第一章 MIPS 导论：汇编指令集

不同的核（Cluster）之间的传输通过总线，吞吐降低。改善架构存在必要。

1.1 什么是汇编语言？

汇编语言（Assembly Language）是 CPU 可以接收的基本操作，各个 CPU 系列存在不同。

1.2 指令集（Instruction Set Architectures）

随着计算机的发展，需要不同的功能，对应着生成许多的指令集不同的实现。

最初出现的 VAX 有许多的指令，可以执行很大的运算。对应的 RISC 指令集将指令变成更细粒度的实现，虽然很多的问题需要巨量的指令数目，但是速度优于 VAX ，更小的指令用量更大，带来更规整的芯片布局，从而时钟周期会更小。RISC 阵营包括：ARM，MIPS 以及 RISC-V 。

MIPS 汇编语言贴近硬件的实现，没有变量类型的概念，操作的单元是寄存器，算数操作的来源只能是寄存器。寄存器的速度与其硬件开销存在制衡，MIPS 中只有 32 位寄存器，满足大部分的需求，并且硬件便于实现。那么这样的 32-bit 称为一个字（word）。

寄存器可以用数字或者名称引用，数字形式：`$1, $2, ... ,$32`

定义如下：

- `$16 - $23` → `$s0 - $s7` 对应 C 变量
- `$8 - $15` → `$t0 - $t7` 对应临时变量

在汇编语言中，寄存器没有类型，通过操作判断其类型。

在写 MIPS 时，需要注意添加注释（#）。

1.3 运算指令格式

规整的格式：一个操作符加上三个操作数 `1  2,3,4`，其中

1. 操作符号
2. 目标操作数：dest
3. 第一源操作数：src1
4. 第二源操作数：src2

如果需要 0 ，我们可以直接引用一个特殊的零寄存器：`$zero$`。MIPS 中没有原生的 `mov` 而是使用 `add $s0, $s1, $s2` 。同样地，可以用 `add $zero, $zero, $s0` 用来产生流水线的气泡。

如果需要常数，我们可以使用立即数指令：`addi $s0, $s1, 10`。

1.4 内存与寄存器

内存大而慢，寄存器小而快，有一些和内存进行交互的指令也就是数据传输指令。这类的指令要求源与目标的地址，此外还有一个偏移量 offset：`8($t0)` 指向的是指针为 `$t0 + 8` 的内存。

规整的格式：一个`lw`操作符加上三个操作数 `1  2,3(4)`，其中

1. 操作符号
2. 目标寄存器位置：dest
3. 偏移量：offset
4. 源内存位置基址：src

规整的格式：一个`sw`操作符加上三个操作数 `1  2,3(4)`，其中

1. 操作符号
2. 目标内存位置：dest
3. 偏移量：offset
4. 源寄存器位置基址：src

1.5 数据对齐

为了保证取字的迅速以及地址的规整性，需要规定内存地址的对齐。

1.6 条件分支

为了支持 `for-loop/while-loop/do-while-loop/if-else/switch-case` 的实现，定义一系列的条件分支指令。

`j label` 会跳转（jump）到标记了 `label` 的位置。类似的还有 `beq, bne, slt, slti`（branch if equal，branch if not equal, set on less than, set on less than immediate）。

1.7 对字节的操作

由于对字节的操作十分常见，提供了字节级别的操作，如 `lb, sb`，不进行符号位的扩展。可以使用 `addu` 类的指令来停止对溢出的处理（抛出异常）。

1.8 逻辑操作

比如有左移右移指令，可以分为逻辑型以及算数型用来区分右移的符号扩展。

1.9 函数调用

我们需要明确，函数的参数传递方式以及返回方式。MIPS 支持 4 个寄存器的函数调用，更多的参数通过栈进行调用。

函数作为程序的一部分，也会加载到内存中，需要在调用前进行参数准备，并且返回到调用的位置，所以需要将调用位置保存下来通过 `jr` 移交控制。那么这里的 `jr` 和之前的 `j` 有什么区别呢？由于操作数是来自寄存器，也就是编程者可以控制的，更加灵活的跳转到不同的调用位置。

引入了 `jal` ，将返回位置隐式存储到 `$ra` ，在调用后可以直接返回。

为了保护函数调用的上下文，需要使用栈维护变量以及函数返回地址。

按照规则，saved regs 由被访问者进行维护，temp regs 由访问者进行保存。

1.10 机器级表示