中山大學
SUN YAT-SEN UNIVERSITY

MIPS

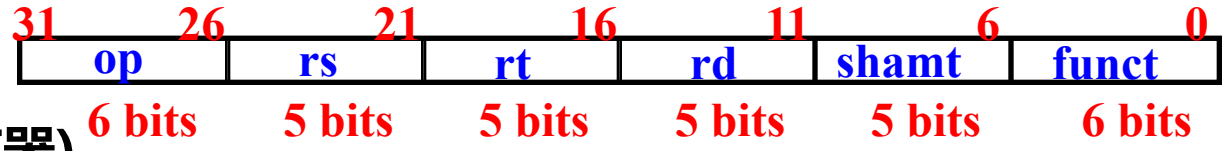
Microprocessor without Interlocked Pipeline Stage

无内部互锁流水级的微处理器

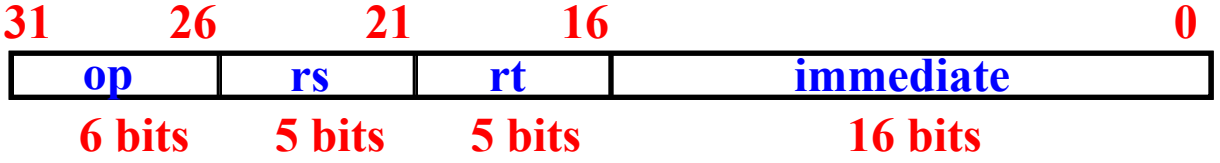
- 1. MIPS指令格式
- 2. 寄存器与存储器数据
- 3. MIPS指令类型
- 4. MIPS寻址方式
- 5. MIPS汇编语言程序结构框架

回顾——1. MIPS指令格式

- 所有指令都是32位宽，按字地址对齐
- 三种指令格式



  - R-Type(用于寄存器)
    - 两个操作数都是寄存器的运算指令。如：sub rd, rs, rt

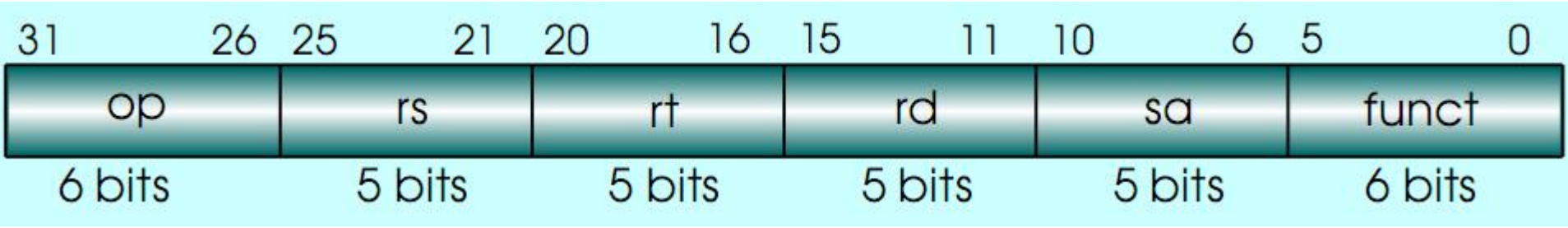


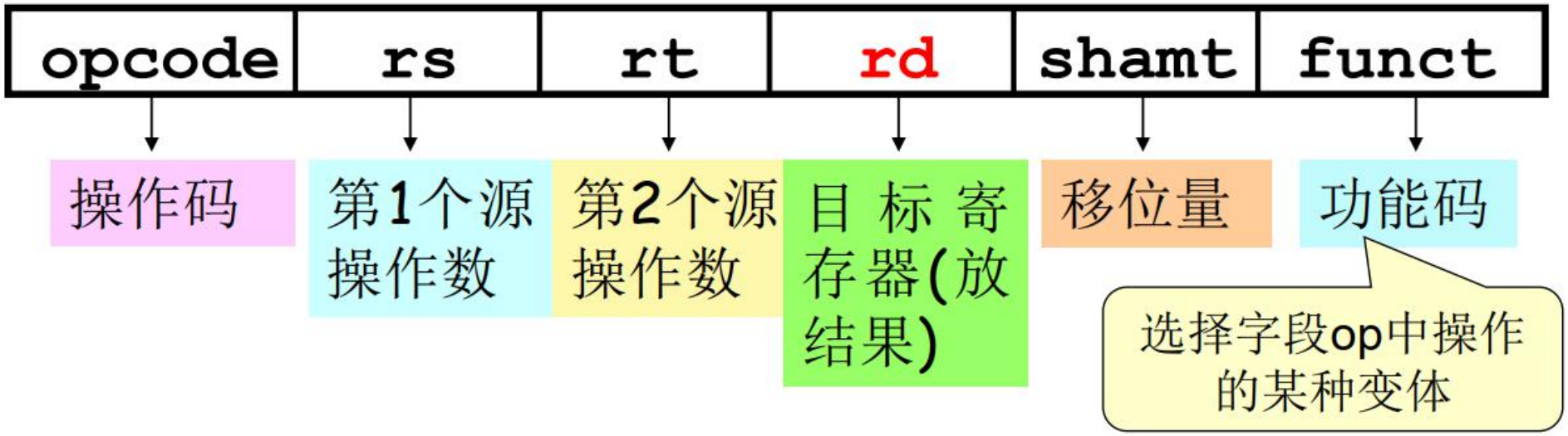
  - I-Type
    - 运算指令：如：ori rt, rs, imm16
    - Load和Store指令。如：lw rt, rs, imm16
    - 条件分支指令。如：beq rs, rt, imm16

  - J-Type
    - 无条件跳转指令。如：j target

1. MIPS指令格式

R型指令

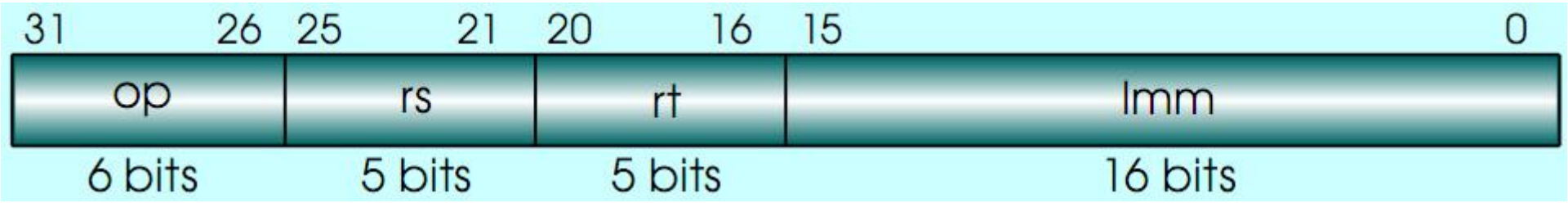


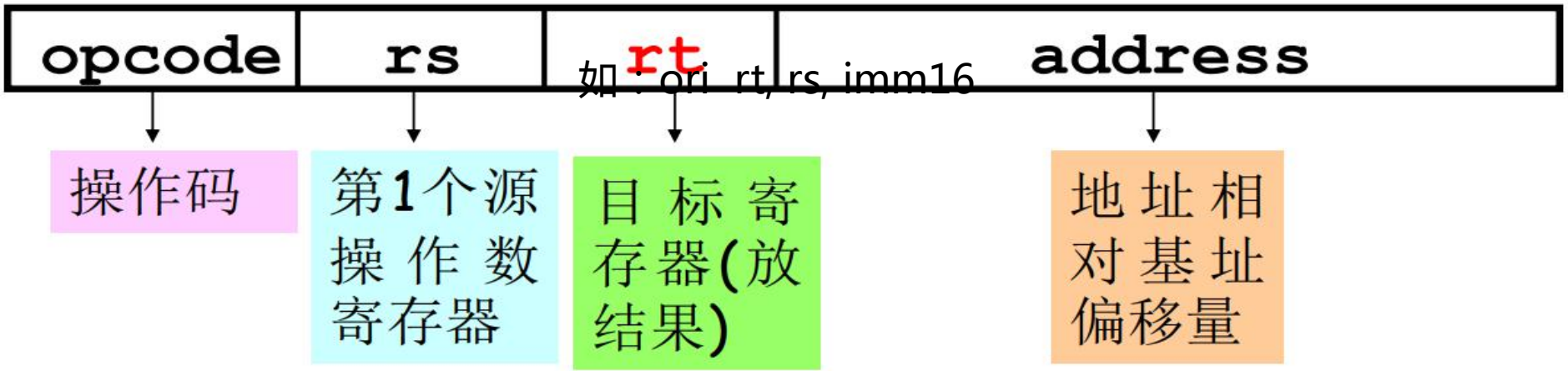


如运算指令：sub rd, rs, rt

1. MIPS指令格式

I型指令



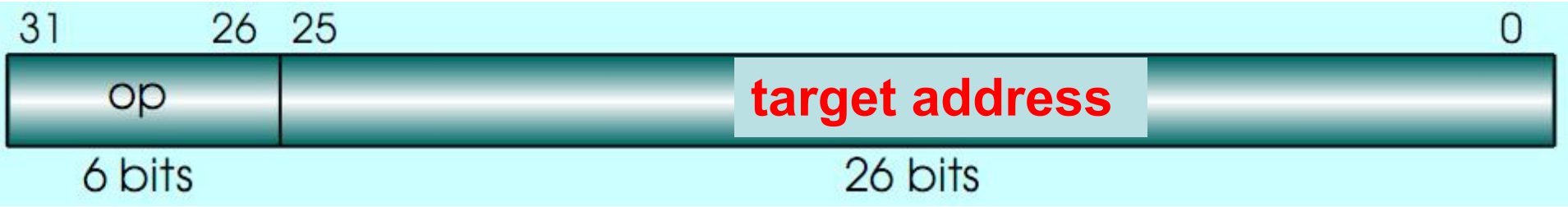


- 运算指令：如：ori rt, rs, imm16
- Load和Store指令。如：lw rt, rs, imm16
- 条件分支指令。如：beq rs, rt, imm16

1. MIPS指令格式

J型指令

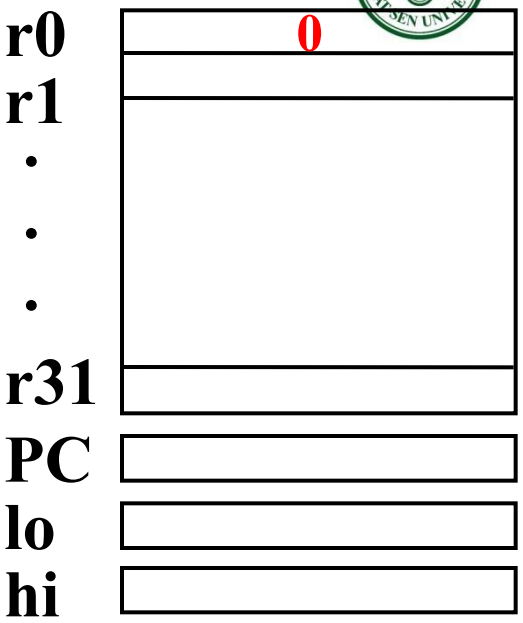
新的PC = { PC[31..28], target address,



- 无条件跳转指令。如：j target

➢2. 指令中寄存器数据和存储器数据的指定

- 寄存器数据指定
  - 31×32-bit GPRs (r0 = 0)
  - 寄存器编号占 5 bits
  - 32×32-bit FP Regs (f0 - f31, paired DP)
  - HI、LO、PC：特殊寄存器
  - 寄存器功能和2种汇编表示方式



➢MIPS寄存器功能定义和两种汇编表示

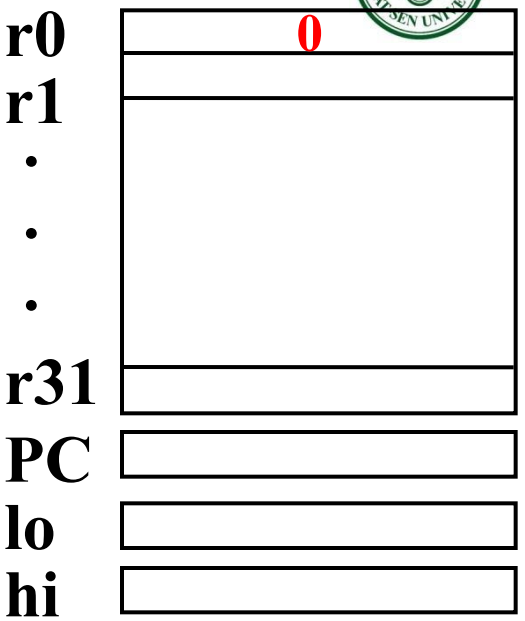
Name	number	Usage	Reserved on call?
zero	0	constant value =0(恒为0)	n.a.
at	1	reserved for assembler编程时避免用	n.a.
v0 – v1	2 – 3	values for results (过程调用返回值)	no
a0 – a3	4 – 7	Arguments (传参数寄存器)	yes
t0 – t7	8 – 15	Temporaries(临时变量)	no
s0 – s7	16 – 23	Saved(被调用者保存寄存器)	yes
t8 – t9	24 – 25	more temporaries(其他临时变量)	no
k0 – k1	26 – 27	reserved for kernel(为OS保留)	n.a.
gp	28	global pointer(全局指针)	yes
sp	29	stack pointer (栈指针)	yes
fp	30	frame pointer (帧指针)	yes
ra	31	return address (过程调用返回地址)	yes

Registers:用数字表示—$0…$31或 用名称表示 —$t0,$s1…$ra

zero	at	v0-v1	a0 - a3	t0 - t7	s0 - s7	t8 - t9	k0 - k1	gp	sp	fp	ra
0	1	2 - 3	4 - 7	8 …… 15	16 …… 23	24 - 25	26 - 27	28	29	30	31

➢2. MIPS指令中寄存器数据和存储器数据的指定

- 寄存器数据指定
  - 31×32-bit GPRs (r0 = 0)
  - 寄存器编号占 5 bits
  - 32×32-bit FP Regs (f0 - f31, paired DP)
  - HI、LO、PC：特殊寄存器
  - 寄存器功能和2种汇编表示方式



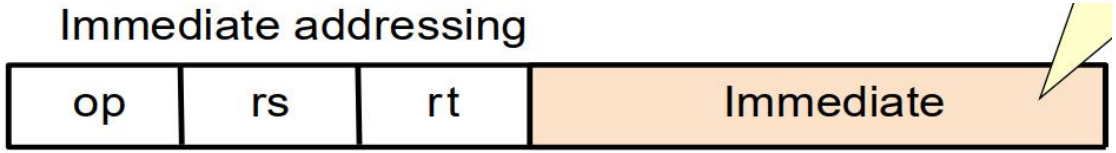
- 存储器数据指定
  - 32-bit machine → 可访问空间： 2^{32} bytes=4GB
  - Big Endian(大端方式)
  - 只能通过Load/Store指令访问存储器数据
  - 访存地址通过一个32位寄存器内容加16位偏移量得到
  - 16位偏移量是有符号整数(补码表示)
  - 数据要求按边界对齐

3. MIPS寻址方式

- 寄存器寻址
- 立即数寻址
- 基址或偏移寻址
- PC相对寻址
- 伪直接寻址

3. MIPS寻址方式

- 寄存器寻址：
  - **MIPS**算术运算指令的操作数必须从**32**个**32**位寄存器中选取
- 立即数寻址



  - 怎样把一个**32**位长的常数装入寄存器**$ s0**中？

0000 0000 0011 1101 0000 1001 0000 0000

61 **2304**

Load Upper Immediate

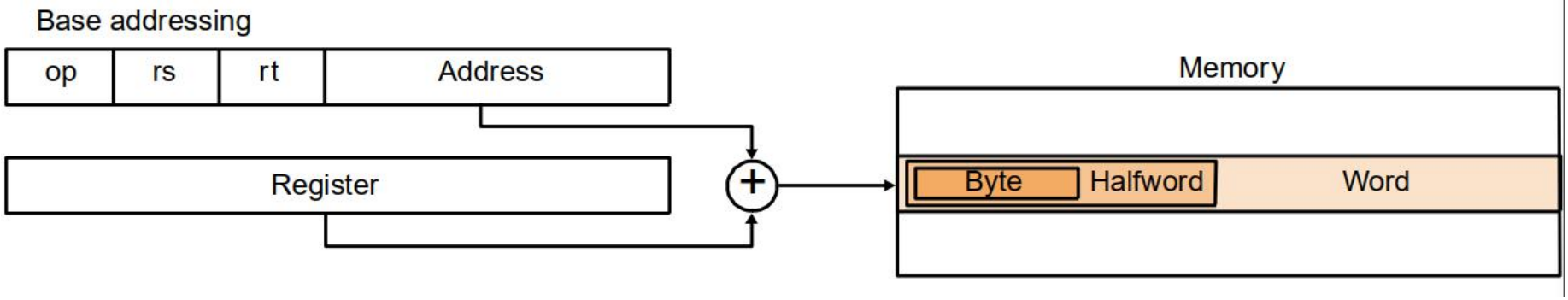
```
Lui  $ s0, 61      # $ s0= 0000 0000 0011 1101 0000 0000 0000 0000
addi $ s0, $s0, 2304  # $ s0= 0000 0000 0011 1101 0000 1001 0000 0000
```

寄存器以$符号开始；注释以#符号开始

3. MIPS寻址方式

- **基址或偏移寻址**
  - 操作数在存储器中，且存储器地址是某寄存器与指令中某常量的和
  - **c(rx)：** 立即数**c** + 寄存器**rx**中的值

Lw $ t0, 8 ($ s0)　　#$ s0中装的是存储器中的地址

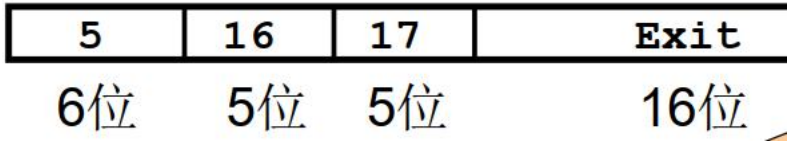


3. MIPS寻址方式

- PC相对寻址

•条件分支指令

bne $ s0, $ s1, Exit



#如果$ s0不等于$s1，则跳转到Exit

程序计数器 = PC寄存器的值 + 分支地址

- 伪直接寻址

•J型指令

跳转地址= { PC[31..28], 指令中的26位, 00 }

4. MIPS指令类型

算术运算

add, sub, addi, subi

逻辑运算

and, or, xor, andi, ori, xori, sll, srl, sra, lui

数据传送

lw, sw

条件分支

beq, bne

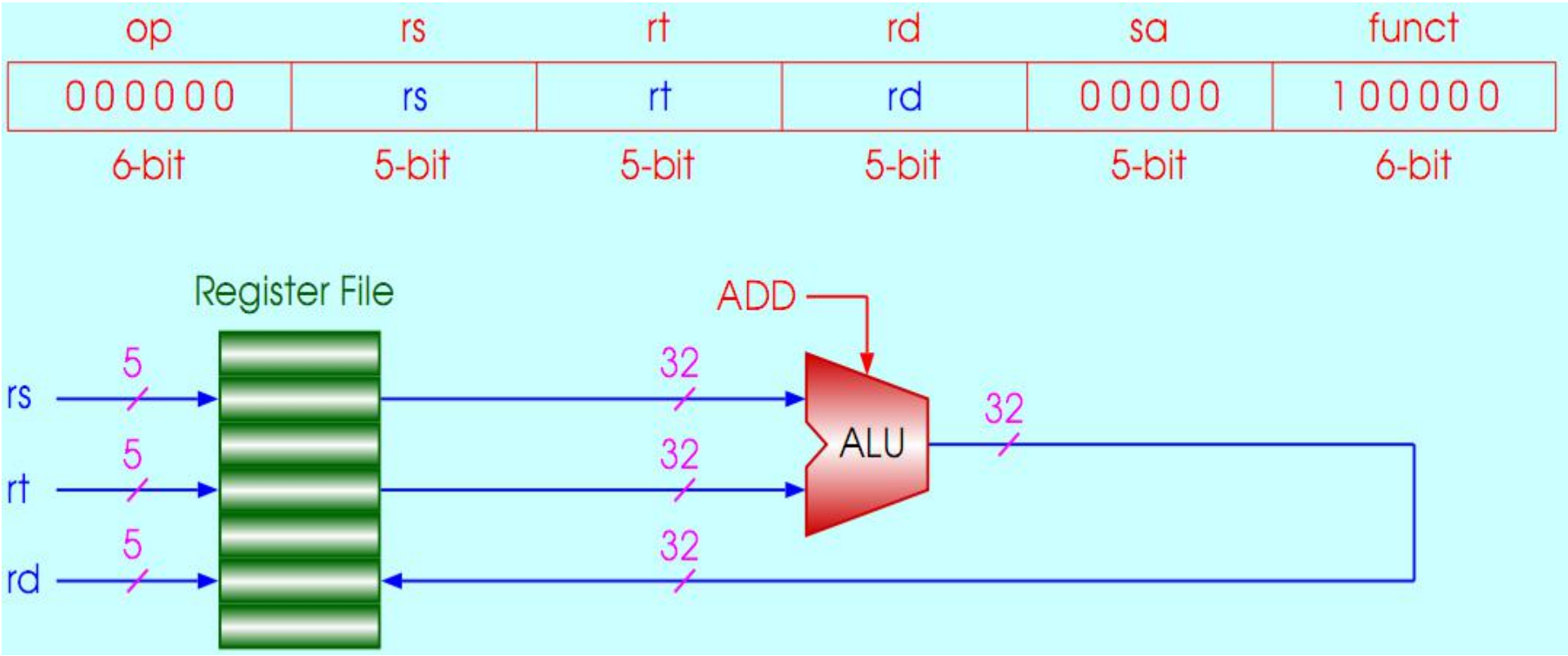
无条件跳转

j, jr, jal

```
op     rs    rt    rd    sa    func   ; R format instructions
000000 rs    rt    rd    00000 100000 ; add  rd, rs, rt
000000 rs    rt    rd    00000 100010 ; sub  rd, rs, rt
000000 rs    rt    rd    00000 100100 ; and  rd, rs, rt
000000 rs    rt    rd    00000 100101 ; or   rd, rs, rt
000000 rs    rt    rd    00000 100110 ; xor  rd, rs, rt
000000 00000 rt    rd    sa    000000 ; sll  rd, rt, sa
000000 00000 rt    rd    sa    000010 ; srl  rd, rt, sa
000000 00000 rt    rd    sa    000011 ; sra  rd, st, sa
000000 rs    00000 00000 00000 001000 ; jr   rs
op     rs    rt    imm                ; I format instructions
001000 rs    rt    imm                ; addi rt, rs, imm
001100 rs    rt    imm                ; andi rt, rs, imm
001101 rs    rt    imm                ; ori  rt, rs, imm
001110 rs    rt    imm                ; xori rt, rs, imm
100011 rs    rt    imm                ; lw   rt, imm(rs)
101011 rs    rt    imm                ; sw   rt, imm(rs)
000100 rs    rt    imm                ; beq  rs, rt, imm
000101 rs    rt    imm                ; bne  rs, rt, imm
001111 00000 rt    imm                ; lui  rt, imm
op     addr                           ; J format instructions
000010 addr                           ; j    addr
000011 addr                           ; jal  addr
```

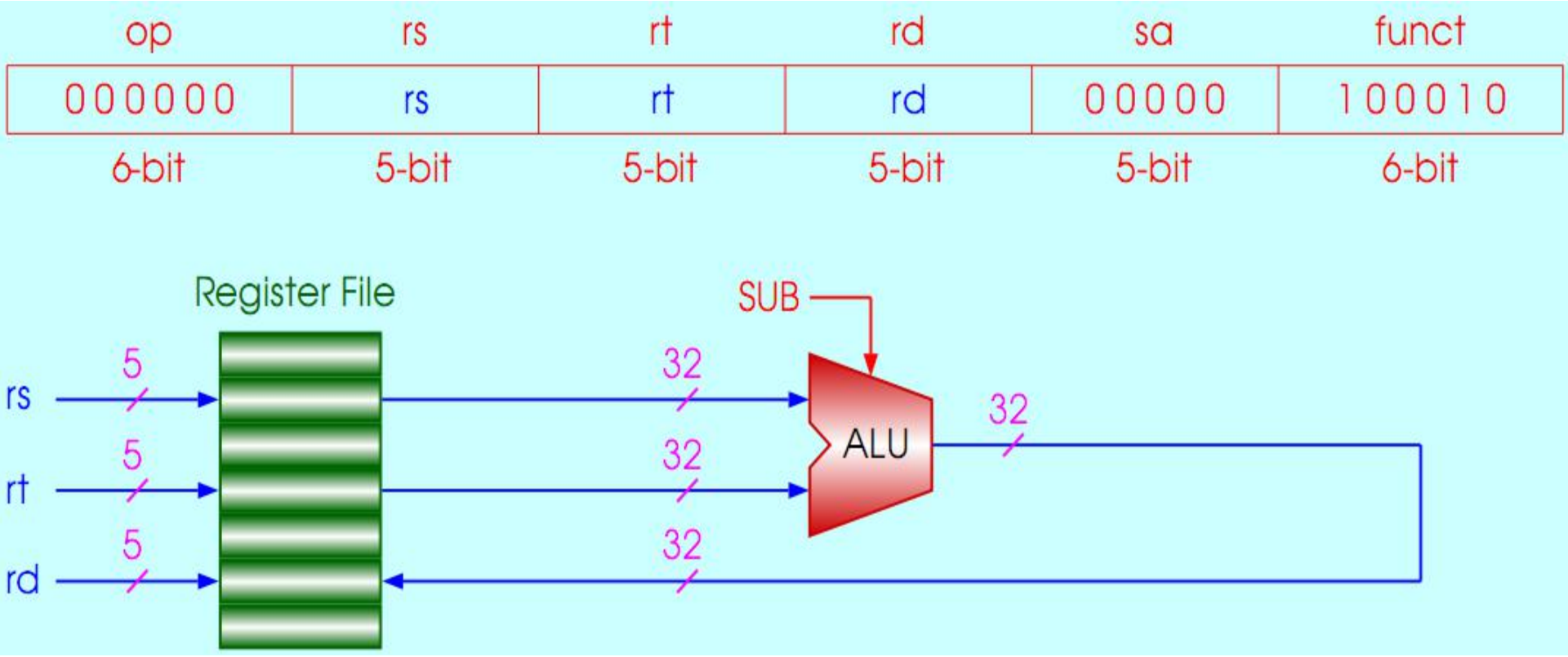
add

- **add rd,rs,rt # rd =rs + rt**



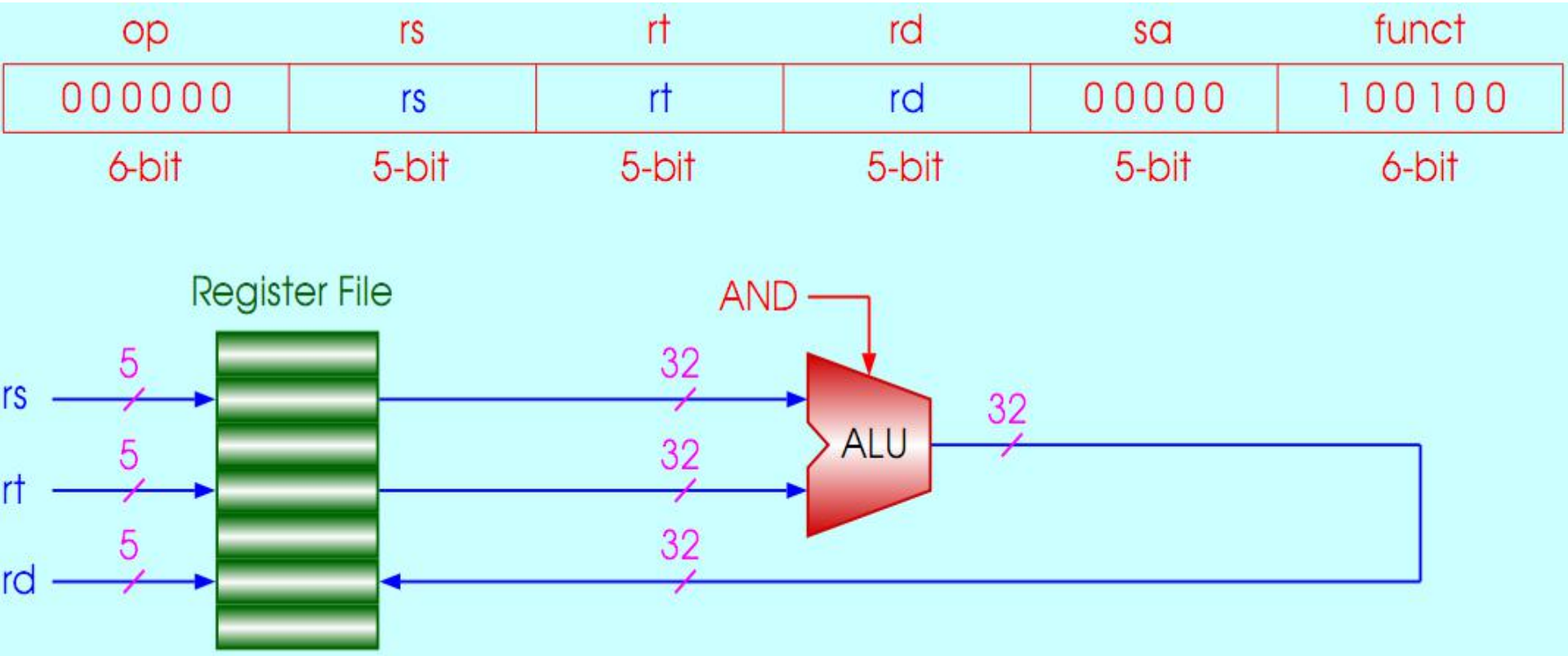
sub

- **sub rd,rs,rt # rd = rs – rt**



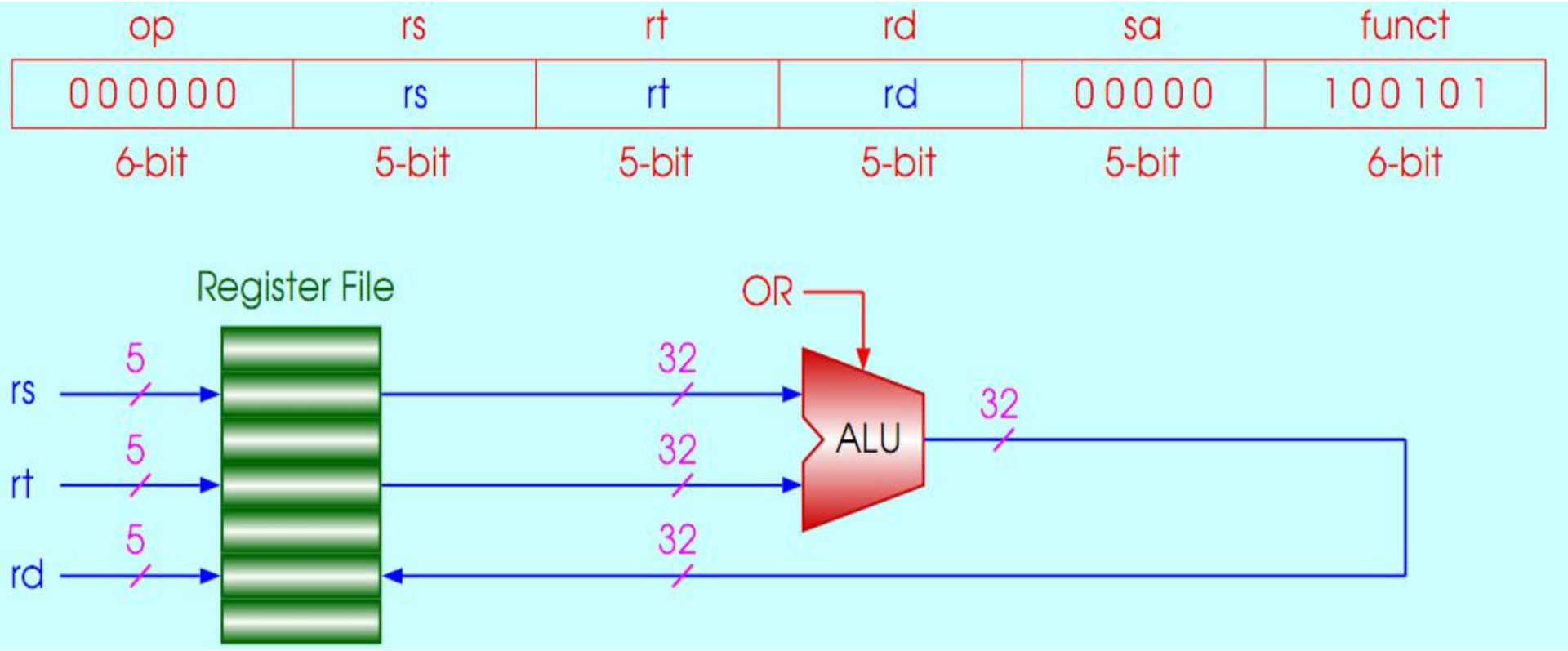
and

- **and rd,rs,rt # rd = rs & rt**



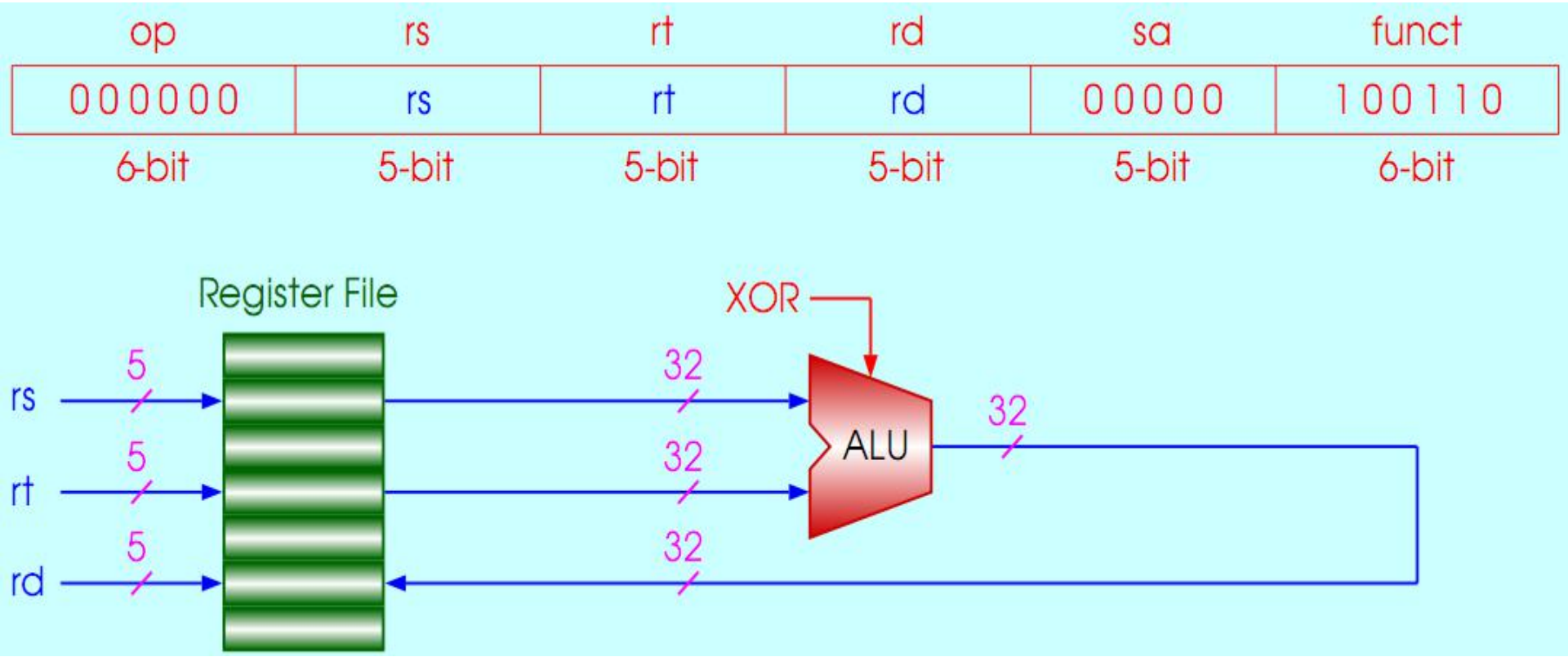
or

- **or rd,rs,rt # rd = rs | rt**



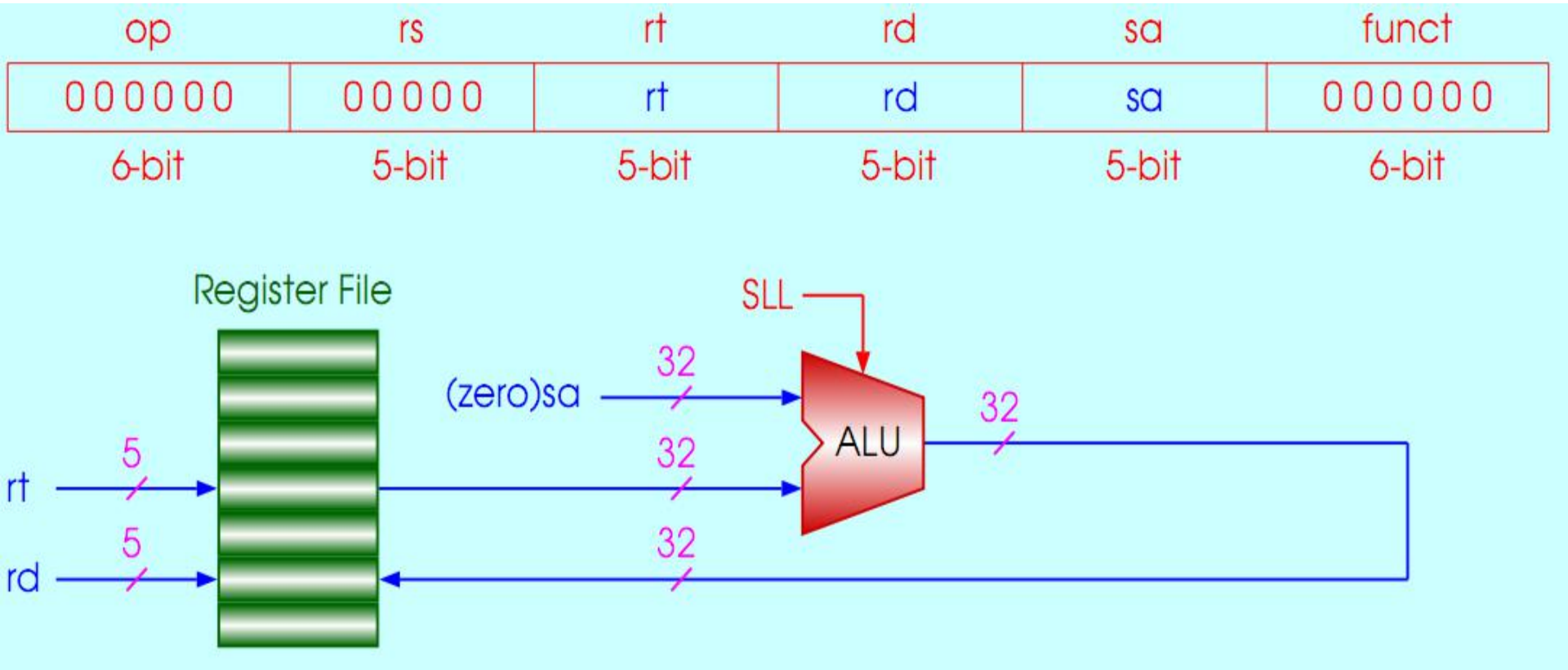
xor

- **xor rd,rs,rt # rd = rs ^ rt**



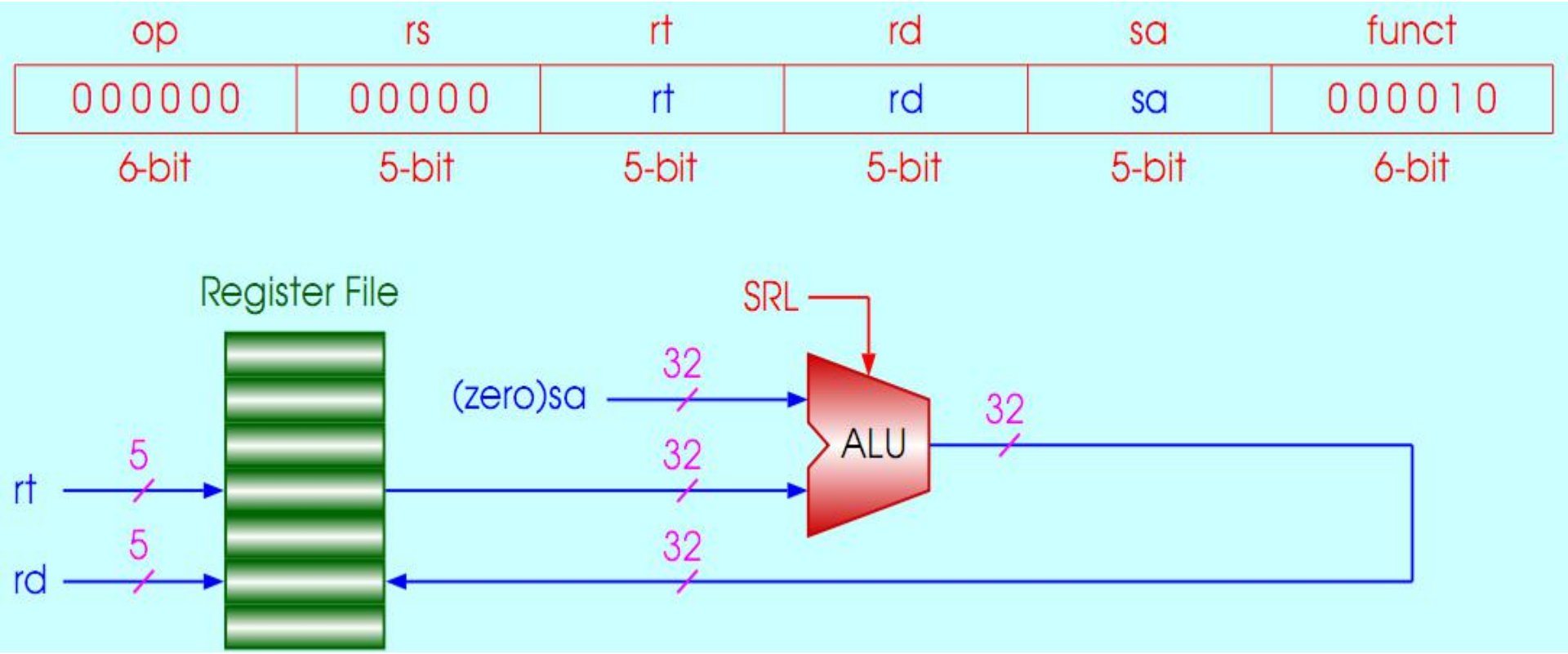
sll

- **sll rd,rt,sa # rd = rt << sa**



srl

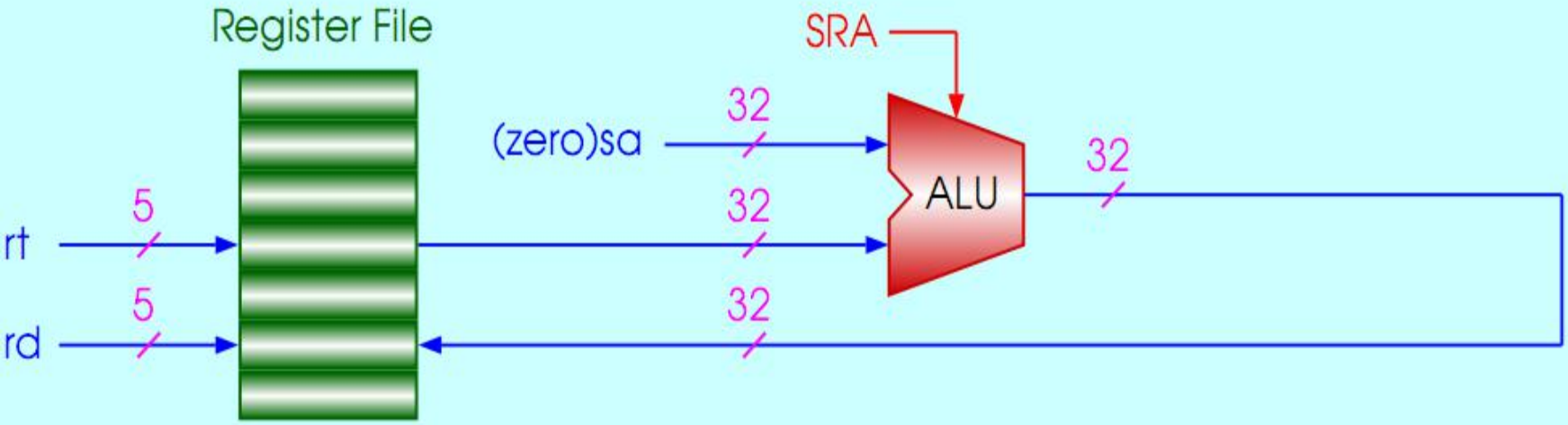
- **srl rd,rt,sa # rd = rt >> sa(逻辑)**



sra

- **sra rd,rt,sa # rd = rt >> sa(算术)**

op	rs	rt	rd	sa	funct
000000	00000	rt	rd	sa	000011
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit



32位移位指令

移位量在某个寄存器中

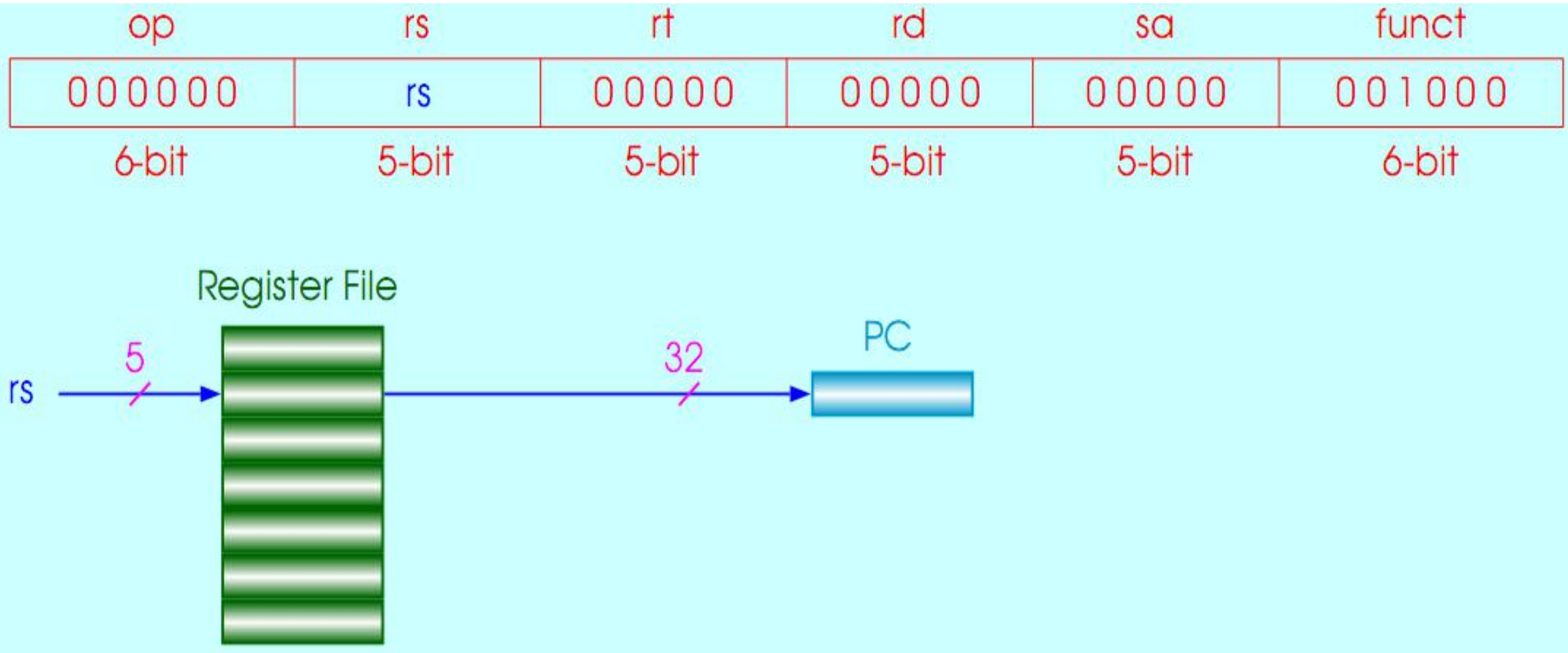
```
sllv $t0, $t1, $t3          # $t0 = $t1 << ($t3 % 32)

srlv $t0, $t1, $t3          # $t0 = $t1 >> ($t3 % 32)

srav $t0, $t1, $t3          # $t0 = $t1 >> ($t3 % 32)
```

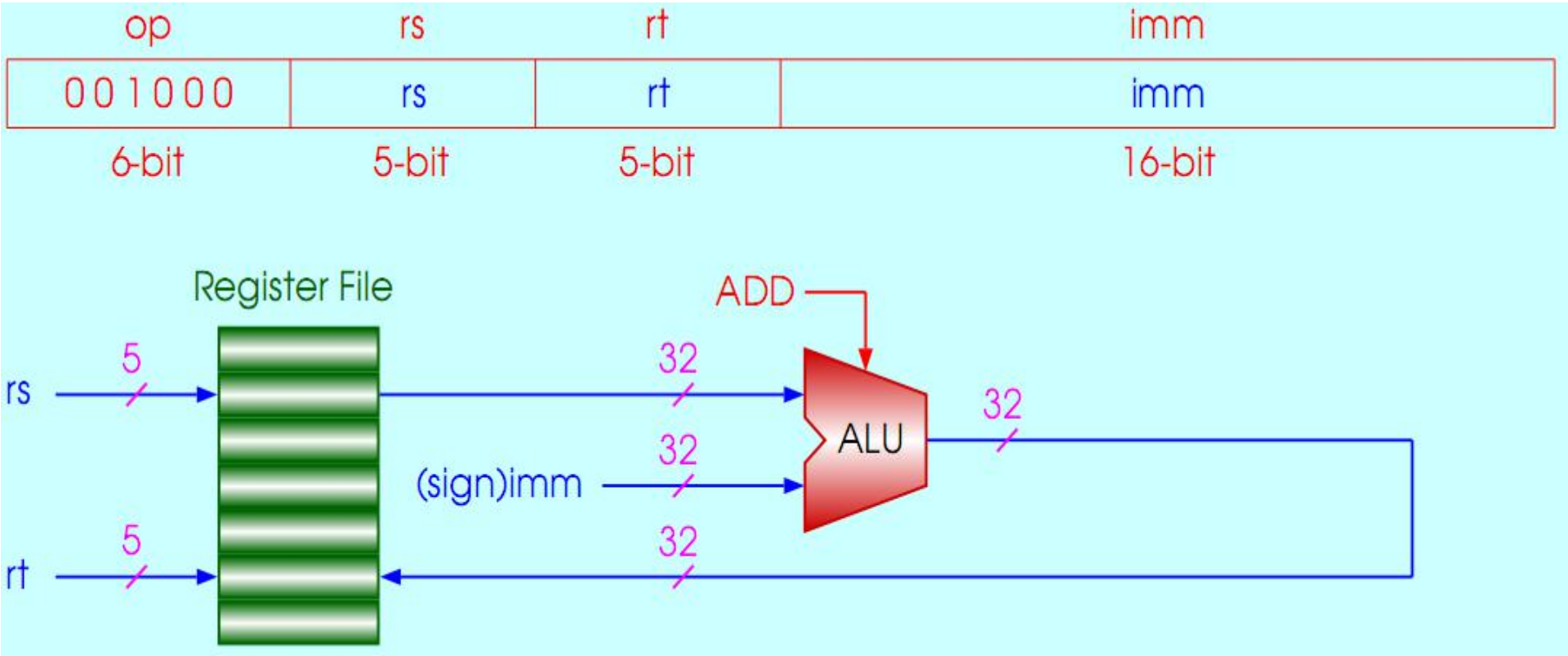
jr

- **jr rs # PC = rs**


op
rs
rt
rd
sa
funct
000000
rs
00000
00000
00000
001000
6-bit
5-bit
5-bit
5-bit
5-bit
6-bit
Register File
rs
5
32
PC

addi

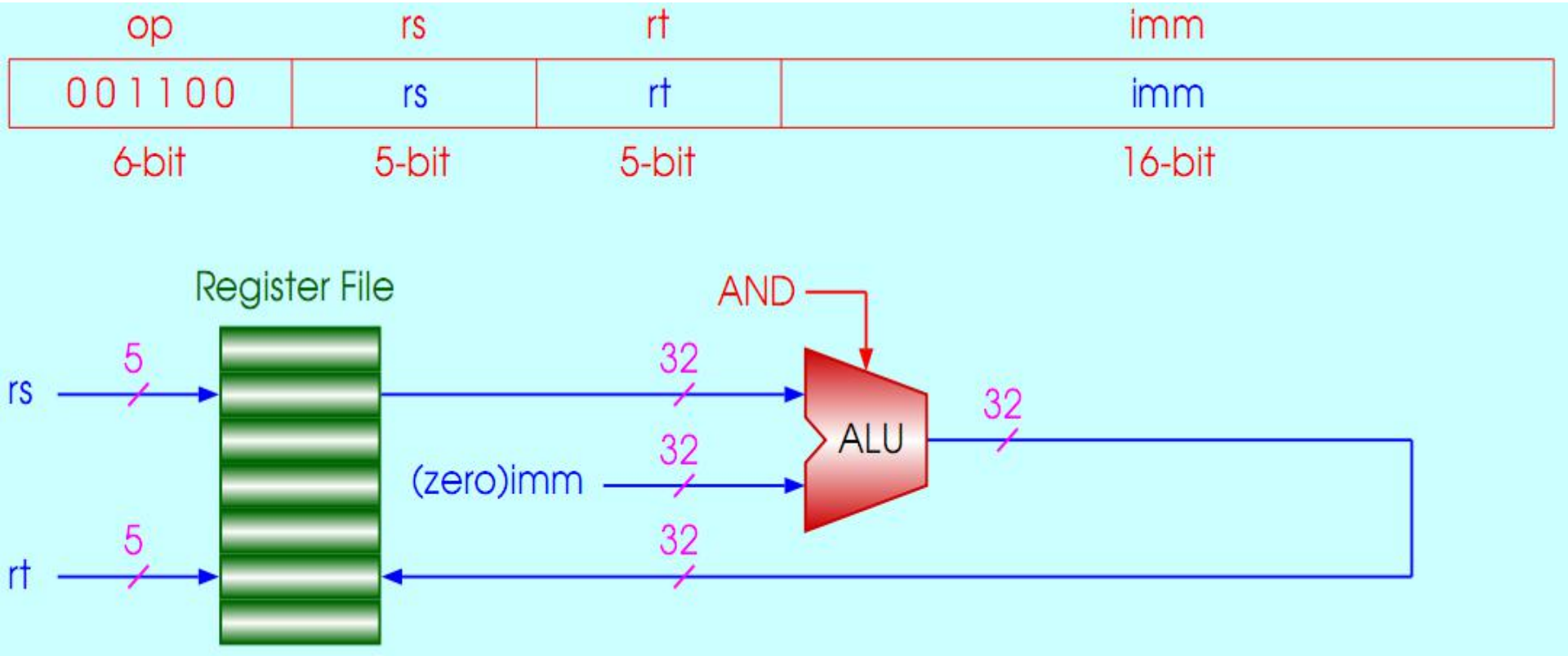
- **addi rt,rs,imm # rt = rs + (sign)imm**



算术指令总是将立即数做符号位扩展

andi

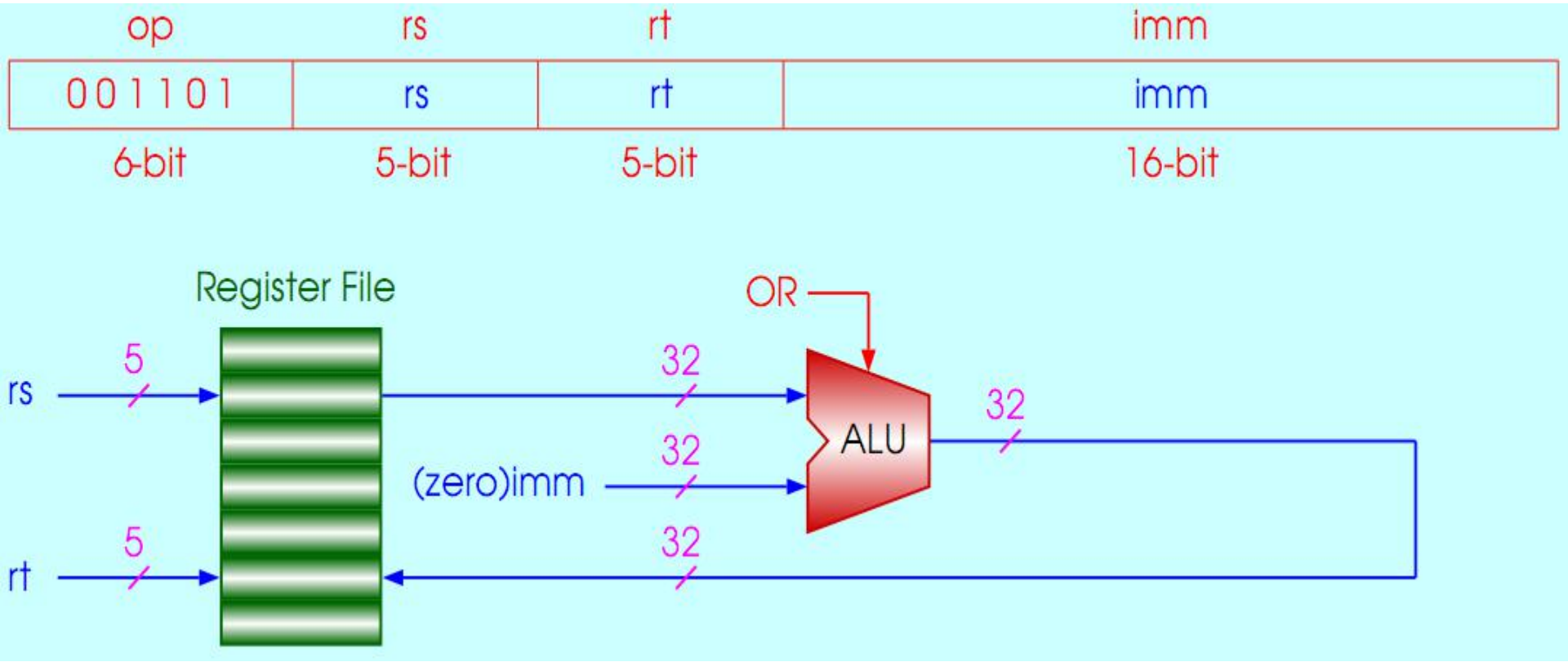
- **andi rt,rs,imm # rt = rs & (zero)imm**



逻辑指令（andi, ori通常处理无符号数）做零扩展

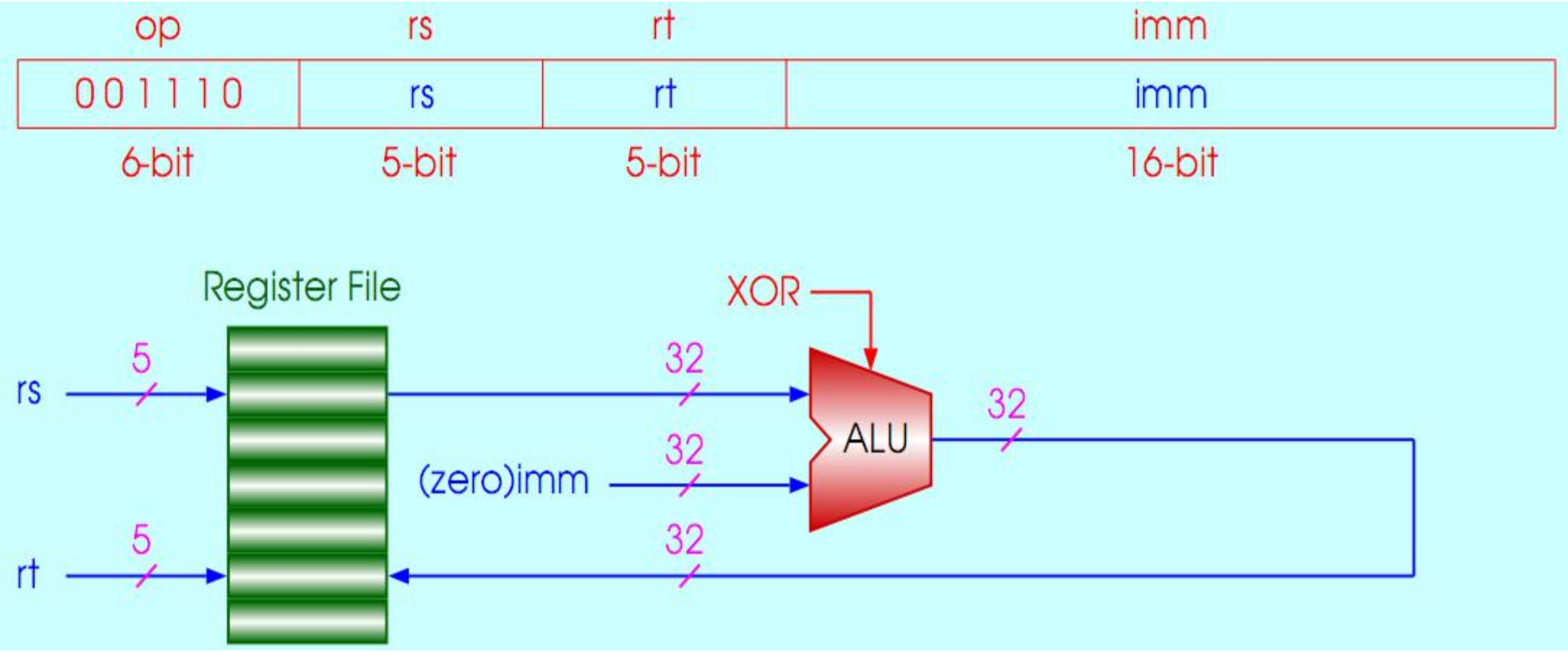
ori

- **ori rt,rs,imm # rt = rs | (zero)imm**



xori

- **xor i rt,rs,imm #rt = rs ^ (zero)imm**



比较指令slt, sltu, slti, sltui

- 比较两个寄存器的内容，并根据比较结果设置第三个寄存器

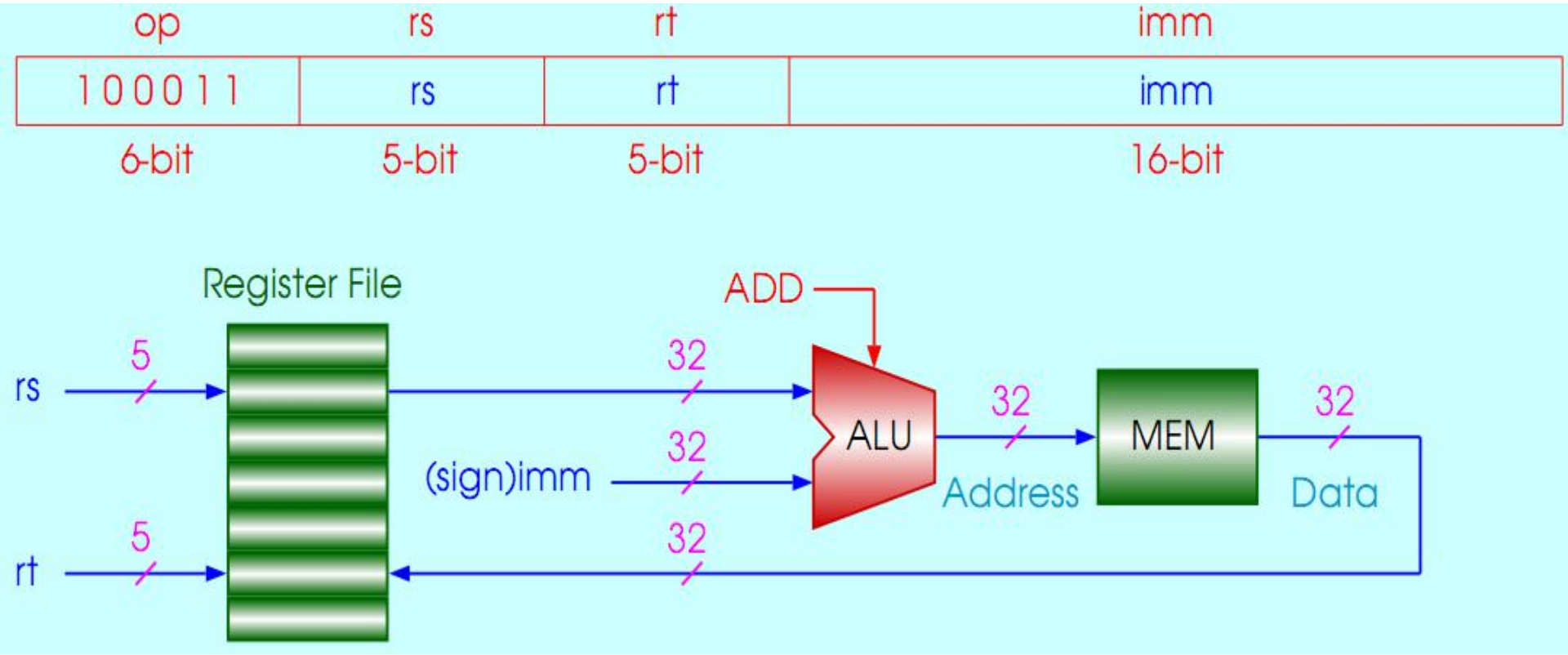
```
slt $t1,$t2,$t3        # if ($t2 < $t3) $t1=1;
                       # else $t1=0
sltu $t1,$t2,$t3       # 无符号比较
```

- 寄存器与立即数比较

```
slti $t1,$t2,10        # 与立即数比较
sltui $t1,$t2,10       # 与无符号立即数比较
```

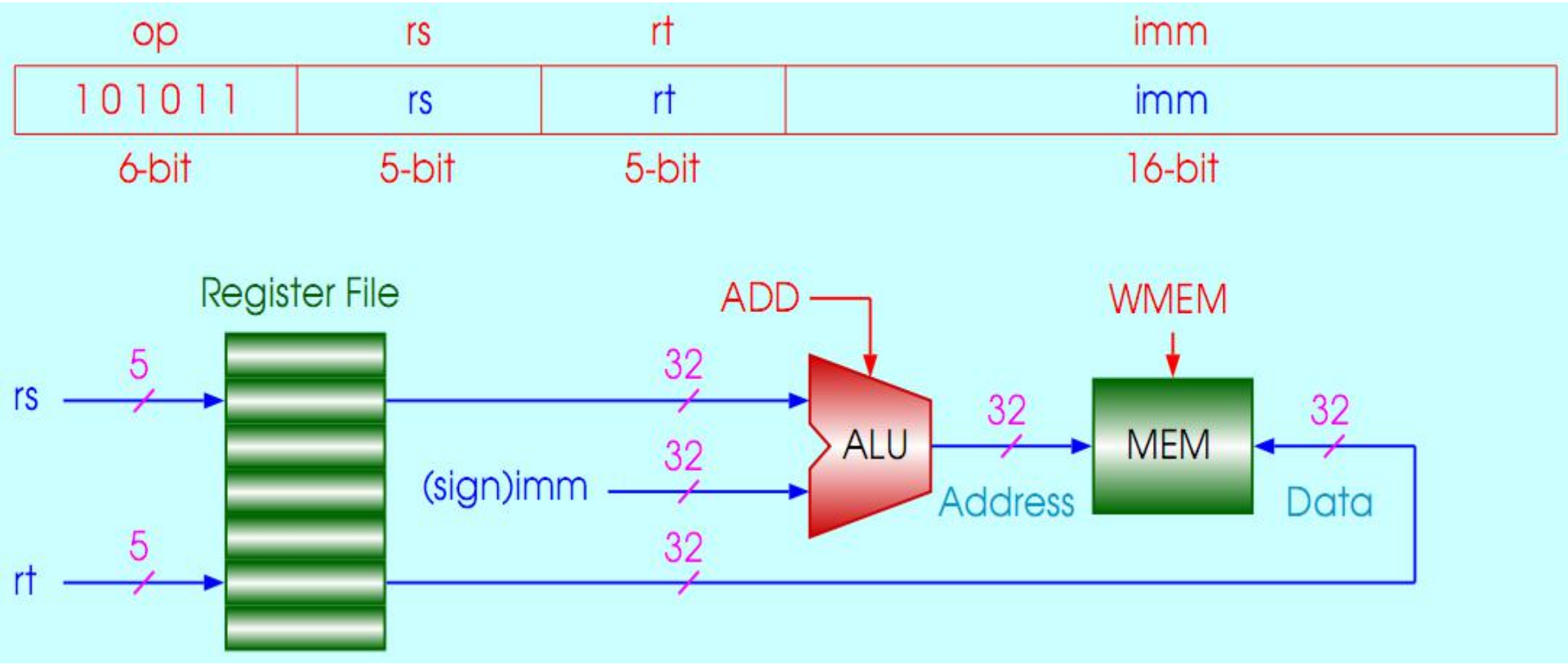
lw

- **lw rt, imm(rs) # rt = memory[rs + (sign)imm]**



MIPS只支持基址(变址)+位移量的内存寻址方式

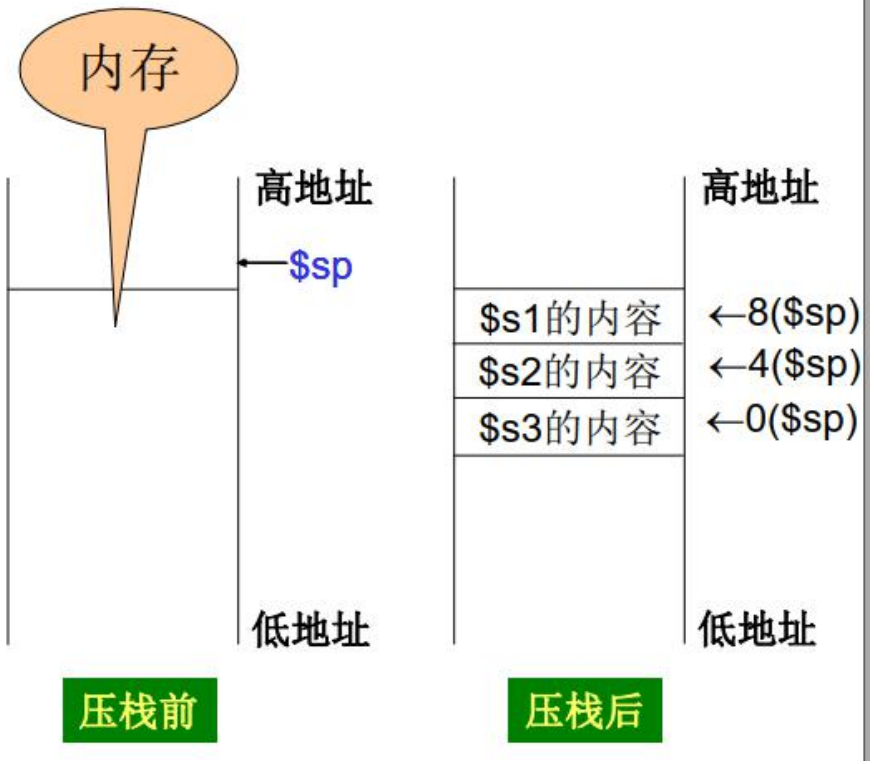
sw rt, imm(rs) # memory[rs + (sign)imm] = rt



栈操作

- MIPS没有单独的栈操作指令,所有对栈的操作都是统一的内存访问方式
- MIPS有一个$sp寄存器可以用做栈指针
-
  - 例，将$s1、 $s2、$s3寄存器的内容压入栈

```
addi $sp, $sp, -12
sw   $s1, 8($sp)
sw   $s2, 4($sp)
sw   $s3, 0($sp)
```

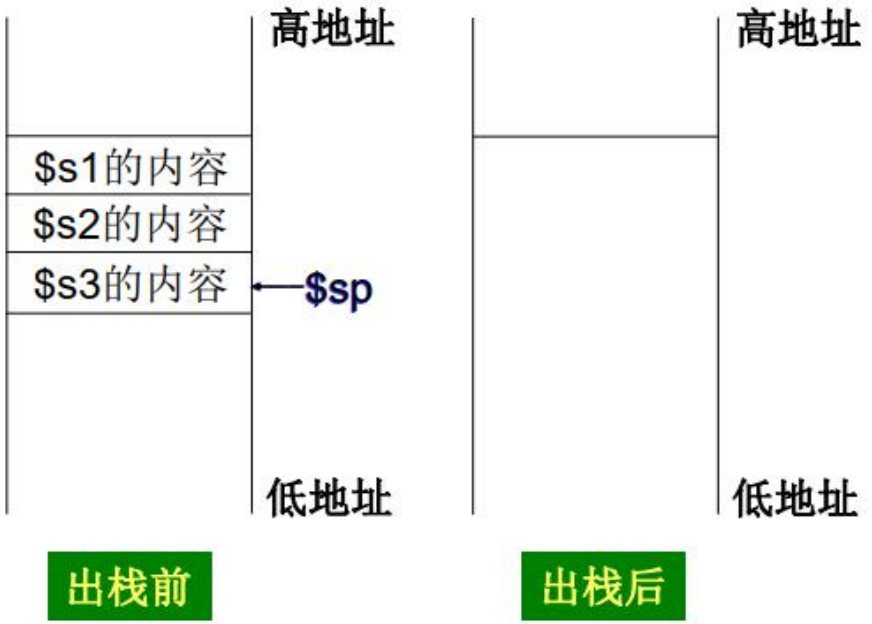


习惯上，栈按照从高到到低的地址顺序增长

栈操作

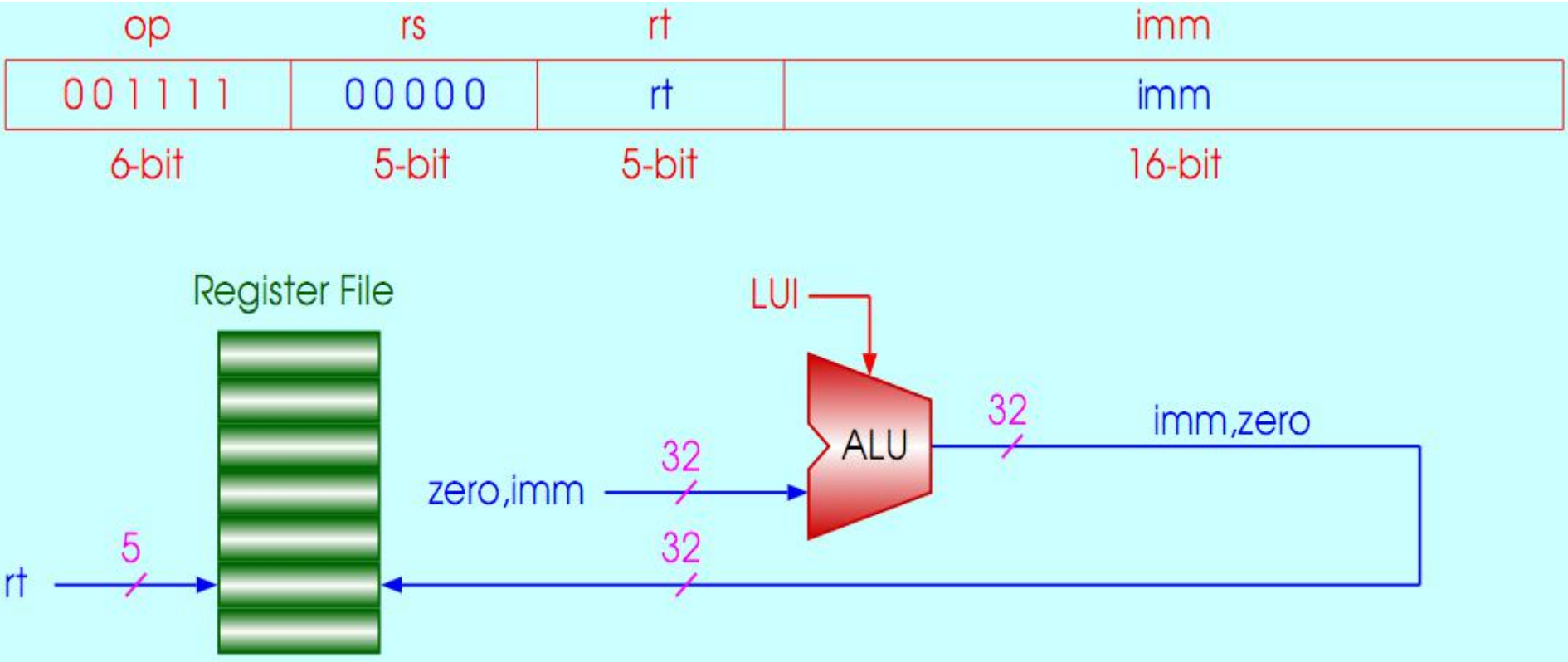
- MIPS没有单独的栈操作指令,所有对栈的操作都是统一的内存访问方式
- MIPS有一个$sp寄存器可以用做栈指针
- 出栈操作

```
lw    $s1, 8($sp)
lw    $s2, 4($sp)
lw    $s3, 0($sp)
addi $sp, $sp, 12
```



lui

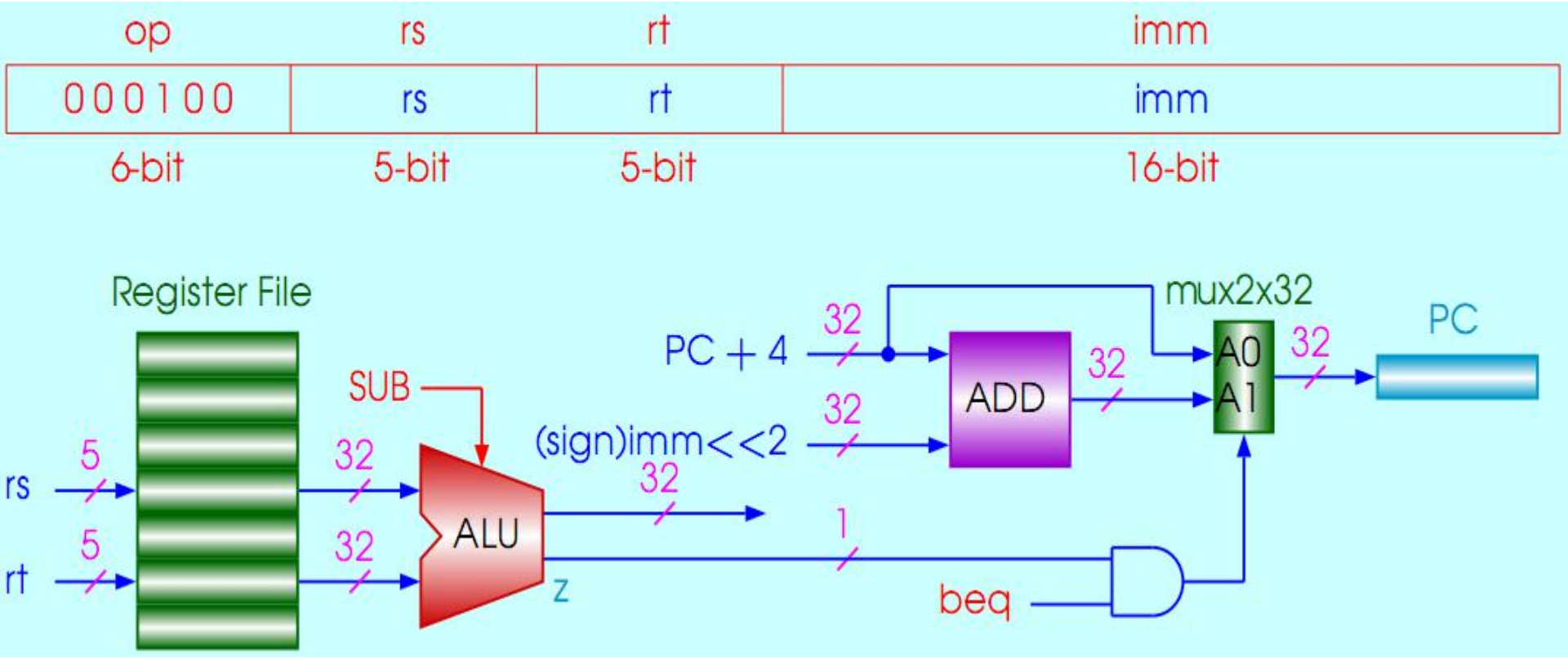
- 装入高位立即数*load upper immediate*
- **lui rt, imm # rt = imm<<16**



beq

- **beq rs,rt,imm**

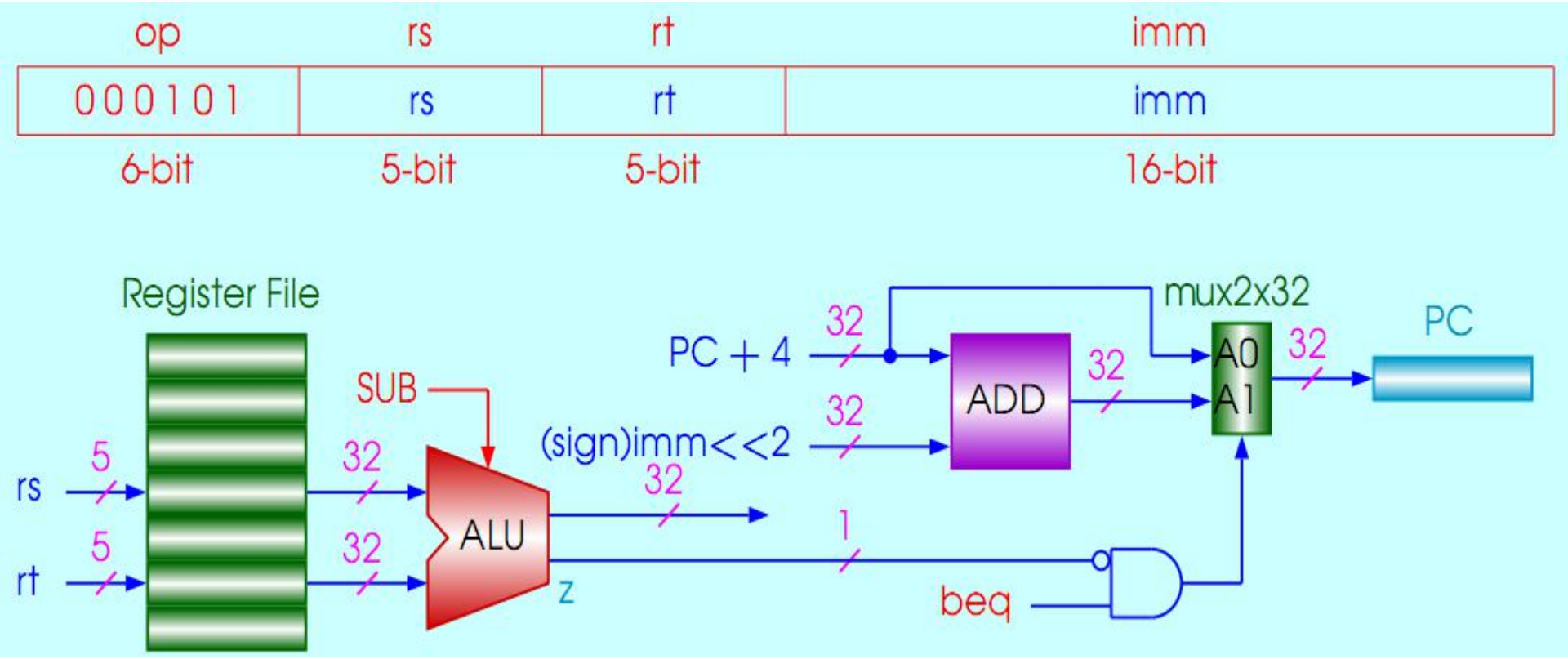
  # if(rs==rt) PC=PC+4+(sign)imm<<2



bne

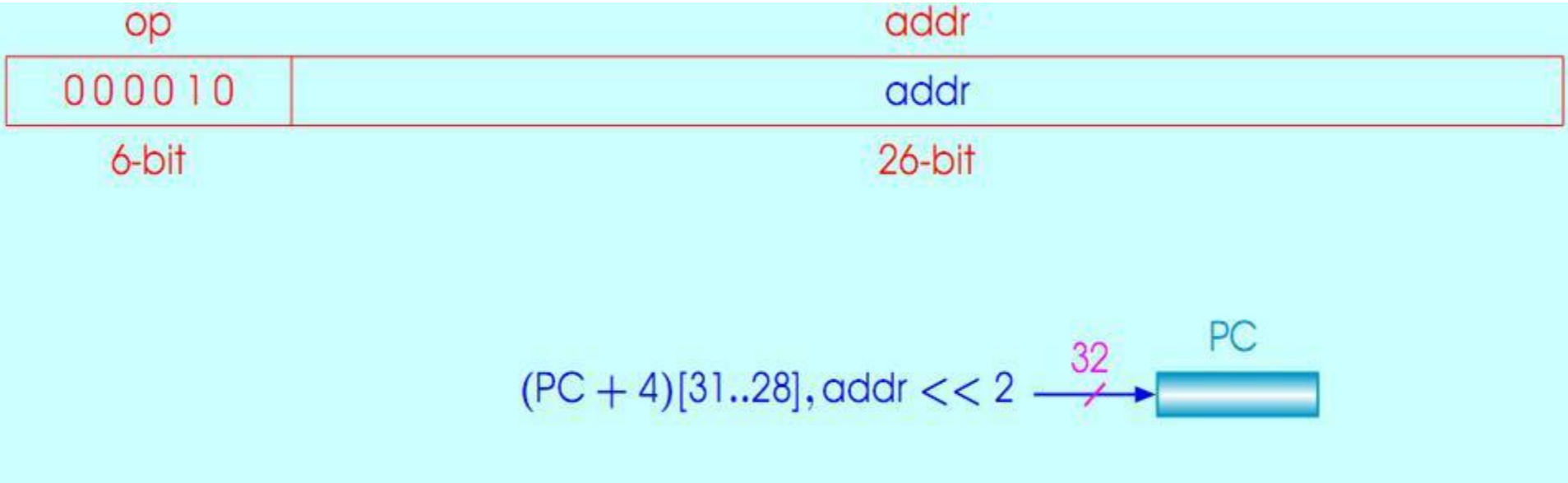
- **bne rs,rt,imm**

  # if(rs!=rt) PC=PC+4+(sign)imm<<2



j

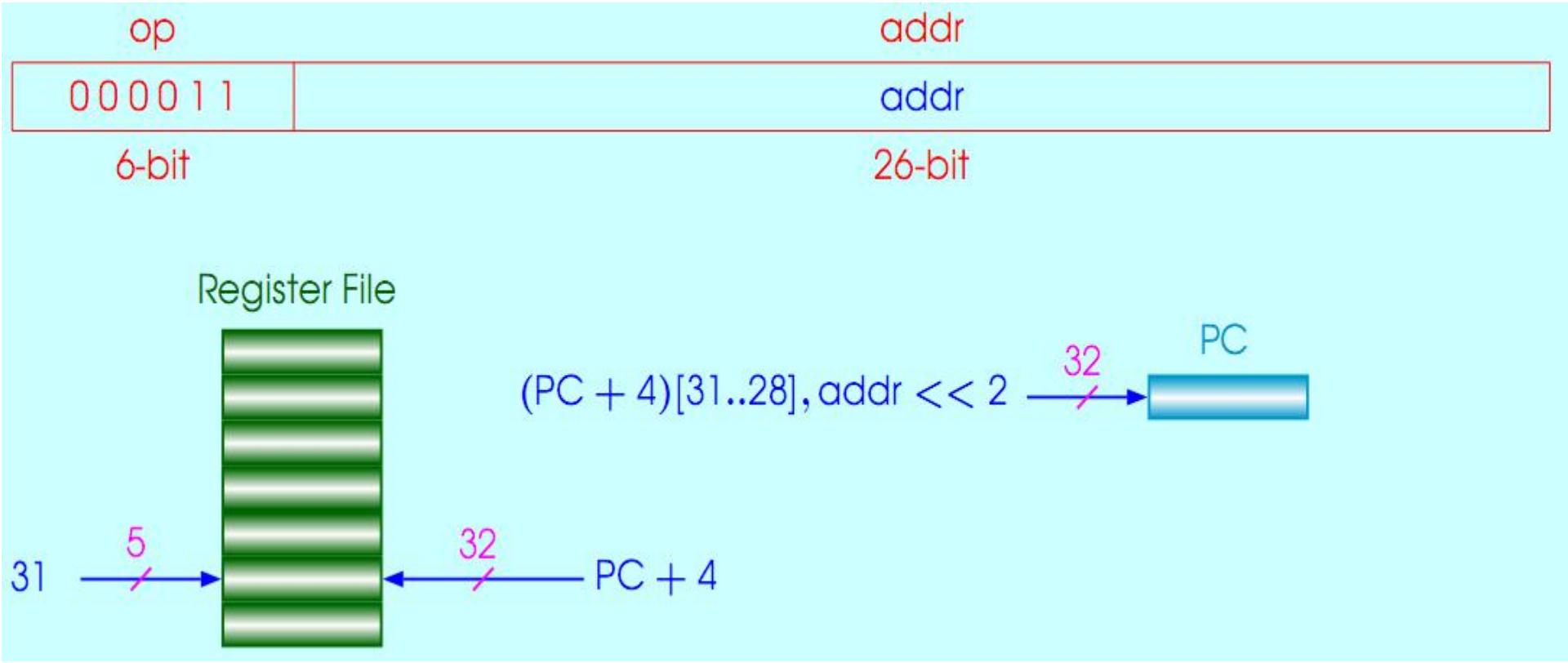
- **j addr # PC = (PC+4)[31..28] , addr<<2**



jal

- **jal addr**

  # $31 = PC+4; PC=(PC+4)[31..28] , addr<<2



符号扩展和零扩展

- **addi rt,rs,imm # rt = rs + (sign)imm**

  寄存器rs中的数据与2的补码表示的立即数imm相加，结果存放在寄存器rt中

- **andi rt,rs,imm # rt = rs & (zero)imm**

  寄存器rs中的数据与无符号立即数imm相与，结果存放在寄存器rt中

伪指令

- 伪指令（pseudo instructions） 是为了编程方便而对指令集进行的扩展，汇编器负责将伪指令翻译成mips指令

MIPS 伪指令	功能描述	实现前伪指令功能所需相应的 MIPS 指令及方法	功能描述
move $t2,$t4	$t2<-$t4	addu $t2,$zero,$t4	$t2<-$zero + $t4
b imm16	无条件转移，pc<-pc+4+sign-extend(imm16)	beq $zero,$zero,imm16	$zero=$zero，pc<-pc+4+sign-extend(imm16)
li $t1,10	$t1<-10	ori $t1,$zero,10	$t1<-$zero | 10
la $t2,0x1000056c	$t2<-0x1000056c	lui $at,0x1000	0x1000=4096，0x10000=65536 $at <- 4096*65536 ；将 16 位立即数放到目标寄存器($at)高 16 位，目标寄存器的低 16 位填 0
		ori $t2,$at,0x056c	$t2<-$at | 0x056c，$t2<-重新合并该数

过程调用

- 过程调用只允许传递参数(输入值)和返回结果，MIPS过程调用遵循如下约定：
  - MIPS指令系统为过程调用分配了7个32位寄存器
  - $a0~$a3：四个参数寄存器，用于传递参数
  - $v0~$v1：两个储值寄存器，用于保存返回值
  - $ra：一个返回地址寄存器，用于保存返回地址
- MIPS汇编语言包含一个跳转与链接指令jal：跳转到某个地址的同时将下一条指令地址保存在$ra中
  - 跳转—链接指令：jal Procedure Address
  - 子程序返回通过寄存器跳转指令jr进行

```
jr   $ra    #挑转到寄存器$ra指令的地址
```

5. MIPS汇编语言程序结构框架

```
#数据声明+普通文本+程序编码
# Template.s
#Bare-bones outline of MIPS assembly language program

.data     # variable declarations follow this line
              # 数据变量声明


# ...
.text       # instructions follow this line  代码段部分
main:    # indicates start of code 程序入口
              # 主程序
              # ...
# End of program
```

一个简单MIPS汇编程序

```
.text                  # 代码段 声明
.globl  main           # globl指明程序的入口地址main ，为全局名
main:                  # 主程序名：main
    la   $a0, str          # 取字符串地址
    li   $v0, 4            # 4号功能调用，输出字符串
    syscall              # 系统调用，输出字符串

    li   $v0, 10           # 退出
    syscall              # 系统调用，

.data                  # 数据段，变量定义部分
  str:                 # 字符串变量名称
        .asciiz  "hello world\n"     # 该字符串为 str 的初值 ， 以“00”为终止符结束
```

谢 谢！