

# 计算机组成原理

**Instructor: 施慧彬（院楼 224室）**

**Email: [hshi@nuaa.edu.cn](mailto:hshi@nuaa.edu.cn)**

**Phone: 15905176918**

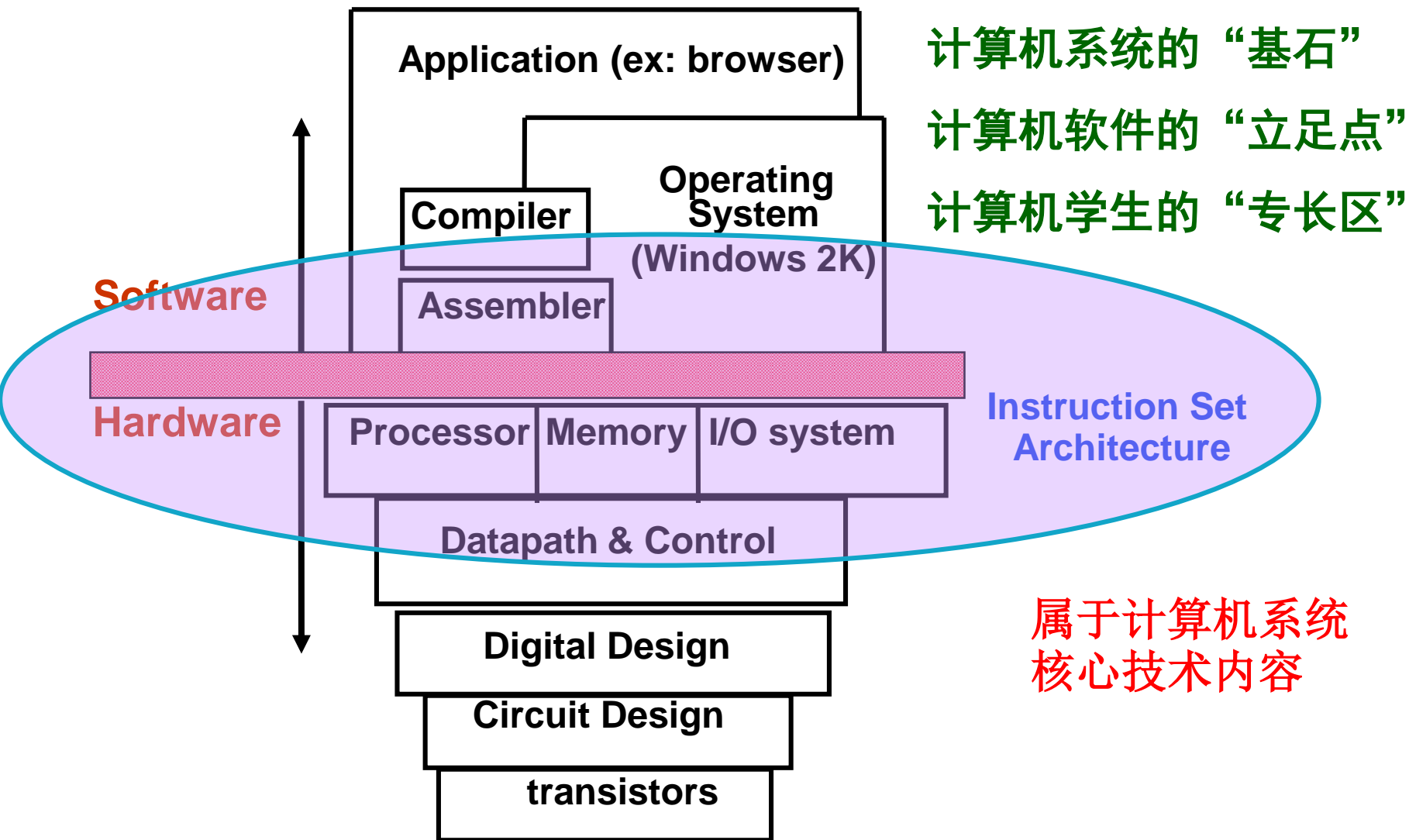
**QQ: 503780014**

# 关于课程名称

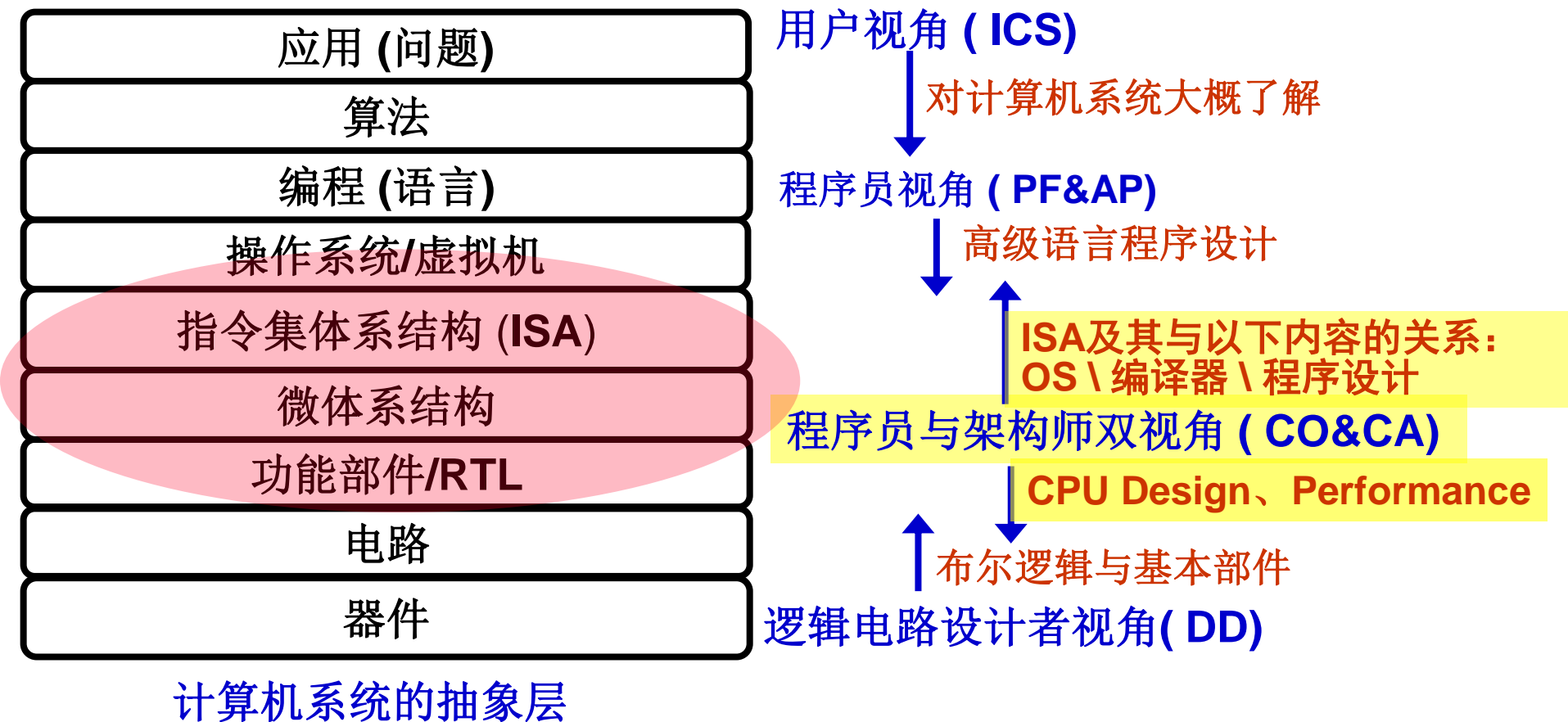
---

- **Computation Structure (MIT)**
- **Machine Structure (Great Ideas in Computer Architecture ) (UC-Berkeley)**
- **Computer Organization and Systems (Stanford)**
- **Introduction to Computer System (CMU)**
- **计算机组成原理 (国内大多数高校)**
- **计算机组成与结构 (或 计算机组织与结构)**
- **计算机组成与系统结构 (或 计算机组织与系统结构)**
- **计算机结构原理**
- **.....**

# 课程内容在计算机系统中的位置



# 本课程与其他课程的关系

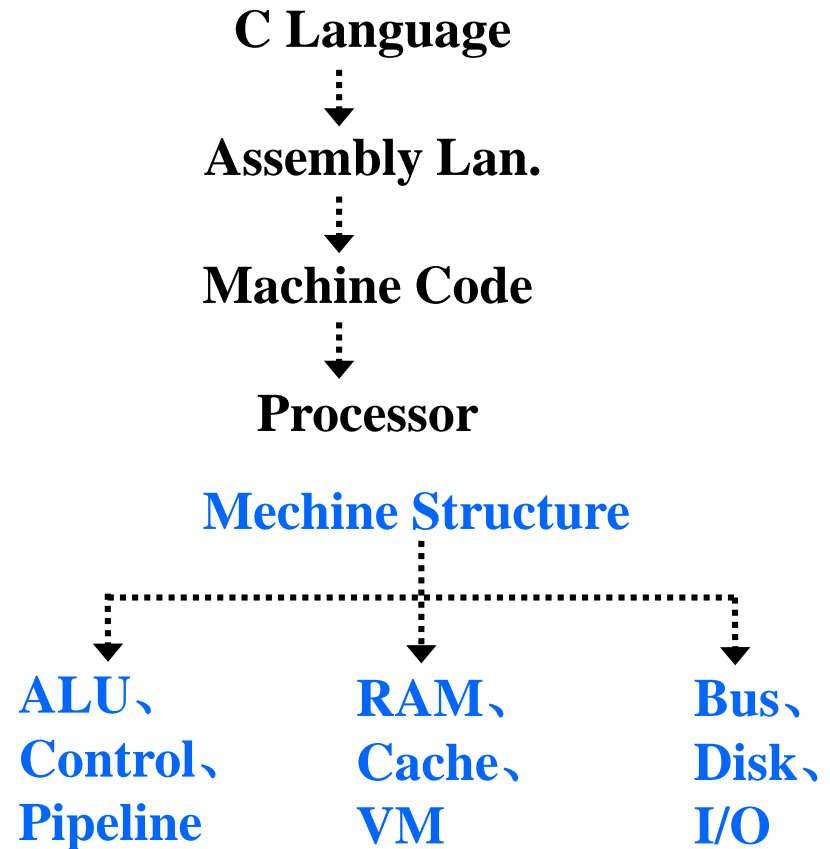


先行课（概论(ICS)、程设(PF)、高级程设(AP)、数电设计(DD)等）：

计算机组成与系统结构：构建系统框架，以建立完整计算机系统概念

# 主要教学内容

- Number Representations
- Floating Point
- Caches
- Virtual Memory
- Assembly Programming
- Logic Design
- CPU Organization
- Pipelining
- Bus
- Disks, I/O
- Performance



Explain how higher level programs are translated into machine language, the general structure of computers, exceptions, interrupts, caches, address translation, CPU design, and related topics.

**Focus on:** a single-processor computer system

# 课程教学目标

## 基本目标是什么？

- ISA设计原理
- 计算机硬件设计原理
- 计算机整机概念的建立

实验课要求设计流水线CPU、并使用总线连接存储器、键盘和VGA接口，以完整实现一个计算机

## 要求掌握的重要知识和能力有哪些？

- 利用硬件知识提高调试程序的能力  
(如：数据格式转换、大端/小端方式)
- 从硬件角度出发编制高效程序的能力  
(如：Cache的局部性、函数调用)

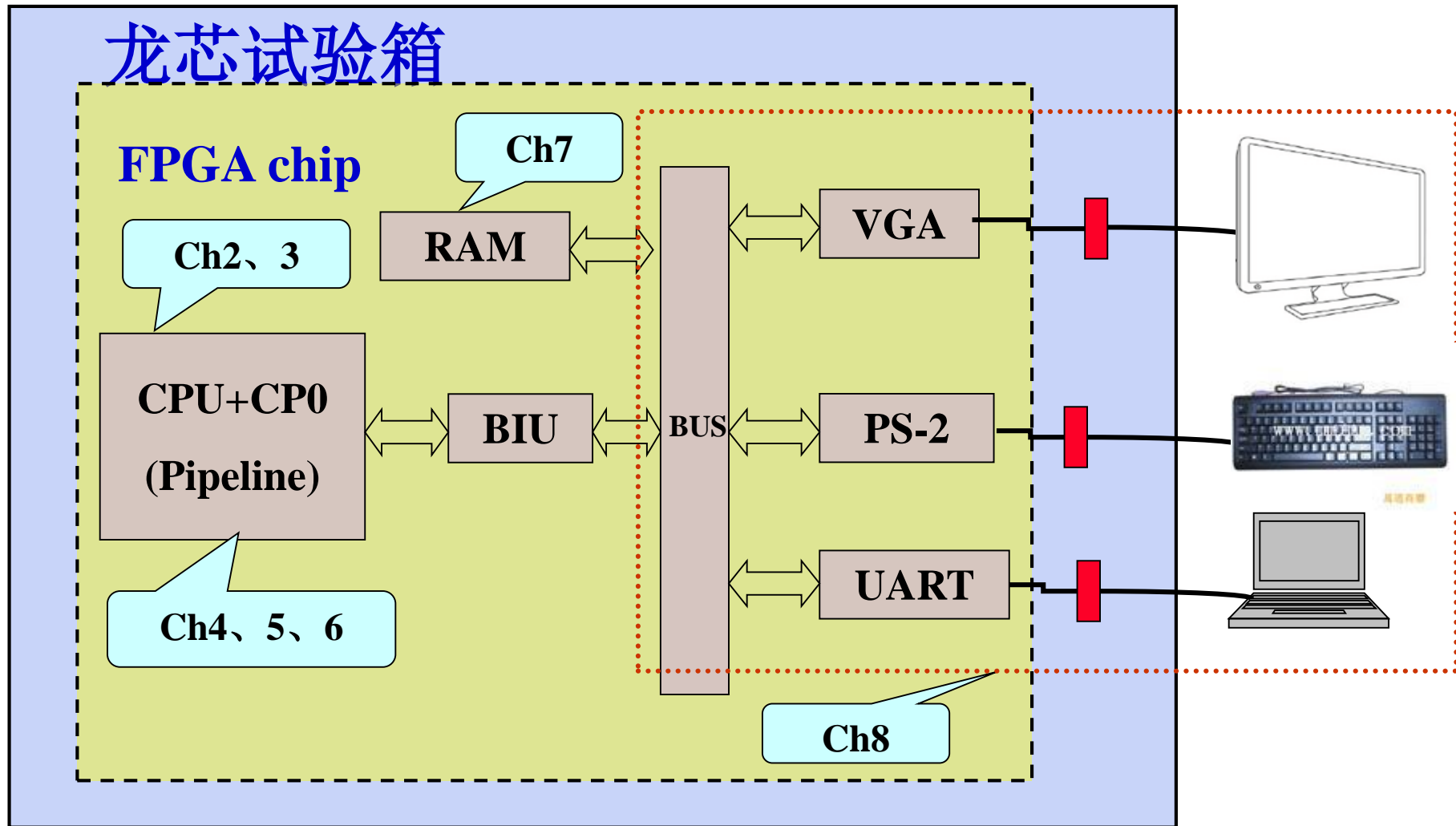
通过本学期的编程实验，以使大家能将理论知识转化为运用实践能力

- OS和硬件如何分工、如何衔接？  
(如：异常、中断)
- 如何从硬件角度出发进行编译优化？  
(如：流水线调度)
- 为后续课程打下坚实基础

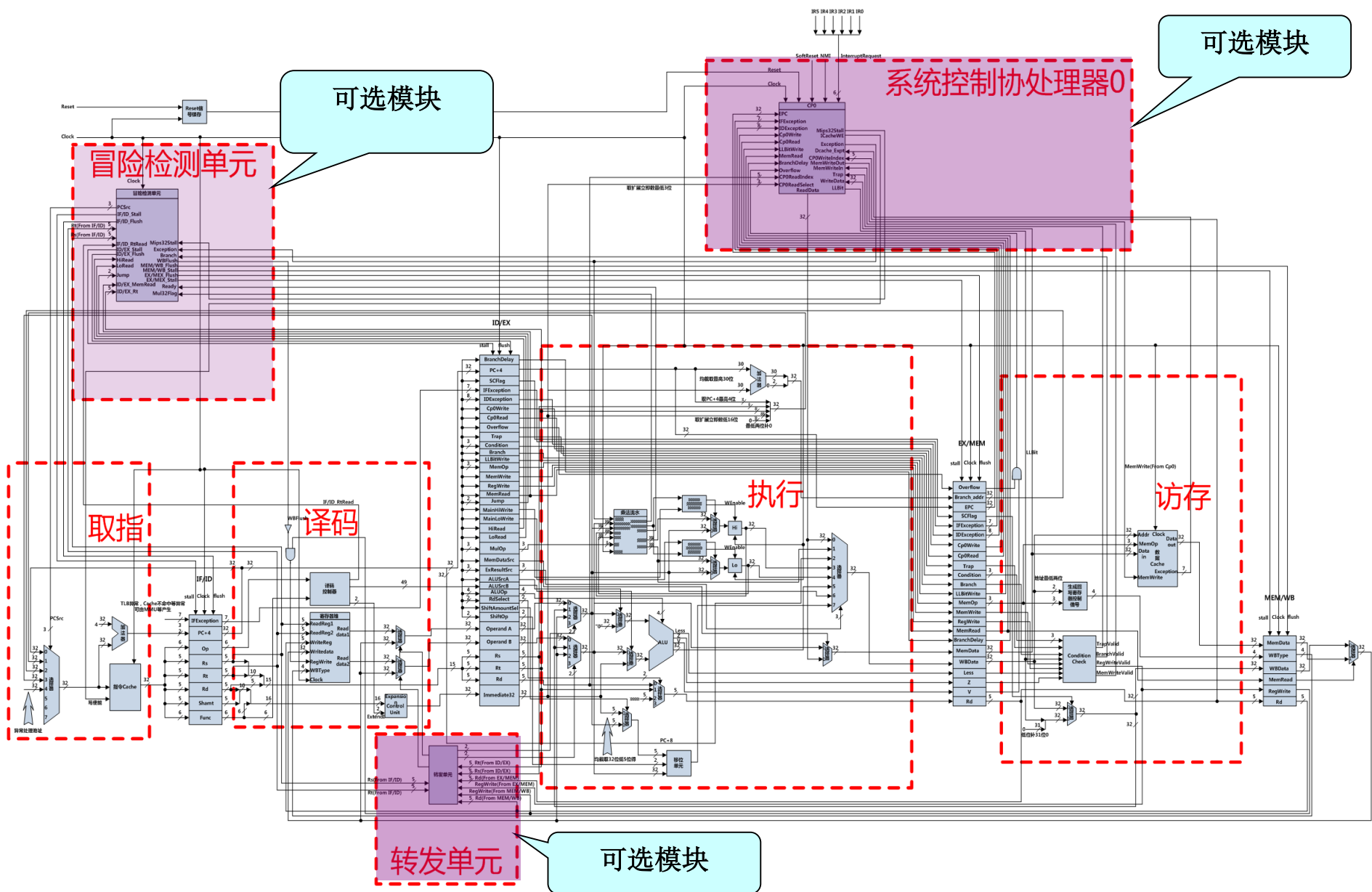
(如：OS、编译原理、体系结构、微机原理、嵌入式技术等)

# 实验课的大作业框架

## 龙芯试验箱



# 5级基本流水线CPU实验 (+可选模块)





# 本课程的重要作用

## 专业基础及其能力主要包括：

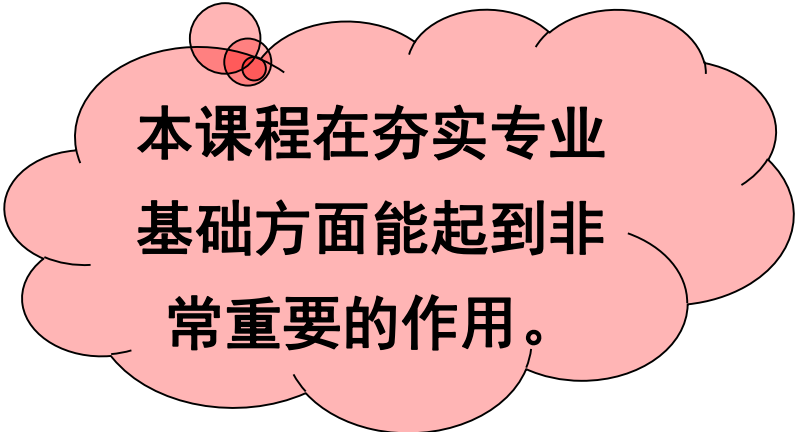
- 对计算机系统有全面的认识
  - 硬件和软件的分工和协作
  - 层次化特点：屏蔽细节
  - 抽象、转换和分而治之

问题→算法→程序→指令集体系结构→微结构→电路

- 应用软件开发能力
- 算法设计和分析能力
- 系统软件的开发和使用能力
- 硬件知识和硬件设计能力
  - HDL、FPGA、SOP、ISA、...

## 本课程内容是：

计算机系统的“基石”  
计算机软件的“立足点”  
计算机学生的“专长区”



本课程在夯实专业基础方面能起到非常重要的作用。

# 如何学好本课程？

## 本课程特点

- 不利点：概念抽象、繁琐、面广、与其他课程关联度高
- 有利点：清澈见底（只要想搞明白就一定能明白）

位→门电路→部件→CPU→微结构→ISA →  
汇编语言程序→高级语言程序

## 学习要领

- 站在系统的高度  
从程序员视角想问题，由粗到细建立系统整体
- 运用联系的观点  
关注软件和硬件的关联、本课程与其他课程的关联
- 试图去理解概念而不是死记硬背  
前后概念串起来理解，抓主干（细节为主干服务）
- 多练习和动手实践  
作业和实验报告自己独立完成
- 上课听讲非常重要，有问题及时解决  
不来上课而“挂科”一定是你的责任！

# Books

---

- **Textbook: 《计算机组成与系统结构（第2版）》袁春风等，清华大学出版社，2015**
- **主要参考书：**
  - **《深入理解计算机系统》（第2版），Randal E. Bryant, david R. O'Hallaron著，龚奕利，雷迎春译，机械工业出版社，2011 年**
  - **《计算机组成与设计》（第4版），David A. Patterson, John L. Hennessy著，康继昌，樊晓桢，安建峰等译，机械工业出版社，2012年**

# 美国名校相关课程网站

- 美国UC Berkeley大学 “Machine Structure”2013年课程网站：  
<http://inst.eecs.berkeley.edu/~cs61c/sp13/>
- 美国UC Berkeley大学 “Components and Design Techniques for Digital System”2013年课程网站：  
<http://inst.eecs.berkeley.edu/~cs150/sp13/>
- 美国UC Berkeley大学 “Computer Architecture and Engineering”2013课程网站：  
<http://inst.eecs.berkeley.edu/~cs152/sp13/>
- 美国Stanford大学 “Computer Organization and Systems” 2013年课程网站：  
<http://www.stanford.edu/class/cs107/>
- 美国Stanford大学 “Digital Systems II”2013年课程网站：  
<http://www.stanford.edu/class/ee108b/>
- 美国Carnegie Mellon 大学 “Introduction to Computer Architecture”2013年课程网站：  
<http://www.ece.cmu.edu/~ece447/>
- 美国麻省理工学院(MIT)“Computation Structures”2013年课程网站：  
<http://6004.csail.mit.edu>

# 教学安排

---

- 课堂教学环节
  - 抓总体框架、讲主干概念、结合例子、提问互动
- 其它环节
  - 作业和习题报告（随机抽查上台讲解）
  - 随堂测验（兼作考勤记录）
  - 没有期中考试
  - 课后交流（网上讨论区、邮件、课间答疑）

# Grading Policy

---

- 平时成绩（包括测验(出勤)、作业和实验）：**30%**
- 考试成绩：**70%**

**重要说明：作业、测验和考试等过程中，发现雷同或笔迹一样时，作0分处理。**

**请大家最好先看书，再提问。**

**每条作业答案需要提供你是根据书本中第几页中的哪（些）小节（段）来得到你的答案（的依据），越细越好。**

# Any Questions ?

---



# Ch1: Computer Abstractions

## 计算机系统概述

第1讲：计算机系统概述

第2讲：计算机性能评价



# 第一讲 计算机系统概述

- 计算机发展简史
  - IAS通用计算机模型机：冯.诺依曼结构
  - IBM360系列机：引入兼容性（系列机）概念
  - DEC PDP-8：引入总线结构
- 计算机系统的组成
  - 计算机硬件：CPU + MM + I/O
  - 计算机软件：系统软件+应用软件
- 计算机层次结构
  - 计算机硬件和软件的接口：指令系统
  - 计算机软件如何在硬件上执行
- 本课程主要内容

# 计算机发展简史

第一代：真空管（电子管Vacuum Tube）1946~57年

- 46年（2月公布）诞生第1台电子计算机 **ENIAC**（**Electronic Numerical Integrator And Computer**，即电子数字积分计算机）

- 体积大，重30吨，有18000多个真空管，5000次加法/s
- 十进制表示/运算，存储器由20个累加器组成，每个累加器存10位十进制数，每一位由10个真空管表示（不是真正意义上的存储器）。
- 采用手动编程，通过设置开关和插拔电缆来实现。

- 冯·诺依曼机（**Von Neumann Machine**）

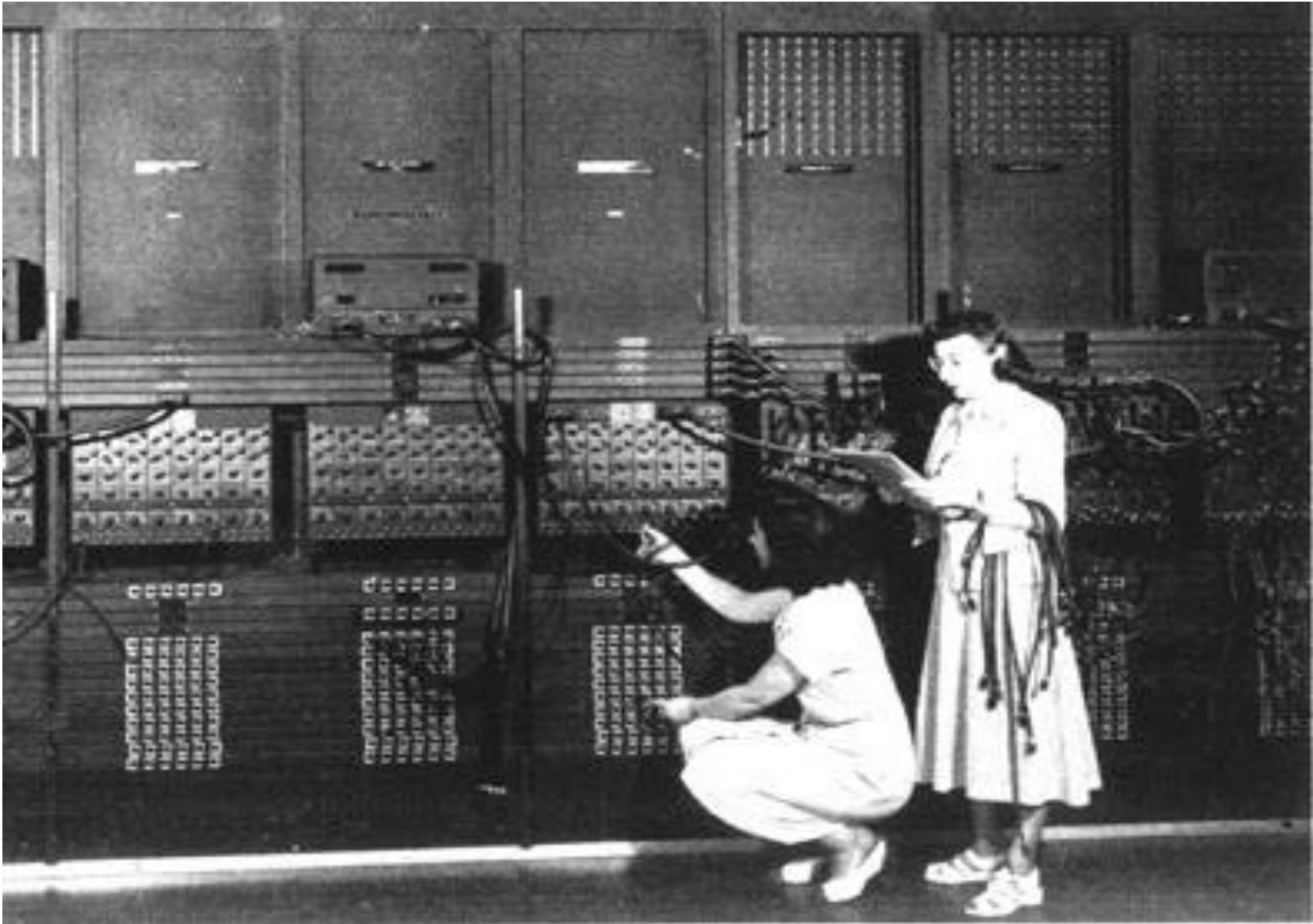
- 45年6月冯·诺依曼提出“**存储程序(Stored-program)**”思想，并于46年在IAS(Institute for Advanced Study)开始设计“存储程序”计算机。
- “存储程序”思想：将事先编好的程序和原始数据送入主存中，然后启动执行。计算机能在不需操作人员干预下，自动完成逐条取出指令和执行指令的任务。

# The First Generation: Vacuum Tube Computers (1946 - 1957)



The first *general-purpose computer* - **ENIAC**

# ENIAC--Not Typical von Neumann Model



[BACK](#)



# 冯·诺依曼的故事

- 1944年，冯·诺依曼参加原子弹的研制工作，涉及到极为困难的计算。1944年夏的一天，诺依曼巧遇美国弹道实验室的军方负责人戈尔斯坦，他正参与ENIAC的研制工作。
- 冯·诺依曼被戈尔斯坦介绍加入ENIAC研制组，1945年，在共同讨论的基础上，冯·诺依曼以“关于EDVAC的报告草案”为题，起草了长达101页的总结报告，发表了全新的“**存储程序通用电子计算机方案**”。
- 一向专搞理论研究的**普林斯顿高等研究院（IAS）**批准让冯·诺依曼建造计算机，其依据就是这份报告。
- EDVAC于1949年8月交付给弹道研究实验室，在解决许多新发现的问题之后，1951年才开始运行，而且局限于基本功能。



**Electronic  
Discrete  
Variable  
Automatic  
Computer**

# 冯·诺依曼结构的主要思想

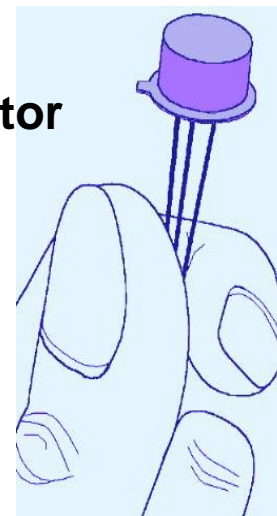
1. 计算机应由运算器、控制器、存储器、输入设备和输出设备五个基本部件组成。
2. 各基本部件的功能是：
  - **存储器**不仅能存放数据，而且也能存放指令，形式上两者没有区别，但计算机应能区分数据还是指令；
  - **控制器**应能自动执行指令；
  - **运算器**应能进行加/减/乘/除四种基本算术运算，并且也能进行一些逻辑运算和附加运算；
  - 操作人员可以通过**输入设备**、**输出设备**和主机进行通信。
3. 内部以**二进制表示**指令和数据。每条指令由操作码和地址码两部分组成。操作码指出操作类型，地址码指出操作数的地址。由一串指令组成程序。
4. 采用“**存储程序**”工作方式。

# 计算机发展简史

## ◦ 第二代：晶体管 1958~64年

- 元器件：逻辑元件采用晶体管，内存由磁芯构成，外存为磁鼓与磁带。
- 特点：变址，浮点运算，多路存储器，I/O处理机，中央交换结构(非总线结构)。
- 软件：使用高级语言，提供了系统软件。
- 代表机种：IBM 7094 (scientific)、1401 (business)和 DEC公司的 PDP-1

晶体管：  
Transistor



DEC PDP-1



# 计算机发展简史

◦ 第三代：中小集成电路SSI/MSI(Small-scale integration Medium Scale Integration) 1965~71年

- 元器件：逻辑元件与主存储器均由集成电路（IC）（Integrated Circuit）实现。
- 特点：微程序控制，Cache，虚拟存储器，流水线等。
- 代表机种：IBM 360和DEC PDP-8（大/巨型机与小型机同时发展）
  - 巨型机(Supercomputer): Cray-1
  - 大型机(Mainframe): IBM360系列
  - 小型机(Minicomputer): DEC PDP-8





# IBM System/360系列计算机

- IBM公司于1964年研制成功
- 引入“兼容机”(系列机)概念
  - 兼容机的特征：
    - 相同的或相似的指令集
    - 相同或相似的操作系统
    - 更高的速度
    - 更多的I/O端口数
    - 更大的内存容量
    - 更高的价格



IBM 360

低端机指令集是高端机的一个子集，称为“**向后兼容**”。原来机器上的程序可以不改动而在新机器上运行，但性能不同。

问题1：引入“兼容机”有什么好处？

问题2：保持“兼容”的关键是什么？

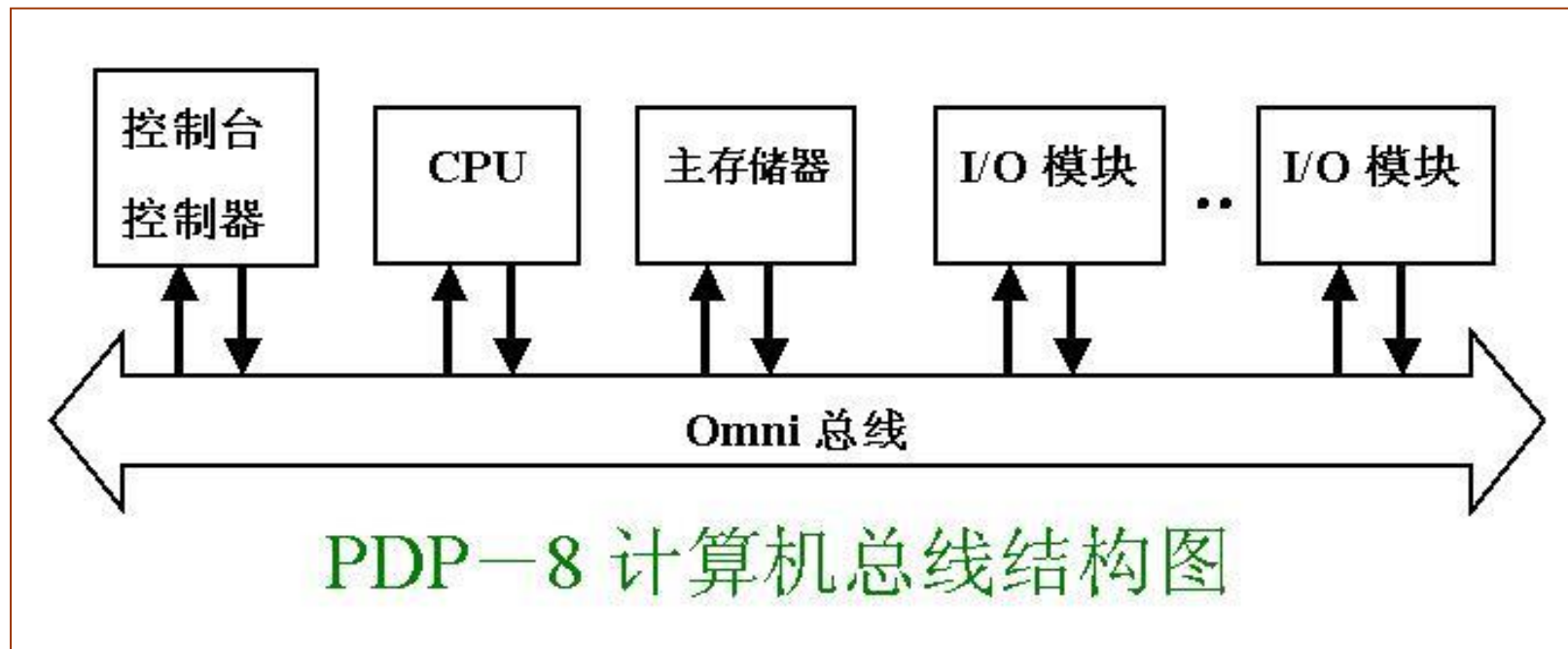
# DEC公司的PDP-8机

- 同在64年出现。与IBM 360相比，价格更低、更小巧，因而被称为小型机（Minicomputer）
- PDP-8“创造了小型机概念，并使之成为数十亿美元的工业”，使DEC成为了最大的小型机制造商。
- 主要特点：首次采用总线结构。

具有高度的灵活性，允许将模块插入总线以形成各种配置。

# PDP-8/E计算机系统框图

Omnibus总线包含了96个独立的信号通道，用以传送控制、地址和数据信号。



**问题：“总线结构”有什么好处？**

可扩充性好（允许将新的符合标准的模块插入总线形成各种配置）、节省器件，体积小，价格便宜

# 计算机发展简史

第四代：（标准、意见不一）有称是VLSI,从80年代开始,也有称是LSI,从72年开始至今, Large/Very Large/Ultra Large Scale Integration ( LSI/VLSI/ULSI )

- 微处理器和半导体存储器技术发展迅猛，微型计算机出现。  
使计算机以办公设备和个人电脑的方式走向普通用户。

## 半导体存储器

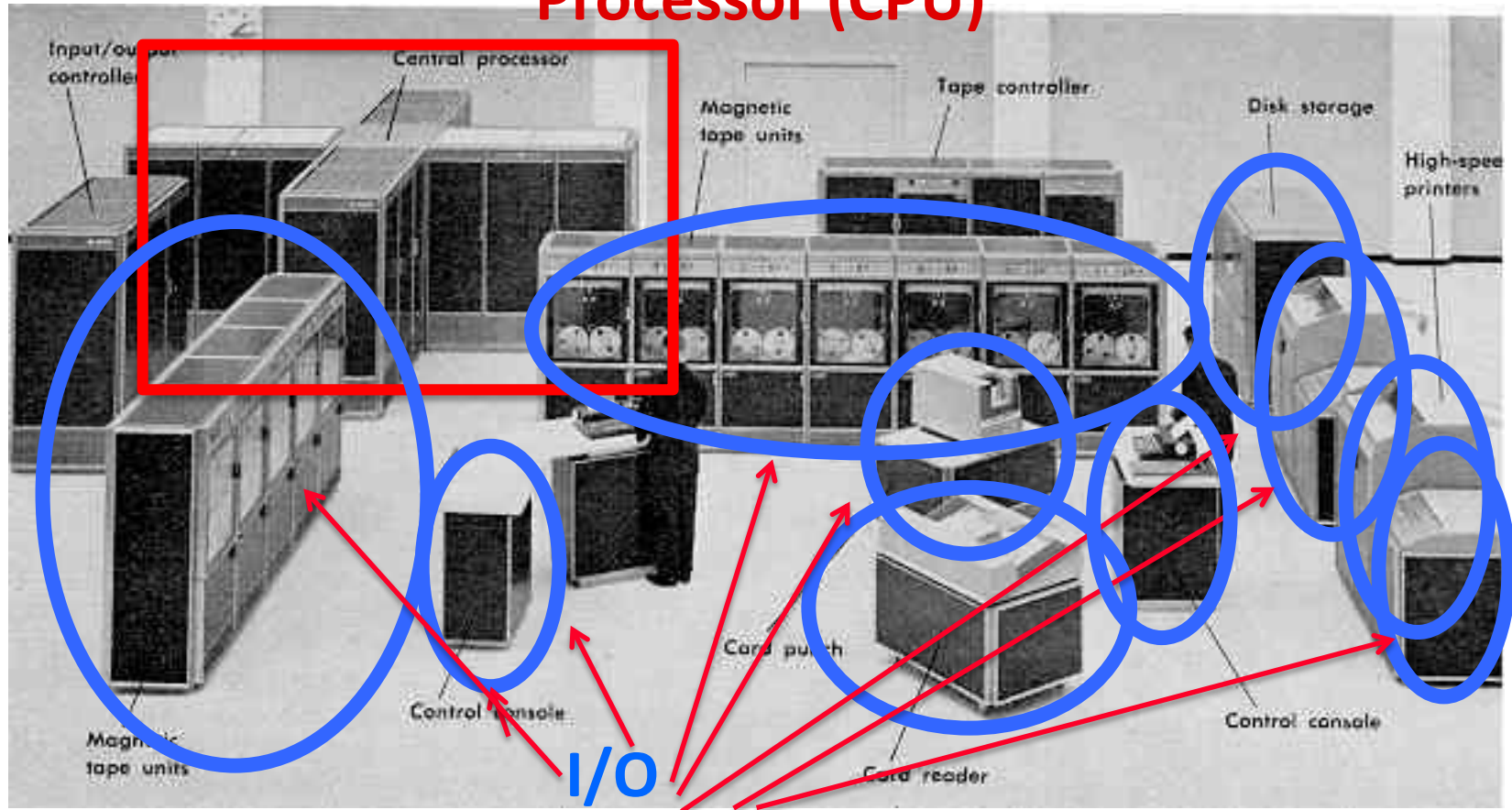
- 70年Fairchild公司生产出第一个相对大容量半导体存储器
- 74年位价格低于磁芯的半导体存储器出现，并快速下跌
- 从70年起，存储密度呈4倍提高（几乎是每3年）

## 微处理器

- 微处理器芯片密度不断增加，使CPU中所有元件放在一块芯片上成为可能。71年开发出第一个微处理器芯片4004。
- 特点：共享存储器，分布式存储器及大规模并行处理系统,计算机网络发展与广泛应用。

# Mainframe(是一种大型商业服务器) Eras : 1950s-60s

## Processor (CPU)



“Big Iron”: IBM, UNIVAC, ... build \$1M computers for businesses => COBOL, Fortran, timesharing OS

# Minicomputer(小型计算机) Eras: 1970s



Using integrated circuits, Digital, HP... build \$10k computers for labs, universities => C, UNIX OS



## PC Era: Mid 1980s - Mid 2000s



Using microprocessors, Apple, IBM, ... build \$1k computer for 1 person => Basic, Java, Windows OS

# PostPC Era: Late 2000s - ??



iPhone  
iPod  
iPad

**Personal Mobile Devices (PMD):** Relying on wireless networking

Apple, Nokia, ... build \$500 smartphone and tablet computers for individuals  
=> Objective C, Android OS

**Cloud Computing:**

Using Local Area Networks, Amazon, Google, ... build \$200M **Warehouse Scale Computers**

with 100,000 servers for Internet Services for PMDs

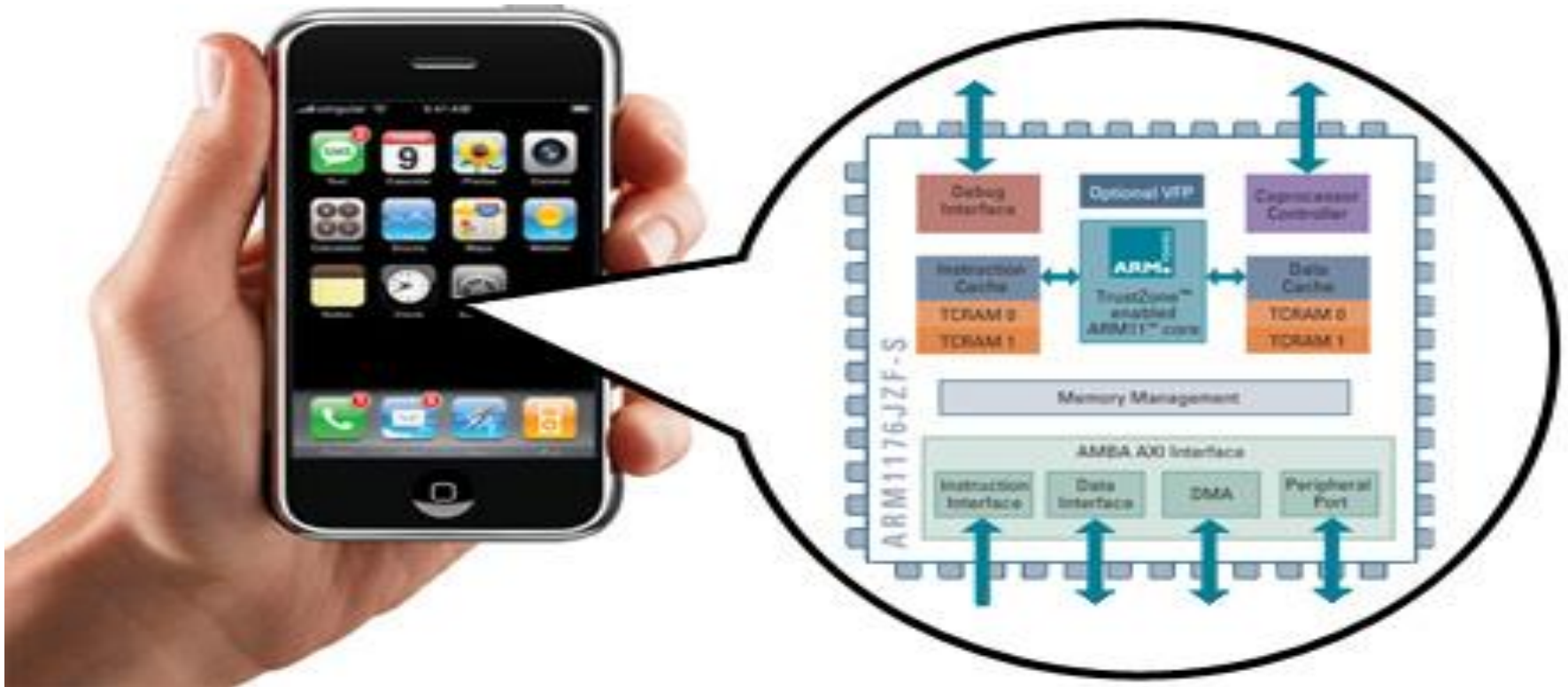
=> MapReduce, Ruby on Rails





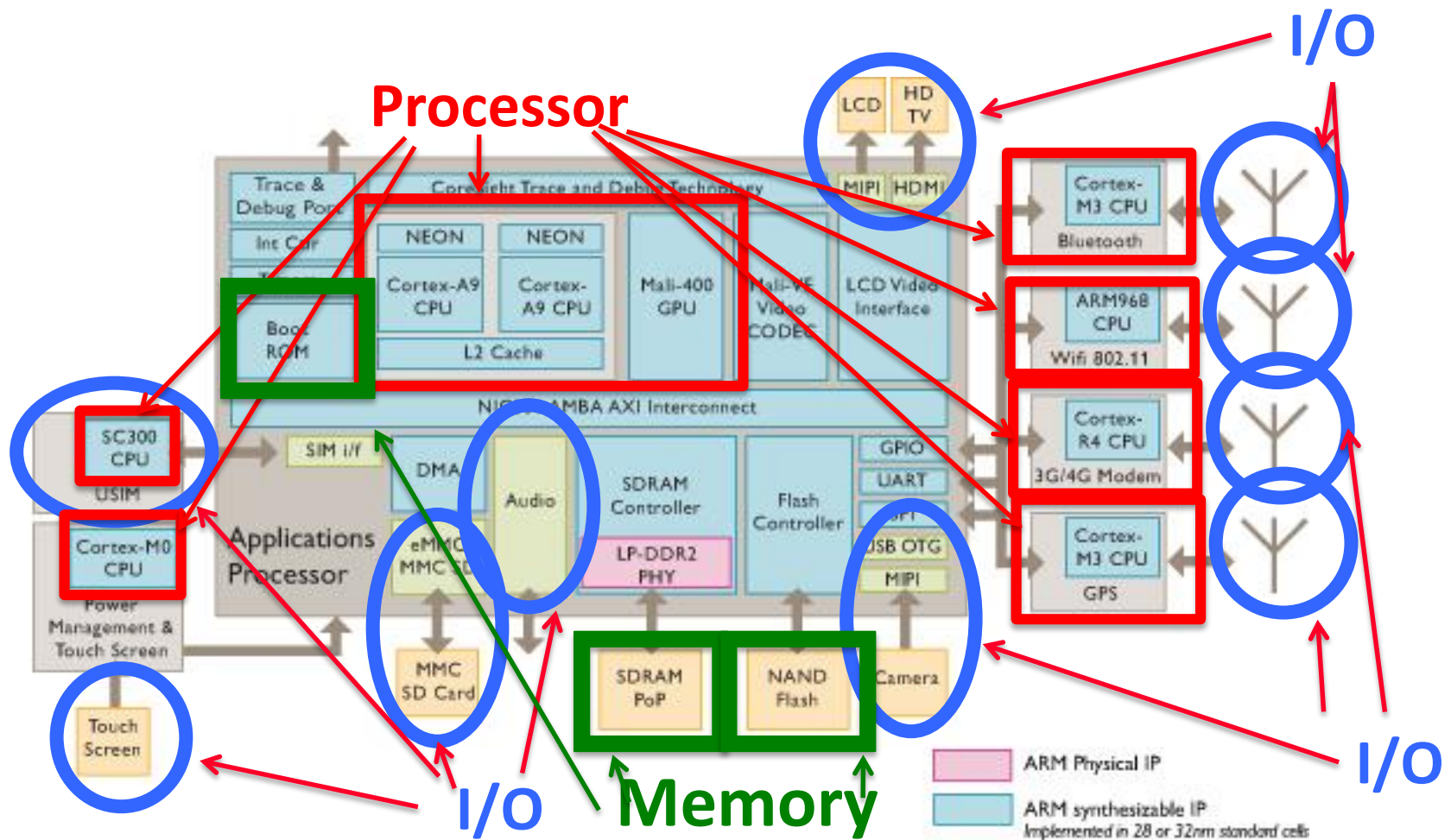
# Advanced RISC Machine (ARM) instruction set inside the iPhone

RISC : Reduced Instruction Set Computing



# iPhone Innards

麻雀虽小，五脏俱全！



You will about multiple processors, data level parallelism, caches in 61C

# Containers in WSCs

Inside WSC(Warehouse Scale Computers)



Inside Container





# Server, Rack, Array



# 计算机的基本组成与基本功能

## ◦ 什么是计算机？

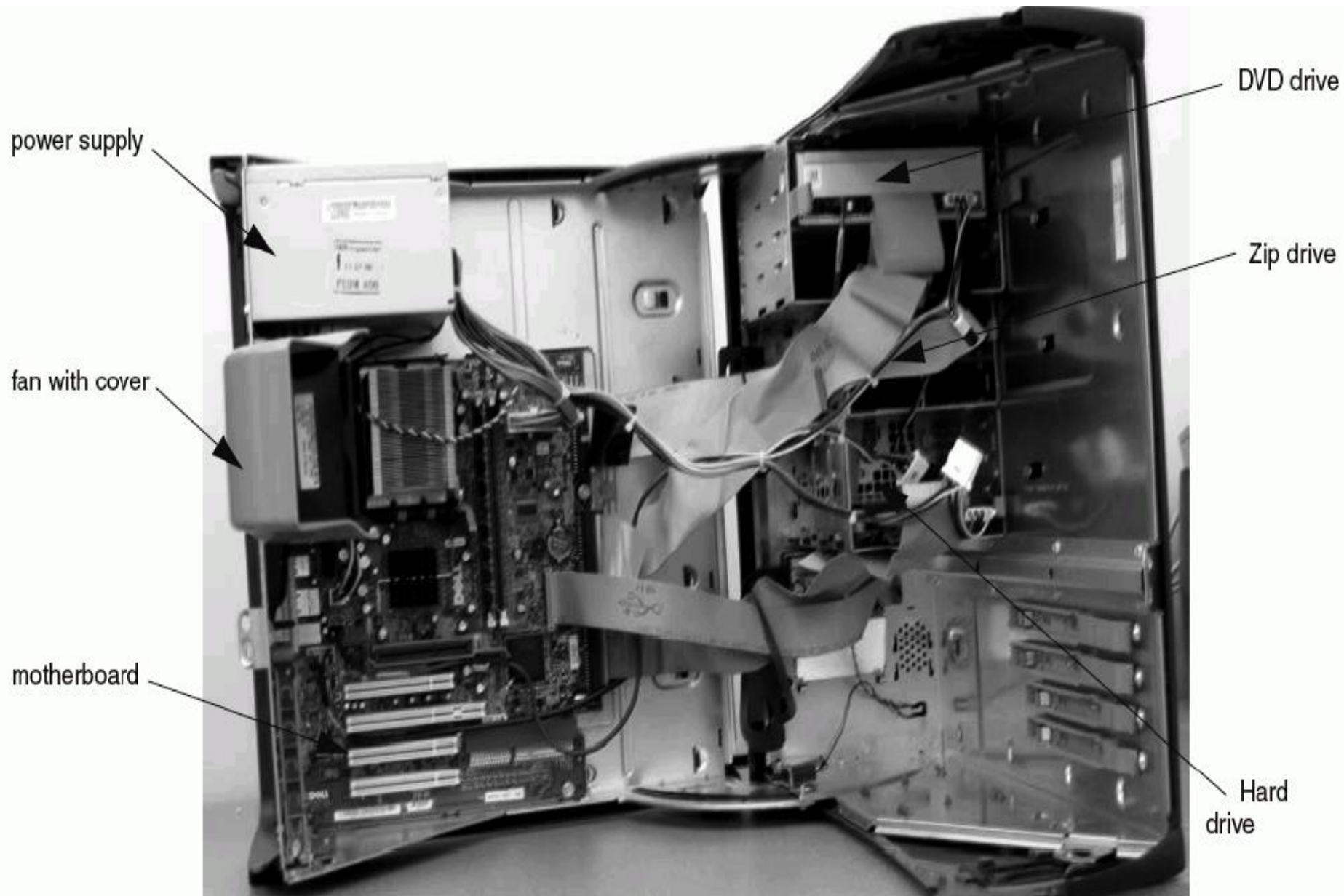
- 计算机是一种能对数字化信息进行自动、高速算术和逻辑运算的处理装置。

## ◦ 计算机的基本部件及功能：

- 运算器（数据运算）：ALU、通用寄存器(General Purpose Registers, GPR)、标志寄存器等
- 存储器（数据存储）：存储阵列、地址译码器、读写控制电路
- 总线（数据传送）：数据(Memory data register,MDR)、地址(Memory address register,MAR)和控制线
- 控制器（控制）：对指令译码生成控制信号

## ◦ 计算机实现的所有任务都是通过执行一条一条指令完成的！

# 计算机硬件：打开PC来看看

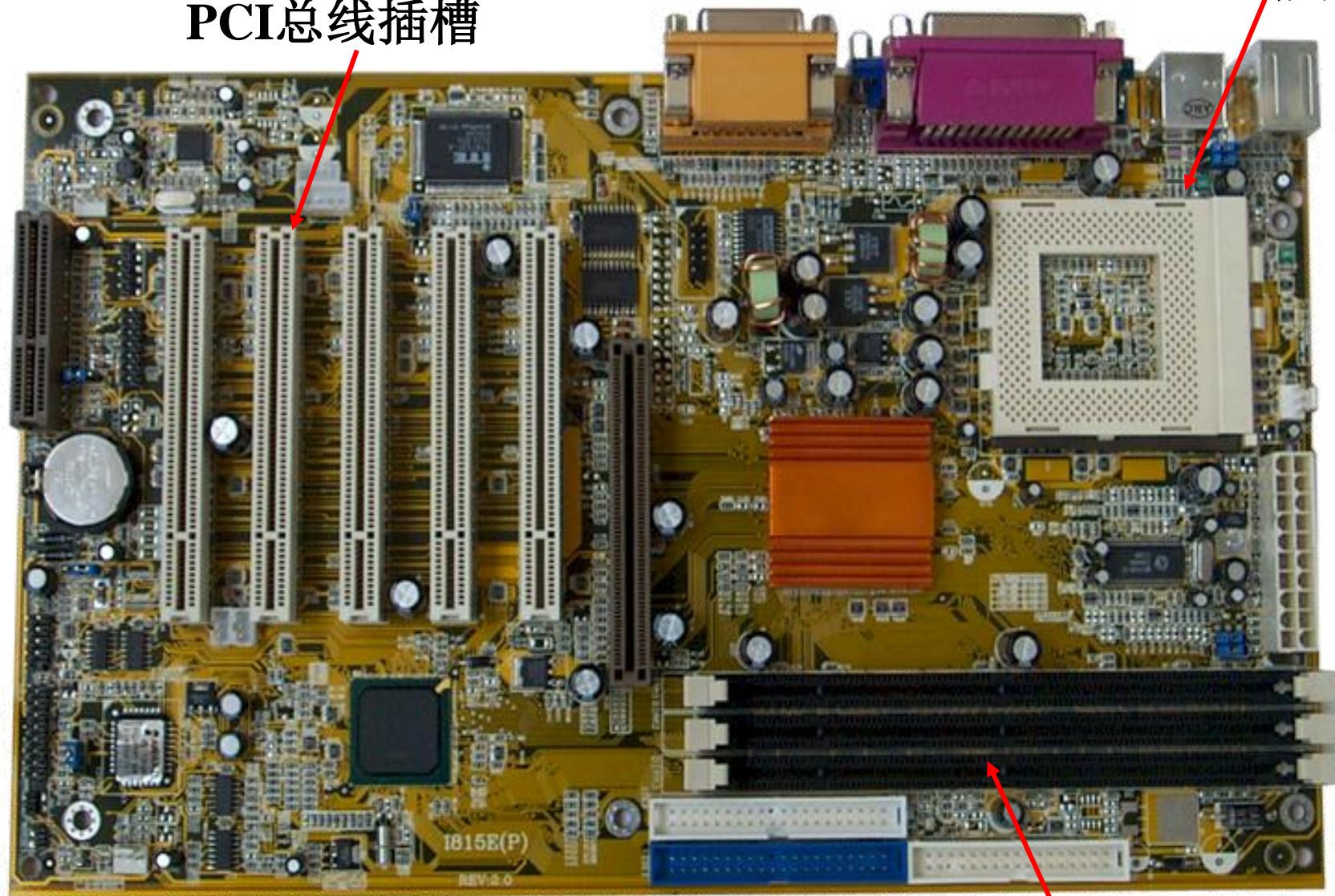




# PC主板

PCI总线插槽

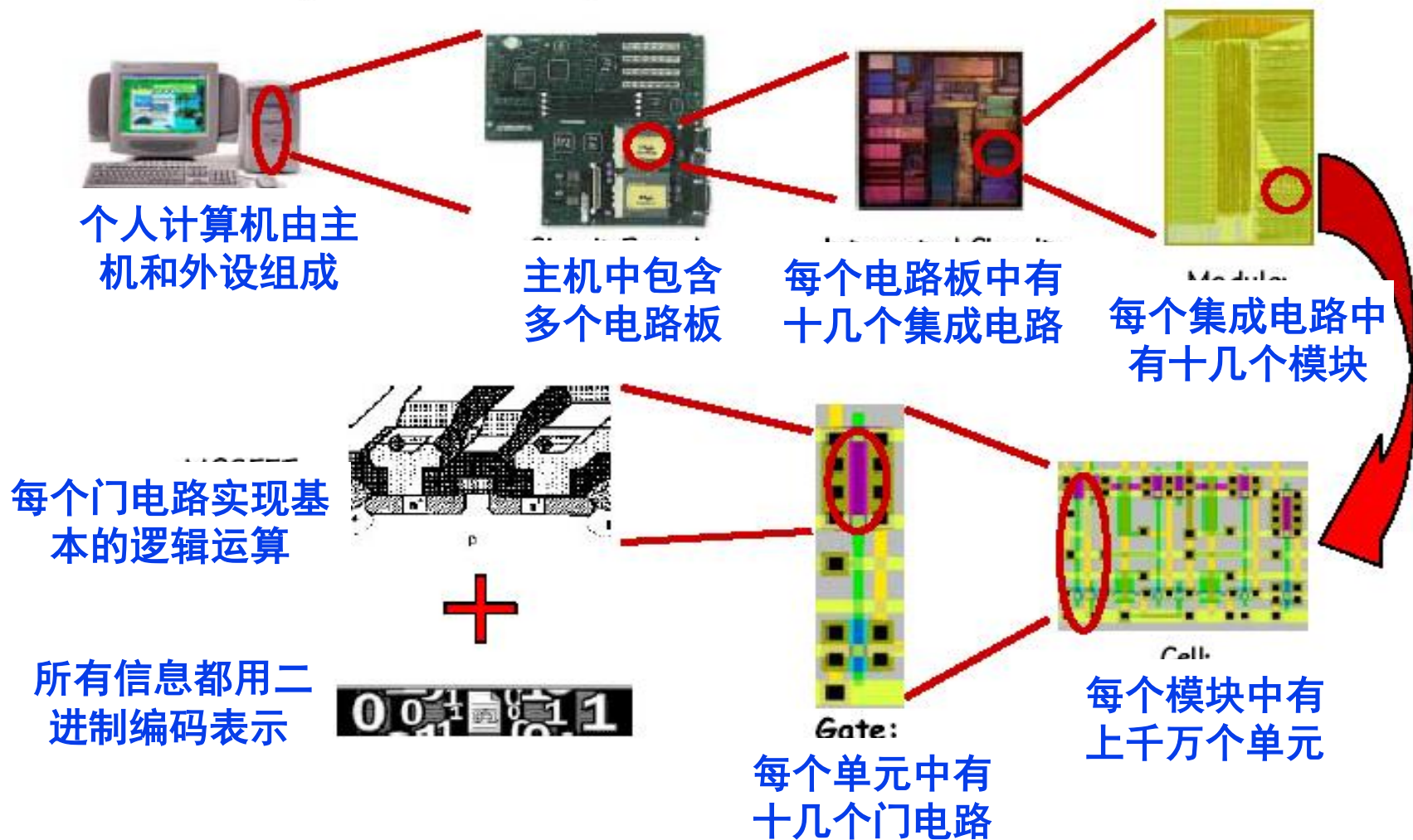
CPU插座



内存条

# 解剖一台计算机（分而治之）

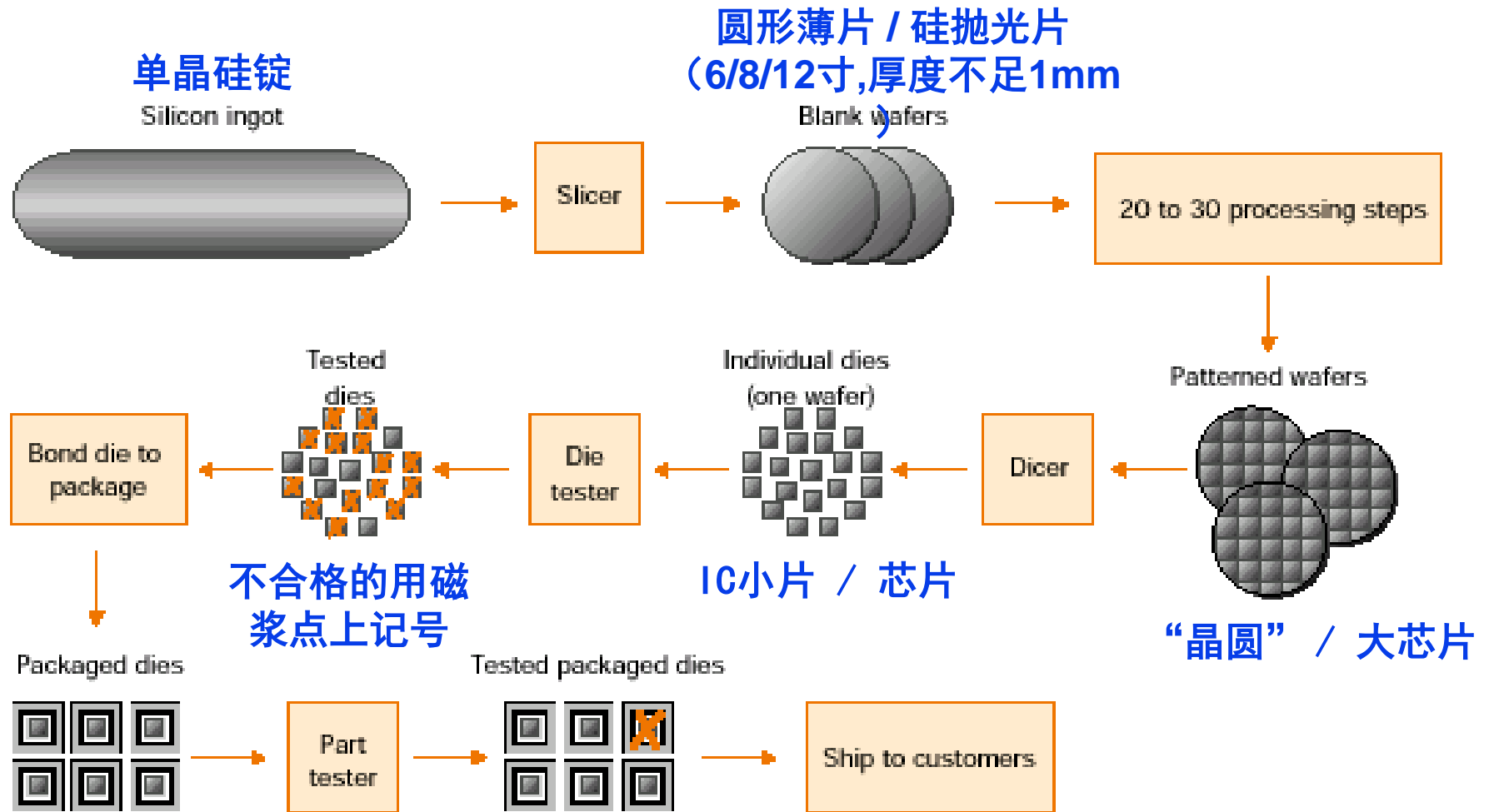
How do you build systems with >1G components?



金属-氧化物半导体场效应晶体管，简称金氧半场效晶体管（Metal-Oxide-Semiconductor Field-Effect Transistor, MOSFET）



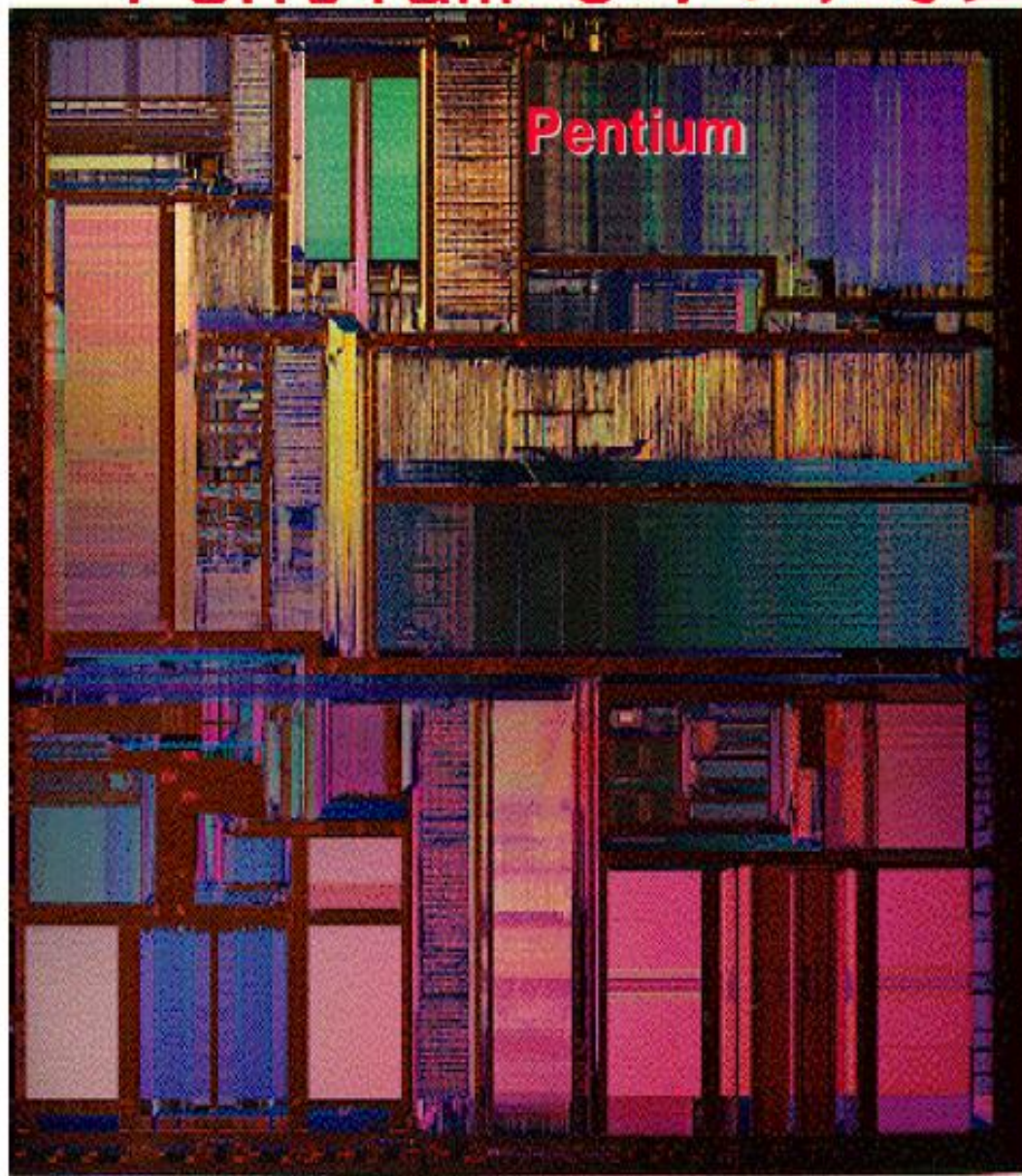
# Integrated Circuits manufacturing process



封装：将芯片固定在塑胶或陶瓷基座上，把芯片上蚀刻出来的引线与基座底部伸出的引脚连接，盖上盖板并封焊成芯片

约需400多道工序！

# Pentium芯片内的主要功能块



Die Area: 91 mm<sup>2</sup>

直径8 inch(200mm)的  
Wafer最多可做 196个Die

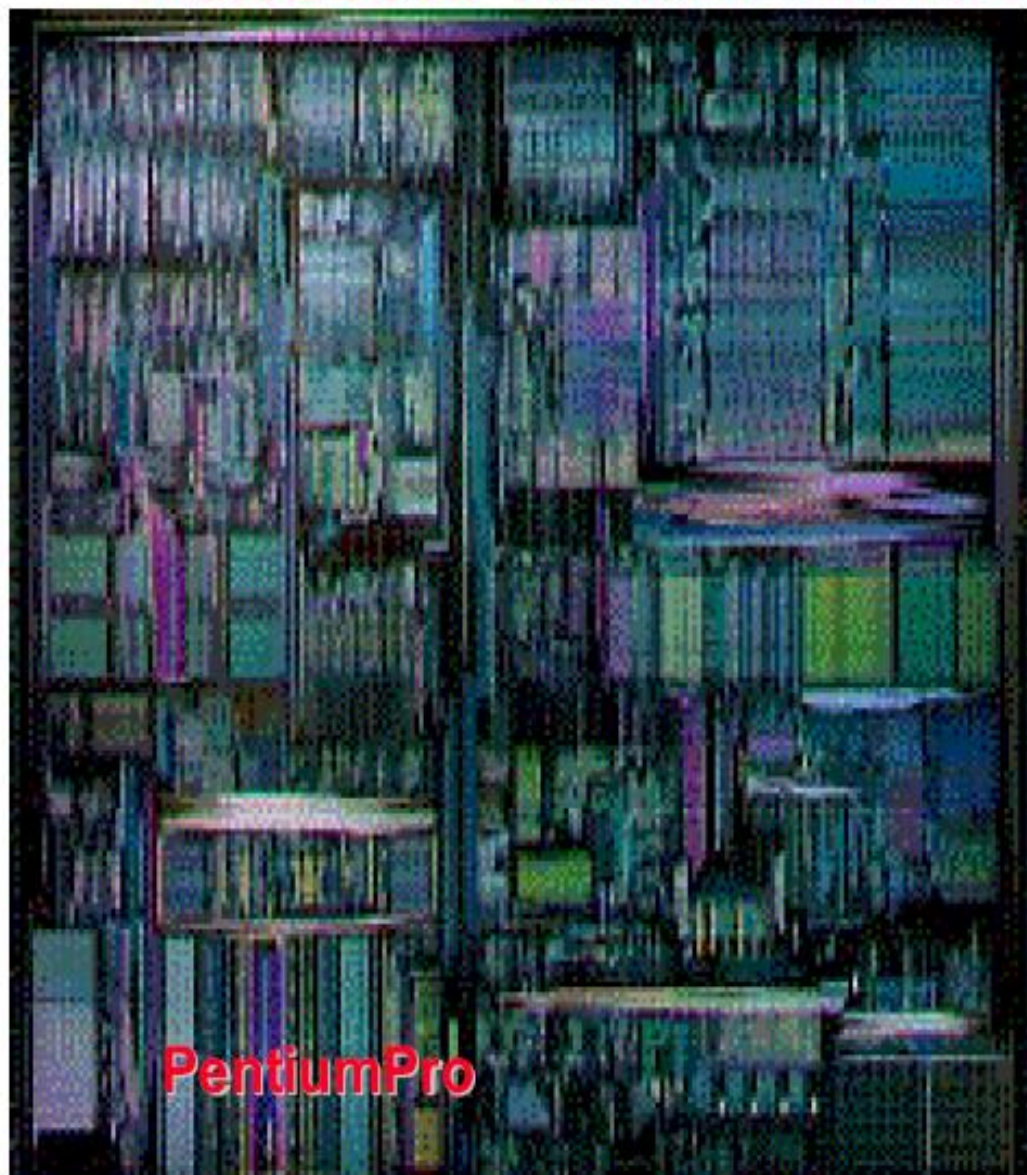
≈ 3,300,000 Transistors

Cache: ≈1M Transistors

296 Pins

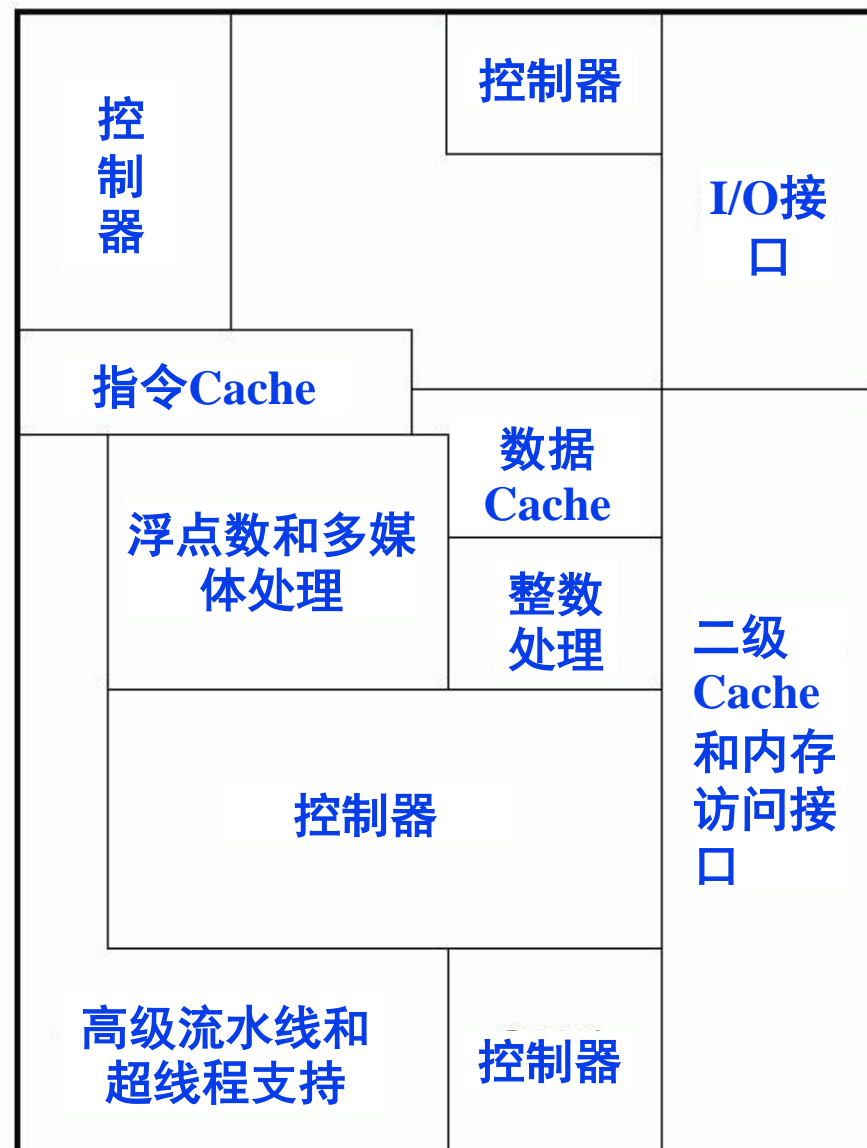
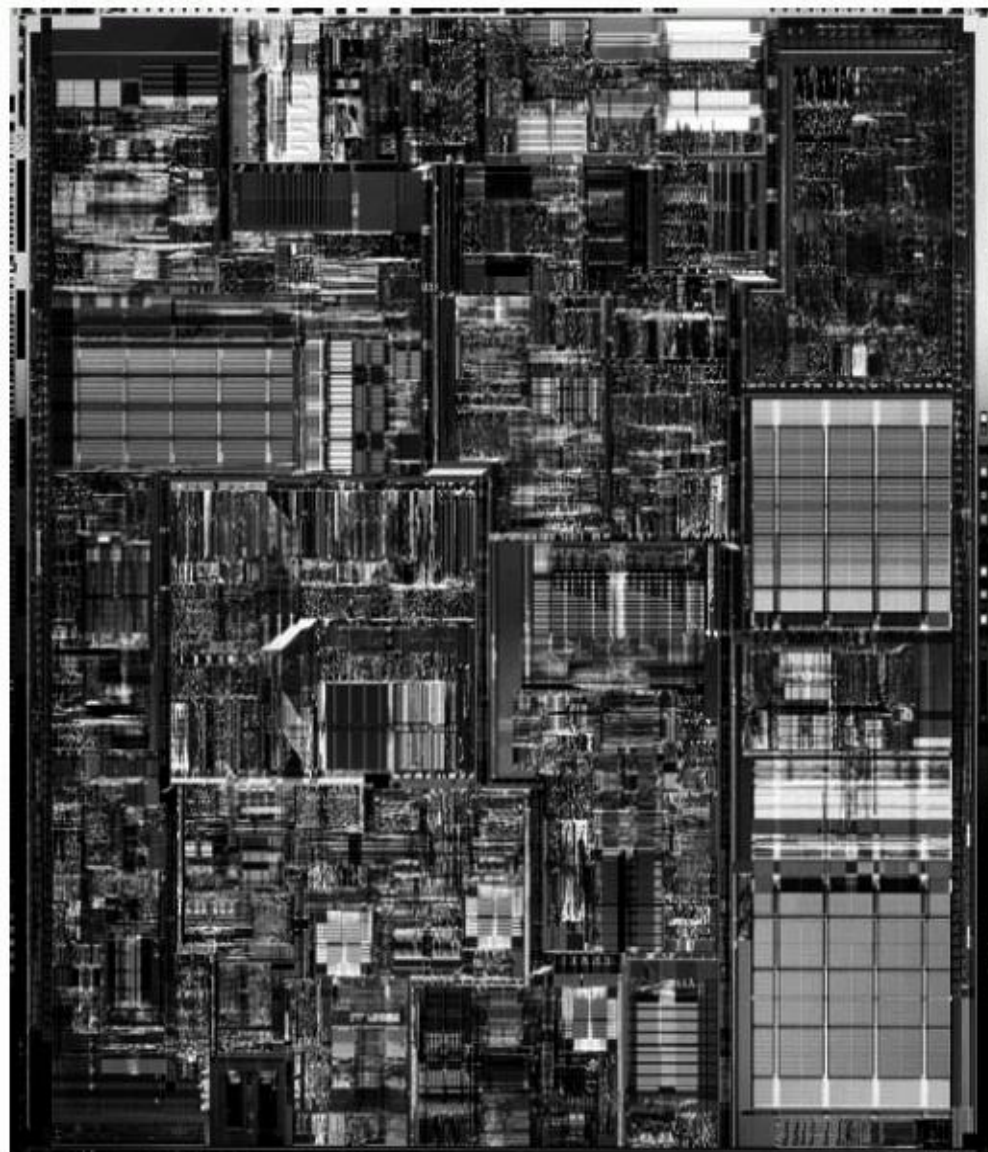


# PentiumPro芯片内的主要功能块



- Die Area: 306 mm<sup>2</sup>
  - 直径8 inch(200mm)的 Wafer最多可做 78个Die
  - $\approx 5,500,000$  Transistors
  - Cache:  $\approx 1\text{M}$  Transistors
  - External Cache:  
31M Transistors
- PentiumPro Package =  
PentiumPro+ExternalCache  
387 Pins

# Pentium4处理器内部布局



# 现代计算机的原型

1946年，普林斯顿高等研究院（ the Institute for Advance Study at Princeton , IAS ）开始设计“**存储程序**”计算机，被称为**IAS计算机**（ 1951年才完成，它并不是第一台存储程序计算机，1949年由英国剑桥大学完成的EDSAC是第一台）。

- 在那个“**关于EDVAC的报告草案**”报告中提出的计算机结构被称为**冯·诺依曼结构**。冯·诺依曼结构计算机也称为冯·诺依曼机器（ Von Neumann Machine ）。
- 冯·诺依曼结构最重要的思想是什么？

**“存储程序(Stored-program)”** 工作方式：

任何要计算机完成的工作都要先被编写成程序，然后将程序和原始数据送入主存并启动执行。一旦程序被启动，计算机应能在不需操作人员干预下，自动完成逐条取出指令和执行指令的任务。

- 几乎现代所有的通用计算机大都采用冯·诺依曼结构，因此，IAS计算机是现代计算机的原型机。



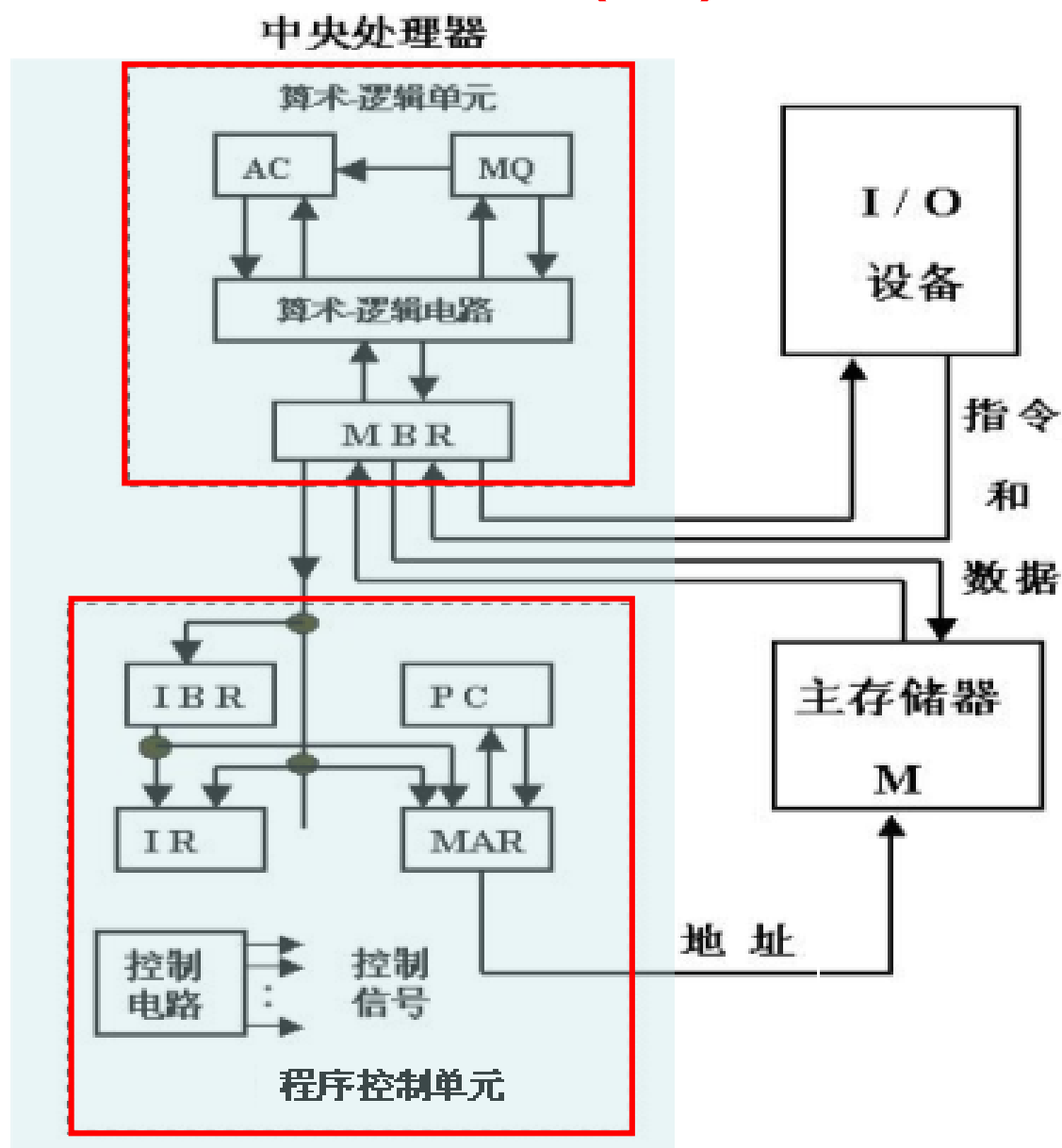
# 你认为冯·诺依曼结构是怎样的？

## 普林斯顿高等研究院(IAS)计算机结构

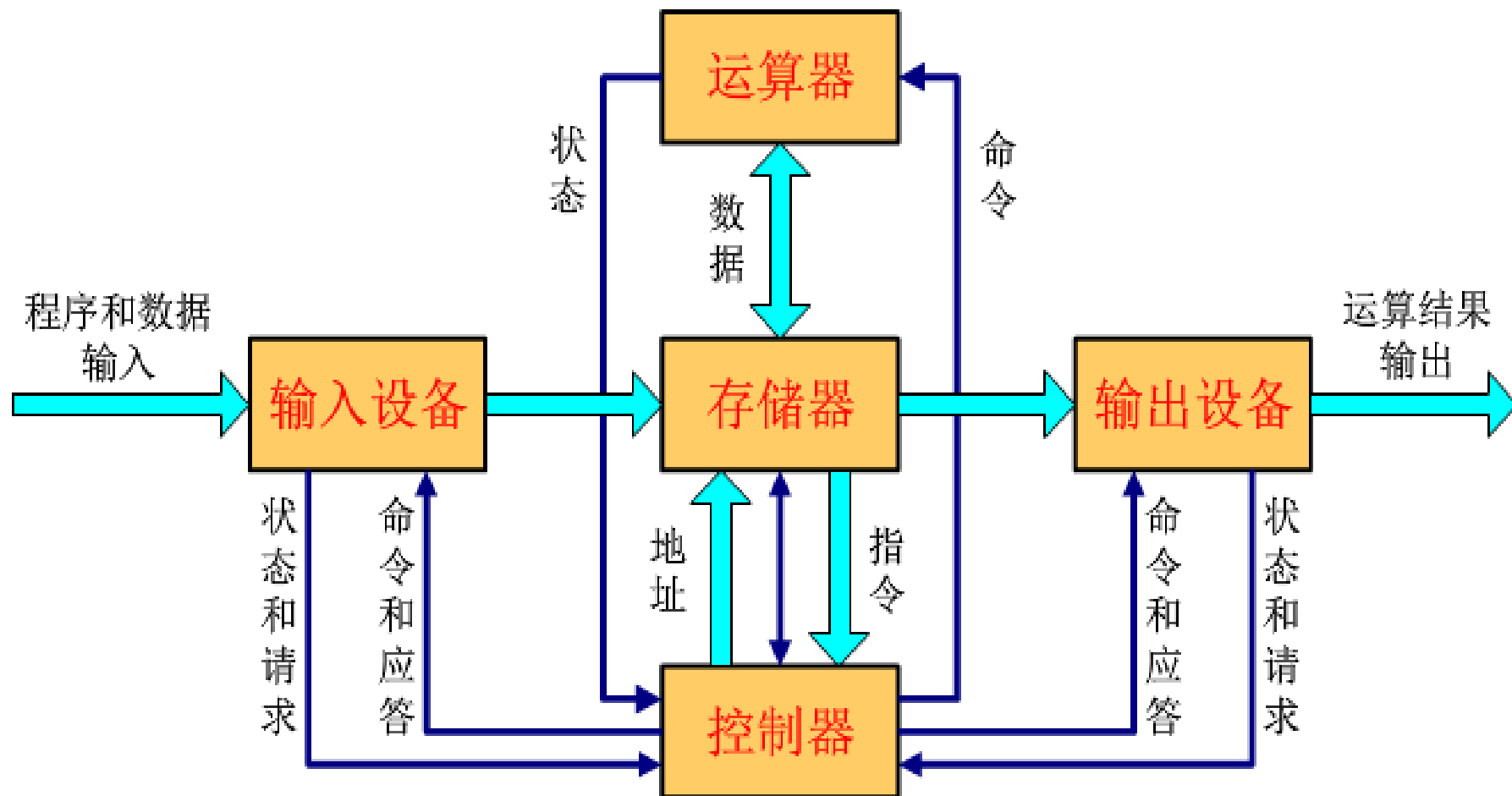
- 应该有个主存，用来存放程序和数据
- 应该有一个自动逐条取出指令的部件
- 还应该具体执行指令（即运算）的部件
- 程序由指令构成
- 指令描述如何对数据进行处理
- 应该有将程序和原始数据输入计算机的部件
- 应该有将运算结果输出计算机的部件

你还能想出更多吗？

你猜得八九不离十了😊



# 冯.诺依曼结构计算机模型



早期，部件之间用**分散方式**相连

现在，部件之间大多用**总线方式**相连

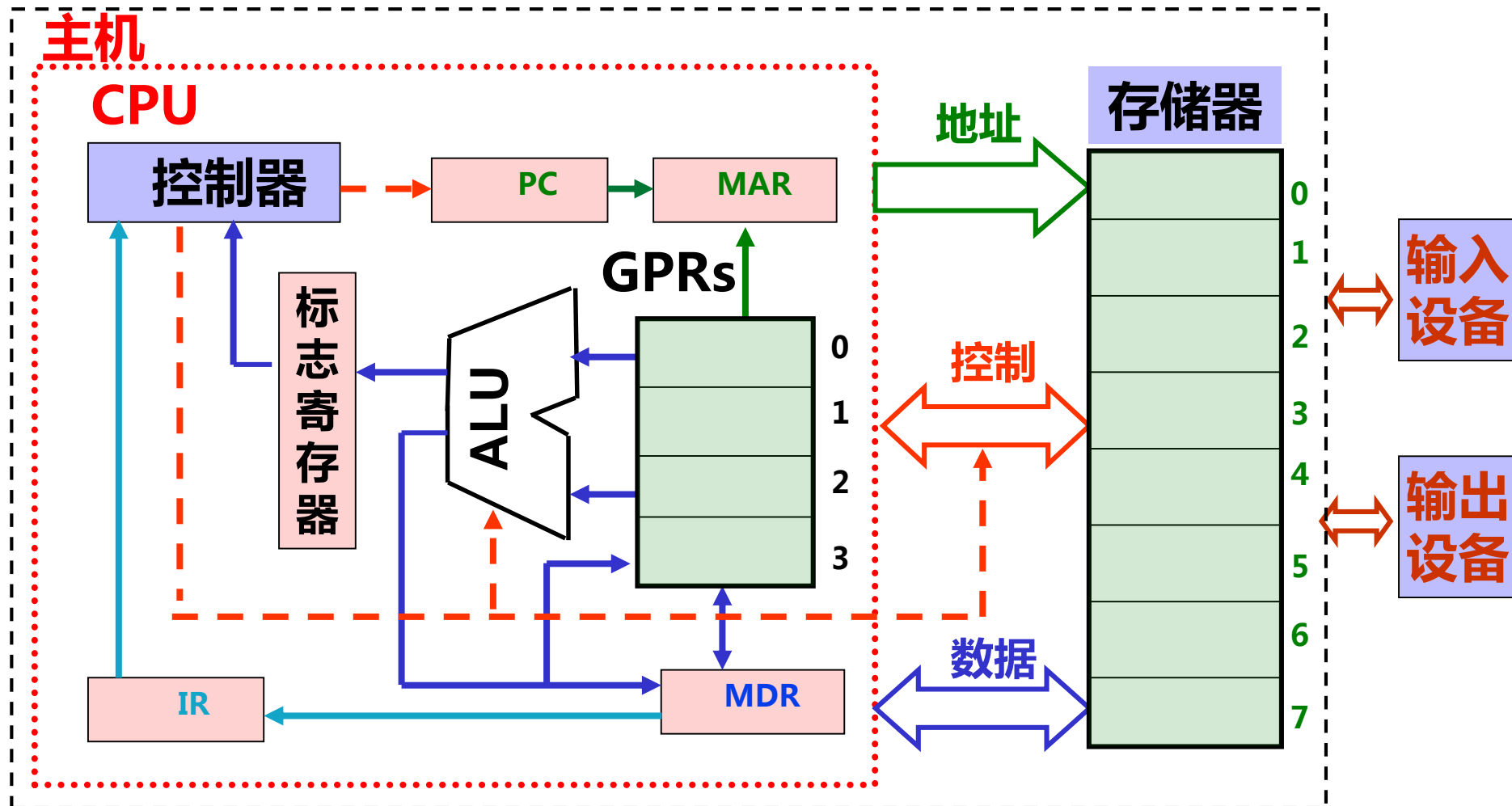
趋势，点对点（**分散方式**）高速连接

# 现代计算机结构模型—其中最基本的部件

**CPU**：中央处理器；**PC**：程序计数器；**MAR**：存储器地址寄存器

**ALU**：算术逻辑部件；**IR**：指令寄存器；**MDR**：存储器数据寄存器

**GPRs**：通用寄存器组（由若干通用寄存器组成，早期就是累加器）





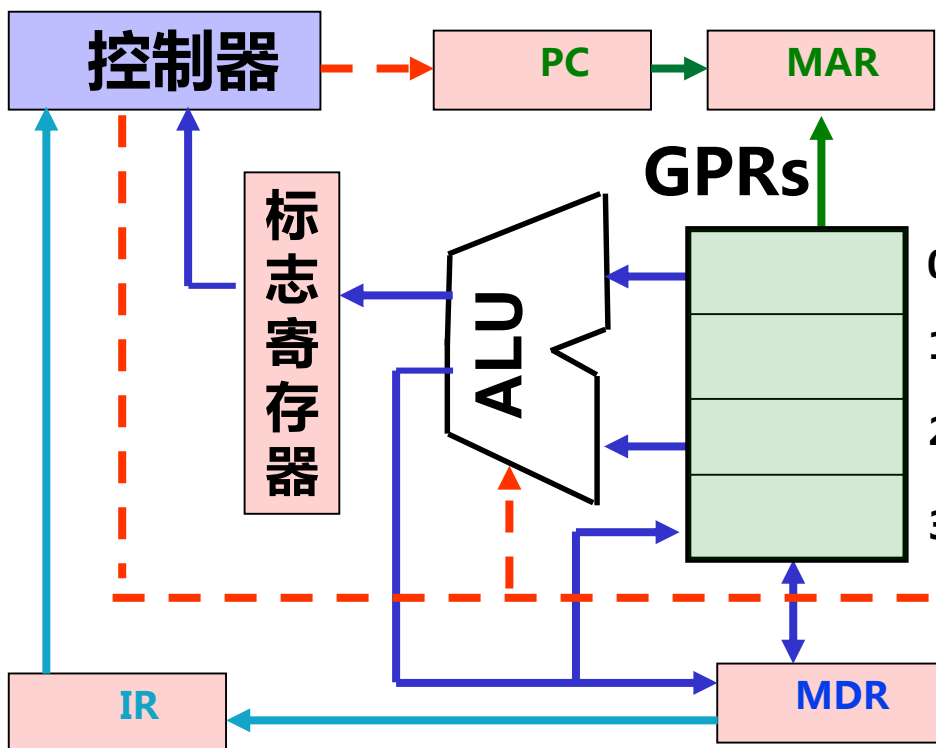
# 计算机是如何工作的？“存储程序”工作方式！

计算机相当于现实生活中的？工厂、饭店？如何工作的呢？如果你知道厨师(或家人)是如何做饭的，你就已经知道计算机是如何工作的！

厨房-CPU，厨师-控制器，盘-GPRs，锅灶等-ALU，架子-存储器

## 主机

### CPU



地址

存储器

0  
1  
2  
3  
4  
5  
6  
7

控制

数据

输入设备

输出设备

# 计算机是如何工作的？

## 类似“存储程序”工作方式

### ● 做菜前

原材料（**数据**）和菜谱（**指令**）都按序放在厨房外的架子（**存储器**）上，每个架子有编号（**存储单元地址**）。

菜谱上信息：原料位置、做法、做好的菜放在哪里等

例如，把10、11号架子上的原料一起炒，并装入3号盘

然后，我告诉厨师从第5个架上（**起始PC=5**）指定菜谱开始做

### ● 开始做菜

第一步：从5号架上取菜谱（**根据PC取指令**）

第二步：看菜谱（**指令译码**）

第三步：从架上或盘中取原材料（**取操作数**）

第四步：洗、切、炒等具体操作（**指令执行**）

第五步：装盘或直接送桌（**回写结果**）

第六步：算出下一菜谱所在架子号 $6 = 5 + 1$ （**修改PC的值**）

继续做下一道菜（**执行下一条指令**）

# 计算机是如何工作的？

## 程序由指令组成（菜单由菜谱组成）

- 程序在执行前

数据和指令事先存放在存储器中，每条指令和每个数据都有地址，指令按序存放，指令由OP、ADDR字段组成，程序起始地址置PC（原材料和菜谱都放在厨房外的架子上，每个架子有编号。厨师从第5个架上指定菜谱开始做）

- 开始执行程序

第一步：根据PC取指令（从5号架上取菜谱）

第二步：指令译码（看菜谱）

第三步：取操作数（从架上或盘中取原材料）

第四步：指令执行（洗、切、炒等具体操作）

第五步：回写结果（装盘或直接送桌）

第六步：修改PC的值（算出下一菜谱所在架子号 $6=5+1$ ）

继续执行下一条指令（继续做下一道菜）

# 指令和数据

- **程序启动前**，指令和数据都存放在存储器中，形式上没有差别，都是0/1序列
- 采用“**存储程序**”工作方式：
  - 程序由指令组成，程序被启动后，计算机能自动取出一条一条指令执行，在执行过程中无需人的干预。
- **指令执行过程中**，指令和数据被从存储器取到CPU，存放在CPU内的寄存器中，指令在**IR ( Instruction Register )**中，数据在**GPR**  
**指令中需给出的信息：**

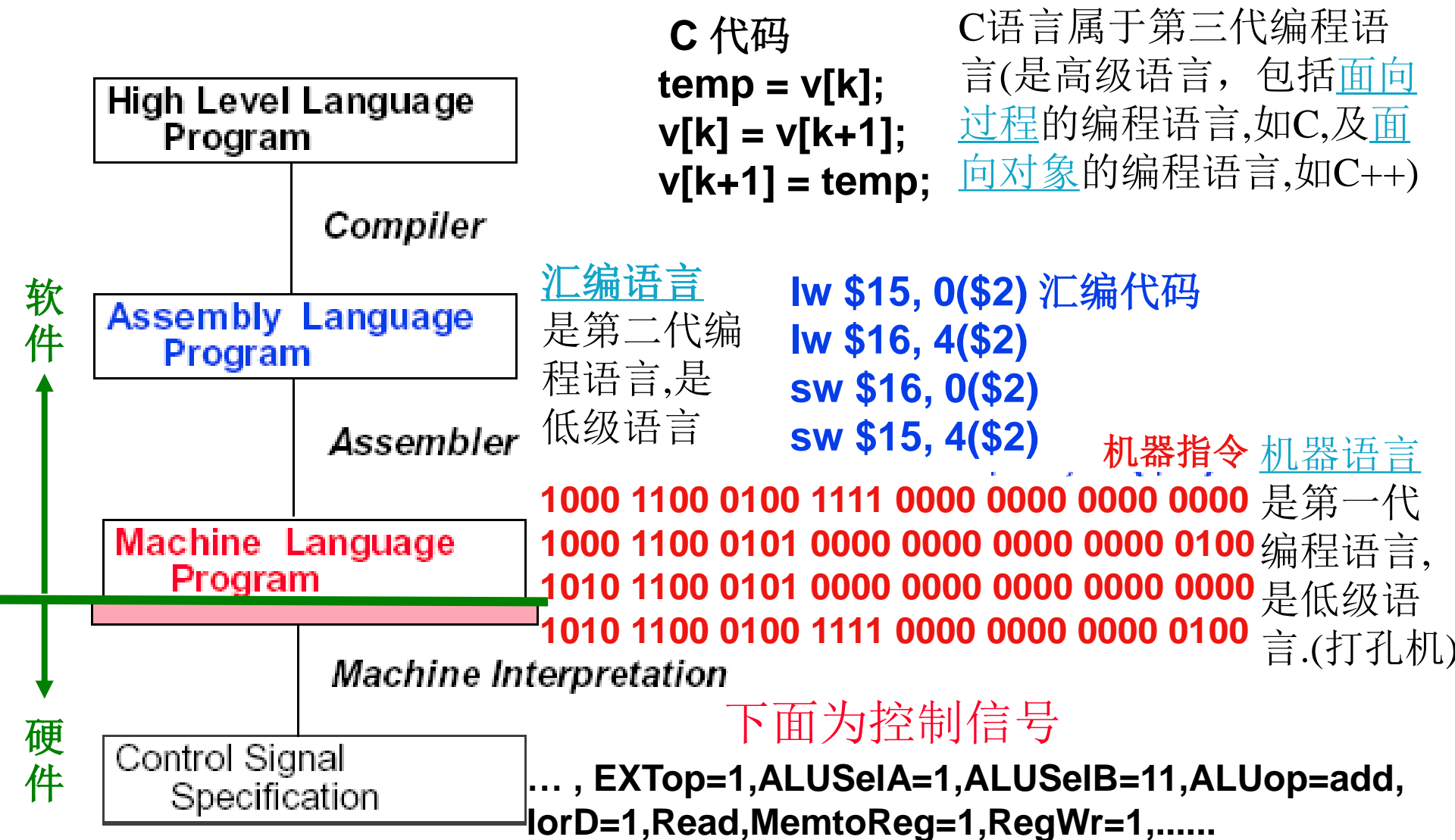
**操作性质（操作码）**

**源操作数1 或/和 源操作数2 （立即数、寄存器编号、存储地址）**

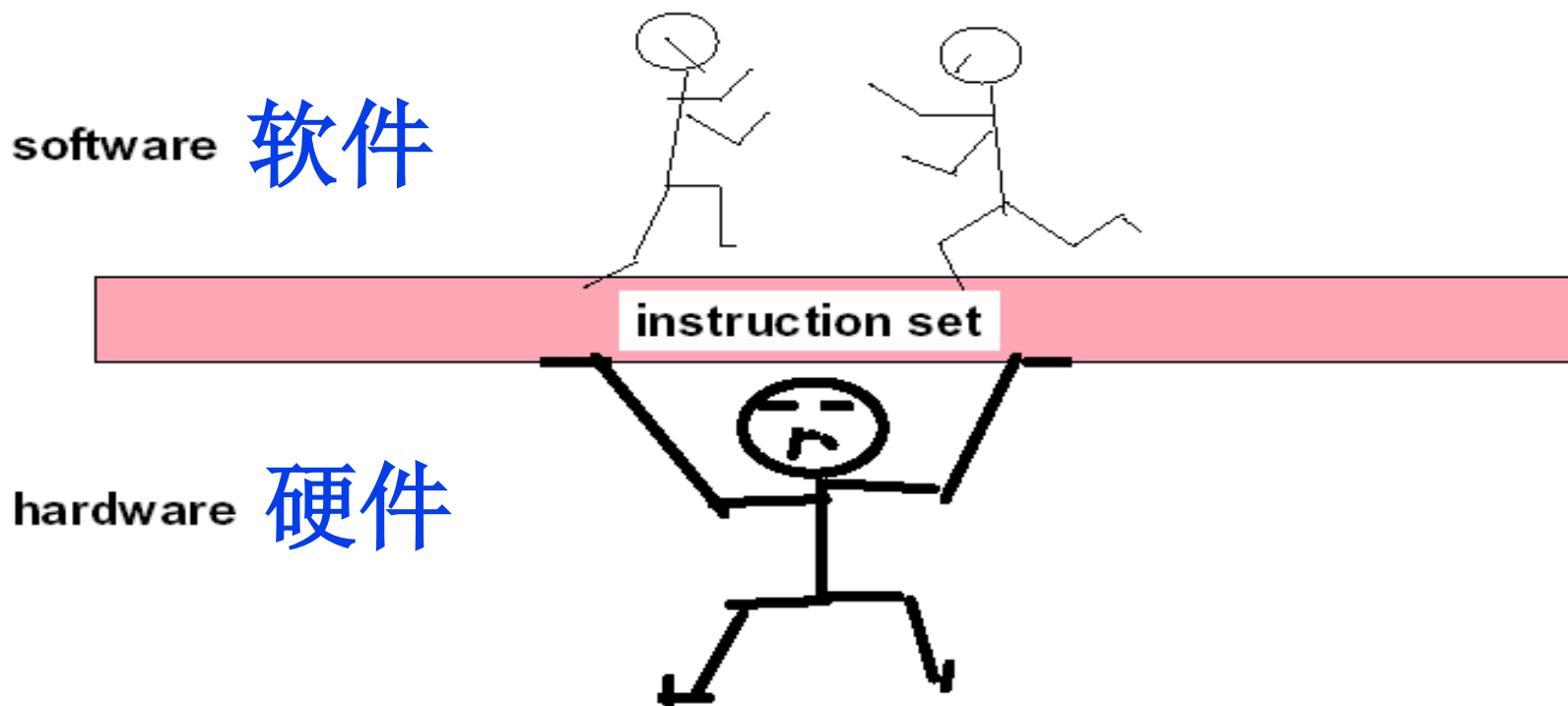
**目的操作数地址 （寄存器编号、存储地址）**

**存储地址的描述与操作数的数据结构有关！**

# Hardware/Software Interface



# Hardware/Software Interface (界面)



软件和硬件的界面： ISA (Instruction Set Architecture)  
指令集体系结构

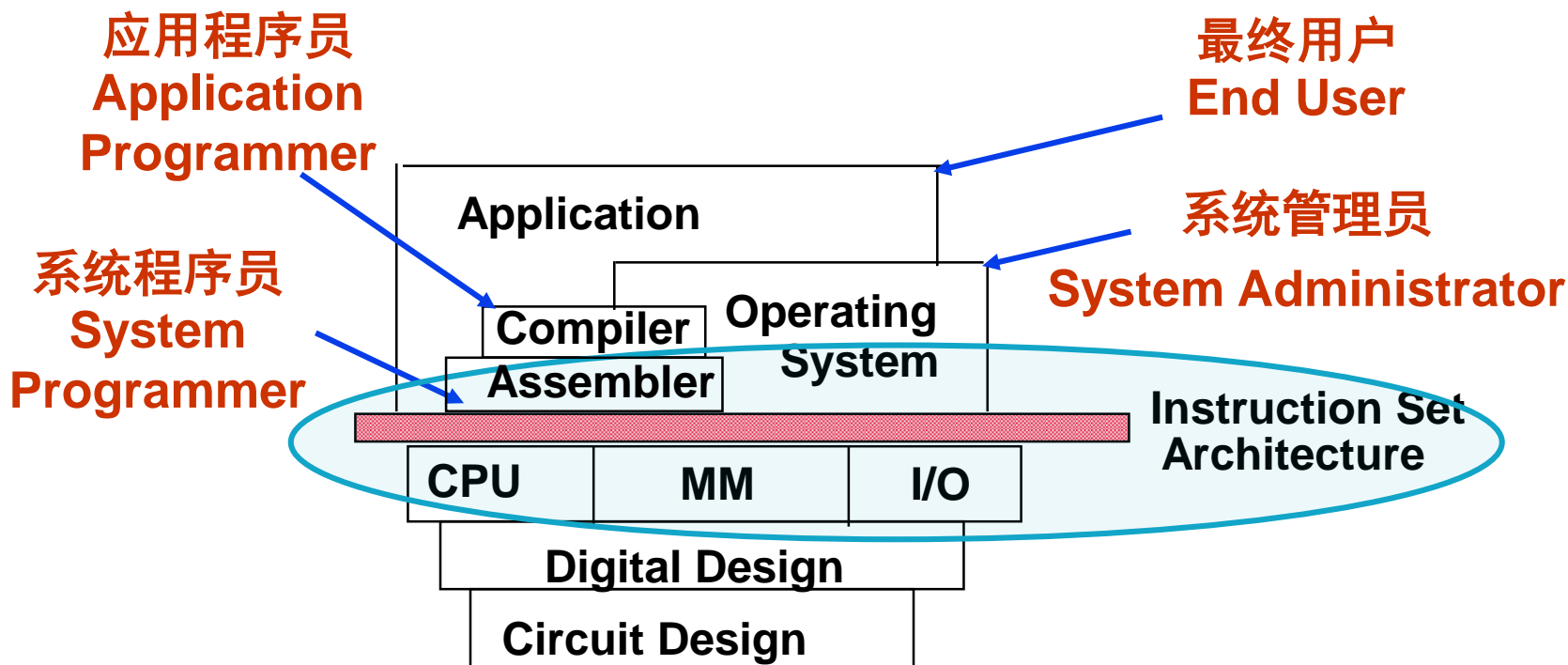
机器语言由指令代码构成，能被硬件直接执行。

# Software

- **System software(系统软件)** - 简化编程，并使硬件资源被有效利用
  - 操作系统 ( Operating System ) : 硬件资源管理，用户接口
  - 语言处理系统：翻译程序+ Linker, Debug, etc ...
    - 翻译程序(Translator)有三类：
      - 汇编程序(Assembler)：汇编语言源程序→机器目标程序
      - 编译程序(Compiler)：高级语言源程序→汇编/机器目标程序
      - 解释程序(Interpreter)：将高级语言语句逐条翻译成机器指令并立即执行,不生成目标文件。
  - 其他实用程序: 如：磁盘碎片整理程序、备份程序等
- **Application software(应用软件)** - 解决具体应用问题/完成具体应用
  - 各类媒体处理程序：Word/ Image/ Graphics/...
  - 管理信息系统 (MIS)
  - Game, ...

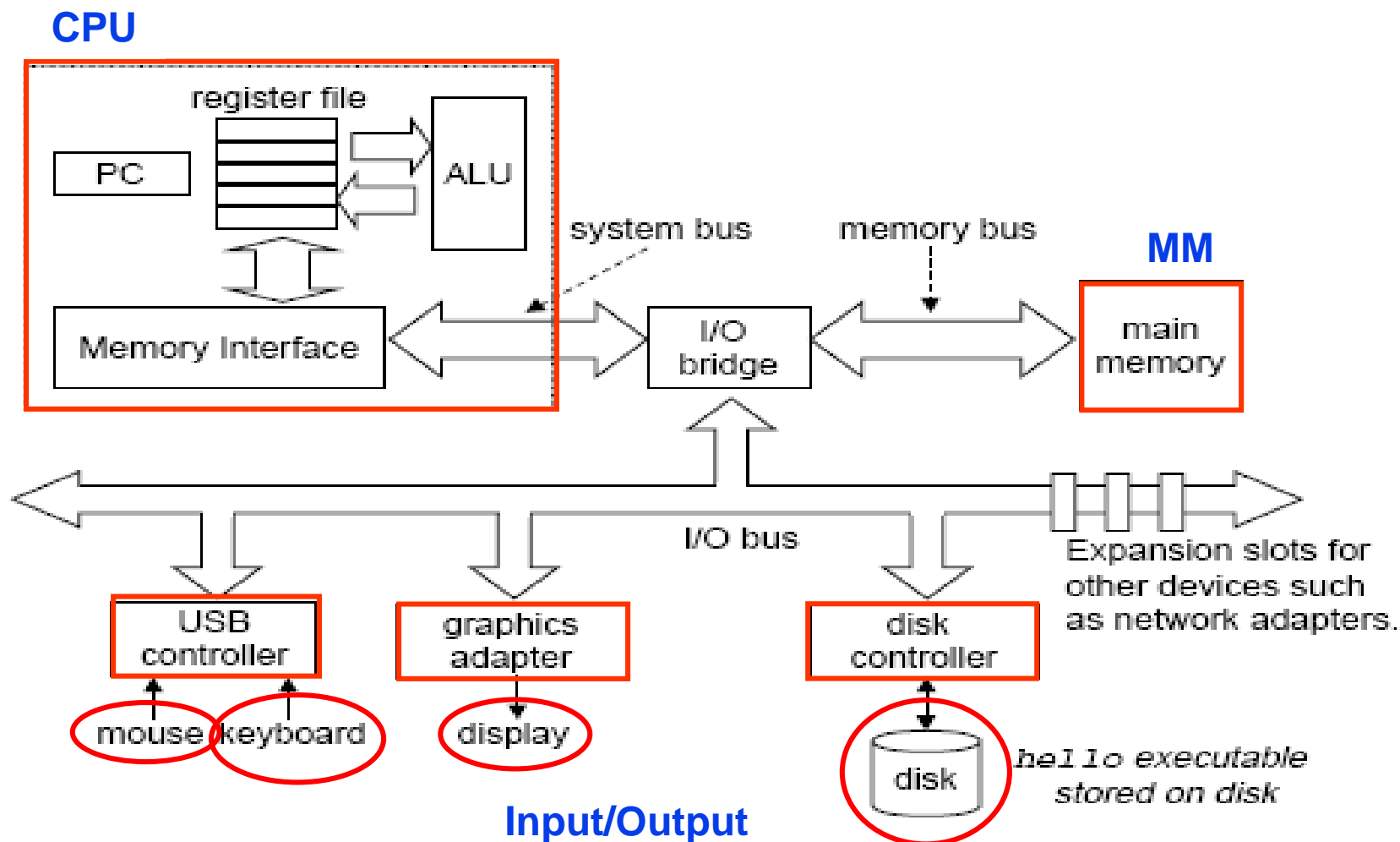


# Computer Hierarchy (计算机系统层次)



- 。 上图给出的是计算机系统的层次结构  
指令系统（即ISA）是软/硬件的交界面
- 。 不同用户工作在不同层次，所看到的计算机不一样
- 。 中间阴影部分就是本课程主要内容，处于最核心的部分！

# 一个典型系统的硬件组成

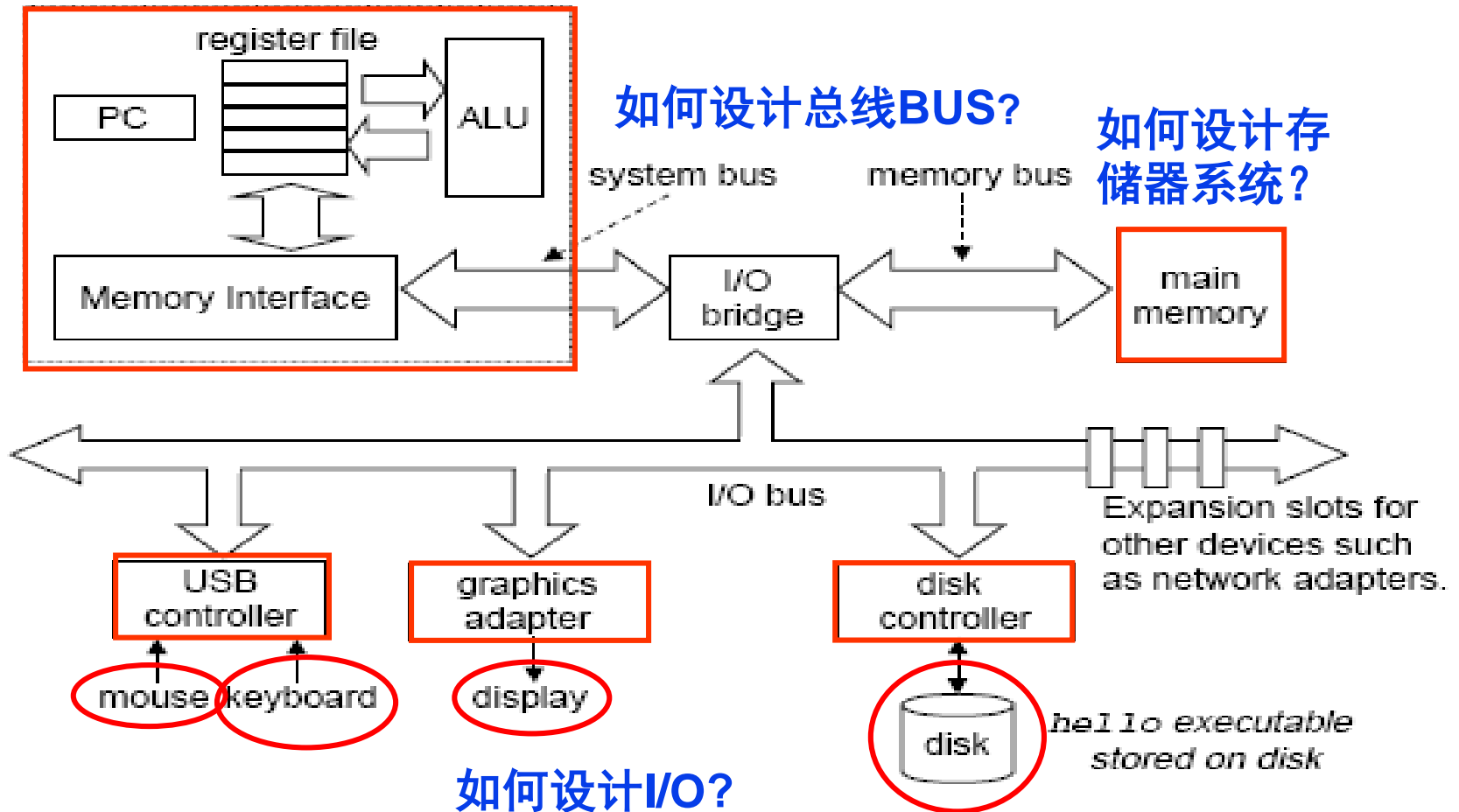


**PC (program counter) :** 程序计数器;

**ALU:** 算术/逻辑单元; **USB:** 通用串行总线

# 本课程及其实验课程的主要学习内容

如何设计高性能CPU?



信息（指令和数据）在计算机中如何表示？  
指令系统如何设计？

# 早期计算机系统的层次

- 最早的计算机用机器语言编程(低级语言)  
机器语言称为第一代程序设计语言 ( First generation programming language , 1GL ) (打孔机)

应用程序

指令集体系结构

计算机硬件

- 后来用汇编语言编程(低级语言)

汇编语言称为第二代程序设计语言 ( Second generation programming language , 2GL )

汇编程序

操作系统

指令集体系结构

计算机硬件

# 现代（传统）计算机系统的层次

° 现代计算机用高级语言编程,包括:

° 第三代程序设计语言（3GL,高级语言）为过程式语言，编码时需要描述实现过程，即“如何做”，其中分为一、面向过程的编程语言（如C，BASIC,pascal），二、面向对象的编程语言（如C++，java ,c#）

° 第四代程序设计语言（4GL,高级语言）为非过程化语言，编码时只需说明“做什么”，不需要描述具体的算法实现细节。数据库编程语言，如各种数据库的SQL语言

可以看出：语言的发展是一个不断“抽象”的过程，因而，相应的计算机系统也不断有新的层次出现

应用程序

语言处理系统

操作系统

指令集体系结构

计算机硬件

**语言处理系统**包括：各种语言处理程序（如编译、汇编、链接）、运行时系统（如库函数，调试、优化等功能）

**操作系统**包括人机交互界面、提供服务功能的内核例程

# 一个典型程序的转换处理过程

## 经典的 “hello.c” C-源程序

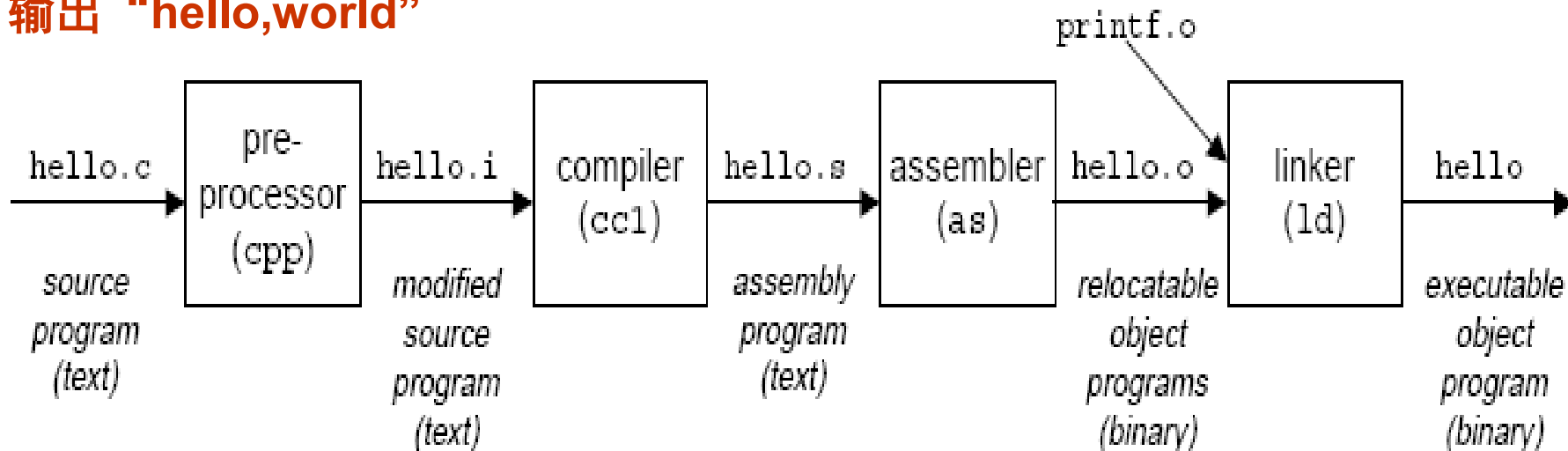
```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello, world\n");
6 }
```

程序的功能是：

输出 “hello,world”

## hello.c的ASCII文本表示

```
# i n c l u d e < s p > < s t d i o .
35 105 110 99 108 117 100 101 32 60 115 116 100 105 111 46
h > \n \n i n t < s p > m a i n ( ) \n {
104 62 10 10 105 110 116 32 109 97 105 110 40 41 10 123
\n < s p > < s p > < s p > < s p > p r i n t f ( " h e l
10 32 32 32 32 112 114 105 110 116 102 40 34 104 101 108
l o , < s p > w o r l d \n " ) ; \n }
108 111 44 32 119 111 114 108 100 92 110 34 41 59 10 125
```



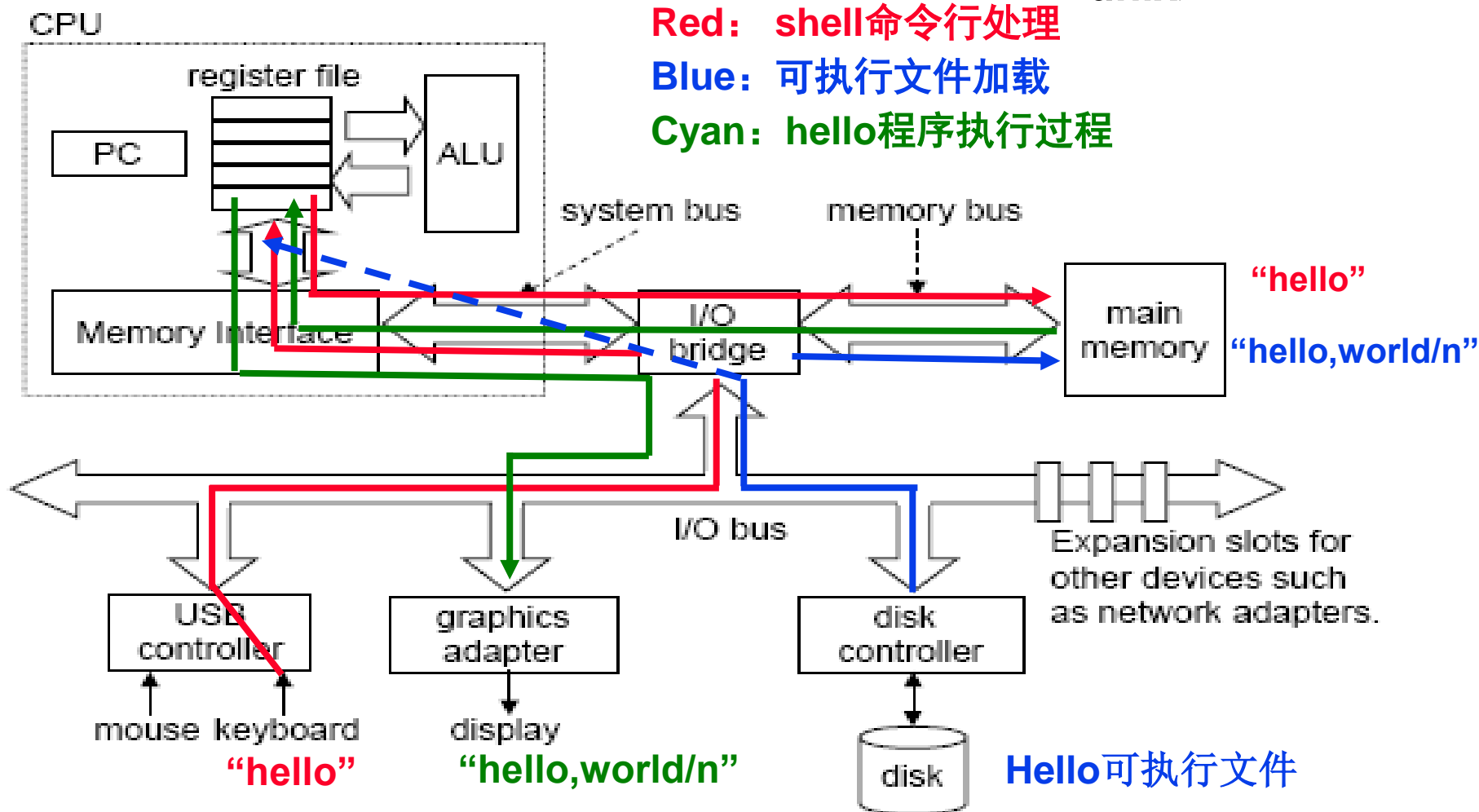
# Hello程序的数据流动过程

```
Unix>./hello [Enter]  
hello, world  
unix>
```

Red: shell命令行处理

Blue: 可执行文件加载

Cyan: hello程序执行过程



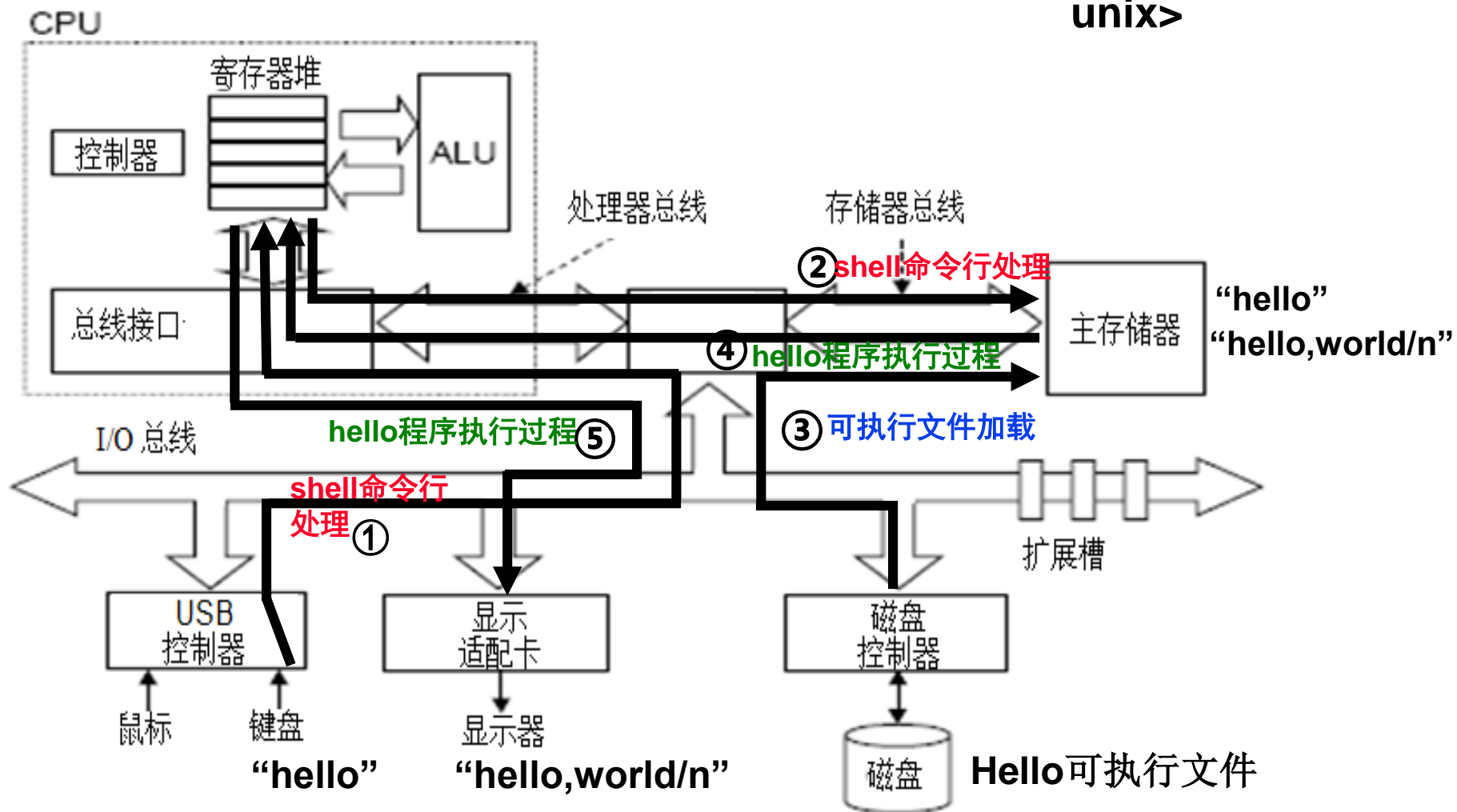
数据经常在各存储部件间传送。故现代计算机大多采用“缓存”技术！

所有过程都是在CPU执行指令所产生的控制信号的作用下进行的。



# Hello程序的数据流动过程

```
Unix> ./hello [Enter]  
hello, world  
unix>
```



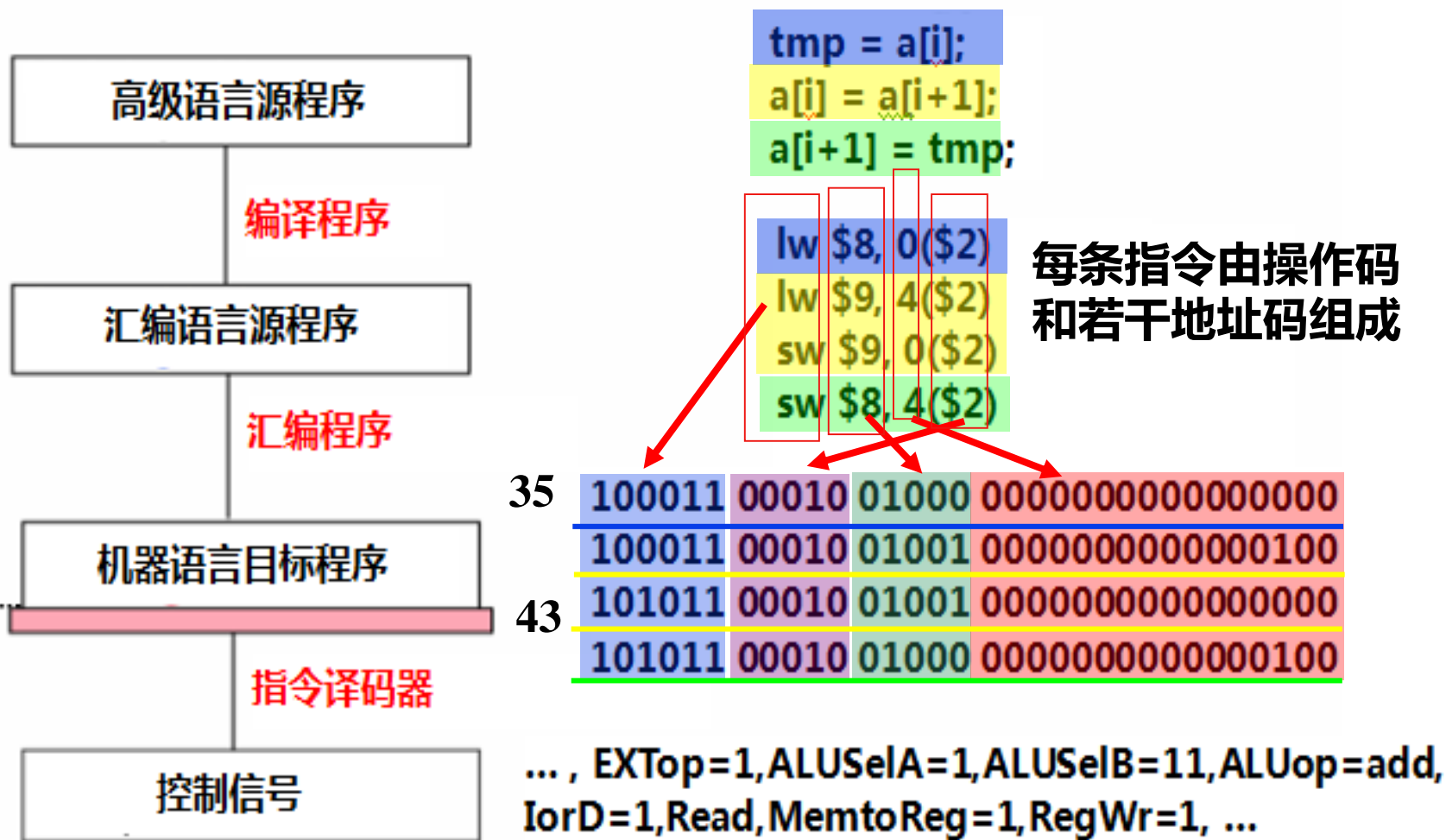
# Course Outline

---

- 性能评价 (Performance measurement)
- 计算机算术 (Arithmetic for Computer)
  - 数据的表示和运算
- 存储器层次结构 (Memory Hierarchies)
- 指令集体系结构 (Instruction Set Architecture)
- CPU设计
  - 数据通路 (Data path) 和控制器 (Control Unit)
- 流水线技术 (Pipelining)
- 系统总线 (System Buses)
- 输入/输出系统 (Input / Output system)

# 不同层次语言之间的等价转换

计算机软件  
-----  
计算机硬件



**任何高级语言程序最终通过执行若干条指令来完成！**

# 开发和运行程序需什么支撑？

◦ 最早的程序开发很简单（怎样简单？）

- 直接输入指令和数据，启动后把第一条指令地址送PC开始执行

◦ 用高级语言开发程序需要复杂的支撑环境（怎样的环境？）

- 需要编辑器编写源程序
- 需要一套翻译转换软件处理各类源程序

- 编译方式：预处理程序、编译器、汇编器、链接器
- 解释方式：解释程序

语言  
处理  
程序

语言处理系统 +

- 需要一个可以执行程序的界面（环境）

- GUI方式：图形用户界面
- CUI方式：命令行用户界面

人机  
接口

+  
操作  
系统

语言的运行时系统

操作系统内核

指令集体系结构

计算机硬件

支撑程序开发和运行的环境由系统软件提供

最重要的系统软件是操作系统和语言处理系统

语言处理系统运行在操作系统之上，操作系统利用指令管理硬件

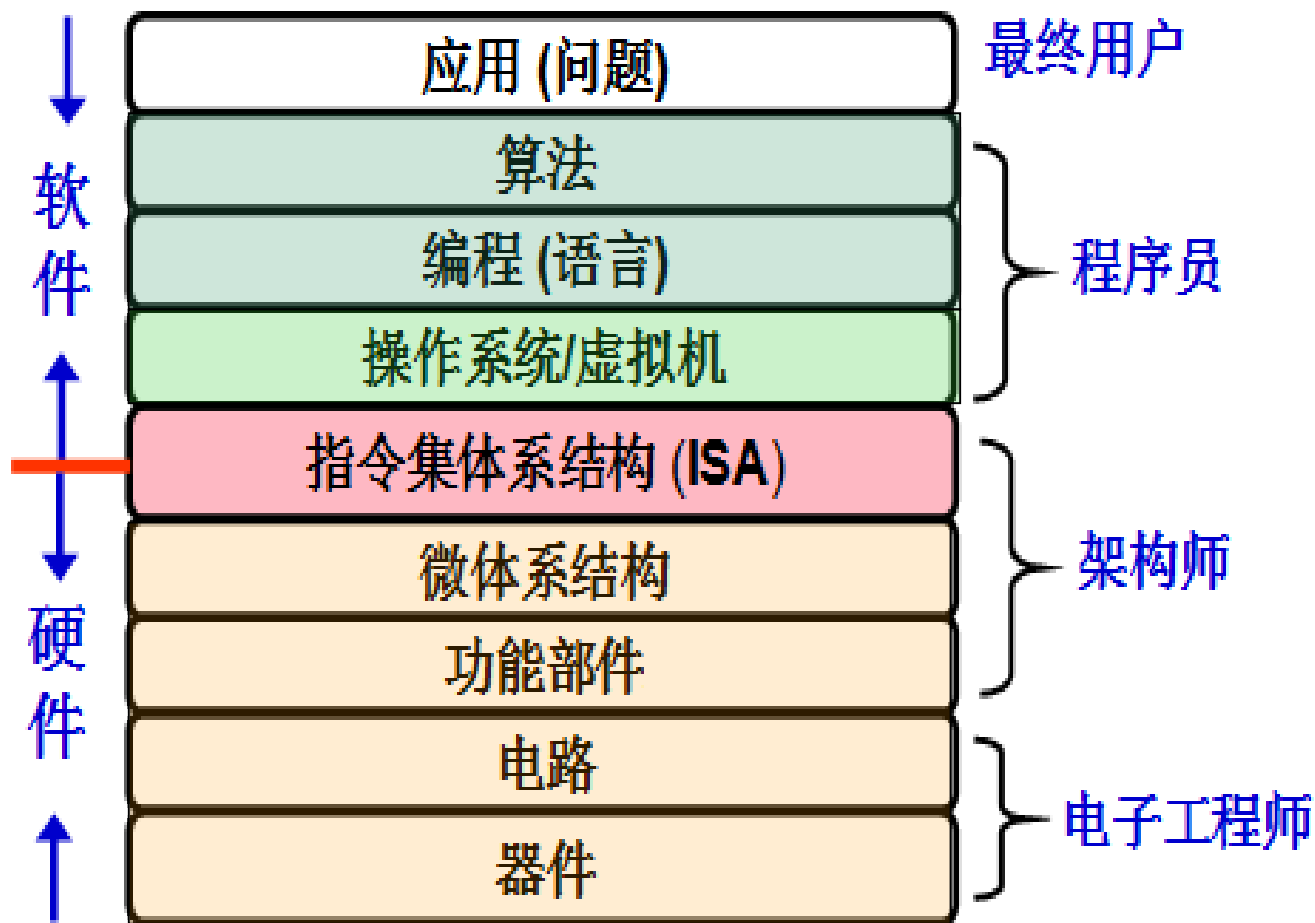
# 计算机系统抽象层的转换

**程序执行结果**  
不仅取决于  
**算法、程序编写**  
而且取决于  
**语言处理系统**  
**操作系统**  
**ISA**  
**微体系结构**

**不同计算机课程**  
**处于不同层次**

**必须将各层次关**  
**联起来解决问题**

**功能转换**：上层是下层的**抽象**，下层是上层的**实现**  
**底层为上层提供支撑环境！**



**最高层抽象就是点点鼠标、拖拖图标、敲敲键盘，但这背后有多少层转化啊！**

# 计算机系统的不同用户

**最终用户**工作在由应用程序提供的最上面的抽象层

**系统管理员**工作在由操作系统提供的抽象层

**应用程序员**工作在由语言处理系统（**主要有编译器和汇编器**）的抽象层

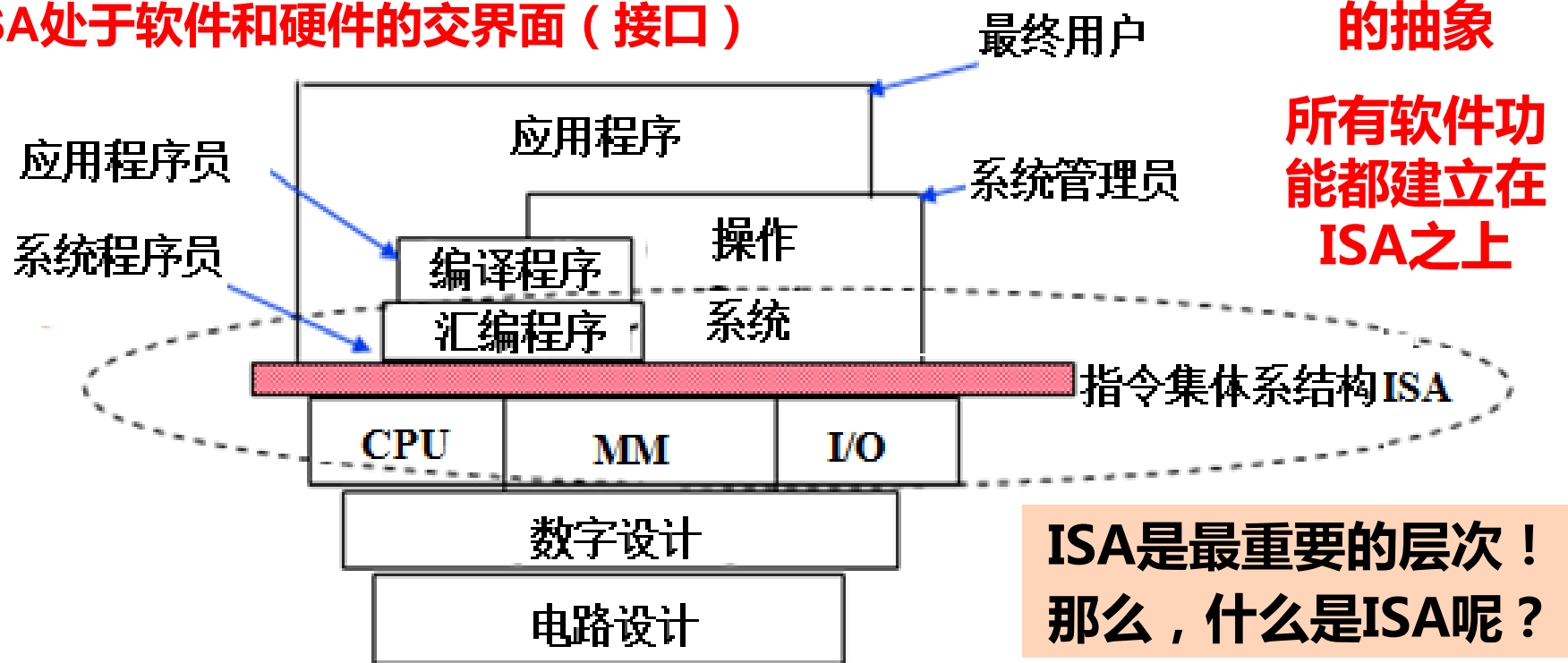
**语言处理系统**建立在**操作系统**之上

**系统程序员**（实现系统软件）工作在ISA层次，必须对ISA非常了解

**编译器和汇编器的目标程序**由**机器级代码**组成

**操作系统**通过**指令**直接对**硬件**进行编程控制

**ISA处于软件和硬件的交界面（接口）**

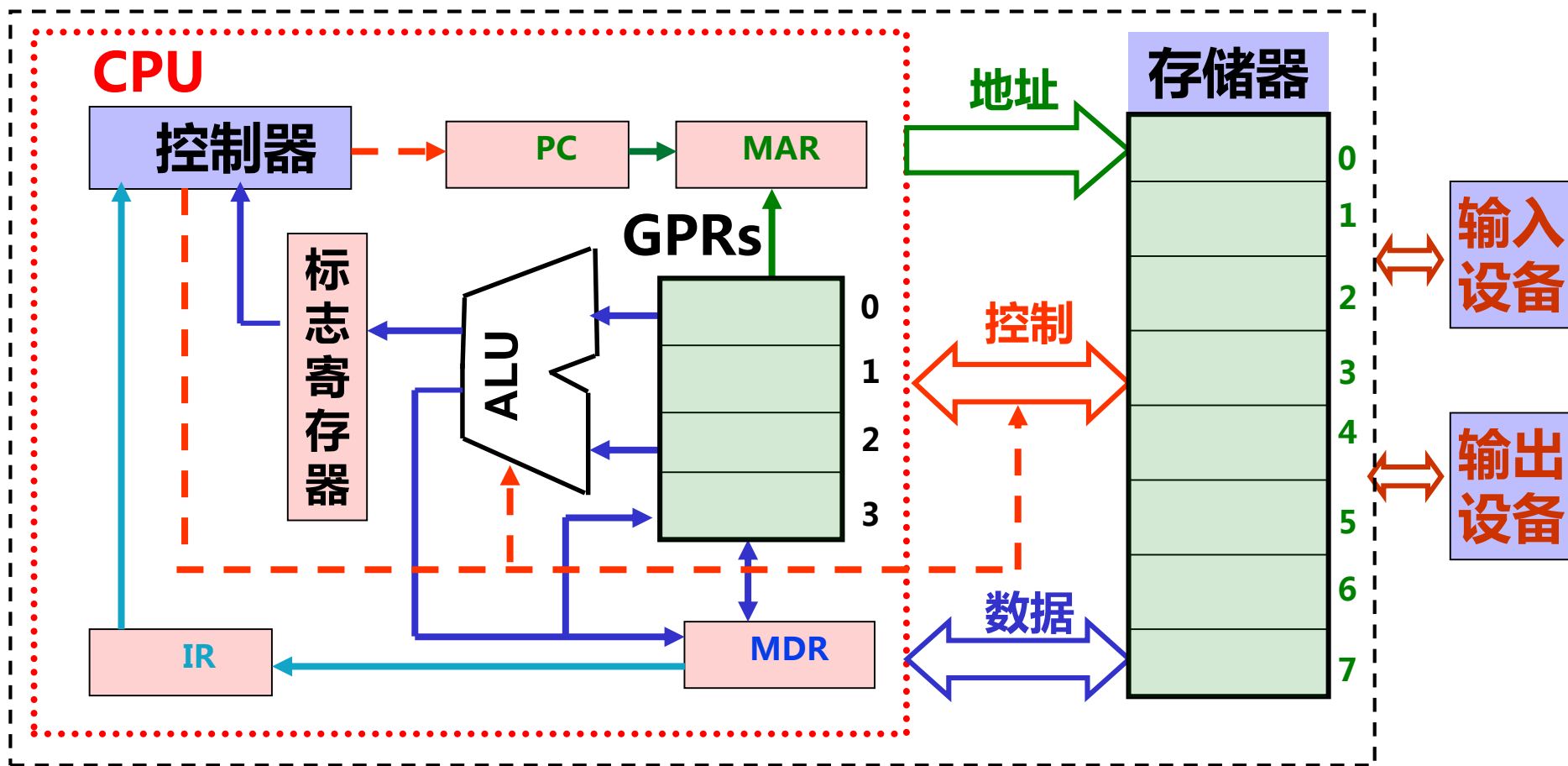




# 指令集体系结构（ISA）

- ISA指Instruction Set Architecture，即指令集体系结构
- ISA是一种规约（Specification），它规定了**如何使用硬件**
  - 可执行的指令的集合，包括**指令格式、操作种类以及每种操作对应的操作数的相应规定**；
  - 指令可以接受的**操作数的类型**；
  - 操作数所能存放的**寄存器组的结构，包括每个寄存器的名称、编号、长度和用途**；
  - 操作数所能存放的**存储空间的大小和编址方式**；
  - 操作数在存储空间存放时按照**大端还是小端方式存放**；
  - 指令获取操作数的方式，即**寻址方式**；
  - 指令执行过程的控制方式，包括**程序计数器、条件码定义等**。
- ISA在计算机系统中是必不可少的一个抽象层，Why？
  - 没有它，软件无法使用计算机硬件！
  - 没有它，一台计算机不能称为“通用计算机”

# ISA和计算机组成（微结构）之间的关系



不同ISA规定的指令集不同，如，IA-32、MIPS、ARM等  
计算机组成必须能够实现ISA规定的功能，如提供GPR、标志、运算电路等  
同一种ISA可以有不同的计算机组成，如乘法指令可用ALU或乘法器实现

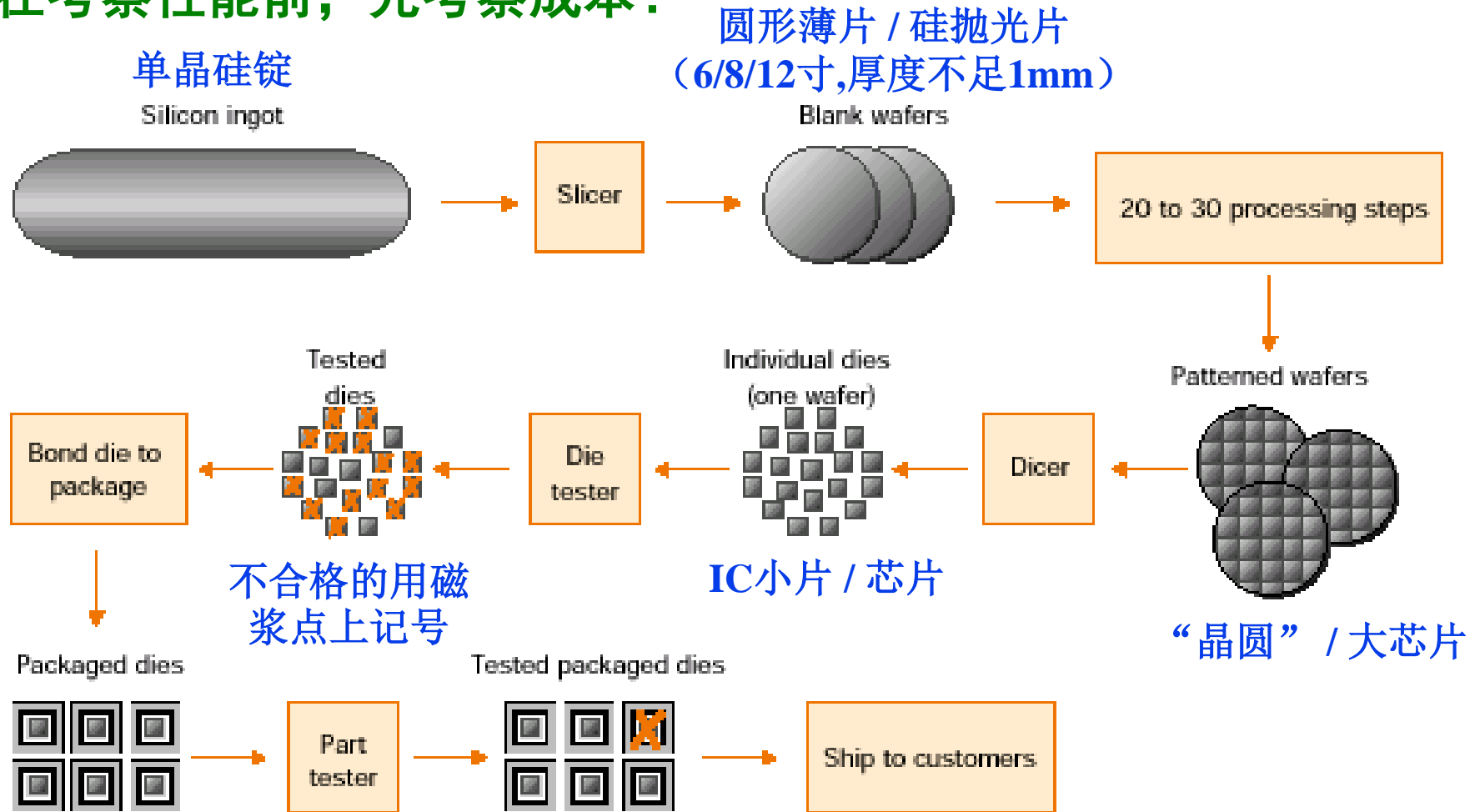
**ISA是计算机  
组成的抽象**

# 第二讲 计算机性能评价

- 制造成本 (manufacturing cost)
- 衡量计算机性能的基本指标
  - 响应时间 (response time)
    - 执行时间 (execution Time)、等待时间 (latency)
  - 吞吐量 (throughput)
    - 带宽 (bandwidth)
- 计算机性能测量
- 指令执行速度 (MIPS、MFLOPS/TFLOPS/PFLOPS)
- 基准程序 (Benchmark)
  - SPEC (Systems Performance Evaluation Committee)
  - Linpack (Linear system package): 线性系统软件包基准测试程序

# 回顾：Integrated Circuits manufacturing process

在考察性能前，先考察成本！



封装：将芯片固定在塑胶或陶瓷基座上，把芯片上蚀刻出来的引线与基座底部伸出的引脚连接，盖上盖板并封焊成芯片

约需400多道工序！

# Integrated Circuits Costs 公式

芯片成本与以下因素有关：

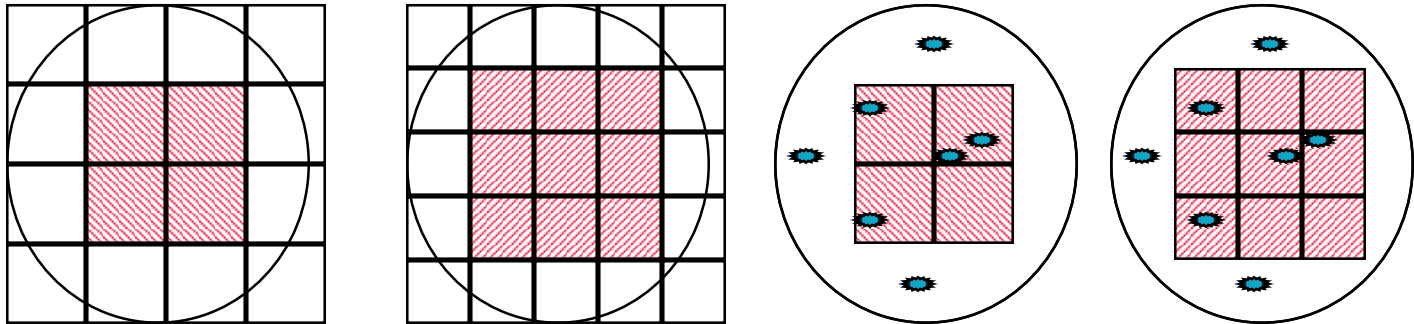
- 圆晶 (wafer) 价格
- 圆晶 (wafer) 所含小片 (die) 数
- 小片合格率

IC小片的成本

$$\text{Die cost} = \frac{\text{Cost\_per\_wafer}}{\text{Die\_per\_wafer} \times \text{Yield}}$$

每片晶圆能够有多少 IC小片

$$\text{Dies per wafer} = \frac{\text{wafer\_area}}{\text{Die\_area}}$$



小片合格率

$$\text{Die Yield} = \frac{1}{(1 + (\text{Defect\_per\_area} \times \text{Die\_area}))^2}$$

由此看出：每个圆晶片上的小片数、集成电路成本都与芯片面积有关！

# 计算机性能的基本评价指标

- 计算机有两种不同的性能

不同应用场合用户关心的性能不同：

- Time to do the task

-要求吞吐率高的场合，例如：

- 响应时间（response time）

多媒体应用（音/视频播放要流畅）

- 执行时间（execution time）

-要求响应时间短的场合：例如：

- 等待时间或时延（latency）

事务处理系统（存/取款速度要快）

- Tasks per day, hour, sec, ns. ..

-要求吞吐率高且响应时间短的场合：

- 吞吐率（throughput）

ATM、文件服务器、Web服务器等

- 带宽（bandwidth）

- 基本的性能评价标准是：CPU的执行时间

程序由指令构成。

“机器X的速度（性能）是Y的n倍”的含义：

CPU执行时间就是  
执行程序中每条指令的时间。

$$\frac{\text{ExTime}(Y)}{\text{ExTime}(X)} = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = n$$

相对性能用执行时间的倒数来表示！



# 计算机性能的测量

- 比较计算机的性能时，用执行时间来衡量
  - 完成同样工作量所需时间最短的那台计算机就是性能最好的
  - 处理器时间往往被多个程序共享使用，因此，用户感觉到的程序执行时间并不是程序真正的执行时间（从hello程序执行过程可知）
  - 通常把用户感觉到的响应时间分成以下两个时间：
    - CPU时间：指CPU真正花在程序执行上的时间。又包括两部分：
      - 用户CPU时间：用来运行用户代码的时间
      - 系统CPU时间：为了执行用户程序而需要运行操作系统程序的时间
    - 其他时间：指等待I/O操作完成或CPU花在其他用户程序的时间
  - 系统性能和CPU性能不等价，有一定的区别
    - 系统性能(System performance)：系统响应时间，与CPU外的其他部分也都有关系
    - CPU性能(CPU performance)：用户CPU时间
  - 本章主要讨论CPU性能，即：CPU真正用在用户程序执行上的时间
- 问题：用户CPU时间与系统响应时间哪个更长？

# CPU执行时间的计算

**CPI(Cycles Per Instruction)=(总运行)时钟周期数/ (总运行)指令数**

$$\begin{aligned}\text{CPU 执行时间} &= (\text{CPU时钟周期数} / \text{程序}) \times \text{时钟周期} \\ &= (\text{CPU时钟周期数} / \text{程序}) \div \text{时钟频率} \\ &= (\text{指令条数} / \text{程序}) \times \text{CPI} \times \text{时钟周期}\end{aligned}$$

$$\text{CPU时钟周期数} / \text{程序} = (\text{指令条数} / \text{程序}) \times \text{CPI}$$

$$\begin{aligned}\text{CPI} &= (\text{CPU时钟周期数} / \text{程序}) \div (\text{指令条数} / \text{程序}) \\ &= (\text{总运行时钟周期数}) \div (\text{总运行指令条数})\end{aligned}$$

CPI 用来衡量以下各方面的综合结果

- Instruction Set Architecture (ISA)
- Implementation of that architecture  
(Organization & Technology)
- Program (Compiler、Algorithm)

# Architecture = Instruction Set Arch. + Organization

## Computer Design

### Instruction Set Design

- Machine Language
- Compiler View
- "Computer Architecture"
- "Instruction Set Architecture"

"Building Architect"

“建筑设计师”：功能定义与设计

### Computer Hardware Design

- logic & physical Implementation
- Logic Designer's View
- "Micro-Architecture"
- "Computer Organization"

"Construction Engineer"

“建造工程师”：具体逻辑结构设计和物理实现技术

例如，是否提供“乘法指令”是ISA设计要考虑的问题；如何实现乘法指令（用专门乘法器还是用一个加法器+移位器实现）是组成(Organization)考虑的问题；如何布线、用什么材料和工艺设计等是实现技术(Technology)考虑的问题。

# Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

三个因素与哪些方面有关？

例如，{.....

y=4\*x; Programming

}

Compiler

Instr. Set Arch.

Organization

Technology

	instr. count	CPI	clock rate
Programming	✓	✓	
Compiler	✓	( ✓ )	
Instr. Set Arch.	✓	✓	
Organization		✓	✓
Technology			✓

问题：计算机性能与ISA、计算机组织（Organization）、计算机实现技术（Technology）三者的关系是什么？

# 如何计算CPI?

对于某一条特定的指令而言，其CPI是一个确定的值。但是，对于某一个程序或一台机器而言，其CPI是一个平均值，表示该程序或该机器指令集中每条指令执行时平均需要多少时钟周期。

时钟频率=每秒执行多少个时钟周期, 1/时钟频率=每时钟周期执行了多少秒

假定  $CPI_i$  和  $C_i$  分别为第  $i$  类指令的CPI和指令条数(count)，则程序的总时钟数为：

$$\text{总时钟数} = \sum_{i=1}^n CPI_i \times C_i \quad \text{所以,} \quad \text{CPU时间} = \text{时钟周期} \times \sum_{i=1}^n CPI_i \times C_i$$

假定  $CPI_i$ 、 $F_i$  是各指令CPI和在程序中的出现频率(Frequency), 则程序综合CPI为:

$$CPI = \sum_{i=1}^n CPI_i \times F_i \quad \text{where} \quad F_i = \frac{C_i}{(\text{程序中总})Instruction\_Count}$$

已知CPU时间、时钟频率、总时钟数、指令条数，则程序综合CPI为:

$$CPI = (\text{CPU(执行)时间} \times \text{时钟频率}) / \text{指令条数} = \text{总时钟周期数} / \text{指令条数}$$

**问题：指令的CPI、机器的CPI、程序的CPI各能反映哪方面的性能？**

**单靠CPI不能反映CPU性能！为什么？**

**例如，单周期处理器CPI=1，但性能差！**

# Example1

程序P在机器A上运行需10秒，机器A的时钟频率为400MHz。  
现在要设计一台机器B，希望该程序在B上运行只需6秒。

机器B时钟频率的提高导致了其CPI的增加，使得程序P在机器B上时钟周期数是在机器A上的1.2倍。机器B的时钟频率达到A的多少倍才能使程序P在B上执行速度是A上的 $10/6=1.67$ 倍？

**Answer:**

$$\text{CPU时间A} = (\text{时钟周期数A}) / (\text{时钟频率A})$$

$$\text{时钟周期数A} = (10 \text{ sec}) \times (400\text{MHz}) = 4000\text{M个}$$

$$\begin{aligned}\text{时钟频率B} &= (\text{时钟周期数B}) / (\text{CPU时间B}) \\ &= (1.2 \times 4000\text{M}) / (6 \text{ sec}) = 800 \text{ MHz}\end{aligned}$$

机器B的频率是A的两倍，但机器B的速度只是A的1.67倍，  
并不是A的两倍！



# Marketing Metrics (产品宣称指标)

**MIPS** = Instruction Count / Second  $\times (1/10^6)$

注:  $10^6 = 1$ 百万

= Clock Rate / CPI  $\times 1/10^6$

**Million Instructions Per Second** (定点指令执行速度)

因为每条指令执行时间不同, 所以MIPS总是一个平均值。

- 不同机器的指令集不同
- 程序由不同的指令混合而成
- 指令使用的频度动态变化
- Peak MIPS: (不实用)

用MIPS数表示性能有没有局限?

所以MIPS数不能说明性能的好坏 (用下页中的例子来说明)

**MFLOPS** = FP Operations / Second  $\times 1/10^6$

**Million Floating-point Operations Per Second** (浮点操作速度)

- 不一定是程序中花时间的部分

用MFLOPS数表示性能也有一定局限!

问题: TFLOPS、PFLOPS等的含义是什么?

FLOPS: floating-point operations per second, 每秒浮点运算次数(亦称每秒峰值速度)。 **TFLOPS**= 每秒万亿 ( $=10^{12}$ ) 次的浮点运算

# Example: MIPS数不可靠!

(书中例1.3) Assume we build **an optimizing compiler** for the load/store machine. The compiler discards 50% of the ALU instructions.

- 1) What is the CPI ? **仅在软件上优化，没涉及到任何硬件措施。**
- 2) Assuming a 20 ns clock cycle time (50 MHz clock rate). What is the MIPS rating for optimized code versus unoptimized code? Does the MIPS rating agree with the rating of execution time?

Op	Freq	Cycle	Optimizing compiler	New Freq
ALU	43%	1	减少50% → $21.5 / (21.5 + 21 + 12 + 24) = 27\%$	27%
Load	21%	2	$21 / (21.5 + 21 + 12 + 24) = 27\%$	27%
Store	12%	2	$12 / (21.5 + 21 + 12 + 24) = 15\%$	15%
Branch	24%	2	$24 / (21.5 + 21 + 12 + 24) = 31\%$	31%

1.57是如何算出来的?

CPI 1.57

MIPS 31.8 (50M/1.57=31.8)

CPI 1.73

50M/1.73=28.9 MIPS

**结果：因为优化后减少了ALU指令（其他指令数没变），所以程序执行时间一定减少了，但优化后的MIPS数反而降低了。**

# 选择性能评价程序（Benchmarks）

## ◦ 用基准程序来评测计算机的性能

- 基准测试程序是专门用来进行性能评价的一组程序
- 不同用户使用的计算机用不同的基准程序
- 基准程序通过运行实际负载来反映计算机的性能
- 最好的基准程序是用户实际使用的程序或典型的简单程序

## ◦ 基准程序的缺陷

- 现象：基准程序的性能与某段短代码密切相关时，会被利用以得到不当的性能评测结果
- 手段：硬件系统设计人员或编译器开发者针对这些代码片段进行特殊的优化，使得执行这段代码的速度非常快
  - 例1：Intel Pentium处理器运行SPECint时用了公司内部使用的特殊编译器，使其性能极高
  - 例2：矩阵乘法程序SPECmatrix300有99%的时间运行在一行语句上，有些厂商用特殊编译器优化该语句，使性能达VAX11/780的729.8倍！

# Successful Benchmark: SPEC

- 1988年，5家公司（Sun, MIPS, HP, Apollo, DEC）联合提出了SPEC (Systems Performance Evaluation Committee)
- SPEC给出了一组标准的测试程序、标准输入和测试报告。它们是一些实际的程序，包括 OS calls、I/O等。
- 版本 89：10 programs = 4 for integer + 6 for FP, 用每个程序的执行时间求出一个综合性能指标
- 版本92：SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs)
  - 整数和浮点数单独提供衡量指标：SPECInt92和SPECfp92
  - 增加 SPECbase: 禁止使用任何与程序有关的编译优化开关
- 版本95：8 int + 10fp
- 较新版本：include SPEC HPC96, SPEC JVM98, SPEC WEB99, SPEC OMP2001. SPEC CPU2000, See <http://www.spec.org>
  - “benchmarks useful for 3 years”
  - Base machine is changed from VAX-11/780 to Sun SPARC 10/40

# 如何给出综合评价结果？

问题：如果用一组基准程序在不同机器上测出了运行时间，那么如何综合评价机器的性能呢？

先看一个例子：

**Program 1:** 1 sec on machine A, 10 sec on machine B

**Program 2:** 1000 sec on A, 100 sec on B

What are your conclusions?

- A is 10 times faster than B for program1.
- B is 10 times faster than A for Program2.

这个结论无法比较A和B的好坏  
必须用一个综合的值来表示！

**Total exec. time**是一个综合度量值，可以据此得出结论：

**B is  $1001/110=9.1$  times faster than A**

实际上，可考虑每个程序在作业中的使用频度，即加权平均

# 综合性能评价的方法

- 可用以下两种平均值来评价：
  - Arithmetic mean(算术平均): 求和后, 再除n
  - Geometric mean(几何平均): n个观察值连乘积的n次方根就是几何平均数
- 根据算术平均执行时间能得到总平均执行时间
- 根据几何平均执行时间不能得到程序总的执行时间
- 执行时间的规格化 (测试机器相对于参考机器的性能) :
  - $\text{time on reference machine} \div \text{time on measured machine}$
- 平均规格化执行时间不能用算术平均计算, 而应该用几何平均
  - program A going from 2s to 1s as important as program B going from 2000s to 1000s.

(算术平均值不能反映这一点!)

综上所述, 算术平均和几何平均各有长处, 可灵活使用!



# 第二讲小结

- 性能的定义：一般用程序的响应时间或系统的吞吐率表示机器或系统整体性能。
- CPU性能的测量（用户程序的CPU执行时间）：
  - 一般把程序的响应时间划分成CPU时间和等待时间，CPU时间又分成用户CPU时间和系统CPU时间。
  - 因为操作系统对自己所花费的时间进行测量时，不十分准确，所以，对CPU性能的测算一般通过测算用户CPU时间来进行。
- 各种性能指标之间的关系：
  - $\text{CPU执行时间} = \text{CPU时钟周期数} \times \text{时钟周期}$
  - 时钟周期和时钟频率互为倒数
  - $\text{CPU时钟周期数} = \text{程序指令数} \times \text{每条指令的平均时钟周期数CPI}$
- MIPS数在有些情况下不能说明问题，不具有可比性！
- 性能评价程序的选择：
  - 采用一组基准测试程序进行综合（算术（加权）平均/几何平均）评测。
  - 有些制造商会针对评测程序中频繁出现的语句采用专门的编译器，使评测程序的运行效率大幅提高。因此有时基准评测程序也不能说明问题。
- 对于某种特定的指令集体系结构，提高计算机性能的主要途径有：
  - 提高时钟频率（第七章 流水线）
  - 优化处理器中数据通路的结构以降低CPI（单周期/多周期/流水线）
  - 采用编译优化措施来减少指令条数或降低指令复杂度（第五章 指令系统）

# 作业

2.4

3

4

6

8

10