# MIPS 汇编研究

SWD 聂勇

## 版 本 历 史

| 版本/状态 | 责任人 | 起止日期 | 备注 |
|---|---|---|---|
| V1.0/正式 | 聂勇 | 26Oct2010 | 研究 MIPS 汇编指令，汇编语法等方面 |
| V1.1/正式 | 聂勇 | 12Nov2010 | 添加实例，理论联系实践 |

# 目　录

# 1. 说明

本文档是在 BSP 学习过程中一些 MIPS 汇编指令知识的总结，作为自己的一个备忘录。

# 2. MIPS 汇编常见语法

## 2.1 提示符

.set noreorder：

要注意 mips 具有流水线可见性，所以跟在跳转指令后的下一条指令，在执行跳转到的地方前，都会执行，这个叫分支延迟。但是编译器会隐藏该特性，但可以通过设置”.set noreorder”来禁止编译器重新组织代码顺序。

.set reorder：

为何在 0xbfc0.0400 处有一个.set reorder 的提示符号呢，放置在这里有什么用处？

.ent _xxxx 一个代码段的开始

.end _xxxx 该代码段的结束

所以，一个完整的代码段的写法一般是如下所示：

| 示例 |
| --- |
| .ent _mtest /* 代码段_mtest 开始 */<br>_mtest:<br>……<br>……<br>.end _mtest /* 代码段_mtest 结束 */ |

,align 是用来做什么的？

## 2.2 标签

使用数字作为标签的时候，汇编器会把这个当成一个本地局部标签（local label）。对于标签的引用，使用 1f 表示当前位置的后一个数字 1 标签，f 表示 forward 的意思，使用 1b 表示当前位置的前一个数字 1 标签，b 表示 back 的意思。

# 3. Global Pointer 寄存器

全局指针寄存器(gp)是 MIPS 的通用寄存器$28。这个寄存器的用途主要有两种：

1. 在 PIC 中，gp 用来指向 GOT(Global Offset Table)。注意，这里的 PIC 是指的 Linux 中共享库中的 PIC，而在 vxWorks 的 BSP 中的 PIC 只是简单的代码和地址无关，并不涉及到共享库，所以 BSP 中的 gp 的用法并不属于此类。

2. 在嵌入式开发中，gp 用来指向链接时决定的静态数据的地址。这样，对在 gp 所指地址正负各 32K 范围内数据的 load 和 store（其实就是 ld 和 sw 指令），就可使用 gp 作为基址寄存器。

在 romInit()函数向 C 函数 romStart()函数跳转，usrStart()函数最开始两处都有 gp 的初始化。其实现的代码都一样，如下所示。

gp 初始化

```
……
la    gp, _gp              # set global ptr from compiler
……
```

首先，了解_gp 是什么呢？通过了解编译链接的过程，查看 bootrom 的符号表，可以看到,_gp 就是链接器在链接时确定的一个静态数据的存放地址。在 bootrom 的符号表中，大概是 0x801656a0。

其次，这两次的初始化有什么不一样呢？对 tmp.o 的符号表进行查看，我们可以看到，此处，_gp 的值为 0x81ce.a3e0。为什么不一样呢？因为 tmp.o 和 bootrom 是分别链接的，并且其 -Ttext 参数一个是 RAM_LOW_ADRS （ 0x8010.0000 ），另外一个是 ROM_HIGH_ADRS（0x81c0.0000），它们的静态数据的地址当然不一样，所以，_gp 的值不一样，也需要在代码中初始化两次。

# 4. stack pointer

堆栈指针寄存器(sp)，MIPS 使用的是直接的指令(例如 addiu) 来升降堆栈，请注意区别在 X86 中使用指令 POP 和 PUSH 来升降堆栈的做法。

在子程序的入口，sp 会被升到该子程序需要用到的最大堆栈的位置。在子程序中，堆栈的升降的汇编代码一般都大同小异，下面将 romStart()函数开头和结尾的堆栈升降提取出来，代码如下所示：

gp 初始化

```
……
addiu     $sp,$sp,-32
sw   $ra,28($sp)
```

```
sw   $s8,24($sp)
move     $s8,$sp
……
move     $sp,$s8
lw   $ra,28($sp)
lw   $s8,24($sp)
jr   $ra
addiu    $sp,$sp,32
……
```

第一句代码的-32 就说明，romStart()函数最多用到的堆栈空间为 32bytes。然后就是将返回地址 ra 和调用函数的堆栈位置（存放在$s8 中）放入堆栈中，然后将堆栈位置放入$s8 中 move $s8,$sp。在函数返回就是一个逆操作，回复调用函数的堆栈现场。有关$s8（又叫做帧指针 fp）的知识，请参考下一节。

# 5. $s8/frame pointer

第九个通用寄存器$8，又叫做帧指针（frame pointer,fp）。

# 6. 跳转指令

了解跳转指令，对阅读汇编代码，了解 romInit.s 的地址无关代码设计（PIC），理解编译和链接过程很有帮助。

## 6.1 备忘笔记

MIPS CPU 架构对跳转指令使用 Motorola 命名规则。

PC 相对跳转指令叫做 branch；

绝对地址跳转指令叫做 jump；

子程序的调用叫做 jump and link 或者 branch and link。操作码以 al 结束。与普通的跳转的区别就是，子程序调用跳转会保存返回地址（return address）；

所有的 PC 相对跳转指令都是有条件的，例如需要比较两个寄存器。例如：

b label 指令本质上是：beq $zero,$zero,off

j 指令：最大跳转范围是 256M，28 根地址线，高 4 位使用当前 PC 寄存器的值。所以，如果涉及到更大范围的跳转，需要 j 指令配合使用寄存器 regs。例如：j r 或者 jr r。这也是唯一可以在整个 4G 地址空间内任意跳转的方法。

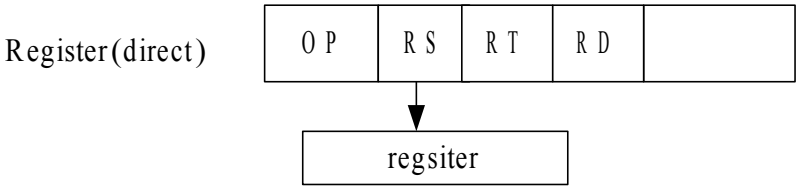
jal 指令：返回地址为当前 PC+8，涉及到延迟槽的相关问题。如果代码需要编译成与地址无关的代码（PIC），则不能够使用 jal label 这样的指令。可以看到，在整个 romInit.s 文件中，只有一处使用了 jal 指令，Line 587

# 7. 寻址方式

## 7.1 寻址方式之指令格式

MIPS 一共有四种寻址方式，下面对每一种方式进行解释和示例说明。

1. 寄存器直接寻址 Register(direct)。其操作数就是寄存器中的值，这也是速度最快的一种寻址方式。

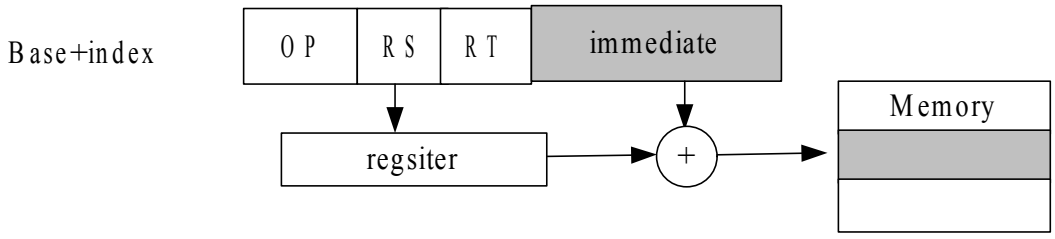



例如：add $S0,$S1,$S2。只对寄存器进行操作的算术指令就是这一类寻址方式。

2. 立即数寻址 Immediate。

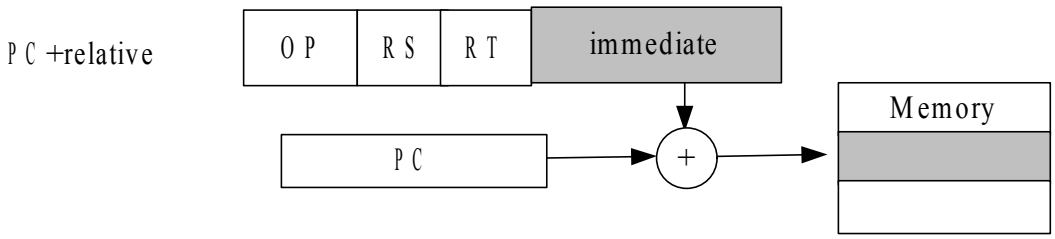

例如：

3. 基址+索引 Base+index。




例如：ld $S0,5($S1)。图中的立即数就是 5。当然，这个立即数是 16 位的，也就是 64K 的范围。

4. 程序指针+偏移量　　PC+relative。




例如：bal label。同理，这里立即数的范围也是 64K。

## 7.2 寻址方式之 load/store

对于 load/store 指令，只有唯一一种寻址方式，那就是 base_reg+offset 的方式，也就上一节中的方式 3。