1.引言

MIPS 操作数

名字	示例	注释
32 个寄存器	$s0 - $s7, $t0 - $t9, $zero, $a0 - $a3, $v0 - $v1, $gp, $fp, $sp, $ra, $at	寄存器用于数据的快速存取。在 MIPS 中，只能对存放在寄存器中的数据执行算术操作，寄存器 $zero的值恒为 0，寄存器 $at 被汇编器保留，用于处理大的常数
2^{30}个存储器字	Memory [0], Memory [4], …, Memory [4294967292]	存储器只能通过数据传输指令访问。MIPS 使用字节编址，所以连续的字地址相差 4。存储器用于保存数据结构、数组和溢出的寄存器

$ra 存储返回地址 $fp,$sp 与栈相关 $gp 全局变量指针 注意字地址

类别	指令	示例	含义	注释
算术	加法	add $s1, $s2, $s3	$s1 = $s2 + $s3	三个寄存器操作数
	减法	sub $s1, $s2, $s3	$s1 = $s2 - $s3	三个寄存器操作数
	立即数加法	addi $s1, $s2,20	$s1 = $s2 +20	用于加常数数据
数据传输	存字	sw $s1,20($s2)	Memory[$s2 +20] = $s1	将一个字从寄存器中取到内存中
	取半字	lh $s1,20($s2)	$s1 = Memory[$s2 +20]	将半个字从内存中取到寄存器中
	取无符号半字	lhu $s1,20($s2)	$s1 = Memory[$s2 +20]	将半个字从内存中取到寄存器中
	存半字	sh $s1,20($s2)	Memory[$s2 +20] = $s1	将半个字从寄存器存到内存中
	取字节	lb $s1,20($s2)	$s1 = Memory[$s2 +20]	将一个字节从内存取到寄存器中
	取无符号字节	lbu $s1,20($s2)	$s1 = Memory[$s2 +20]	将一个字节从内存取到寄存器中
	存字节	sb $s1,20($s2)	Memory[$s2 +20] = $s1	将一个字节从寄存器存到内存中
	取链接字	ll $s1,20($s2)	$s1 = Memory[$s2 +20]	取字作为原子交换的前半部
	存条件字	sc $s1,20($s2)	Memory[$s2 +20] = $s1; $s1 =0 or 1	存字作为原子交换的后半部分
	取立即数的高位	lui $s1,20	$s1 =20 * 2^{16}	取立即数并放到高 16 位
逻辑	与	and $s1, $s2, $s3	$s1 = $s2 & $s3	三个寄存器操作数按位与
	或	or $s1, $s2, $s3	$s1 = $s2 | $s3	三个寄存器操作数按位或
	或非	nor $s1, $s2, $s3	$s1 = ~($s2 | $s3)	三个寄存器操作数按位或非
	立即数与	andi $s1, $s2,20	$s1 = $s2 & 20	和常数按位与
	立即数或	ori $s1, $s2,20	$s1 = $s2 | 20	和常数按位或
	逻辑左移	sll $s1, $s2,10	$s1 = $s2 <<10	根据常数左移相应位
	逻辑右移	srl $s1, $s2,10	$s1 = $s2 >>10	根据常数右移相应位
条件分支	相等时跳转	beq $s1, $s2,25	if($s1 == $s2) go to PC +4 +100	相等检测；和 PC 相关的跳转
	不相等时跳转	bne $s1, $s2,25	if($s1 ! = $s2) go to PC +4 +100	不相等检测；和 PC 相关的跳转
	小于时置位	slt $s1, $s2, $s3	if($s2 < $s3) $s1 =1; else $s1 =0	比较是否小于；beq, bne
	无符号数比较小于时置位	sltu $s1, $s2, $s3	if($s2 < $s3) $s1 =1; else $s1 =0	比较是否小于无符号数
	无符号数比较小于立即数时置位	slti $s1, $s2,20	if($s2 <20) $s1 =1; else $s1 =0	比较是否小于常数
	无符号数比较小于无符号立即数时置位	sltiu $s1, $s2,20	if($s2 <20) $s1 =1; else $s1 =0	比较是否小于无符号常数
无条件跳转	跳转	j 2500	go to 10000	跳转到目标地址
	跳转至寄存器所指位置	jr $ra	go to $ra	用于 switch 语句，以及过程调用
	跳转并链接	jal 2500	$ra = PC +4; go to 10000	用于过程调用

图 2-1 （续）

h->half u->unsigned i->立即数 b->byte

2.一些概念

存储器操作数

大端：在大端序中，最高有效字节（Most Significant Byte，MSB）存储在最低的内存地址，而最低有效字节（Least Significant Byte，LSB）存储在最高的内存地址。这就好像数字的表示是从左到右，高位到低位的顺序一样。

举例：十六进制数 0x12345678 在大端序中的存储方式为 12 34 56 78。

小端：在小端序中，最低有效字节存储在最低的内存地址，而最高有效字节存储在最高的内存地址。这就好像数字的表示是从右到左，低位到高位的顺序一样。

举例：十六进制数 0x12345678 在小端序中的存储方式为 78 56 34 12

常数或立即数操作数

有符号数和无符号数

LSB:最低有效位 MSB:最高有效位

求反捷径：(2)10 到(-2)10 按位取反加 1

符号扩展：最高位是 1 就补 1，最高位是 0 就补 0

3.指令

字。在 MIPS 汇编语言中，寄存器 $s0 ~ $s7 映射到寄存器 16 ~ 23，同时，寄存器 $t0 ~ $t7 映射到寄存器 8 ~ 15。因此，$s0 表示寄存器 16，$s1 表示寄存器 17，$s2 表示寄存器 18…… $t0 表示寄存器 8，$t1 表示寄存器 9，依次类推。在下面几节中，我们将介绍 32 个寄存器中其余寄存器的映射。

MIPS 字段

R 型指令

为了使讨论变得简单，给 MIPS 字段命名如下：

op	rs	rt	rd	shamt	funct
6位	5位	5位	5位	5位	6位

MIPS 指令中各字段名称及含义如下：

- op：指令的基本操作，通常称为**操作码**（opcode）。
- rs：第一个源操作数寄存器。
- rt：第二个源操作数寄存器。
- rd：用于存放操作结果的目的寄存器。
- shamt：位移量。（在 2.6 节中介绍移位指令和该术语，在此之前，指令都不使用这个字段，故此字段的内容为 0。）
- funct：功能。一般称为*功能码*（function code），用于指明 op 字段中操作的特定变式。

I 型指令

MIPS 设计者选择这样一种折中方案：保持所有的指令长度相同，但不同类型的指令采用不同的指令格式。例如，上述格式称为 R 型（用于寄存器）。另一种指令格式称为 I 型（用于立即数），立即数和数据传送指令用的就是这种格式。I 型的字段如下所示：

op	rs	rt	constant or address
6位	5位	5位	16位

16 位的地址字段意味着取字指令可以取相对于基址寄存器地址偏移 $\pm 2^{15}$ 或者 32 768 个字节（$\pm 2^{13}$ 或者 8192 个字）范围内的任意数据字。类似地，加立即数指令中常数也被限制不超过 $\pm 2^{15}$。可以看到在这种格式下，很难设置 32 个以上的寄存器，因为 rs 和 rt 字段都必须增加额外的位，这样就导致 32 位字长的指令很难满足要求。

指令	格式	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{10}	n. a.
sub (subtract)	R	0	reg	reg	reg	0	34_{10}	n. a.
add immediate	I	8_{10}	reg	reg	n. a.	n. a.	n. a.	常数
lw (load word)	I	35_{10}	reg	reg	n. a.	n. a.	n. a.	address
sw (store word)	I	43_{10}	reg	reg	n. a.	n. a.	n. a.	address

图 2-5 MIPS 指令编码。在上表中，“reg”代表寄存器的标号（从 0～31），“address”表示 16 位地址，“n. a.”（not applicable）表示这个字段在该指令格式中不出现。注意，add 和 sub 指令具有相同的 op 字段值，硬件根据 funct 字段的值来决定所进行的操作：add（32）或 substract（34）

例 1

我们来分析一下 2. 3. 1 节例子中的取字指令：

```
lw   $t0,32($s3)   # Temporary reg $t0 gets A[8]
```

这里，19（寄存器 $s3）存放于 rs 字段，8（寄存器 $t0）存放于 rt 字段，32 存放于 address 字段。注意，对于这条指令 rt 字段的意思已经改变：在一条取字指令中，rt 字段用于指明接收取数结果的目的寄存器。

注意 rs,rt 到底是谁，rs 只能作源操作数寄存器，rt 可以作源操作数寄存器也可以作目的寄存器（在 I 型指令中），rd 只能作目的寄存器

例 2

现在可以给出一个例子来描述从程序员所编程序到机器执行指令的整个转换过程。如果数组 A 的基址存放在 $t1 中，h 存放在 $s2 中，下面的 C 赋值语句：

```
A[300] = h + A[300];
```

被编译成如下汇编语言：

```
lw   $t0,1200($t1) # Temporary reg $t0 gets A[300]
add  $t0,$s2,$t0   # Temporary reg $t0 gets h + A[300]
sw   $t0,1200($t1) # Stores h + A[300] back into A[300]
```

一般题目如果给了数组基本上就是默认这个数组元素大小是一个字（四个字节），但是 lw 指令存的是字节地址，注意看汇编部分

4.一些概念

AND OR SLL SLR

NOR（或非） A NOR 0 =NOT(A OR 0)=NOT(A)

BEQ BNE

slt slti

过程（其实就是函数调用）

与栈有关的操作

名称	寄存器号	用途	调用时是否保存
$zero	0	常数 0	不适用
$v0 ~ $v1	2 ~ 3	计算结果和表达式求值	否
$a0 ~ $a3	4 ~ 7	参数	否
$t0 ~ $t7	8 ~ 15	临时变量	否
$s0 ~ $s7	16 ~ 23	保存的寄存器	是
$t8 ~ $t9	24 ~ 25	更多临时变量	否
$gp	28	全局指针	是
$sp	29	栈指针	是
$fp	30	帧指针	是
$ra	31	返回地址	是

5.分支和跳转中的寻址

J 型指令

这个例子中的 J 10000 是错的

MIPS 跳转指令寻址采用最简单的寻址方式。它们使用最后一种 MIPS 指令格式，称为 J 型。J 型除了 6 位操作码之外，其余位都是地址字段。所以

```
j    10000    # go to location 10000
```

可以汇编为下面的格式（实际中要更加复杂一些，正如我们后面将看到的那样）：

2	10000
6位	26位

其中跳转操作码的值为 2，跳转地址为 10000。

和跳转指令不同，条件分支指令除了规定分支地址之外还必须指定两个操作数。因此

```
bne  $s0,$s1,Exit  # go to Exit if $s0 ≠ $s1
```

被汇编为下面的指令，只保留了 16 位用于指定分支地址：

5	16	17	Exit
6位	5位	5位	16位

PC 相对寻址

答案取决于条件分支是如何使用的。条件分支在循环和 if 语句中都可以找到，它们倾向于转到附近的指令。例如，在 SPEC 基准测试程序中，大概一半条件分支的跳转距离小于 16 条指令。因为程序计数器（Program Counter，PC）包含当前指令的地址，如果我们使用 PC 来作为

增加地址的寄存器，我们可转移到离当前指令距离为 $\pm 2^{15}$ 个字的地方。几乎所有循环和 if 语句都远远小于 2^{16} 个字，因此 PC 是一个理想的选择。

这种分支寻址形式称为 **PC 相对寻址**（PC-relative addressing）。正如在第 4 章中将会看到的那样，提前递增 PC 来指向下一条指令会对硬件带来很多方便。所以，MIPS 寻址实际上是相对于下一条指令的地址（PC +4），而不是相对于当前指令（PC）。寻址附近的指令是**加速大概率事件**的另外一个例子。

PC 相对寻址：一种寻址方式，它将 PC 和指令中的常数相加作为寻址结果。

6.不同的寻址方式

1.立即数寻址

操作数为指令自身中的常数立即数

addi r2,r1,1

r2=r1+1

2.寄存器寻址

操作数为寄存器中的值

add r2,r1,r0

r2=r1+r0

3.基址寻址（lw 指令）

以基址寄存器中的值和立即数常数之和作为地址，该地址指向的内存的值作为操作数

lw r2,8(r1)

r2=M[r1+8]

4.PC 相对寻址（分支指令）

以 PC 程序计数器和指令中常数之和作为地址，即 16 位地址左移 2 位（即乘以 4）与 PC 计数器相加

beq r2,r1,label

如果 r2==r1 则程序跳到 label 处

5.伪直接寻址

跳转地址由指令中 26 位地址左移两位（即乘以 4）与 PC 计数器的高 4 位相连所组成