



计算机体系架构

作者：Pannenets.F

时间：December 12, 2020

分类：笔记

Je reviendrai et je serai des millions. ——«Spartacus»

特别声明

北航微电子学院在 2020 年秋季学期开设的计算机体系结构课程，课程教师为成元庆老师。

Pannenets F December 12, 2020

目录

1	课程介绍	1
1.1	课程简介	1
1.2	电脑是智能的吗?	1
1.3	计算机表示的层次	1
1.4	计算机的结构	2
1.5	集成电路	2
1.6	教学方向	2
2	数据表示	4
2.1	表示方式	4
3	MIPS 导论: 汇编指令集	5
3.1	什么是汇编语言?	5
3.2	指令集 (Instruction Set Architectures)	5
3.3	运算指令格式	5
3.4	内存与寄存器	6
3.5	数据对齐	6
3.6	条件分支	6
3.7	对字节的操作	6
3.8	逻辑操作	6
3.9	函数调用	7
3.10	机器级表示	7
4	浮点数	8
4.1	定点数	8
4.2	二进制科学计数法	8
4.3	IEEE 标准	8
5	反汇编	9
5.1	反汇编	9
5.2	人工指令	9
6	运行程序	10
6.1	解释与编译	10
6.2	链接与加载	10

7 硬件实现	11
7.1 组合逻辑电路	11
7.2 CPU 设计	11

第一章 课程介绍

任课教师：成元庆

联系方式：yuanqing@ieee.org 18911370169

课程助教：倪嘉诚

1.1 课程简介

计算机体系架构是连接软硬件的一门课程，向下连接集成电路设计，向上联系计算机高层的软件、编译器、操作系统等。

近年来，出现了几种新型的 CPU 架构，但是整体来说，设计思路没有大的变化，本质来讲，目前的计算架构采用的仍然是冯·诺依曼架构。

本课程采用 MIPS 架构进行讲解，与 RISC-V 以及 ARM 等架构几乎同源，我国的龙芯项目也是采用的本指令集。教材采用《Computer Organization and Design》。

本课程的课件采用英文编写，因为大量的术语尚未有确切的中文翻译，并且大量的文献都是英文编写。

本课程有一定的前置课程，采用 C 语言进行教授，需要使用到部分数据结构的知识以及一些数字逻辑设计。

1.2 电脑是智能的吗？

对于编程者，是不需要考虑具体的实现的，如考虑数据管理、函数调用等操作，编程者可以按照思维设计软件的结构。但是对于最底层的电路来说，实际支持的可能只有与或非等基本操作，只支持电路的二值化，硬件并不会自动的进行上述的这些工作。由操作系统这种底层的软件进行一系列的复杂调度，来作为沟通软硬件的接口。层次划分特点，为计算机体系提供了广泛的抽象，这种抽象可以使不同层次的使用者不需要了解过于细节的实现。

软件的执行需要最终通过汇编器翻译成机器码，执行机器码需要操作系统对硬件的调用。

1.3 计算机表示的层次

高等语言会通过 Compile 转换为汇编语言，再通过 Assemble 转为机器码，准备在机器上执行。硬件的架构也是一种抽象，如各种模块 ALU、RegFile 等，最终需要落实到不同的逻辑门上。

1.4 计算机的结构

计算机的结构基本如下：

- 处理器
 - 控制部分
 - 数据通路
- 主存
- 输入
- 输出

1.5 集成电路

对于裸片（Bare Die）存在不同的 CMOS 工艺，最终需要封装到 PCB 板上。PCB 一般是树脂或者塑料的衬板。到目前来说，集成电路的发展基本吻合摩尔定律（2X/18 Months）。单位处理器的速度以 1.20-1.52 倍每年的倍率上升，但是随着单核处理器的某些瓶颈出现，速度逐渐放缓。

DRAM 也就是动态随机存储器，用于计算机的主存从 1980 年代到如今存储密度已经上升了超过 8000 倍。

1.6 教学方向

计算机体系的速度限制是什么？

基本的教学方向：

- 计算机层级的抽象
- 五个基本的计算机组成部分

具体来说

- 计算机的机器表示
 - 数字表示
 - 汇编语言
 - 编译与汇编
- 处理器与硬件
 - 逻辑电路设计
 - CPU 组织
 - 流水线
- 存储组织
 - 缓存
 - 虚拟内存
- 输入输出

- 中断
- 硬盘与网络
- 其他
 - 性能
 - 虚拟化
 - 并行化

单词

brawn 肌肉

anatomy 解剖

chassis 底盘

dramatic 急剧的

第二章 数据表示

数据需要数字化之后才可以在计算机进行表示。核心的折中点：如何用尽可能少的位尽可能精确表示一个数字？

2.1 表示方式

定义 2.1 (原码) $s|bbb$: 数字大小的变化方向不一致，存在两个 0：0000，1111

定义 2.2 (反码) $sssbb$: 数字大小的变化方向一致，存在两个 0：0000，1111

定义 2.3 (补码) $sbbbb$: 数字大小的变化方向一致，存在一个 0：0000

在进行位变换时，需要进行符号扩展。

第三章 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。

3.1 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。

3.2 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（#）。

3.3 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。

3.4 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`

3.5 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。

3.6 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`)。

3.7 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）。

3.8 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。

3.9 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。

3.10 机器级表示

I, J, R 形式的指令格式分别对应立即数、跳转与寄存器。

对 R 指令来说：

- opcode: 6, 指令码
- rs: 5, 第一操作数
- rt: 5, 第二操作数
- rd: 5, 目标操作数
- shamt: 5, 移位量
- funct: 函数

但是 `mult` 与 `div` 的目的寄存器在 `hi` 和 `low`。

I 型指令：

- opcode: 6, 指令码
- rs: 5, 第一操作数
- rt: 5, 第二操作数
- immediate: 16

为了使用高于 16 位的立即数，汇编器会为我们自动转换一部分数据，这用到了 `$at` (assembler temporary)，因此使用者不应该使用它。

由于立即数的限制，条件跳转存在跳转的范围，在上下文进行相对寻址。

J 型指令使用 PC 的计数器的高位以及乘四的地址。

第四章 浮点数

4.1 定点数

定点数的表示方法与整型数据基本一致，只是计量基准从 1 转换为 2^k 。

4.2 二进制科学计数法

表示为以有效数字 1 开头的尾数乘以指数。

规格化的数决定了有效性。

4.3 IEEE 标准

Float 类型 1 位符号，8 位阶码，23 位尾数。存在向上向下两种溢出。

对于规格化的值，exp 不全为 0 或 1，此时阶码为 $E = e - Bias$ ，尾数隐含 1。

对于非规格化的值，exp 全为 0，此时阶码为 $E = 1 - Bias$ ，尾数不含隐藏位。

对于特殊值，exp 全为 1，若尾数为 0，则为无穷，不为 0 为 NaN。

Word

mantissa 尾数

第五章 反汇编

5.1 反汇编

可以通过二进制数的方式进行反汇编，得到原来的汇编代码。

5.2 人工指令

有些满足二进制指令代码的格式要求的代码可以转换为真汇编指令。

第六章 运行程序

6.1 解释与编译

直接将源代码解释为可执行的文件。如 Java 转换为 Byte Code 。为了提高解释性程序的性能，通过统计进行优化，可以将自己编译成对应平台的二进制码。解释型的语言有跨平台的特性。

编译性的程序报错更多，错误更难定位与调试。而优化后的代码会产生源码的不对应。最终产生的可执行文件相当小。

对于汇编，需要进行扫描来确定不确定的那些地址。对于 Object 中的绝对地址，只能通过链接器进行确定，具体由符号表 (Symbol Table) 实现。一些不可到达的大距离跳转，以及改变的被分配地址，通过 Relocation Table 重定位得到绝对地址。

对于可执行文件需要有头部信息，代码段，数据段，调试信息。

Java 使用解释器与翻译器的原因是跨平台支持。

6.2 链接与加载

链接可以提高编译的效率和可维护性。

为了解决函数的引用问题，从源代码、库进行搜索。

静态链接库可以方便的运行，但是打包了所有的库，体积较大。动态库虽然需要反复的加载，但是速度一般更快。

加载器加载库并运行。会加载命令行参数、清空寄存器、跳转到开始代码加载参数之后调用主函数。

第七章 硬件实现

对于硬件系统来说，一般分为同步系统和异步系统，同步系统可以做到更高的速度。

逻辑设计的技巧与规则对系统性能的影响十分明显。通过逻辑设计的语言进行硬件的描述，来实现整体的功能。

硬件和软件是相互耦合的，也因此可以进行优化，但是也出现了相应的平台依赖问题。

7.1 组合逻辑电路

组合逻辑电路都可以通过真值表实现。

7.2 CPU 设计

数据通路是对 ISA 的模块化实现。

过程可以分为

- 取指
- 译码
- 读存
- 计算
- 写存

术语

MMU: Memory Management Unit

最后的扯淡

讲道理，这课都是抄的 UCB 的，考试也是直接拷过来的，所以建议把 20 年真题刷一遍就完事了！