**MIPS寄存器介绍**

MIPS有32个通用寄存器（$0-$31），各寄存器的功能及汇编程序中使用约定如下：

下表描述32个通用寄存器的别名和用途

|  |  |  |
| --- | --- | --- |
| REGISTER | NAME | USAGE |
| $0 | $zero | 常量0(constant value 0) |
| $1 | $at | **保留**给汇编器(Reserved for assembler) |
| $2-$3 | $v0-$v1 | 函数调用返回值(values for results and expression evaluation) |
| $4-$7 | $a0-$a3 | 函数调用参数(arguments) |
| $8-$15 | $t0-$t7 | 暂时的(或随便用的) |
| $16-$23 | $s0-$s7 | 保存的(或如果用，需要SAVE/RESTORE的)(saved) |
| $24-$25 | $t8-$t9 | 暂时的(或随便用的) |
| $28 | $gp | 全局指针(Global Pointer) |
| $29 | $sp | 堆栈指针(Stack Pointer) |
| $30 | $fp | 帧指针(Frame Pointer) |
| $31 | $ra | 返回地址(return address) |

下面给以详细说明：

* $0:即$zero,该寄存器总是返回零，为0这个有用常数提供了一个简洁的编码形式。

          move $t0,$t1  
       实际为  
           add $t0,$0,$t1  
       使用伪指令可以简化任务，汇编程序提供了比硬件更丰富的指令集。

* $1:即$at，该寄存器为汇编保留，由于I型指令的立即数字段只有16位，在加载大常数时，编译器或汇编程序需要把大常数拆开，然后重新组合到寄存器里。比如加载一个32位立即数需要lui（装入高位立即数）和addi两条指令。像MIPS程序拆散和重装大常数由汇编程序来完成，汇编程序必需一个临时寄存器来重组大常数，这也是为汇编保留$at的原因之一。
* $2..$3:($v0-$v1)用于子程序的非浮点结果或返回值，对于子程序如何传递参数及如何返回，MIPS范围有一套约定，堆栈中少数几个位置处的内容装入CPU寄存器，其相应内存位置保留未做定义，当这两个寄存器不够存放返回值时，编译器通过内存来完成。  
  $4..$7:($a0-$a3)用来传递前四个参数给子程序，不够的用堆栈。a0-a3和v0-v1以及ra一起来支持子程序／过程调用，分别用以传递参数，返回结果和存放返回地址。当需要使用更多的寄存器时，就需要堆栈（stack)了,MIPS编译器总是为参数在堆栈中留有空间以防有参数需要存储。
* $8..$15:($t0-$t7)临时寄存器，子程序可以使用它们而不用保留。
* $16..$23:($s0-$s7)保存寄存器，在过程调用过程中需要保留（被调用者保存和恢复，还包括$fp和$ra），MIPS提供了临时寄存器和保存寄存器，这样就减少了寄存器溢出（spilling,即将不常用的变量放到存储器的过程), 编译器在编译一个叶（leaf)过程（不调用其它过程的过程）的时候，总是在临时寄存器分配完了才使用需要保存的寄存器。
* $24..$25:($t8-$t9)同($t0-$t7)
* $26..$27:($k0,$k1)为操作系统／异常处理保留，至少要预留一个。异常（或中断）是一种不需要在程序中显示调用的过程。MIPS有个叫异常程序计数器（exception program ounter,EPC)的寄存器，属于CP0寄存器，用于保存造成异常的那条指令的地址。查看控制寄存器的唯一方法是把它复制到通用寄存器里，指令mfc0(move from system control)可以将EPC中的地址复制到某个通用寄存器中，通过跳转语句（jr)，程序可以返回到造成异常的那条指令处继续执行。MIPS程序员都必须保留两个寄存器$k0和$k1，供操作系统使用。发生异常时，这两个寄存器的值不会被恢复，编译器也不使用k0和k1,异常处理函数可以将返回地址放到这两个中的任何一个，然后使用jr跳转到造成异常的指令处继续执行。
* $28:($gp)为了简化静态数据的访问，MIPS软件保留了一个寄存器：全局指针gp(global pointer,$gp)，全局指针只想静态数据区中的运行时决定的地址，在存取位于gp值上下32KB范围内的数据时，只需要一条以gp为基指针的指令即可。在编译时，数据须在以gp为基指针的64KB范围内。
* $29:($sp)MIPS硬件并不直接支持堆栈，你可以把它用于别的目的，但为了使用别人的程序或让别人使用你的程序，还是要遵守这个约定的，但这和硬件没有关系。
* $30:($fp)GNU MIPS C编译器使用了帧指针(frame pointer),而SGI的C编译器没有使用，而把这个寄存器当作保存寄存器使用（$s8),这节省了调用和返回开销，但增加了代码生成的复杂性。
* $31:($ra)存放返回地址，MIPS有个jal(jump-and-link,跳转并链接)指令，在跳转到某个地址时，把下一条指令的地址放到$ra中。用于支持子程序，例如调用程序把参数放到$a0~$a3,然后jal X跳到X过程，被调过程完成后把结果放到$v0,$v1,然后使用jr $ra返回。另外，MIPS对CPU的控制通过协处理器0（CP0）来完成。
* 参考资料： http://hi.baidu.com/qq520131714/blog/item/f28933245603072cd40742a6.html     MIPS汇编小贴士http://blog.csdn.net/jerryutscn/archive/2010/03/10/5365263.aspx   基于MIPS架构的BackTrace实现