FSE598 前沿计算技术

模块 1　计算思维
单元 1　计算机系统设计
第 2 讲　计算机组成与架构

本讲座的英文版内容基于教材：
The English version of the lectures are partly based on the book:
Patterson and Hennessy, Computer Organization and Design: The Hardware Software Interface

本讲提要

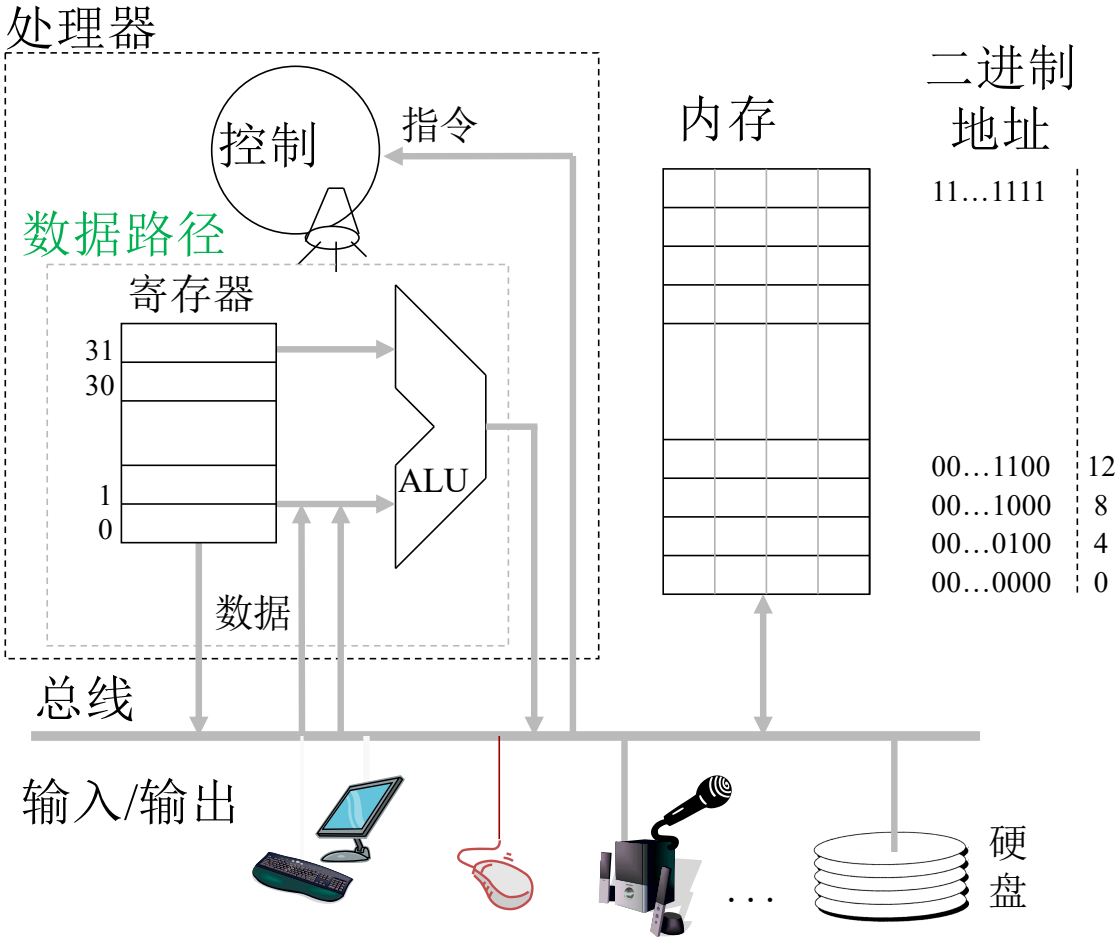
学习

- 计算机的主要部件
- 计算机系统的抽象和层次
- 性能
- 计算机组成及其部件
- 不同的架构与编程模型

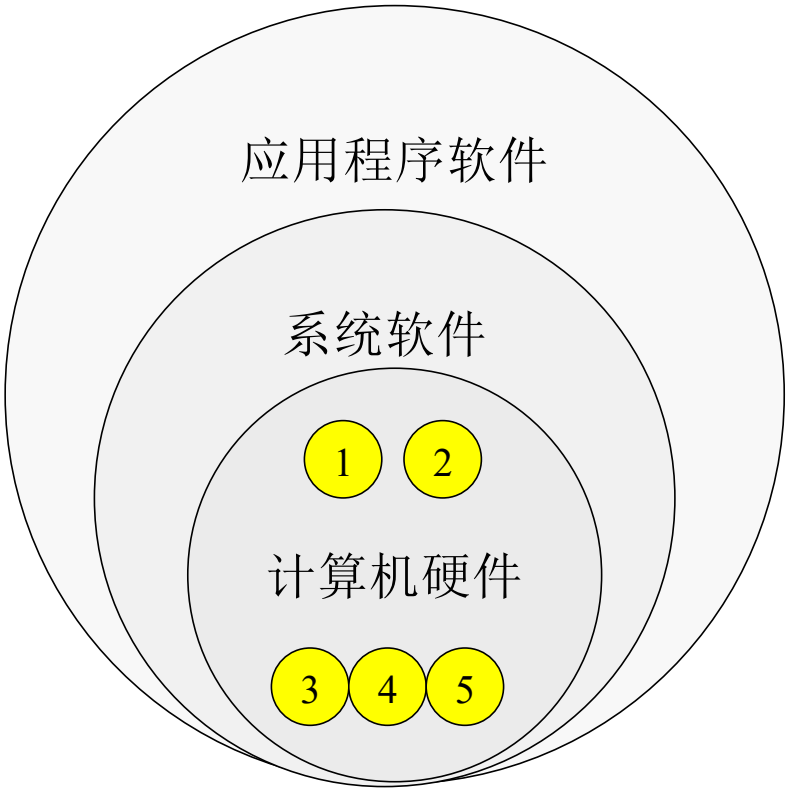
计算机的五个模块

- **CPU（中央处理器）：**控制整个系统并执行中央操作：
  1. **数据路径：**执行算术运算。主要功能单元是算术/逻辑单元 (ALU)。
  2. **控制单元：**根据程序指令的指示，告知数据路径、内存和输入/输出设备应该做什么。
- **内存模块：**存储指令和数据
- **外围设备**
  4. **输入模块：**接受输入并将其写入内存
  5. **输出模块：**从内存读取数据，并发送给外部

关于计算机模块的更多详情



硬件和软件分层简化视图

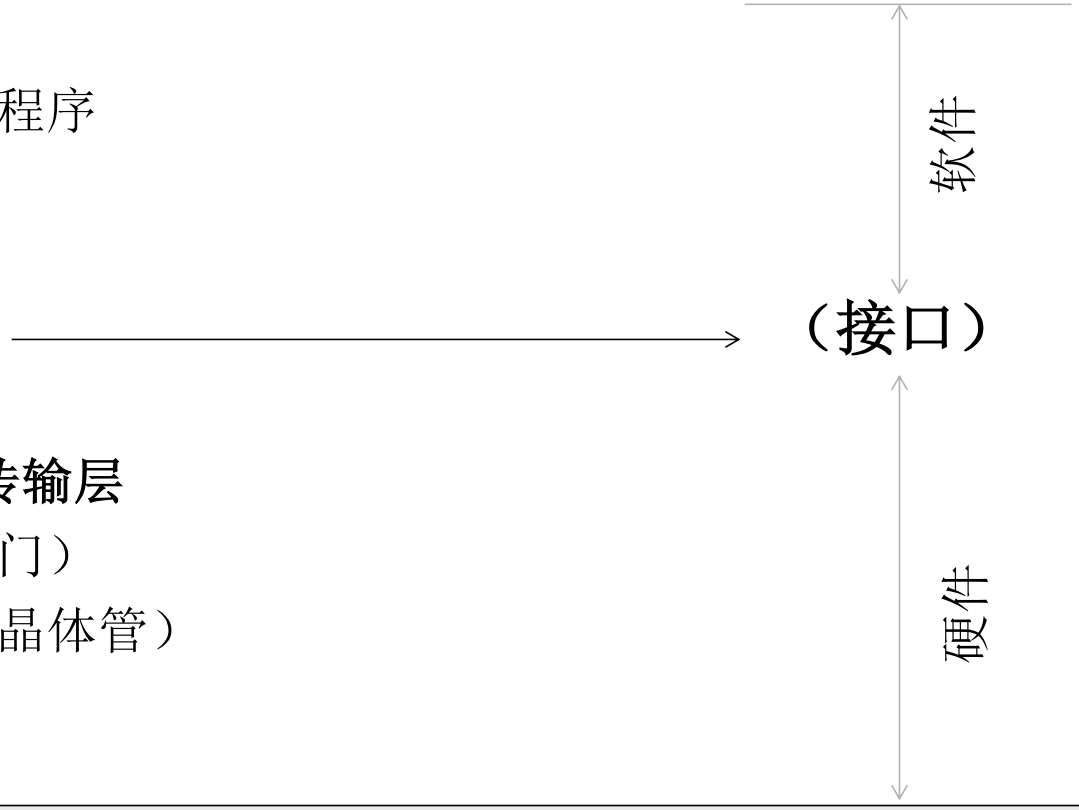


计算机系统的层（抽象技术）

应用程序软件（例如 Web 浏览器、游戏、文本编辑器）
系统软件（例如操作系统、编译器）
高级语言程序（例如 C++、Java、C#、Python）
汇编程序（例如 Intel、MIPS、Motorola Assembly ）
机器码
指令集 → **架构**
模块层
寄存器-传输层 → **组成**
逻辑层（门）
电路层（晶体管） → **实现**
半导体层
……

计算机系统的层（抽象技术）

应用程序软件
系统软件
高级语言程序
汇编程序
机器码
指令集
模块层
寄存器-传输层
逻辑层（门）
电路层（晶体管）
半导体层
……



计算机架构里程碑

- *存储程序概念*
  （冯诺依曼机）
  - 以数字表示的指令
  - 像数字一样存储在内存中的程序
- *系列概念*：将架构与其组成分离：相同的指令集，不同的实施方式。1964 年首次在 IBM S/360 中引入，随后是 DEC PDP-8， M68000 系列。
- *流水线*：将并行性引入指令执行的一种方法。
- 多核：当今大多数计算机所采用的技术

架构与组成分离

- 处理器设计的三个方面：
  - **架构**：处理器在指令集，机器语言/汇编语言级别的规范
  - **实施（组成）**：处理器在框图（门和模块）级别的描述
  - **实现**：电子/机械设备 (CMOS, TTL) 级别的描述
- 通过分离可以针对相同的处理器架构采用不同的实施方式。
- 分离还可以对相同的实施采取不同的实现方式。

架构里程碑（续）

- *CISC*（复杂指令集计算机）架构：强大的指令和寻址模式。编写汇编语言程序的可写性和表达性。
  M6800 - M68000、Intel 8086 – 486。
- *RISC*（精简指令集计算机）架构：指令集对底层硬件比对汇编语言程序更友好，对流水线更友好。
  PowerPC、MIPS、Pentium、Core、Multi-Core

计算机性能

从用户的角度来看性能

响应时间（对于用户程序）

- CPU 时间
  - 用户 CPU 时间
  - 系统 CPU 时间
- 输入/输入等待时间
- 运行其他程序时间

速度相关性能：

$$\text{性能} = \text{速度} = \frac{1}{\text{执行时间}}$$

$$\text{CPU 性能} = \frac{1}{\text{用户 CPU 时间}}$$

$$s = \frac{d}{t}$$

$$s = \frac{1}{t} \quad if \quad d = 1$$

相对性能

- 定义性能 = 1/执行时间
- “X 比 Y 快 *n* 倍”

$$\text{性能}_X / \text{性能}_Y = \text{执行时间}_Y / \text{执行时间}_X = n$$

- 示例：运行某个程序所需的时间
  - A 是 10 秒，B 是 15 秒
  - $\text{执行时间}_B / \text{执行时间}_A$ = 15 秒 / 10 秒 = 1.5
  - 因此，A 的速度是 B 的 1.5 倍

某程序的 CPU 时间

$$\text{CPU 时间} = \text{CPU 时钟周期} \times \text{时钟周期时间}$$

$$= \frac{\text{CPU 时钟周期}}{\text{时钟频率}}$$

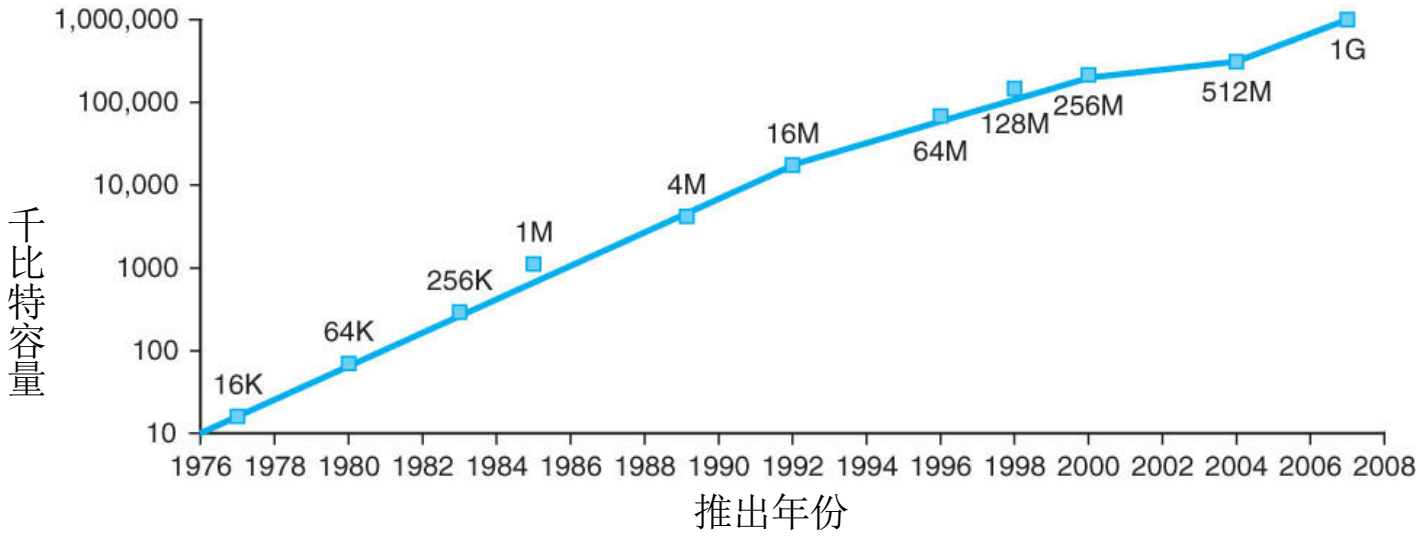
时钟周期

- 可以通过以下途径提高计算机的性能
  - 减少所需的时钟周期数
  - 提高时钟频率
  - 硬件设计人员必须经常会针对每条指令在时钟频率与周期数之间进行权衡

摩尔定律

- 英特尔联合创始人戈登·摩尔(Gordon E. Moore)在 1965 年写道：
  - 从 1958 年到 1965 年，集成电路中的元件数量每年翻一番；
  - 这一趋势将持续“至少十年”。
- 事实证明，他的预测非常准确，直到今天这一定律仍然适用！
- 这一定律现在用于半导体行业，为长期规划提供指导，同时也为研究和开发工作设定目标。
- 这一定律同样适用于其他半导体参数。

内存增长规则

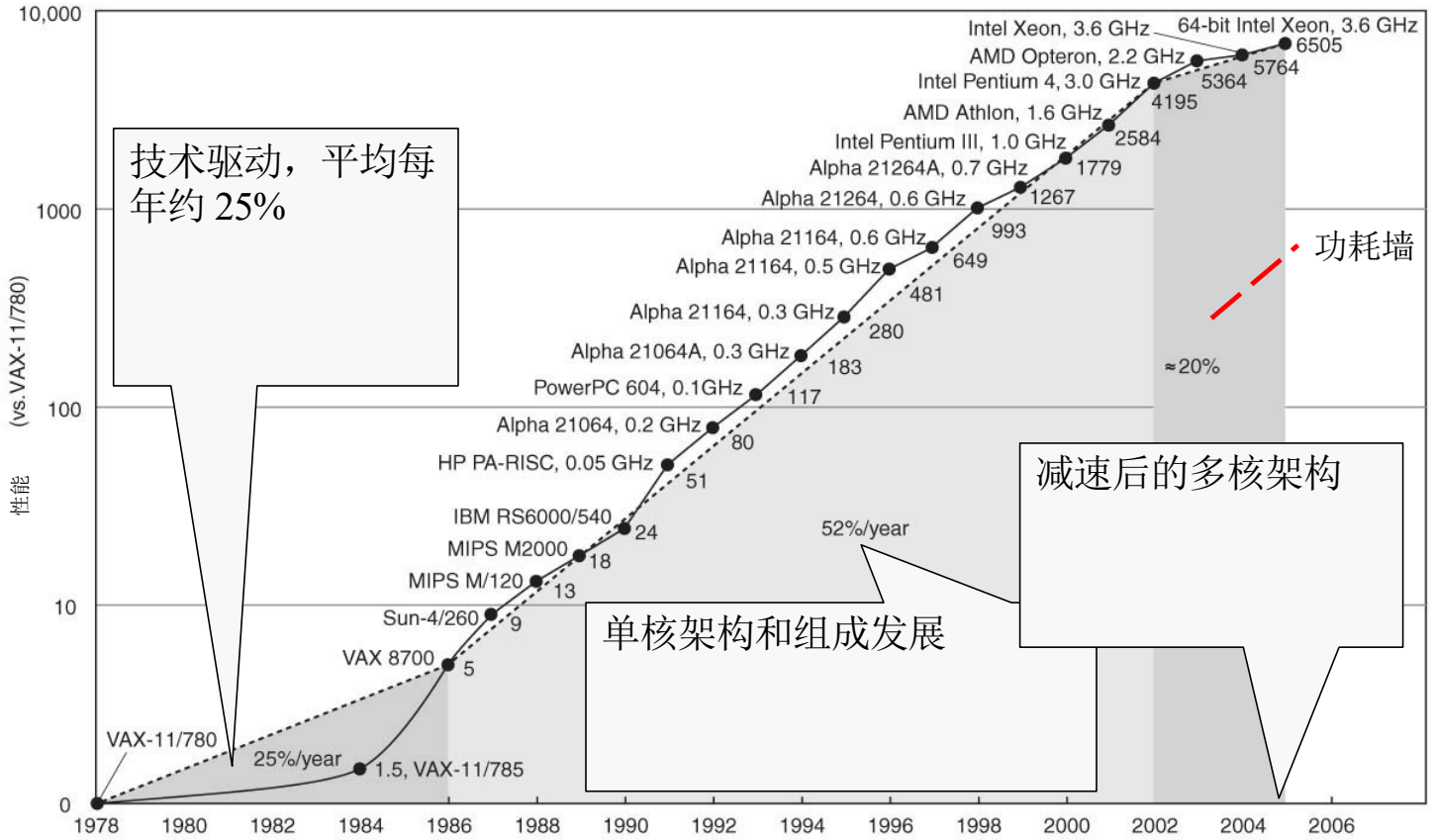


DRAM 增长规则：

DRAM 的容量几乎每三年翻两番，每年增长 60%，这一趋势持续了 20 年。近年来，这一速度有所减缓，基本上接近于每两到三年翻一番。

资料来源：Patterson 和 Hennessy，《计算机组成与设计：硬件/软件接口》(Computer Organization and Design: The Hardware Software Interface)

处理器性能/速度的增长



资料来源：Patterson 和 Hennessy，《计算机组成与设计：硬件/软件接口》(Computer Organization and Design: The Hardware Software Interface)

不同的架构

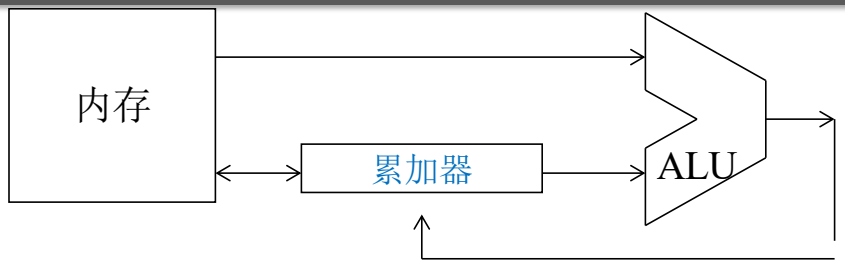
累加器

架构：

一个来自累加器

另一个来自内存

写回累加器

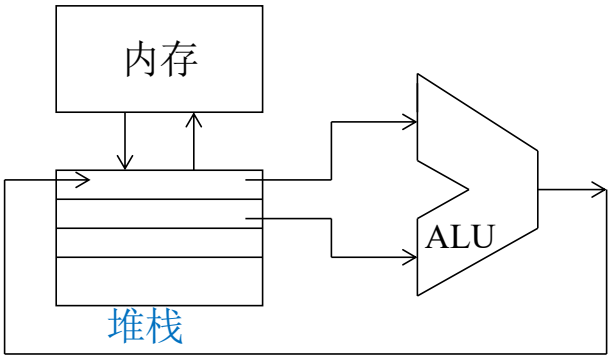


堆栈

架构

都来自堆栈

写回堆栈

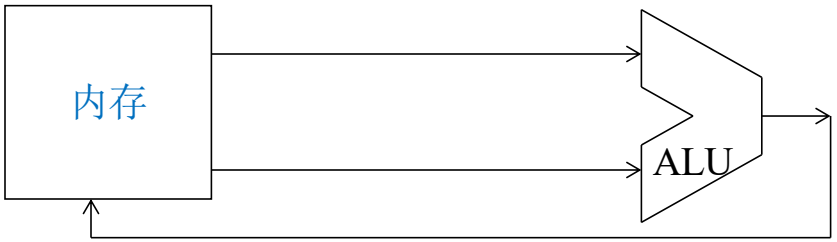


内存-内存

架构：

都来自内存

写回内存

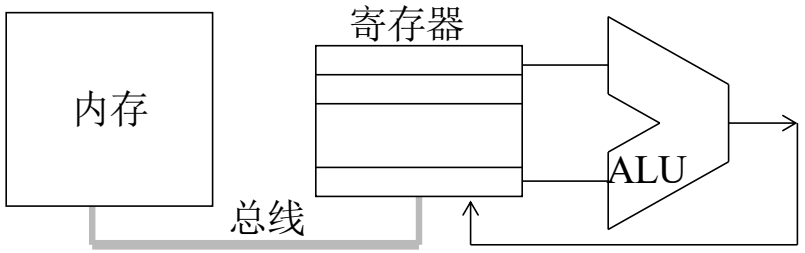


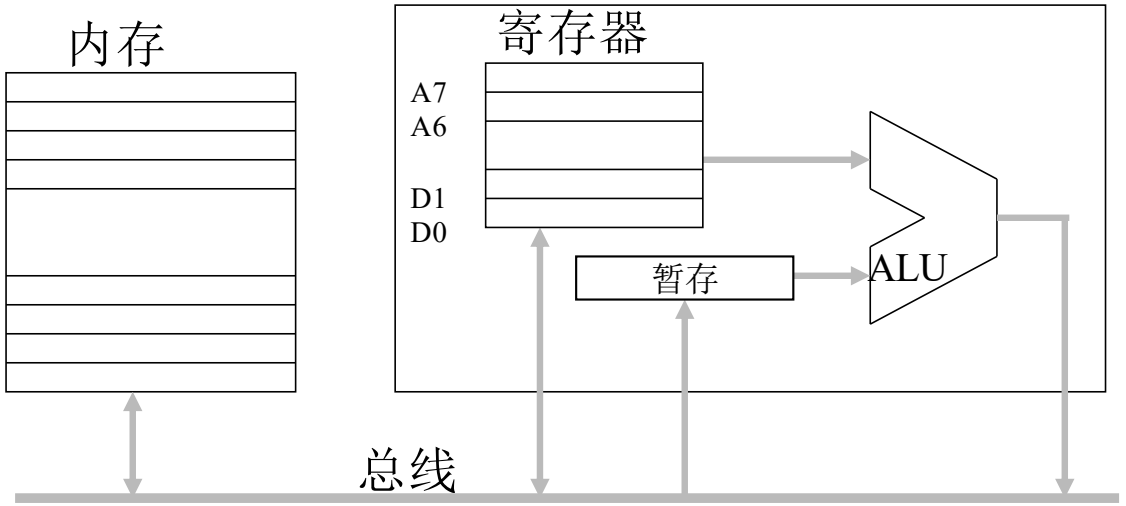
不同的架构 （续）

加载-存储
架构(RISC)

MIPS 架构
都来自寄存器
写回寄存器

复合
架构
(CISC)：
内存-
内存
与加载-存储
架构的组合





摩托罗拉 68000 架构

不同架构上的汇编语言编程

*编程任务：y = x1 * x2 + x3 / x4*

累加器		堆栈		M-M		加载-存储	
Load x1 *mult x2* *Store y* *Load x3* *div x4* *Add y* *Store y*		*push x1* *push x2* *mult* *push x3* *push x4* *div* *Add* *pop y*		*mult x1 x2* *div x3 x4* *Add x2 x4* *move x4 y*		*move x1 R1* *move x2 R2* *move x3 R3* *move x4 R4* *mult R1 R2* *div R3 R4* *Add R2 R4* *move R4 y*	
IC	MA	IC	MA	IC	MA	IC	MA
7	7	8	5	4	11	8	5

IC： 指令数

MA：内存访问

RISC

CISC

真实示例：Java 字节码，基于堆栈的架构

示例：

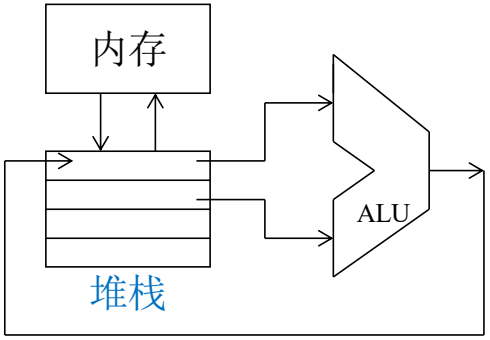
*计算：y = x1 * x2 + x3 / x4*

x1 = 3.0, x2 = 2.5, x3 = 4.5, x4 = 3.1416

```
; use assembly directive
; to declare stack size
.limit stack 4              ; allocate 5 vars

ldc     3.0                 ; load constant
fstore_1                    ; store in x1
ldc     2.5                 ; load constant
fstore_2                    ; store in x2
ldc     4.5                 ; constant
fstore_3                    ; store in x3
ldc     3.1416              ; load constant
fstore_4                    ; store in x4
```

初始化

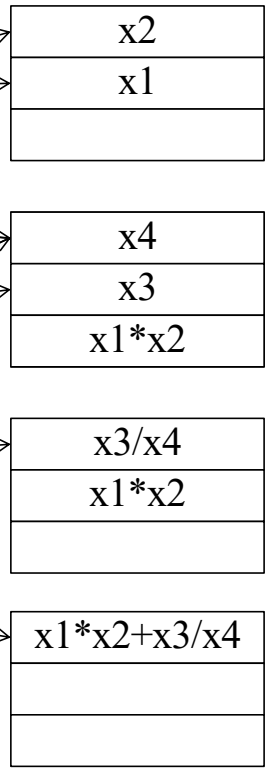


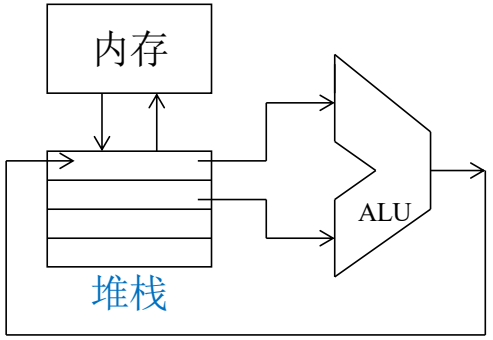
Java 字节码（续）

x1 = 3.0, x2 = 2.5, x3 = 4.5, x4 = 3.1416

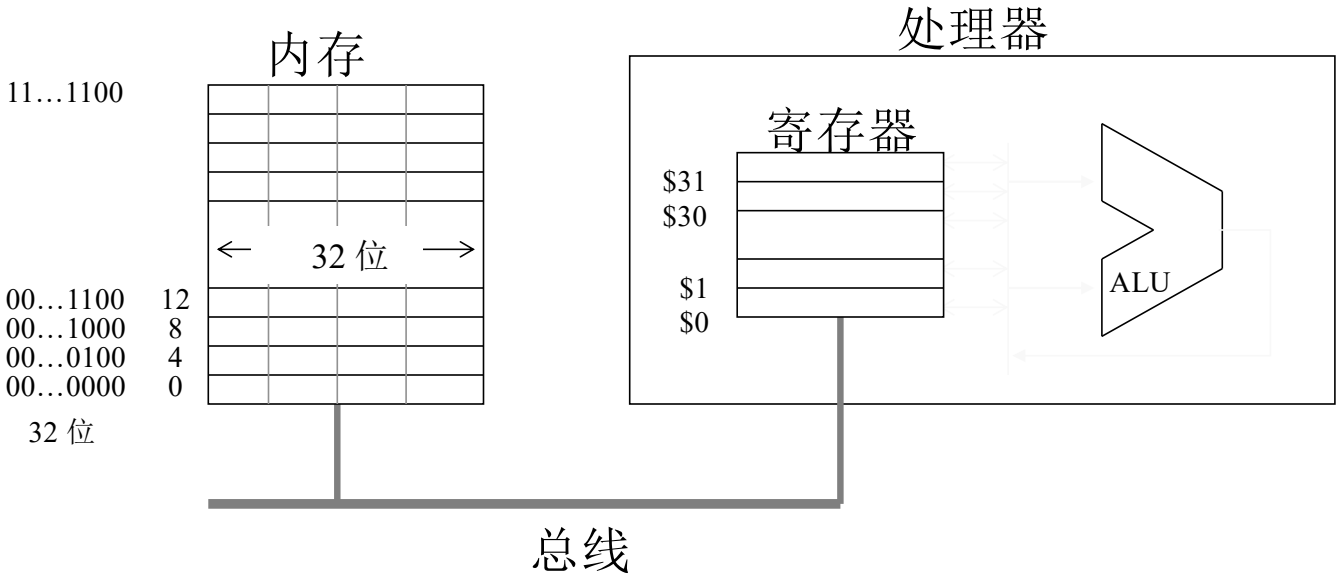
*y = x1 * x2 + x3 / x4*

```
; load data and do multiplication
fload_1          ; load x1
fload_2          ; load x2
fmul             ; float multiply
; load data and do division
fload_3          ; load x3
fload_4          ; load x4
fdiv             ; float division
; the data on the top of stack
fadd             ; float addition
; store final result
fstore_5         ; store in y
```





MIPS：一种加载-存储架构



结论

- 计算机由其组成部分以及它们在模块、寄存器传输和门级别的连接定义。
- 抽象分层
  - 硬件和软件
- 成本/性能比正在提高
  - 由于底层技术的发展
- 功率是一个限制因素
  - 使用并行性进一步提高性能
- 指令集架构
  - 硬件/软件接口
- 计算机架构由指令集架构决定
  - 汇编语言级别的编程模型