Lecture 3 指令系统体系结构1

1. 指令系统体系结构

指令集

要控制一台计算机的硬件，你必须能说它的语言

- **指令 instruction**：计算机语言的单词
- **指令集 instruction set**：命令的词汇表
  一个给定计算机体系结构所包含的指令集合

两种形式的指令集

- **汇编语言**：由人编写的语言
- **机器语言**：由计算机读取

一个程序(比如C语言)被编译成一个由机器指令组成的可执行程序

- 这个可执行程序也必须在未来的机器上运行
- 每个英特尔处理器读取相同的x86指令，但是每个处理器处理指令的方式不同

Java程序被转换成可移植的字节码，在执行期间转换成机器指令(即时编译)

指令集的设计目标

- **易于构建**：硬件和编译器
- **性能最大化**，成本和能源最小化

指令集的设计原则

- **保持硬件简单**：芯片必须只实现基本的原语并运行快速
- **保持指令的规则性**：简化指令的解码/调度

指令集的相似性

不同机器的指令集是**相似**的，因为

- 基于基本原理相似的硬件技术
- 提供了一些基本操作
- 计算机设计师有一个共同的目标：找到一种语言，可以方便硬件和编译器的设计，且使性能最佳，成本和功耗最低

RISC和CISC

- **RISC精简指令集计算机**：MIPS, ARM, PowerPC, RISC-V
- **CISC复杂指令集计算机**：x86

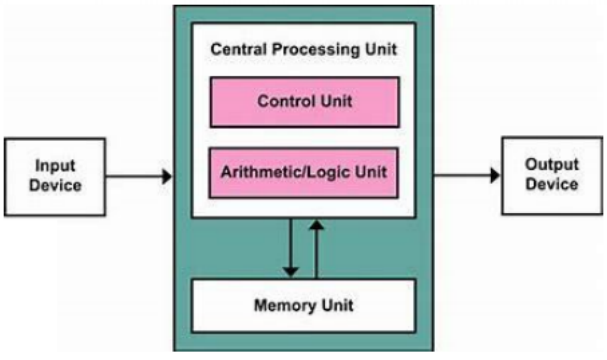
所有的指令集都是相似的

- 一旦学会了一个，就容易学会其它的

冯诺依曼架构

存储程序计算机

- **指令**和**许多类型的数据**可以以数字的形式存储在存储器中



Von Neumann architecture

广泛使用的个人计算机多为冯诺依架构的

哈佛架构

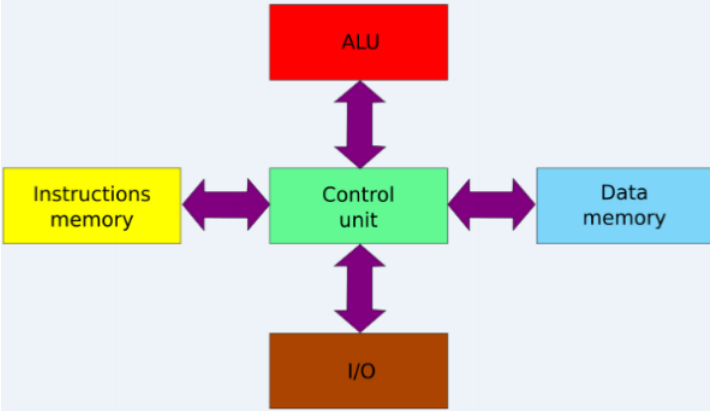
两个独立的内存

- 存储数据的内存
- 存储指令的内存

两种分开的导线

- 数据导线
- 内存导线

比冯·诺伊曼更高效，广泛应用于**嵌入式系统**embedded systems



硬件设计原则

简单和正则性

指令格式简单，形式固定

- 正则性使实现更简单
- 简单性以更低的成本实现更高的性能

越小越快

寄存器个数被严格限制，更大的寄存器可能会使时钟周期边长，花费更长的时间

- 寄存器和内存
- 寄存器的数量很小

加快经常性事件的速度

存在立即数，$zero 寄存器加速大概率事件

- 小的常数是常见的
- 直接操作数避免了load指令（addi）

好的设计需要好的妥协

不同指令的格式存在区别，但长度都是 32-bit

- 不同的格式会使解码复杂化，但却允许统一使用32位指令
- 保持尽可能相似的格式

2. MIPS 操作数 operand

在C语言中，每个“变量”都是内存中的一个位置

在硬件中，每次内存访问都是昂贵的，如果变量 a 被重复访问，它有助于将变量带入芯片上的超高速中间结果存储器并对该寄存器进行操作

注意：在C程序中操作数（变量）的数量非常大，**汇编中的操作数是固定的**，寄存器的数量是有限的

名字	示例	注释
32个寄存器 Register	$s0-$s7,$t0-$t9,$zero $a0-$a3,$v0-$v1	寄存器用于数据的快速存取，在MIPS中，只能对存放在**存储器**中的数据进行算出操作
2^{30}个存储器字 Memory word	Memory[0],Memory[1],... Memory[4294967292]	存储器只能通过数据传输指令访问 MIPS使用字节（1代表一个字节 byte）编址，所以连续的字的地址相差4

MIPS存储约定

MIPS体系结构中，**寄存器、存储器的大小为 32-bit**，由于其经常出现，故在MIPS中被统一称为**字 word**

- **字 word**：**计算机中的基本访问单位**，通常 32-bit 为一组，在MIPS体系结构中与寄存器的 大小相同

寄存器 Registers

寄存器用于数据的快速存取，在MIPS中，只能对存放在**存储器**中的数据进行算出操作

注意

- MIPS中**算数运算指令**的操作数必须来自于**寄存器**
- 寄存器由硬件直接构建且数量**有限**

MIPS ISA有**32个寄存器**（x86有8个寄存器）

- 为什么不更多一些：因为多了需要更多的位宽来表示寄存器，大量的寄存器会使时钟周期变长，且受到指令格式位数限制
- 为什么不更少一些：因为需要更多的移动，会很麻烦

寄存器的种类

名称	寄存器号	用途	调用时是否保存
$zero	0	常数0	不适用
$v0 ~ $v1	2 ~ 3	计算结果和表达式求值	否
$a0 ~ $a3	4 ~ 7	参数	否
$t0 ~ $t7	8 ~ 15	临时变量	否
$s0 ~ $s7	16 ~ 23	保存的寄存器	是
$t8 ~ $t9	24 ~ 25	更多临时变量	否
$gp	28	全局指针	是
$sp	29	栈指针	是
$fp	30	帧指针	是
$ra	31	返回地址	是

除此之外

- $at：被汇编器保留
- $k0~$k1：被操作系统保留

存储器 Memory

处理器只能将少量的数据保存在寄存器中，而存储器有数十亿的数据，因此数据结构（数组和结构）是存放在存储器里的

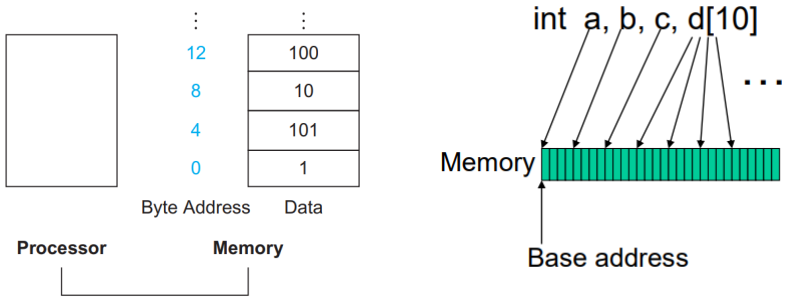
MIPS 包含在存储器和寄存器之间传送数据的指令，这些指令叫做**数据传送指令 data transfer instruction**，为了访问在存储器中的一个字 word，指令必须给出**存储器的地址 address**

地址 address

存储器是一个很大的，下标从 0 开始的一维数组，地址就相当于数组的下标

MIPS 中存储器是**按字节编址**的（地址中的一个 1 对应着 1 个字节 byte），因为存储器也是存放着**字 word** 的，所以地址是 4 个 4 个存放的

一个字 word 的地址必须和它所包括的 4 字节中的某个地址相匹配，且连续字的地址相差 4，所有**字的起始地址必须是 4 的倍数**，这叫做**对齐限制 alignment restriction**



如图所示，其中第三个字 word 的字节地址为 8

有两种字节寻址的计算机

- **最左边（大端 big end）**：使用最左边字节的地址作为字地址
  MIPS 使用**最大端编址**
- **最右边（小端 little end）**：使用最右边字节的地址作为字地址

常数/立即操作数 Constant/Immediate Operands

程序中经常会在某个操作**使用到常数**，包含常数的算数运算**指令执行速度快且能耗低**，

直接指令使用一个常量作为输入之一（而不是寄存器操作数）

根据使用频率来定义常数是**加速大概率事件**的另一个好办法

MIPS支持的常数/立即操作数

- **常数 0** 还有另外的作用，有效使用它可以简化指令集，MIPS 将寄存器 $zero 恒置为 0
- MIPS 支持负常数

寄存器和存储器的比较

寄存器的访问速度比内存快

操作存储器数据需要加载和存储

- 更多的指令要执行

编译器必须尽可能多地使用寄存器

- 只有那些不太常用的变量才会溢出到存储器中
- 寄存器优化很重要

3. 数字表示

计算机硬件中，所有信息都是由二**进制位 bit** 组成的，因此二进制数运算的基本单位是 bit

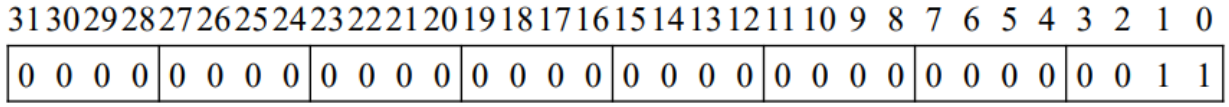
在 MIPS 中，字的大小表示为

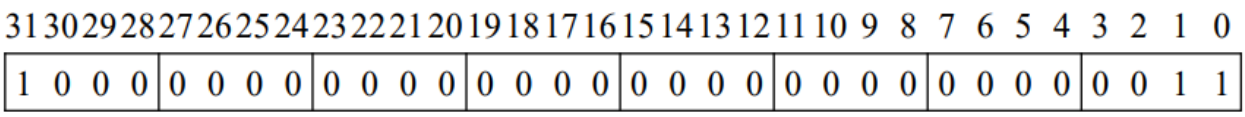
- **最低有效位 least significant bit**：MIPS 字中最右边那一 bit
- **最高有效位 most significant bit**：MIPS 字中最左边那一 bit

MIPS 的字有 32-bit，可以表示 2^{32} 个不同的 32 bit 模式，可以自然组合表示出从 0 到 $2^{32}-1$ 之间的数

$$
\begin{aligned}
&0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 &&= 0_{10}\\
&0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 &&= 1_{10}\\
&0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 &&= 2_{10}\\
&\ldots &&\ldots\\
&1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 &&= 4{,}294{,}967{,}293_{10}\\
&1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 &&= 4{,}294{,}967{,}294_{10}\\
&1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 &&= 4{,}294{,}967{,}295_{10}
\end{aligned}
$$

假设寄存器`$s0`中的位如下，s0的值是多少？





- 第一个：3

- 第二个：要根据有无符号来判定

一般数字表示

- 十进制Decimal：35_{10}
- 二进制Binary：00100011_2
- 十六进制Hexadecimal：23_{hex}

无符号数

给定一个n比特的数字

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \ldots + x_1 2^1 + x_0 2^0$$

范围

$$[0 \sim 2^n - 1]$$

如果是32比特，范围在0到+4,294,967,295

有符号数

如何表示正负号需要一种计算方法，在没有明显更好的解决方案的情况下，选择了一种易于硬件实现的表示方式

前导位为 0 表示正数，前导位为 1 表示负数，这种表示有符号二进制数的方法称为**二进制补码 two's complement**，在看二进制补码之前，我们先来了解一下二进制反码

二进制反码 1's complement

就是把所有的0变成1，1变成0

所以

$$2's\ \ complement = 1's\ \ complement + 1$$

示例

$+2 = 00000000\ldots0010_2$

$-2 = 11111111\ldots1101_2 + 1 == 11111111\ldots1110_2$

二进制补码 2’s complement

给定一个n比特的数字，定义如下值

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \ldots + x_1 2^1 + x_0 2^0$$

范围

$$[0, 2^{n-1} - 1]$$
$$[-2^{n-1}, -1]$$

- 注意，二进制补码中的最小负数 -2^{31} **没有相应的正数对应**

采用二进制补码的优点在于**所有负数的最高有效位都是 1**，硬件只需要检测这一位就知道一个数是正数还是负数

溢出检测

- 该数是负数时符号位是 0
- 该数是正数时符号位是 1

符号拓展

对 MIPS 取数指令来说，有符号数和无符号数是有区别的

将一个数字拓展到更高的比特位时，我们希望

- 保留数值
- 从寄存器取数 8-bit，然后想要扩展它至 16-bit 或 32-bit

MIPS 提供的多种取数方法

- lw（取字）：本身就是 32-bit 的字，不需要符号拓展
- lh（取半字）：取出 16-bit 的半字，需要符号拓展（有符号）
- lb（取字节）：取出 8-bit 的字节，需要符号拓展（有符号）
- addi（扩展立即值）：拓展立即值到寄存器（有符号）
- lbu（取字节）：取出 8-bit 的字节，需要符号拓展（无符号）

符号拓展方法

在左边复制符号位

- 无符号数：扩展0
- 有符号数：扩展符号位

示例 8bit扩展到16bit （有符号）

$+2 : 00000010 => 0000000000000010$

$-2 : 11111110 => 1111111111111110$

4. 指令格式

机器指令

当今计算机基于以下两个重要的原则构建

1. 指令用数的形式表示
2. 和数据一样，程序存储在存储器中，并且可以读写

指令用32-bit(一个字 word)表示

机器指令分为若干字段 field，指令的布局叫做指令格式 instruction format，指令的数字形式称为机器语言machine language，这样的指令序列叫做机器码 machine code

为了避免二进制字串过于冗长，机器码又通常采用16进制表示

寄存器映射

在 MIPS 中

- $zero 映射到 0
- $v0~$v1 映射到 2~3
- $a0~$a3 映射到 4~7
- $t0~$t7 映射到 8~15
- $s0~$s7 映射到 16~23
- $t8~$t9 映射到 24-25
- $gp 映射到 28
- $sp 映射到 29

- $fp 映射到 30
- $ra 映射到 32

R-指令 算数指令格式

用于**基本指令**

OP	RS	RT	RD	SHAMT	FUNCT
6bits	5bits	5bits	5bits	5bits	6bits
操作码 opcode	第一个源操作寄存器	第二个源操作寄存器	用于存放结果的目的寄存器	位移量	功能码，用于指明op字段中操作特定变式

I-指令 数据传输指令格式

用于**立即数/数据传送指令**

OP	RS	RT	CONS
6bits	5bits	5bits	16bit
操作码opcode	第一个源操作寄存器	用于存放结果的目的寄存器	表示地址字段

J-指令 分支跳转指令格式

OP	CONS
6bits	26bit
操作码 opcode	表示地址字段

MIPS转机器语言示例

MIPS 机器语言

名字	格式	举例						注释
add	R	0	18	19	17	0	32	add $s1, $s2, $s3
sub	R	0	18	19	17	0	34	sub $s1, $s2, $s3
addi	I	8	18	17	100			addi $s1, $s2, 100
lw	I	35	18	17	100			lw $s1, 100($s2)
sw	I	43	18	17	100			sw $s1, 100($s2)
字段宽度		6 位	5 位	5 位	5 位	5 位	6 位	所有 MIPS 指令均为32 位
R 型	R	op	rs	rt	rd	shamt	funct	算术指令格式
I 型	I	op	rs	rt	address			数据传送指令格式

例题1 将MIPS汇编语言翻译成机器语言

下面三条 MIPS 指令的机器语言代码是什么

```
lw $t0, 1200($t1)
add $t0, $s2, $t0
sw $t0, 1200($t1)
```

机器语言（十进制表示）：

OP	RS	RT	CONS
35	9	8	1200

OP	RS	RT	RD	SHAMT	FUNCT
0	18	8	8	0	32

OP	RS	RT	CONS
43	9	8	1200