

# 计算机组成理论第四期

## —指令集架构（一）—

大连理工大学立命馆大学国际信息软件学部大森孝之

# 讲座内容

---

## 指令集架构

- 概述

## MIPS

- 寄存器和基本算术指令存储器访问指令

-

# 从源代码到机器语言

转贴

C语言

```
int a,b,c,d;  
a = b + c;  
d = a * b;  
...
```

照原样  
跑不起来

汇编语言

```
lw $9, 4($1)  
lw $10, 8($1)  
add $8,$9,$10  
...
```

编译器  
转换者

机器语言

```
01001101  
00100100  
01001000  
00100000
```

汇编器  
转换者

在电路  
上可以  
执行

- 高级语言程序是编译器，通过汇编程序等转换为机器语言代码。
- 汇编语言和机器语言指令一一对应

# 电脑字

---

- 操作说明：

- 计算机语言词

- 计算机可以解释哪些词

- 指令系统：

- 计算机词汇（一组指令）

# RISC 和

---

## ■ 指令集设计策略

- 增加您可以通过一条指令执行的操作

- 用一条指令简化您的工作

## ■ RISC（精简指令集计算机）一条指令能做的事情很少，但很简单

- 每条指令执行速度快

## ■ CISC（复杂指令集计算机）一条指令可以做很多事情，但是很复杂

- 每条指令的执行速度很慢

## ■ 近年来，两者之间的差异正在消失。

# MIPS

- 由 MIPS 计算机系统（现为 MIPS 技术）开发的指令集
  - 适用于 NINTENDO64、PlayStation 等。
  - 1条指令是32位（最新版本是64位）
  - 字：计算机处理的数据量的单位
    - 32 位用于 MIPS
    - = 寄存器或 ALU 一次处理的数据的位宽

指令

**add a, b, c**

**sub d, e, f**

每个“变量”的值是实际存储在寄存器中  
... 将 b+c 的结果存入 a

... 将 e-f 的结果存入 d

MIPS汇编指令 → 可翻译成与机器语言一一对应

■ 每条指令的操作对象称为操作数。

■ 如果加上a, b, c, 则有三个操作数(a, b, c)

■ 如果你把三个数字相加, 你必须把它们分成两个指令

**add a, b, c      #a←b+c**

**add a, a, d      #a←a+d**

设计原则  
(一): 简单导致规律

# 指令集差异（示例）

## ■ MIPS 加减法指令

■ `add a, b, c`

将#  $b + c$  的结果存入

■ 一个子 `a, b, c`

将#  $b - c$  的结果存入 `a`

## ■ x86 加法/减法指令

■ `add a, b`

将#  $a + b$  的结果存入 `a`

■ `sub a, b`

将#  $a - b$  的结果存入 `a`



# 登记

**add a, b, c**



**add \$s0, \$s1, \$s2**

每个“变量”的值是  
实际上存储在寄存器中

寄存器\$s1和\$s2中的  
值

## ■ 登记

- 用于存储 CPU 中要计算的数据的存储区
- 快速访问

## ■ 在 MIPS 中，寄存器是 32bit

- 字：一种数据量单位（32 位），表示寄存器或 ALU 一次处理的位宽。

## ■ 有32个寄存器（+a）

# 确认问题

- 回答符合下列解释的术语。
- (1) 计算机语言词
- (2) 计算机词汇 (一组指令)
- (3) 一条指令能做的事情很少, 简化指令的指令集设计策略
- (4) 通常你可以用一条指令来做, 使指令复杂化的指令集设计策略
- (5) MIPS 技术 (原 MIPS 计算机) 由 Systems 开发的指令集)
- (6) 将要计算的数据存储在 CPU 中 储藏区域
- (7) 表示寄存器或 ALU 一次处理的位宽的数据量单位
- (8) 指令的操作目标



翻译狗  
www.fanyigou.com  
www.translateGo.com

# 讲座内容

---

## ■ 指令集架构

### ■ 概述

## ■ MIPS

### ➡ ■ 寄存器和基本算术指令存储器访问指令

# 注册转帖

**add a, b, c**



**add \$s0, \$s1, \$s2**

每个“变量”的值是  
实际上存储在寄存器中


寄存器\$s1和\$s2中的  
值

- 登记
  - 用于存储 CPU 中要计算的数据的存储区
  - 快速访问
- 在 MIPS 中，寄存器是 32bit
  - 字：寄存器或ALU一次处理的位宽
  - 显示的数据量单位（32bit）
- 有32个寄存器

# 寄存器类型

\$zero	0	始终为零
\$at	1	汇编程序临时使用
\$v0-v1	2-3	对于返回值
\$a0-a3	4-7	对于参数
\$t0-t9	8-15, 24-25	临时寄存器（用于临时变量）
\$s0-s7	16-23	保存寄存器（用于变量）
\$k0-k1	26-27	为操作系统内核保留
\$gp	28	全局指针
\$sp	29	堆栈指针
\$fp	30	帧指针
\$ra	31	退货地址

# 寄存器数

- 寄存器很快
  - 那么，增加寄存器的数量是否更好？
  - 如果增加寄存器的数量。
    - 电路变大了
    - 不得不降低时钟频率，这增加了布线长度和信号延迟。
- 
- 设计原则2：越小越快

## (示例) 使用寄存器

### 编译 C 赋值语句

**f = (g+h) - (i+j) ;**



```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

f: \$s0  
g: \$s1  
h: \$s2  
i: \$s3  
j: \$s4

- 变量的值存放在保存寄存器 (\$s0-\$s7) 中，临时操作的结果就是临时寄存器。
- 存储在 (\$ t0- \$ t9)



# 基本算术指令

## ■ 添加

- 说明: 加  $\$s1$ 、 $\$s2$ 、 $\$s3$

- 含义:  $\$s1 = \$s2 + \$s3$

## ■ 减法

- 说明: `sub  $\$s1$ ,  $\$s2$ ,  $\$s3$`

- 含义:  $\$s1 = \$s2 - \$s3$

# 基本算术指令

## 逻辑运算和

- 说明: `and $s1, $s2, $s3`
- 含义:  $\$s1 = \$s2 \& \$s3$

## 逻辑运算或

- 说明: 或 `$s1, $s2, $s3`
- 含义:  $\$s1 = \$s2 \mid \$s3$

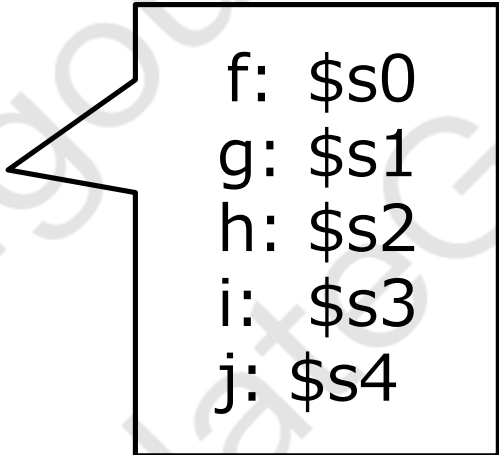
## 逻辑运算也

- 说明: `nor $s1, $s2, $s3`
- 含义:  $\$s1 = \sim(\$s2 \mid \$s3)$

# 确认问题

- 显示对应于以下汇编指令的 C 语句。

```
add $s0, $s1, $s2
sub $s0, $s0, $s3
add $s0, $s0, $s4
```



f:	\$s0
g:	\$s1
h:	\$s2
i:	\$s3
j:	\$s4

\* F, g, h, i, j 是  
预定义的

你可以把它想成一个



翻译狗  
www.fanyigou.com  
www.translateGo.com

# 讲座内容

---

## ■ 指令集架构

### ■ 概述

## ■ MIPS

### ■ 寄存器和基本算术指令存储器

### ➡ ■ 访问指令

# 内存访问

- 从内存中获取不适合寄存器的值或将其存储在内存中

$$g = h + \underline{A[8]}$$



```
lw $t0, 32($s3)  
# load word
```

$$\underline{A[8]} = g + h$$



```
sw $t0, 32($s3)  
# store word
```

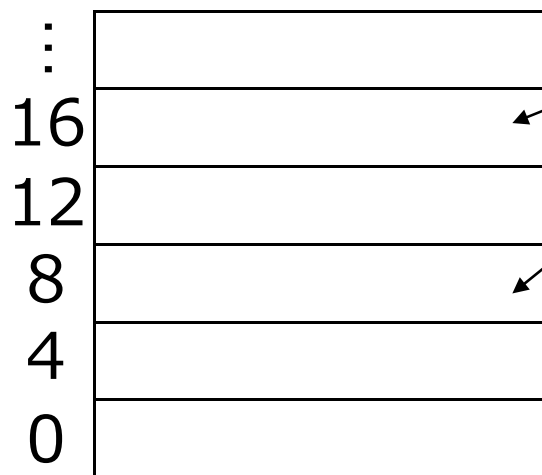
# 内存访问

■ **lw \$s1, 8(\$s2)**

■ **sw \$s1, 8(\$s2)**

抵消

基址寄存器



记忆

$8(\$s2) = 16$

如果  $\$s2 = 8$ ,

■ 每个内存单元都分配一个内存地址 (1字节)

■ 1 个字 = 4 个字节

■ 一般使用DRAM

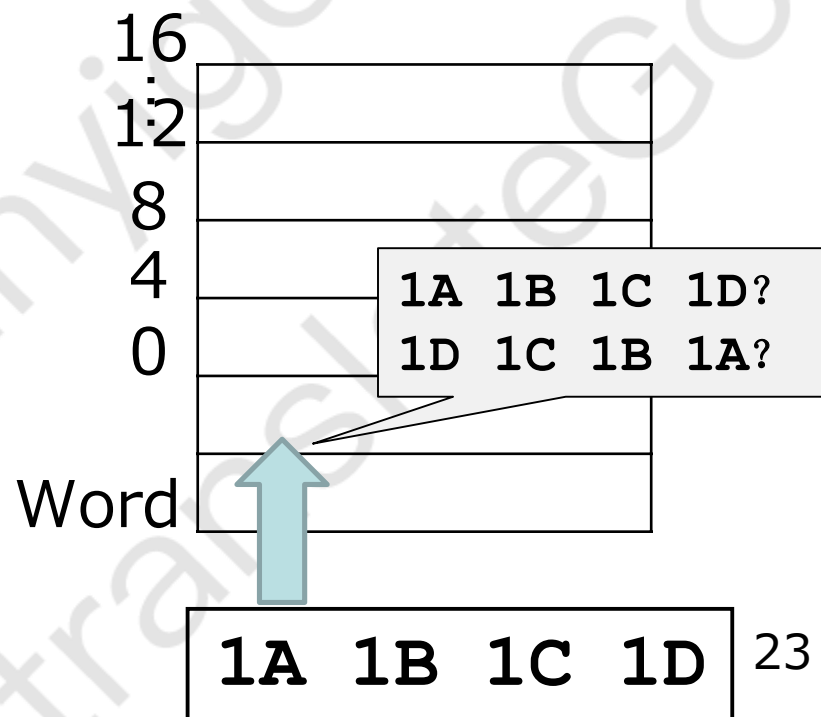
# 单词在内存中的存储

## ■ 对齐约束

- 在 MIPS 中，字必须放在 4 的倍数的地址的开头，以加快内存访问。

## ■ 储藏方法

- 大端大端
  - 从高位字节开始按顺序存储
- 小端小端
  - 从最低字节开始存储
- MIPS 上的大端





# 内存访问指令

## ■ 加载字（从内存传输到寄存器）

■ 说明: `lw $ s1 20 ($ s2)`

■ 含义:  $\$ s1 = \text{内存}[\$ s2 + 20]$

## ■ 存储字（从寄存器传输到内存）

■ 说明: `sw $s1 20 ($s2)`

■ 含义:  $\text{内存}[\$ s2 + 20] = \$ s1$

# 内置程序方法

- 用于在计算机上处理  
一种预先在内存中存储指令和数据的方法
- 当今所有计算机的基本概念
- 也称为诺依曼型

## C言語

```
int a,b,c,d;  
a = b + c;  
d = a * b;  
...
```

## アセンブリ言語

```
lw $9, 4($1)  
lw $10, 8($1)  
add $8,$9,$10  
...
```

## 機械語

```
01001101  
00100100  
01001000  
00100000
```

コンパイラ  
により変換

アセンブラ  
により変換

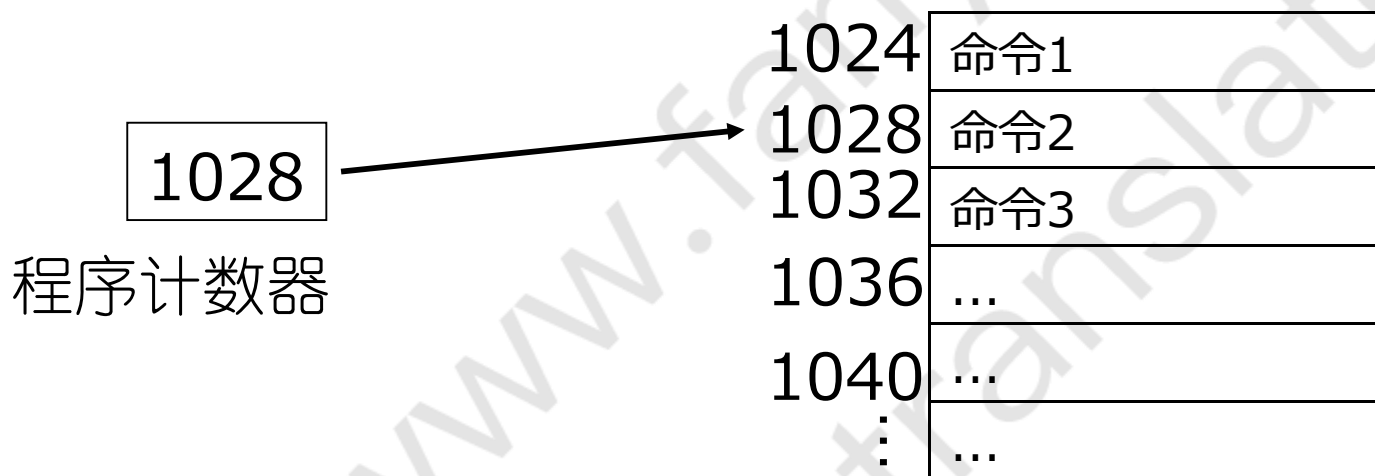


:	
16	...
12	...
8	...
4	...
0	01001101 0010...

记忆

# 程序计数器

- 当前执行的指令的寄存器指示内存地址
  - 根据指令执行的时序指示下一条指令的内存地址
- 也写成程序计数器，PC



# 确认问题

- 响应以下处理对应的MIPS算术指令
  - 添加
  - 减法
  - 从记忆中读出一个词
  - 写一个字记忆
- 内存访问执行多少字节？
- 从最高字节开始按顺序在内存中存储单词的方法叫什么
- 从最低字节开始按顺序在内存中存储单词的方法叫什么
- 下一步要执行的程序的内存地址叫什么
- 用于处理的数据和程序

什么叫预先存入内存的方法



翻译狗  
www.fanyigou.com  
www.translateGo.com

# 确认问题

- (1) 在 MIPS 中声明了一个 int 类型数组 A[100]。当存放 A[0] 的内存地址为 1000 时，回答存放 A[50] 的内存单元的起始地址。

- 提示: int 类型 → 4 字节 (32 位)

- (2) MIPS 是大端的。  
在上述 (1) 的情况下,  $A[0] = \text{FF332211}_{16}$   
当主存地址  
1000、1001、1002、1003 的内容  
用二进制回答每个。

1000	(A[0] 的 值)
	(A[1] 的 值)
?	...
	(A[50] 的 值)

1000	FF332211
	(A[1] 的 值)
	...
	(A[50] 的 值)



翻译狗  
www.fanyigou.com  
www.translateGo.com

# 参考

---

- Computer Configuration and Design 5th Edition by David A. Patterson, John L. Hennessy, 成田光明翻译, Nikkei BP