

MIPS 基础

本文介绍 MIPS 架构的相关知识。首先讨论一些基础背景知识，然后介绍指令。

存储设计

MIPS 采用的是按字节编址，按字访问的方式。使用的字长为 32 位，访问时以字为单位存/取。字有**对齐限制 (Alignment Restriction)**，所有字的首地址必须为 4 的倍数。访问字时访问的是字的首地址，即字所包含的地址中最小的那个。

MIPS 采用的是大端模式。不过这个貌似不是很重要。

MIPS 有一片栈空间专门用来保存函数调用时调用者 (Caller) 的相关信息。

设计原则

记住一些设计原则有利于指令的记忆和理解。

1. 简单即规整 (Simplicity favors regularity)。即指令比较简单，同时同一类型的指令格式统一。
2. 少就是快 (Smaller is faster)。即要平衡一些设计，加的东西太多未必好。
3. 加快经常性事件 (Make the common case fast)。例如寄存器溢出，以及使用零寄存器等。
4. 良好的设计就是合理的折衷 (Good design demands good compromises)。例如 I 型指令，牺牲了立即数的取值范围，保证了指令长度的统一。

汇编基础规则

1. 一行一条指令。
2. 可以有注释，用 # 表示注释的开始，但是注释只能放在一行的最后面。
3. 寄存器名字前加一个 \$ 符号。

操作数

可以是通用寄存器，也可以是常数（一般会被称为立即数）。

通用寄存器

寄存器 (Register) 是位于 CPU 内，距离运算器最近、具有最快访问速度的存储器。

似乎直接来自内存也是可以的，但是 RISC 下为了维护简单性没有这么做。x86 允许操作数直接来源于内存。

通用寄存器总共有 32 个，每一个都具有一定的功能。下面列举一下。

为什么说是通用寄存器？因为它们是对程序员可见的。还有一些寄存器并不对程序员可见，如 PC。在谈论操作数的时候，一般说的寄存器就是指通用寄存器。

- 零寄存器，记作 zero。特点是始终是 0，即使对其进行写入操作也保持是 0，原因在于其是**硬接线的 (Hard-wired)**。也就是硬件层面上保证了其恒定为 0。
- at。其由汇编器控制，用于处理较大的常数。
- v0 和 v1。用于存储子程序返回值的寄存器。
- a0 到 a3。用于存储子程序要用到的参数。可以直接保存参数本身，或者保存参数在内存中的地址。

- t0 到 t9。没什么特定功能的寄存器。
- s0 到 s7。没什么特定功能的寄存器，但是会在子程序调用时被保存在栈中。
- k0 和 k1。操作系统专用。
- gp。用于编译器优化。
- sp。用作栈结构的栈顶指针。
- fp。（不知道干啥用的）
- ra。用于保存子程序执行完之后应当让 PC 返回到的地址。

既然寄存器这么快，为什么不多整几个？只保留 32 个？

原因有几个方面。一是大量的寄存器可能导致电信号的行进距离增长，导致时钟周期变长、时钟频率降低。二是因为增加寄存器可能导致功耗上升。三是因为受到指令格式的限制，如果寄存器变多就需要在指令中给寄存器编号分配更多的位数，这在 RISC 指令定长的限制下是相当难办的事情。

而在事实层面，寄存器的数目增长也是非常缓慢的。因为寄存器的数目和指令集架构有关，指令集架构的发展存在惰性，因而寄存器数目的增长和指令集架构的发展近乎一样缓慢。

总之，平衡好程序员的工作难度和硬件的运行速度很重要。

立即数

立即数也就是常数，可以看作是被硬编码在了指令中的操作数。

使用立即数的好处在于其可以简化很多指令的设计。例如 move 指令就是一条其中一个操作数是 0 的加法指令。

立即数虽然好用，但经常受到定长指令的限制，导致其的取值范围往往达不到 32 位。

指令种类

由于 MIPS 属于 RISC 架构，因此其的指令长度全部固定为 32 位。

MIPS 指令大致可以分成三大类：R 型指令、I 型指令和 J 型指令。

R 型指令

R 型指令和寄存器相关（取 Register 头字母）。其具有以下构成：

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

- op 表示**操作码（Opcode）**，用于识别指令种类。R 型指令的 op 是全 0。I 型、J 型也是开头 6 位用作操作码。
- rs, rt 表示两个操作数。s 指 source, t 指 temporary。
- rd 表示目的操作数，即运算结果的保存位置。

上面三个可以记忆为 std。需要注意的是，汇编代码中的寄存器顺序和这里的寄存器顺序并不一致。例如 `add $1, $2, $3` 表示的是 $\$1 = \$2 + \$3$ 。

- shamt 表示偏移量（Shift Amount）。
- funct 表示指令功能。只有 R 型指令有这个位域，因为其操作码是全 0，只能靠 funct 指定功能。

R 型指令大部分都和算术运算相关，如 `add`、`sub` 等等。最特殊的一个是 `jr`，即跳转到某个寄存器保存的地址上。

I 型指令

I 型指令和立即数相关（取 Immediate 头字母）。其具有以下构成：

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

- op 还是操作码，不是全 0，表示指令功能。
- rs、立即数是两个操作数，rt 是目标操作数。
- 后 16 位表示一个常数，这个常数对于不同的指令而言有着不同的意义。

I 型指令的构成就比较丰富。既有算术指令的扩展版本（即一个运算数变成常数），也有存储器的读写指令（即 lw、sw），还有分支执行指令（即 beq、bne）。它们的功能差异较大。

J 型指令

J 型指令和跳转相关（取 Jump 头字母）。其具有以下构成：

op	address
6 bits	26 bits

- op 还是操作码，不是全 0，表示指令功能。
- address 表示跳转位置。由于其只有 26 位，因此真实的跳转位置为 PC + 4 的高 4 位，再连上 address 左移两位后的结果。

J 型指令只包含两条指令：j 和 jal。j 是无条件跳转，方便构造出死循环，在波形仿真中经常使用；而 jal 表示跳转并链接，将 PC + 4 先保存在 31 号寄存器上（即 ra 寄存器）再进行跳转。它一般用在子程序调用中。

总结

了解 MIPS 指令的构成对于后续扩展 MIPS 指令十分有帮助。