

❖ **汇编助记符**中，紧跟指令符号右边的寄存器是目的寄存器，这和**指令机器码**的排列顺序有差异

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ (3) 构造 CPU 顶层模块：

- ① 新建一个工程；
- ② 新建一个 Veilog Module 作为 CPU 顶层模块；
- ③ 拷贝各个模块的 \*.v 文件到当前工程目录下；
- ④ 将各个 \*.v 文件加到工程中来；
  - 对于指令存储器的 \*.xco，处理较复杂（可重新在工程中生成 ROM 的 IP 核）
- ⑥ 在顶层模块中，引用各个模块的实例；
  - 定义一组信号做为各个模块间的信号连接；
- ⑦ 在顶层模块中，编写有关指令译码和控制单元的程序；

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 注意：

- **寄存器模块：** **\$0** 要始终置为 0，且不允许写操作；
- **指令译码：** **OP=000000** 为 R 型指令，由 **func** 字段识别具体 R 型指令；
- **控制信号赋值：** 依据指令译码，见下表



# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 指令译码与控制： OP=6'b000000

指令	func( 输入 )	ALU_OP( 输出 )	操作
add rd,rs,rt	100000	100	位与
sub rd,rs,rt	100010	101	位或
and rd,rs,rt	100100	000	位异或
or rd,rs,rt	100101	001	位非
xor rd,rs,rt	100110	010	算术加
nor rd,rs,rt	100111	011	算术减
sltu rd,rs,rt	101011	110	小于置位
sllv rd,rt,rs	000100	111	逻辑左移

# 实验八 实现 R 型指令的 CPU 设计

## ❖ 时序设计

- 时钟源为 CPU 的主频脉冲  $\text{clk}$
- $\text{clk}$  的上升沿，启动指令存储器依据 PC 读指令
- $\text{clk}$  高电平持续期间，完成 PC 值的自增、指令译码、寄存器读操作，随后完成 ALU 运算
- $\text{clk}$  的下降沿则完成目的寄存器的写入、PC 值的更新和标志寄存器的更新
- 时序实现时：
  - 指令存储器的  $\text{clk}$  :  $\text{clk}$
  - PC 寄存器的打入  $\text{clk}$  :  $\sim\text{clk}$
  - 寄存器堆的写入  $\text{clk}$  :  $\sim\text{clk}$

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 指令对标志寄存器的影响

- 传送类指令和跳转类指令**不影响**标志位
- 有符号算术运算类指令（包括 `slt` 和算术移位指令）**影响 ZF 和 OF**
- 无符号算术运算类指令和逻辑运算类指令**影响 ZF**，**不影响 OF**
- 条件转移类指令一般会使用**标志位 ZF**

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 标志寄存器的赋值

- 算术运算类指令：影响标志位 ZF 和 OF
- 逻辑运算类指令：只影响标志位 ZF
- 存储器访问指令：不影响标志位
- 无条件跳转指令：不影响标志位
- 有条件分支指令：影响标志位 ZF

## ❖ 影响→赋值

## ❖ 不影响→不赋值

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ (4) 指令测试

- 实现的 CPU 能够支持 8 条 R 型指令子集吗？
- 用 8 条 R 型指令编写一段程序，用于测试 CPU 的功能：

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 测试程序：汇编

```
nor $1, $0, $0;    #$1=FFFF_FFFF
sltu $2, $0, $1;    #$2=0000_0001
add $3, $2, $2;     #$3=0000_0002
add $4, $3, $2;     #$4=0000_0003
add $5, $4, $3;     #$5=0000_0005
add $6, $5, $3;     #$6=0000_0007
sllv $7, $6, $2;     #$7=0000_000E
add $9, $5, $6;     #$9=0000_000C
sllv $8, $6, $9;     #$8=0000_7000
xor $9, $1, $8;     #$9=FFFF_8FFF
add $10, $9, $1;    #$10=FFFF_8FFE
sub $11, $8, $7;    #$11=0000_6FF2
sub $12, $7, $8;    #$12=FFFF_900E
and $13, $9, $12;   #$13=FFFF_800E
or $14, $9, $12;    #$14=FFFF_9FFF
or $15, $6, $7;     #$15=0000_000F
```

```
nor $16, $6, $7;    #$16=FFFF_FFF0
add $17, $7, $3;     #$17=0000_0010
sllv $18, $8, $17;   #$18=7000_0000
sllv $19, $3, $17;   #$19=0002_0000
sllv $20, $19, $7;   #$20=8000_0000
add $21, $20, $1;    #$21=7FFF_FFFF
or $22, $18, $21;    #$22=7FFF_FFFF
add $23, $20, $22;   #$23=FFFF_FFFF
sub $24, $20, $22;   #$24=0000_0001
sub $25, $22, $20;   #$25=FFFF_FFFF
xor $26, $18, $1;    #$26=8FFF_FFFF
sltu $27, $22, $20;  #$27=0000_0001
sltu $28, $26, $20;  #$28=0000_0000
add $29, $22, $2;     #$29=8000_0000
sub $30, $20, $2;     #$30=7FFF_FFFF
add $31, $11, $26;   #$31=9000_6FF1
```

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 测试程序：机器指令编码

- 00000827, 0001102b, 00421820, 00622020, 00832820, 00a33020, 00463804, 00a64820, 01264004, 00284826, 01215020, 01075822, 00e86022, 012c6824, 012c7025, 00c77825, 00c78027, 00e38820, 02289004, 02239804, 00f3a004, 0281a820, 0255b025, 0296b820, 0296c022, 02d4c822, 0241d026, 02d4d82b, 0354e02b, 02c2e820, 0282f022, 017af820

## ❖ ① 将机器代码编辑成 \*.coe 文件（格式化）

；

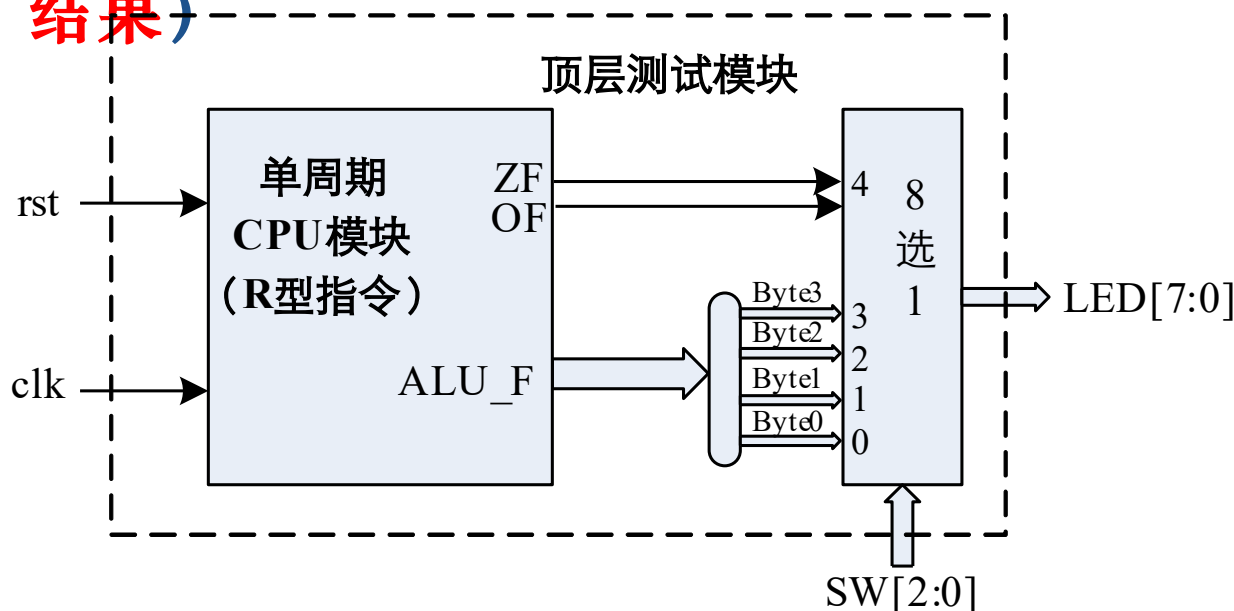
## ❖ ② 与指令存储器关联；

## ❖ ③ 重新生成 ROM 核： Regenerate Core 操

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ (5) 实验验证

- 在板卡上如何看到测试程序的每条指令的执行结果？如何验证功能是否正确？
- 在 CPU 模块之上，再构造一个实验验证模块，其中引用改造过的 CPU 模块（引出 ALU 的运算结果）





# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 3、实验要求

- 在实验三、实验四、实验五和实验六的基础上，**编写一个 CPU 模块，能够实现 8 条指定的 R 型指令。**
- **编写一段测试 8 条指令的汇编程序，使用实验六的汇编器，将其翻译成二进制机器码，并通过关联文件初始化指令存储器。**
- **编写一个实验验证的顶层模块，用于验证实验。**
- **实验室任务：**
  - **配置管脚；**
  - **生成 \*.bit 文件并下载。**
  - **完成板级验证。**
- **撰写实验报告。**

# 实验八实现 R 型指令的 CPU 设计实验

## 实验八信号配置表

	□ □	□ □ □ □ □ □	□ □ □ □
□ □ □ □	rst	1 □ □ □	□ □
	clk	1 □ □ □	□ □ □ □ □ BTND □ □ BTNR □
	□ □ □ □	3 □ □ □ □ □	□ □ □ □ □ ALU □ □ □ □ □ □ □ □ □ □ OF □ ZF □
□ □ □ □	LED[7:0]	8 □ LED □	□ □ □ □ □ □ □ □ □ □

# 实验八实现 R 型指令的 CPU 设计实验


## ❖ 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；**
- 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：**
  - 按动 rst 按钮使 PC、寄存器堆、ZF、OF 清零；
  - 每按动 clk 按钮一次，则执行一条指令；
  - 拨动 3 位逻辑开关选择 ALU 的 32 位结果字节或者标志，并记录

# 实验八实现 R 型指令的 CPU 设计实验

## ❖ 5、思考与探索：必做（1）

- （1）将测试程序（ $\geq 16$  条）的各条指令执行的结果和标志记录到表 6.19 中，分析结果正确与否？如果不正确，请分析原因。
- （2）sll rd, rt, shamt 指令将 rt 寄存器的数据进行逻辑左移，左移的位数则是由字段 shamt 指定。试着实现该指令，谈谈你的实现方法。
- （3）本实验实现的 sltu 指令是对无符号数的比较置位指令，如果需要实现有符号数的比较置位指令——slt 指令，请问应该如何实现？
- （4）sra 是对（有符号）数据的算术右移指令，考虑如何实现它？
- （5）说说你在实验中碰到了哪些问题，你是如何解决的？



The End!