计算机组成原理与系统结构

第六章 指令系统

http://www.icourses.cn/coursestatic/course_2859.html

二、MIPS32 指令格式、寻址方式和指

1. 指令格式：

- MIPS32 的每条指令长度固定是 32 位。

□□□□	□□□□□□ 32 □□						□□
	31-26	25-21	20-16	15-11	10-6	5-0	
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)	□□□□
I	opcode (6)	rs (5)	rt (5)	offset/immediate (16)			□□□□□□□□□□ □□□□
J	opcode (6)	address (26)					□□□□

2、MIPS32 指令格式、寻址方式和指令分

指指指	指指指指指指指					
	31..26	25..21	20..16	15..11	10..6	5..0
R	op	rs	rt	rd	shamt	func

- R(register)型指令，也叫 RR 型指令。该类型指令的源操作数和目的寄存器都是寄存器操作数。最高 6 位是操作码（opcode），第 25-21 位、第 20-16 位和第 15-11 位，连续三个 5 位二进制码来表示三个**寄存器的地址**，第 10-6 位表示**移位的位数**，如果未使用移位操作，则第 10-6 位置全 0，第 5-0 位为 6 位的**功能码**(function)，它与第 31-26 位的 opcode 码共同决定 R 型指令的具体操作方式。算术指令都是 R 型指令。

2、MIPS32 指令格式、寻址方式和指令

指指指	指指指指指指指			
	31..26	25..21	20..16	15..0
I 指指指	op	rs	rt	immediate

- I（immediate）型指令，是立即数型指令。该类型指令的一个源操作数是 16 位立即数，另两个操作数是寄存器操作数，因此也称为 R-I 型指令。最高 6 位是操作码（ opcode ），第 25-21 位和第 20-16 位，连续两个 5 位二进制码分别表示两个寄存器的地址，第 15-0 位是一个 16 位二进制码表示的立即数。指令执行时， 16 位立即数需要进行符号扩展或 0 扩展，变成 32 位操作数才能参与运算。数据传输、分支、立即数指令采用此类指令结构。

二、MIPS32 指令格式、寻址方式和指令

指指指	指指指指指指指	
	31..26	25..0
J	op	address

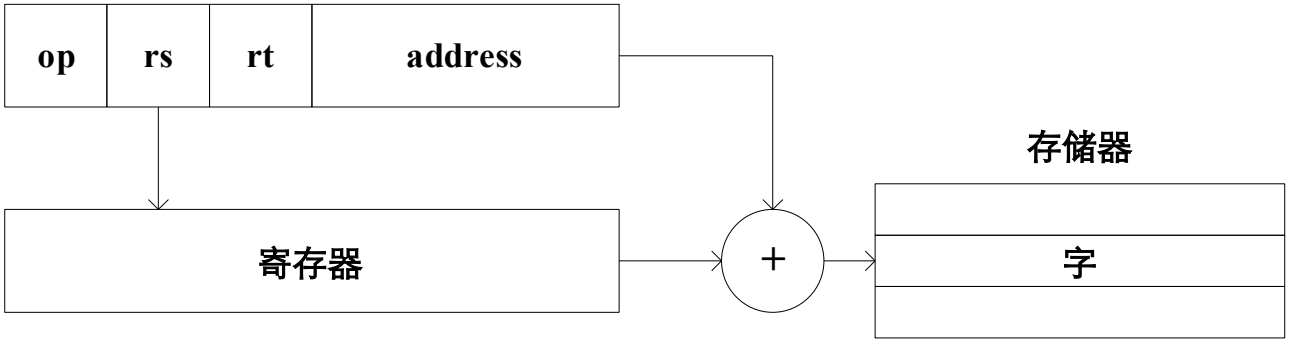
- J(jump)型指令，是无条件转移指令。该类型指令使用一个 26 位的直接地址（第 25-0 位）。因为 MIPS 采用 32 位定长指令，所以每条指令都占 4 个存储单元，指令地址总是 4 的倍数。于是，只要将当前 PC 值的高 4 位拼上 26 位直接地址，末尾两位置 0，即可得到 32 位的目标转移地址。

二、MIPS32 指令格式、寻址方式和指令

2. 寻址方式

①寄存器寻址，操作数是寄存器

②基址寻址或偏移量寻址，操作数在存储器中，存储器地址是指令中基址寄存器和常数之和



二、MIPS32 指令格式、寻址方式和指令

2. 寻址方式

①寄存器寻址，操作数是寄存器

```
or   $s1,$s2,$s3 # $s1=$s2 | $s3
```

①基址寻址（偏移量寻址或寄存器相对寻址），操作数在存储器中，存储器地址是指令中基址寄存器和常数之和

例如：`lw $t0,addr($t3)`

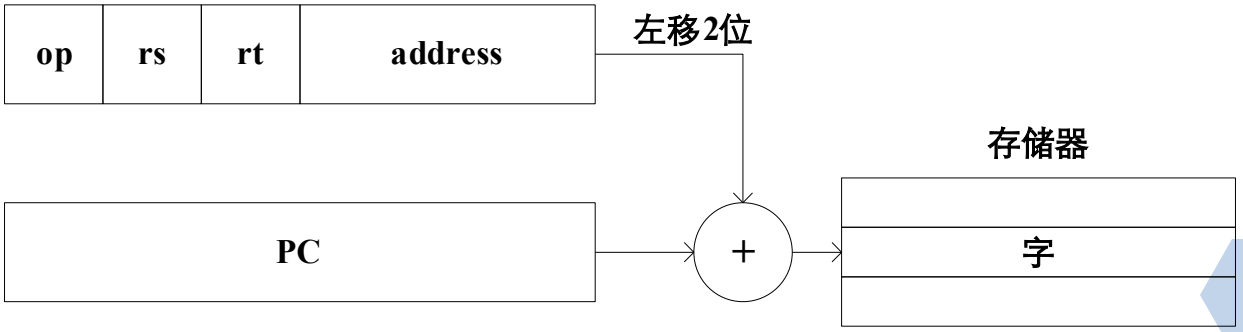
如果 addr=0，也可以写成 `lw $t0,($t3)`

二、MIPS32 指令格式、寻址方式和指令

2. 寻址方式

③立即数寻址，操作数是指令给出的常数

④相对寻址，目标地址是当前 PC 值和指令中的常数之和， 16 位的常数在与 PC 相加之前要左移 2 位（相当于乘以 4 ）



二、MIPS32 指令格式、寻址方式和指

2. 寻址方式

③立即数寻址，操作数是指令给出的常数

例： sra $s1,$s2,10 # $s2 算术右移 10 位 → $s1

addi $s1,$s2,100 # $s1=$s2+100

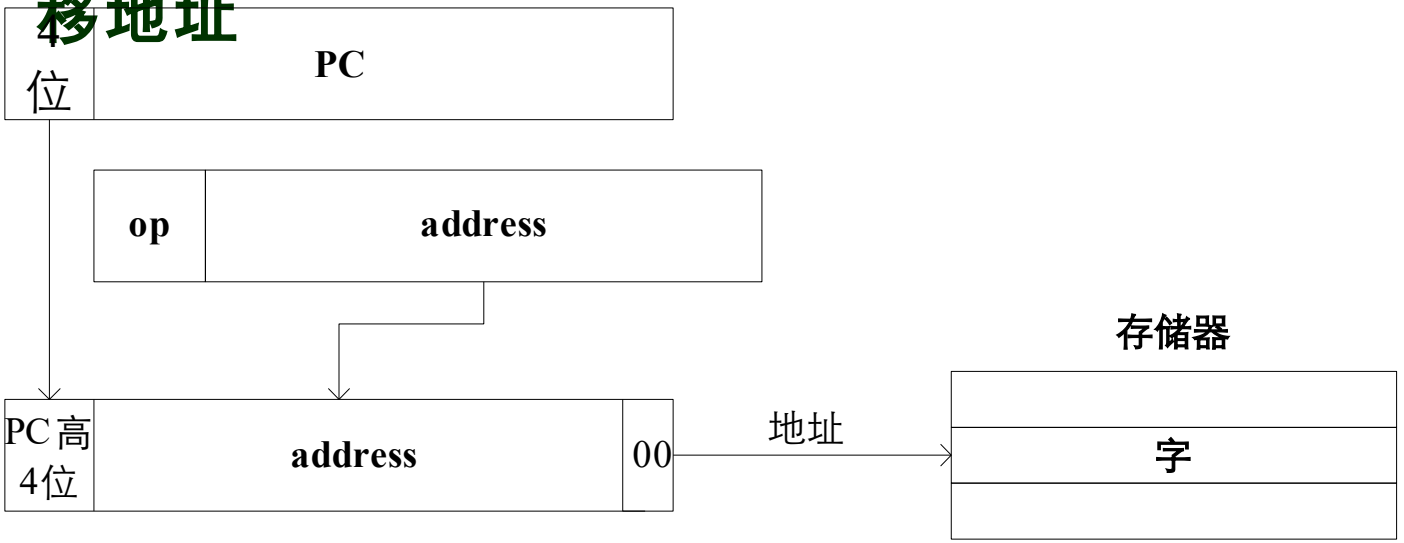
④相对寻址，目标地址是当前 PC 值和指令中的常数之和， 16 位的常数在与 PC 相加之前要左移 2 位（相当于乘以 4 ）

例： bne $s1,$s2,25 # if $s1≠$s2 then goto 25*4+ 当前 pc 值（即 bne 指令的 PC+

二、MIPS32 指令格式、寻址方式和指令

2. 寻址方式

⑤伪直接寻址，将当前 PC 值的高 4 位拼上 26 位直接地址，末尾两位置 0，即可得到 32 位的目标转移地址



二、MIPS32 指令格式、寻址方式和指

2. 寻址方式

- 从 R 、 I 、 J 三种指令类型的角度讨论寻址方式
- R 型指令只有寄存器寻址一种寻址方式。

例如：

```
or   $s1,$s2,$s3
     # $s1=$s2 | $s3
```

二、MIPS32 指令格式、寻址方式和指令

2. 寻址方式

- I 型指令有寄存器寻址、立即数寻址、相对寻址、基址或偏移量寻址四种寻址方式。

若 I 型指令是双目运算指令，则 rs 寄存器和立即数分别作为源操作数， rt 寄存器是目的操作数；

例如：

```
addi  $sp,$sp,4          # $sp=$sp+4 ，立即数寻址
lui  $s0,61                    #把常量（61）10装入 s0 寄存器的高 16 位，  (61)10=(003D)16，
```

二、MIPS32 指令格式、寻址方式和指

2. 寻址方式

- 若 I 型指令是访存指令 load/store，则存储器地址由 rs 寄存器加上符号扩展的立即数得到，load 指令将存储内容读出到 rt 寄存器中，store 指令将 rt 寄存器写入存储器中。例如：

```
lw $t0,32($s1)        # 假如 s1 里是数组 A 的
```

起始地址，存储器按字节编址，因此 $s1+32 指向 A[8]，连续读 4 个字节，读出 A[8] 的值置入 t0 寄存器。这是寄存器相对寻址（基址寻址）方式，s1 是基地址寄存器，用来装载数组的起始地址，常数 32 是偏移量，表示数组元素的下标值。

二、MIPS32 指令格式、寻址方式和指令

2. 寻址方式

- 若 I 型指令是条件转移指令，则对 rs 和 rt 寄存器的内容进行指定运算，根据结果决定是否跳转。转移目标地址等于当前 PC 值加上符号扩展后的立即数，这是相对寻址。偏移量可正可负。在对数组和堆栈寻址时，相对寻址比其他寻址方式更具优势。例如：

- bne $s0,$s1,L0 # 如果 #$s0≠s1，则跳转到 L0 处。这是相对寻址。 PC=(PC)+16 位偏移量，请注意：与偏移量相加的 PC 值是 bne 指令的后一条指令的 PC 值，即 bne 指令的

二、MIPS32 指令格式、寻址方式和指

2. 寻址方式

- J 型指令只有伪直接寻址这一种寻址方式。因为 J 型指令都是无条件转移指令，而 MIPS 系统采用 32 位定长指令，每条指令占 4 个存储单元，指令地址总是 4 的倍数。所以只要将当前 PC 值的高 4 位拼上 26 位直接地址，末尾两位置 0，即可得到 32 位的目标转移地址。

表 6.25　MIPS 的转移指令格式及编码

R 型指令	字段	OP	rs	rt	rd	shamt	func	功能描述
	位数	6	5	5	5	5	6	
								无条件跳转：
j label	000010	address						位,address,0,0}→PC
jal label	000011	address						无条件跳转并链接：(PC+4)→$31, {(PC+4)高 4 位,address,0,0}→PC

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) 高 4 位 ,address,0,0}；

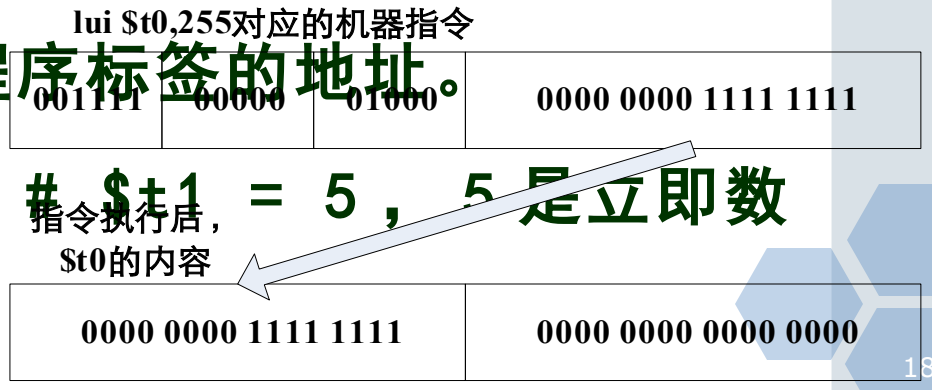
二、MIPS32 指令格式、寻址方式和指

3. 指令分类：

- 指令以 d 开头表示 64 位版本指令，以“u”结尾表示无符号数指令，以“i”结尾表示立即数指令，以“b”、“w”和“d”结尾，分别表示字节、字和双字操作
- **空操作指令**：空操作指令有 nop 和 ssnop
- **数据传送指令**
  - 寄存器间数据传送指令：move、movf、movt、movn、movz，movf、mov：根据浮点条件标志，有条件地进行整数寄存器之间的传递；movn、movz：根据另一个寄存器的状态，有条件地

二、MIPS32 指令格式、寻址方式和指令分类

- **数据传送指令**
  - 常数加载指令，la 是获取某些标号地址或程序中变量地址的宏指令，li 是装入立即数常数的指令，lui 指令把 16 位立即数加载到寄存器的高位，寄存器低 16 位清 0 。
  - la $t1, var　　# 把 var 在主存中的地址装载到 t1 寄存器中， var 也可以是程序中定义的一个子程序标签的地址。
  - li $t1, 5　　# $t1 = 5 ，5 是立即数
  - lui $t0,255



二、MIPS32 指令格式、寻址方式和指令

- **算术/逻辑运算指令**
  - 加减运算指令 add、addu、addiu、subu、sub
  - 混合算术运算指令：取绝对值指令 abs，取反指令 negu、neg
  - 按位逻辑指令：and、andi、or、ori、xor、xori、nor、not
  - 循环和移位指令：rol、ror、sll、sllv、srl、srlv、sra、srav
  - 条件设置指令：slt、slti、sltiu、sltu：硬件指令，如果条件满足则写入1，否则写入0，seq、sge、sgeu、sgt、sgtu、sle、slue、sne 根据更复杂的条件设置目的寄存器

二、MIPS32 指令格式、寻址方式和指令

- 算术/逻辑运算指令
  - mul、mulo、mulou、mult、multu、rem、remu、mfhi、mflo、mthi、mtlo
- 访存指令：
  - lw、lb（寄存器高位填入符号位）、lbu（寄存器高位填入0）、lh（寄存器高位填入符号位）、lhu（寄存器高位填入0）、把数据预取到缓冲的指令pref、sb、sh、sw
  - 浮点存取指令：存取单/双精度浮点数的指令l.s、l.d、s.s、s.d

二、MIPS32 指令格式、寻址方式和指令

- 程序控制类指令
  - 分支指令，与 PC 相关的跳转指令称为“分支指令”，以 b 开头

```
b target    #   基于当前指令地址的无条件相对跳转，跳转范围比较短
beq $t0,$t1,target    #  branch to target if  $t0 = $t1
blt $t0,$t1,target    #  branch to target if  $t0 < $t1
```

二、MIPS32 指令格式、寻址方式和指令

- **程序控制类指令**
  - 分支指令

```
    ble $t0,$t1,target    #  branch to targ
et if  $t0 ≤ $t1

    bgt $t0,$t1,target    #  branch to targ
et if  $t0 > $t1

    bge $t0,$t1,target    #  branch to targ
et if  $t0 ≥ $t1

bne $t0,$t1,target   #  branch to target if
  $t0 ≠ $t1
```

二、MIPS32 指令格式、寻址方式和指令

- **程序控制类指令**

  - 跳转指令（Jumps），绝对地址的跳转称为“跳转指令”，跳转指令以 j 开头。

    j 指令是无条件跳转到一个绝对地址，访问 256M 代码空间。

    jr 指令是跳转到某寄存器指示的地址处。

例如：

```
j target     #   无条件跳转到标号 target 处
jr $t3       #   跳转目标地址在 $t3 寄存
```

二、MIPS32 指令格式、寻址方式和指令

- **子程序调用指令**
  - 子程序调用称为“跳转并链接”或“分支并链接”，指令以...al 结尾。

    Jal/jalr 是直接/间接的子程序调用，跳转到指定地址，并把返回地址放到 $ra 寄存器中，返回地址是当前指令地址 +8 。这是因为紧跟在调用指令 jal 后面立即执行的指令称做调用的延迟槽，返回地址应该是延迟槽指令后面的那条指令。

二、MIPS32 指令格式、寻址方式和指令

- **调用子程序：**

jal sub_label　　#　把返回地址（返回地址是当前指令地址 +8，即当前指令的下下条指令的地址）放到 $ra 寄存器中，然后跳转子程序，sub_label 是子程序的标号

例：返回地址是第三行的指令，而不是 jal 指令后面的 move 指令

```
    jal     printf    #  返回地址是 xxx 指令的地址
    move    $4,$6
    xxx                #  子程序返回到该指令
```

二、MIPS32 指令格式、寻址方式和指令

- **子程序调用指令**

子程序调用返回：

```
jr  $ra                    #   跳转到 $ra
```

寄存器指示的地址处，$ra 中是调用子程序之前保存的主程序返回地址。

注意，返回地址存放在 $ra 寄存器中。如果子程序调用了下一级子程序，或者是递归调用，那么需要将返回地址保存在堆栈中，因为每执行一次 jal 指令就会覆盖 $ra 中的返回地址。

二、MIPS32 指令格式、寻址方式和指令

- **断点及陷阱指令**
  - break 指令产生“断点”类型的异常，可以在宏扩展指令中产生陷阱或用于调试器。
  - sdbbp 指令产生 EJTAG 异常的断点指令。
  - syscall 指令产生一个约定用于系统调用的异常类型。
  - teq、teqi、tge、tgei、tgeiu、tgeu、tlt、tlti、tltiu、tltu、tne、tnei 是条件异常指令，对一个或两个操作数进行条件测试，条件满足则触发异常。

二、MIPS32 指令格式、寻址方式和指令

- **协处理器 0 的功能指令**
  - 协处理器与通用寄存器之间的双向数据传递
    - ◆cfc0 、 ctc0 是把数据拷进 / 拷出协处理器 0 的控制寄存器的指令。
    - ◆mfc0 、 mtc0 是在通用寄存器和协处理器 0 寄存器之间交换数据的指令。
  - 用于 CPU 控制的特殊指令 :eret 是异常返回指令。

二、MIPS32 指令格式、寻址方式和指令

- **浮点操作指令**支持 IEEE754 的单精度和双精度格式。
  - 加法：add.s（单精度），add.d（双精度）
  - 减法：sub.s，sub.d
  - 乘法：mul.s，mul.d，除法：div.s，div.d
  - 比较：bclt（条件为真跳转），bclf（条件为假跳转），c.x.s，c.x.d，其中 x 可能为相等（eq），不等（neq），小于（lt），小于等于（le），大于（gt），大于等于（ge）
  - 条件转移：bclt（条件为真跳转），bclf（条件为假跳转）
  - 数据传送：lwcl（读存储器字到浮点寄存器），swcl（把浮点寄存器内容写入存储器）。

■用户模式下对“底层”硬件的有限访问指令

二、MIPS32 指令格式、寻址方式和指令

4. 常用指令列表及指令编码

表 2-5 MIPS 指令译码表-1

OP(31:26)								
28-26 / 31-29	000	001	010	011	100	101	110	111
000	R 型指令	bltz/gez	j	jal	beq	bne	blez	bgtz
001	addi	addiu	slti	sltiu	andi	ori	xori	lui
010	TLB	FLPt						
011								
100	lb	lh	lwl	lw	lbu	lhu	lwr	
101	sb	sh	swl	sw				
110	lwc0	lwc1						
111	swc0	swc1						

二、MIPS32 指令格式、寻址方式和指令

4. 常用指令列表及指令编码

表 2-6 MIPS 指令译码表-2

OP(31:26）=010000（TLB), rs(25:21)								
23-21 / 25-24	000	001	010	011	100	101	110	111
000	mfc0		cfc0		mtc0		ctc0	
001								
010								
011								
100								
101								
110								
111								

二、MIPS32 指令格式、寻址方式和指令

4. 常用指令列表及指令编码

表 2-7 MIPS 指令译码表-3

OP(31:26）=000000（R 型指令），func(5:0)								
2-0 / 5-3	000	001	010	011	100	101	110	111
000	sll		srl	sra	sllv		srlv	srav
001	jr	jalr			sysycall	break		
010	mfhi	mthi	mflo	mtlo				
011	mult	multu	div	divu				
100	add	addu	sub	subu	and	or	xor	nor
101			slt	sltu				
110								
111								

助助助	助助助助助（位）						助助	助助助助	助助助助助助助
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
add	0	rs	rt	rd	0	100000	add $s1,$s2,$s3	$s1=$s2+$s3	rd←rs+rt；其中 rs = $s2， rt=$s3,rd=$s1，有符号数加
addu	0	rs	rt	rd	0	100001	addu $s1,$s2,$s3	$s1=$s2+$s3	rd←rs+rt；其中 rs = $s2， rt=$s3,rd=$s1，无符号数加
sub	0	rs	rt	rd	0	100010	sub $s1,$s2,$s3	$s1=$s2-$s3	rd←rs-rt；其中 rs = $s2， rt=$s3,rd=$s1，有符号数减
subu	0	rs	rt	rd	0	100011	subu $s1,$s2,$s3	$s1=$s2-$s3	rd←rs-rt；其中 rs = $s2， rt=$s3,rd=$s1，无符号数减
and	0	rs	rt	rd	0	100100	and $s1,$s2,$s3	$s1=$s2&$s3	rd←rs&rt；其中 rs = $s2， rt=$s3,rd=$s1，按位与运算

助助助	助助助助助（位）						助助	助助助助	助助助助助助助
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
or	0	rs	rt	rd	0	100101	or $s1,$s2,$s3	$s1=$s2|$s3	rd←rs|rt；其中 rs = $s2， rt=$s3,rd=$s1，按位或运算
xor	0	rs	rt	rd	0	100110	xor $s1,$s2,$s3	$s1=$s2^$s3	rd←rs xor rt；其中 rs = $s2， rt=$s3,rd=$s1，按位异或运算
nor	0	rs	rt	rd	0	100111	nor $s1,$s2,$s3	$s1=~($s2|$s3)	rd←not(rs|rt)；其中 rs = $s2， rt=$s3,rd=$s1，按位或非运算
slt	0	rs	rt	rd	0	101010	slt $s1,$s2,$s3	if($s2<$s3)$s1=1 else$s1=0	if(rs<rt)rd=1 else rd=0；其中 rs = $s2， rt=$s3,rd=$s1(有符号数)
sltu	0	rs	rt	rd	0	101011	sltu $s1,$s2,$	if($s2<$s3)$s1=1	if(rs<rt)rd=1 else rd=0；其中 rs =

□□□	□□□□（位）						□□	□□□□	□□□□□□
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
sll	0	0	rt	rd	shamt	000000	sll $s1,$s2,10	$s1=$s2<<10	rd←rt<<shamt；逻辑左移。 shamt 中存放移位的位数，即□□□□□□ $s2 rd=$s1
srl	0	0	rt	rd	shamt	000010	srl $s1,$s2,10	$s1=$s2>>10	rd←rt>>shamt；逻辑右移，其中 rt=$s2,rd=$s1
sra	0	0	rt	rd	shamt	000011	sra $s1,$s2,10	$s1=$s2>>10	rd←rt>>shamt；立即数是移位次数，算术右移，保留符号位。□□ $s2 rd=$s1
sllv	0	rs	rt	rd	0	000100	sllv $s1,$s2,$s3	$s1=$s2<<$s3	rd←rt<<rs；逻辑左移，其中 rs = $s3 rt=$s2 rd=$s1
srlv	0	rs	rt	rd	0	000110	srlv $s1,$s2,$s3	$s1=$s2>>$s3	rd←rt>>rs；逻辑右移，其中 rs = $s3， rt=$s2,rd=$s1

□□□	□□□□（位）						□□	□□□□	□□□□□□
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
srav	0	rs	rt	rd	0	111	srav $s1,$s2,$s3	$s1=$s2>>$s3	rd←rt>>rs；移位次数存于寄存器中，算术右移，保留符号位。□□= $s3 rt=$s2 rd=$s1
jr	0	rs	0	0	0	001000	jr $s31	go to $ra	PC←rs, □□ = $s31

□□□	□□□□□□				□□	□□	□□□□□□
	31..26	25..21	20..16	15..0			
| □□□	op	rs	rt	immediate			
addi	001000	rs	rt	immediate	addi $s1,$s2,100	$s1=$s2+100	rt←rs+ □□□ rt=$s1,rs=$s2
addiu	001001	rs	rt	immediate	addiu $s1,$s2,100	$s1=$s2+100	rt←rs+ □□□ rt=$s1,rs=$s2
andi	001100	rs	rt	immediate	andi $s1,$s2,10	$s1=$s2&10	rt←rs& □□□ rt=$s1,rs=$s2
ori	001101	rs	rt	immediate	andi $s1,$s2,10	$s1=$s2|10	rt←rs| □□□ rt=$s1,rs=$s2
xori	001110	rs	rt	immediate	andi $s1,$s2,10	$s1=$s2^10	rt←rs xor □□□ rt=$s1,rs=$s2

□□□	□□□□□□				□□	□□	□□□□□□
	31..26	25..21	20..16	15..0			
	op	rs	rt	immediate			
lui	001111	0	rt	immediate	lui $s1,100	$s1=100*65536	rt←immediate*65536 □□ 16 □□ □□□□□□□□□□ 16 □□ 0
lw	100011	rs	rt	immediate	lw $s1,10($s2)	$s1=memory[$s2+10]	rt←memory[rs+(sign-extend)immediate] □ rt=$s1,rs=$s2
sw	101011	rs	rt	immediate	sw $s1,10($s2)	memory[$s2+10]=$s1	memory[rs+ □ rt=$s1,rs=$s2
bne	000101	rs	rt	immediate	bne $s1,$s2,10	if($s1!=$s2) goto PC+4+40	if(rs!=rt)PC←PC+4+
slti	001010	rs	rt	immediate	slti $s1,$s2,10	if($s2<10) $s1=1 else $s1=0	if(rs< □□□□ $s2 □ rt=$s1
sltiu	001011	rs	rt	immediate	sltiu $s1,$s2,10	if($s2<10) $s1=1 else $s1=0	if(rs< □□□□ $s2 □ rt=$s1

□□□	□□□□□□□		□□	□□□□	□□□□□□
	31..26	25..0			
J	op	address			
j	000010	address	j 10000	goto 10000	PC←(PC+4)[31..28],address,00 □ address=10000/4
jal	000011	address	jal 10000	$s31←PC+4; goto 10000	$s31←PC+4 □ PC←(PC+4)[31..28],address,00

□□□□	□□	□□	□□□□□	
			op31..26	Address25..0
j 10000	jump to 2500*4	□□□	2	2500
		□□□□	000010	00 0000 0000 0000 1001 1100 0100
jal 10000	$s31=PC+4	□□□	3	2500
		□□□□	000011	00 0000 0000 0000 1001 1100 0100