

计算机系统基础

张宇

手机：13971637644

邮箱：zhyu@hust.edu.cn

办公地点：东五楼206房间



群名称：2022计算机系统基础
群 号：522453980

第一章 计算机系统概述

主要内容

- 计算机基本工作原理
- 程序的开发与执行
- 计算机系统的层次结构
- 计算机系统性能评价

第一台通用电子计算机: ENIAC

第一台通用电子计算机的诞生

1946年, 第1台通用电子计算机 ENIAC 诞生。

ENIAC : Electronic Numerical Integrator And Computer
电子数字积分计算机

- 美国宾夕法尼亚大学研制: 莫克利 (Mauchly)、艾克特 (Eckert)
- 由电子真空管组成, 支持5000次加法/s, 50次乘法/s, 平方、立方、sin、cos等, 主要用于解决复杂的弹道计算问题, 使原来的执行时间由20分钟缩短到30秒, 速度提高40倍
- 用十进制表示信息并运算。采用手动编程, 即通过插拔电缆等方式实现

第一台通用电子计算机: ENIAC



占地面积170平方米

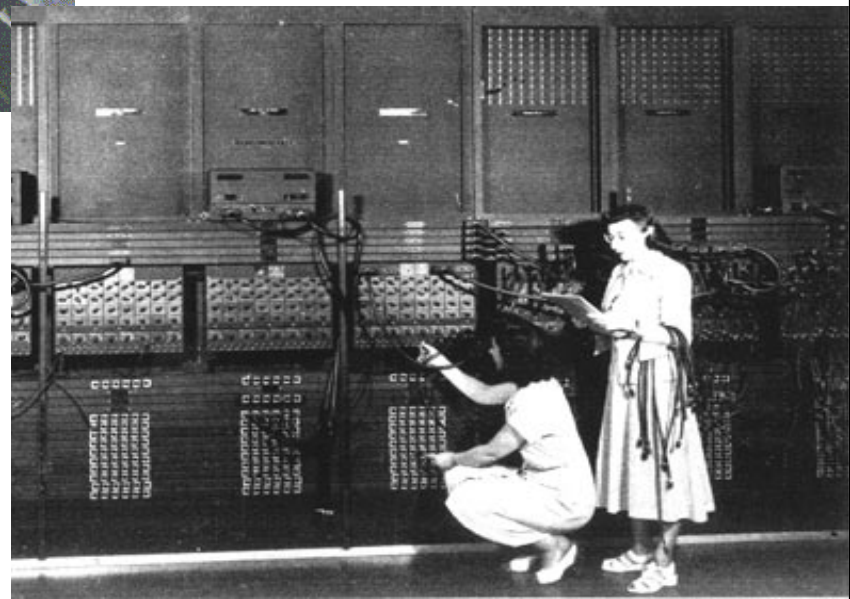
重30吨

有18000多个真空管

耗电160千瓦

该机正式运行到1955年10月2日,
这十年间共运行80223个小时

ENIAC并不具备现代电子计算机的主要特征，没有采用二进制表示和存储程序的工作方式
思考：ENIAC有什么缺点？



现代计算机的原型

- 冯·诺伊曼原籍匈牙利，数学家。
- 1944年，冯·诺依曼在参加原子弹的研制工作，涉及到极为困难的计算。
- 1945年，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**”，宣告现代计算机结构思想的诞生。
- 在这个报告中提出的计算机结构被称为**冯·诺依曼结构**
 - 采用**冯·诺依曼结构**的计算机也称为**冯·诺依曼机器** (Von Neumann Machine)。
 - 现代几乎所有的通用计算机都采用**冯·诺依曼结构**。



**Electronic
Discrete
Variable
Automatic
Computer**
离散变量自动电子计算机

现代计算机的原型

- 冯·诺依曼结构最重要的思想是 “存储程序 (Stored-program)” 工作方式

基本思想：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，计算机能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。

思考：相对于以前没有存储程序的工作方式而言，存储程序工作方式有什么优势？

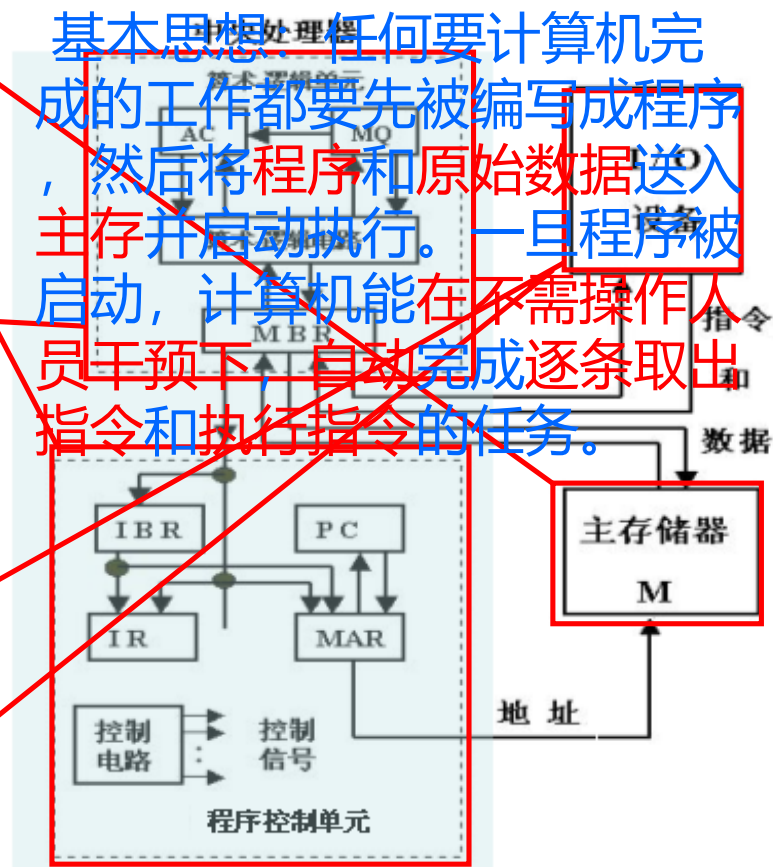
➤可以将用户从繁琐的操作中解脱出来（例如：以前基于卡片的方式只能顺序执行卡片上的操作，而对于跳转指令执行等，需要用户根据中间结果进行后续卡片的选择等。而存储程序的工作方式可以通过设置PC直接实现跳转）

➤减少人工干预，避免人工操作成为性能瓶颈

你认为冯·诺依曼结构是怎样的？

- 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
- 程序由一组指令构成，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

基本思想：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，计算机能自动完成逐条取出指令和执行指令的任务。



你还能想出更多吗？

你猜得八九不离十了☺

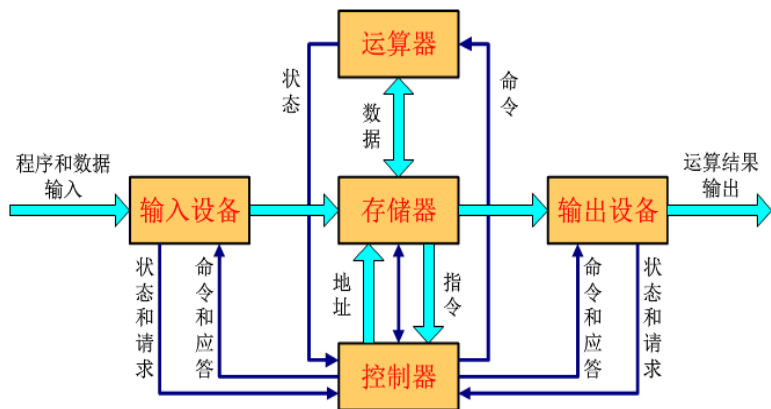
普林斯顿高等研究院“存储程序”计算机（称为IAS计算机）的计算机结构

冯.诺依曼结构计算机模型

1. 工作方式：采用“**存储程序**”式工作方式。
2. 冯.诺依曼结构计算机应由**存储器**、**控制器**、**运算器**、**输入设备**和**输出设备**五个基本部件组成。
3. 各基本部件的功能是：
 - **存储器**：存放数据和指令，两者形式上没有区别，但计算机应能区分数据还是指令；
 - **控制器**：自动取出指令来执行；
 - **运算器**：进行加减乘除基本算术运算及逻辑运算和其它附加运算；
 - 操作人员可以通过**输入设备**、**输出设备**使用计算机。

4. 内部以**二进制**形式表示指令和数据。

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**。程序由一串指令组成。



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

冯·诺依曼计算机是如何工作的？

● 程序在执行前

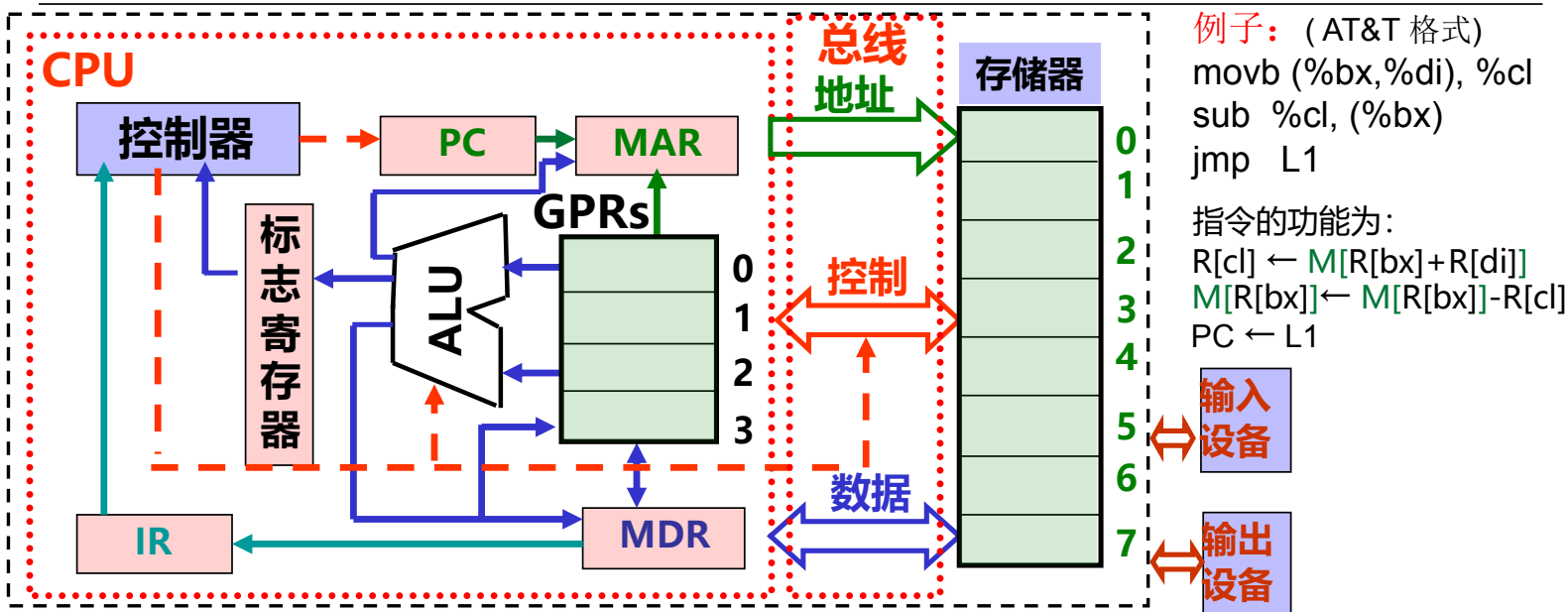
- 数据和指令事先**按序**存放在**存储器**中，其中**每条指令**和**每个数据**都有地址
- 用程序的**起始地址**初始化**程序计数器PC**（总存储下一条要执行的指令的地址）

注意：由于采用“ 存储程序 ”工作方式

在程序启动前： **指令**和**数据**都存放在**存储器**中，因此**形式上没有差别**，都是0/1序列

在程序被启动后： 在指令执行过程中，**指令**和**数据**被从存储器取到CPU，分别**存**放在CPU内的**对应寄存器**中（**指令**存储在**指令寄存器IR**中，**数据**存储在**通用寄存器GPRs**中），从而实现了**指令和数据**的区分。

冯·诺依曼计算机是如何工作的？



CPU：中央处理器；

PC：程序计数器（存储下一条要执行的指令的地址）；

MAR：存储器地址寄存器；MDR：存储器数据寄存器；

IR：指令寄存器；

GPRs（通用寄存器组）：由若干通用寄存器组成，用于数据暂存

算术逻辑部件（ALU）：用于数据运算；

数据通路（datapath）：指的是指令在执行过程中数据所经过的部件以及部件之间的连接线路

控制器：主要用于执行微程序，生成相应控制信号，以控制数据通路进行正确的操作

总线：用于在CPU、主存和输入输出设备之间传送数据

存储器：用于数据存储，其主要部件有存储阵列、地址译码器、读写控制电路

主要内容

- 计算机基本工作原理
- 程序的开发与执行
- 计算机系统的层次结构
- 计算机系统性能评价

1.2 程序的开发和执行过程

最早的程序开发过程：用机器语言编写程序，并记录在纸带或卡片上



穿孔表示0，未穿孔表示1

所有信息都是0/1序列！
输入：开关等；
输出：指示灯等

假设：用0010表示跳转指令jxx

0: 0101 0110

1: 0010 0100

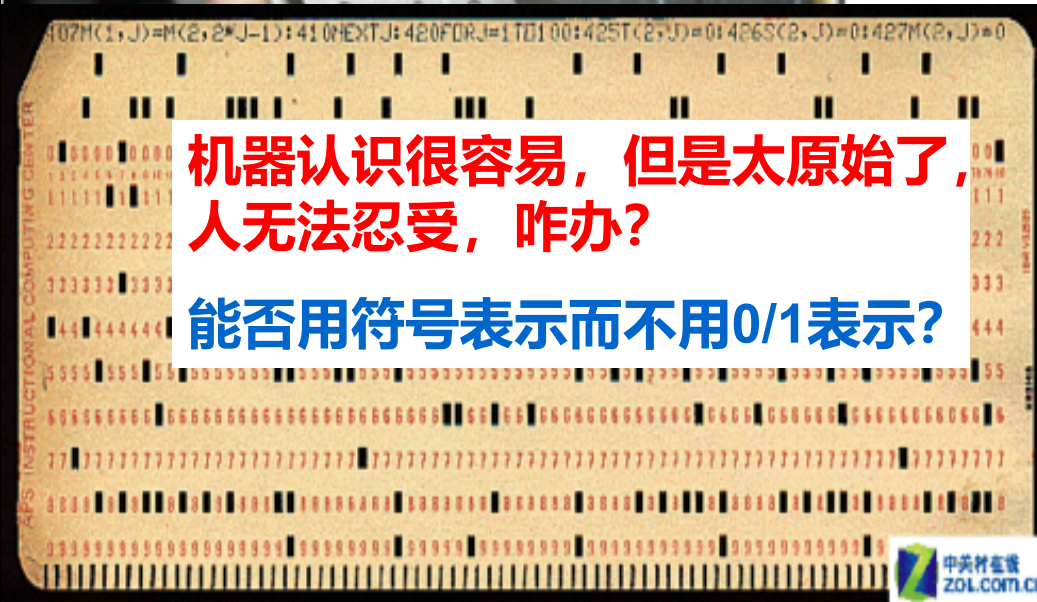
2:

3:

4: 0110 0111

5:

6:



机器认识很容易，但是太原始了，人无法忍受，咋办？

能否用符号表示而不用0/1表示？

若想在第4条指令前加入指令，会怎么样？

需重新计算地址码（如jxx的目标地址），然后重新打孔。

编程不灵活！书写、阅读困难！

用汇编语言开发程序

- 因此，后面设计师们期望使用**符号**表示跳转位置和变量位置等信息，简化机器使用
- 于是，出现汇编语言：它用**助记符**表示操作码、寄存器、位置等

0: 0101 0110

1: 0010 0100

2:

3:

4: 0110 0111

5:

6:

7:

add B

jxx L0

.....

.....

L0: sub C

.....

B:

C:

用汇编语言编写的优点是：

- 不会因为增减指令而需要修改其它指令
- 不需记忆指令码，编写方便，可读性比机器语言强（比机器语言更接近自然语言）

不过，这带来新的问题，是什么呢？

人容易了，可机器不认识这些指令了！

需将汇编语言转换为机器语言！

用汇编程序转换

在第4条指令前加指令时不用改变jxx指令中的地址码！

虽然，**汇编程序**可以将**汇编语言**转换为**机器语言**来解决问题，但是用汇编语言编程很麻烦

- **汇编指令与机器指令一一对应。**

- **机器指令**用**二进制**表示，**汇编指令**用**符号**表示。

- **汇编指令只能描述一些基本的操作**

- 取、存一个数
 - 两个数的算术（加、减、乘、除）或逻辑（与、或等）运算
 - 根据运算结果判断是否转移执行等

- **想象用**汇编语言**编写复杂程序是怎样的情形？**

- （例如，用汇编语言实现排序、矩阵相乘）

- 需要描述的细节太多了！程序会很长很长！
 - 而且在不同结构的机器上不一定都能运行！

机器语言和汇编语言都是面向机器结构的语言，故它们统称为**机器级语言**

结论：用汇编语言比机器语言好，但是，使用时还是很麻烦！

用高级语言开发程序

- 随着技术的发展，出现了许多高级编程语言
 - 它们与具体机器结构无关
 - 高级语言中一条语句对应几条、几十条甚至几百条机器指令
 - 有“面向过程”和“面向对象”的语言之分
- 现在，几乎所有程序员都用高级语言编程，但最终要将高级语言程序转换为机器语言程序，才能在底层机器上执行
 - 有两种转换方式：“编译”和“解释”
 - 编译程序(Compiler): 提前将高级语言源程序转换为机器级目标程序，执行时只要启动目标程序即可
 - 解释程序(Interpreter): 在执行时，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。

一个C语言程序开发、转换和执行过程

经典的“hello.c”C-源程序

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```

功能：输出 “hello,world”

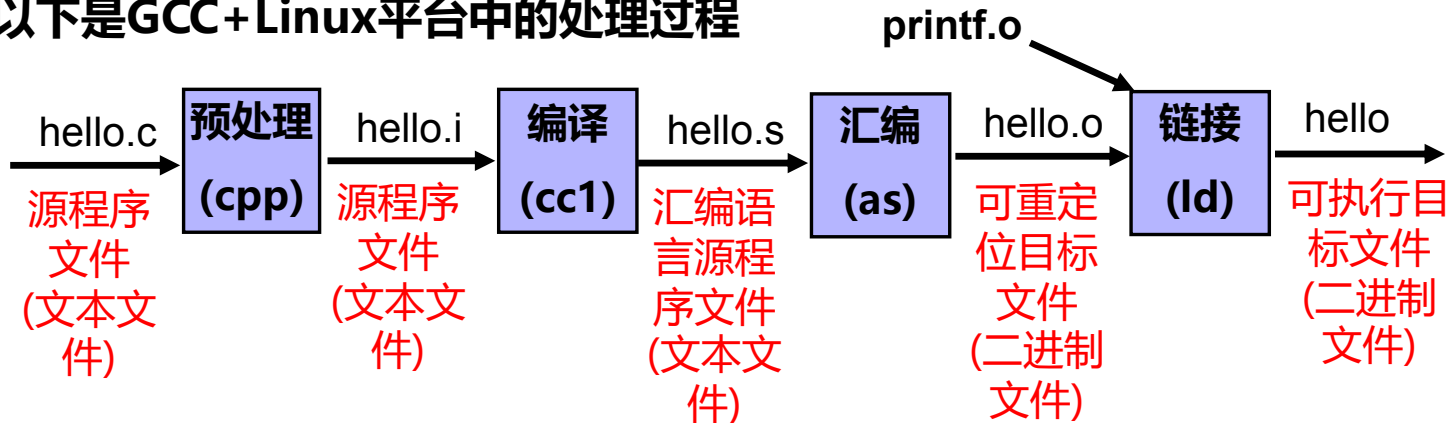
hello.c的ASCII文本表示

```
# 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
```

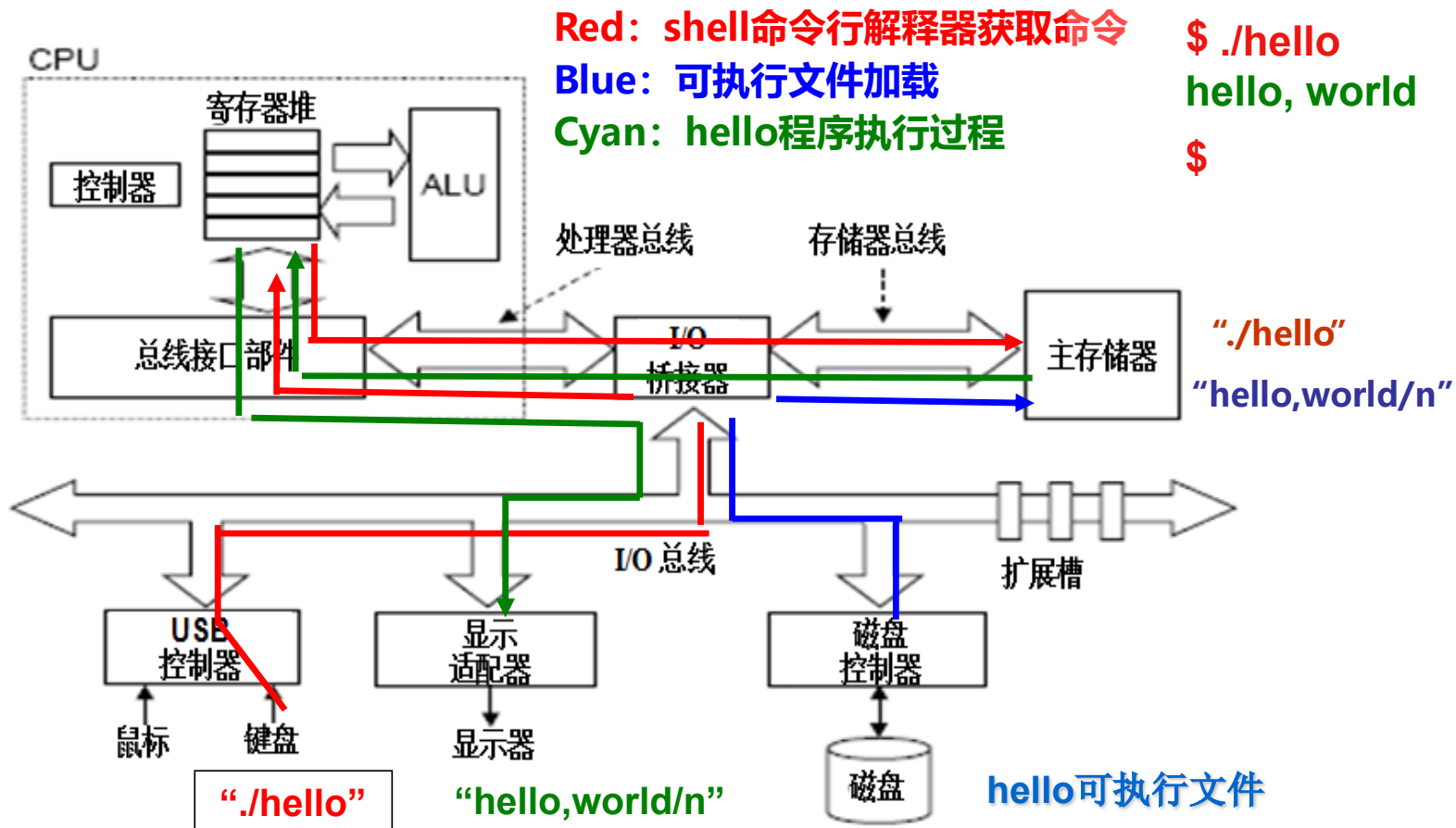
通过文本编辑器编辑生成源程序，并保存到hello.c文件中

计算机不能直接执行hello.c!

以下是GCC+Linux平台中的处理过程



Hello程序执行中的指令/数据流动过程



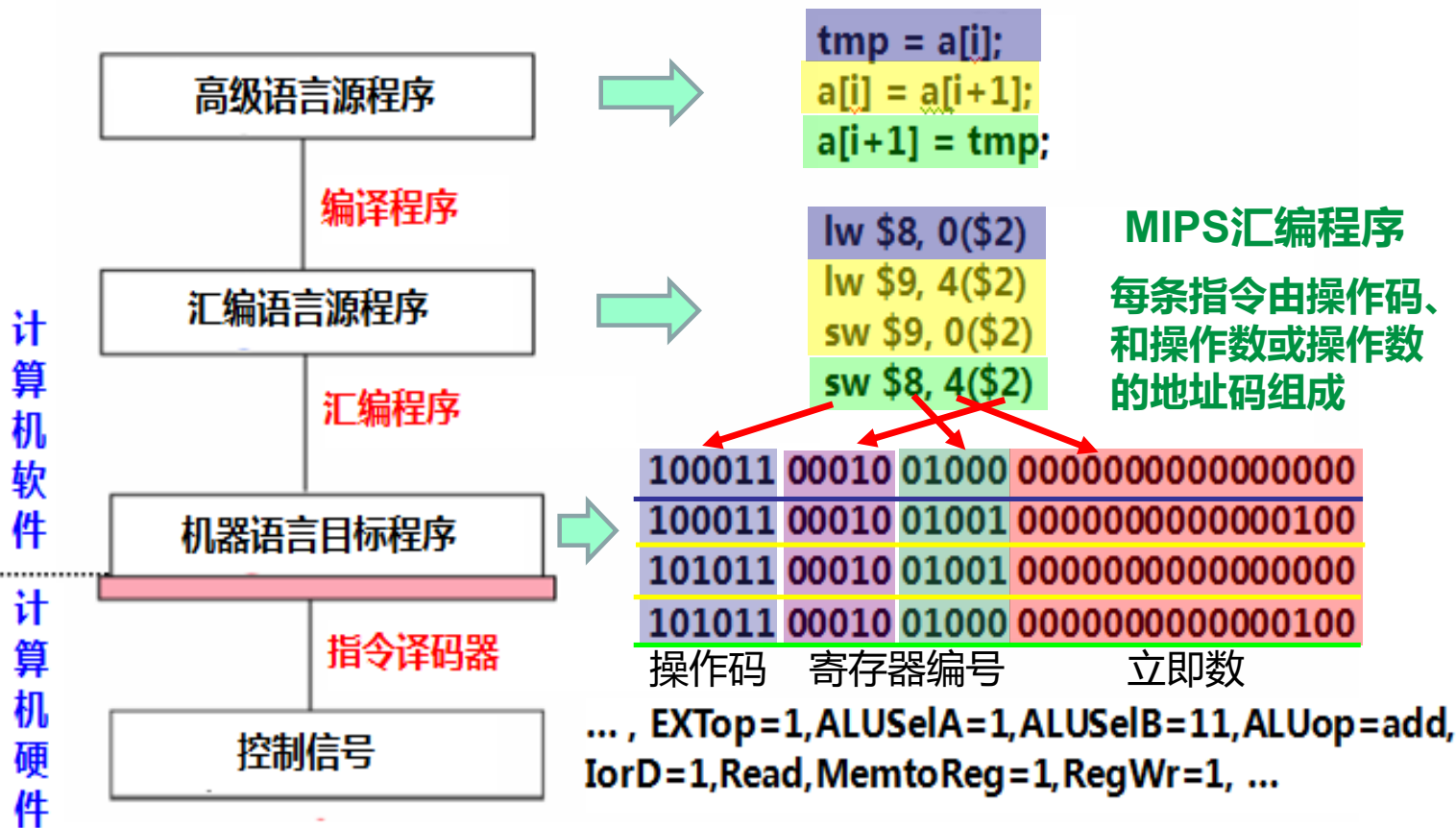
所有过程都是在执行指令时CPU产生的控制信号的作用下进行的

主要内容

- 计算机基本工作原理
- 程序的开发与执行
- 计算机系统的层次结构
- 计算机系统性能评价

不同层次语言之间的等价转换

为了支持高级语言源程序的执行，该高级语言源程序需要在不同层次语言之间进行等价转换，最终转换成控制信号



注释：MIPS是世界上一一种很流行的RISC处理器。MIPS采用RISC指令集

高级语言开发和运行程序需要什么支撑？

- 最早的程序开发很简单——机器语言
- 为了实现前面所述的高级语言程序转换成控制信号从而在计算机硬件上执行，需要复杂的支撑环境
 - 需要编辑器编写源程序
 - 需要一套翻译软件或解释软件处理源程序
 - 编译方式 或 解释方式
 - 需要一个可以执行程序的支撑环境
 - 人机接口（GUI、CUI）
 - 内核服务例程

语言
处理
系统

用于语言
编辑/编译
、链接等

操作系统

进行程序运行时的硬件
资源管理，作业调度、
和提供用户接口等

语言处理系统和操作系统构成支撑程序开发和运行的最重要的系统软件

语言处理系统

操作系统

指令集体系结构

计算机硬件

早期计算机系统的层次

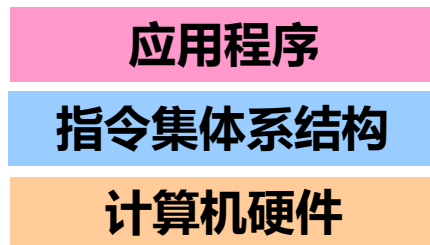
计算机是为了执行程序而设计的。

怎么编写和执行程序决定了计算机系统的基本结构

早期计算机系统层次结构

- 最早的计算机用机器语言编程

机器语言称为第一代程序设计语言 (First generation programming language , 1GL)



- 后来用汇编语言编程

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



现代计算机系统的层次

- 现代计算机用高级语言编程
 - 第三代程序设计语言 (3GL) 为**过程式语言**，编码时需要描述实现过程，即“如何做”
 - 第四代程序设计语言 (4GL) 为非过程化语言，编码时只需说明“做什么”



现代传统计算机
系统层次结构

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

可以看出：语言的发展是一个不断“**抽象**”的过程，以**简化**程序开发。

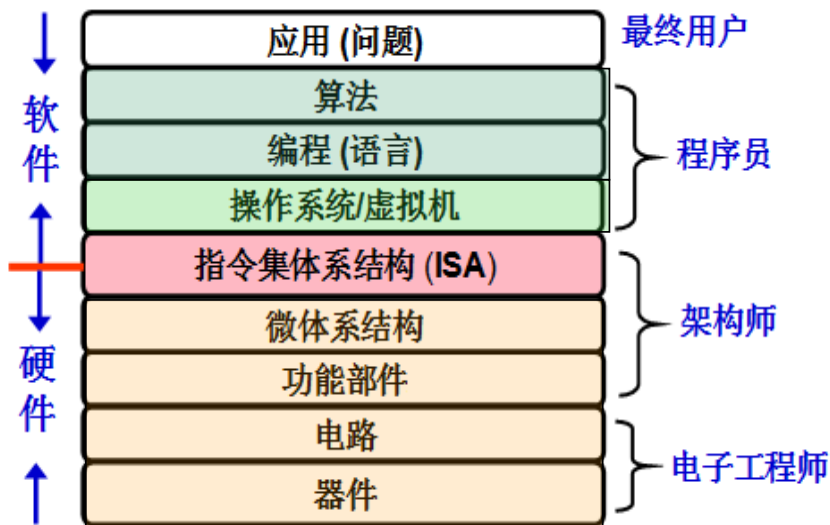
为此，**计算机系统**也不断有**新的层次**出现，**为简化程序开发和运行提供支持**，同时**高效**利用**底层计算机硬件**

思考：为什么现代计算机系统都采用层次结构？

➤方便设计，也可以方便分工合作。

指令集体系结构 (ISA)

- ISA (Instruction Set Architecture) 是计算机系统中必不可少的一个抽象层，位于**计算机系统中软件和硬件的交界面**。它是**对硬件的抽象**，将物理上的计算机硬件抽象成一个逻辑上的虚拟计算机（称为**机器语言级虚拟机**），**来对软件屏蔽底层硬件实现细节**。它是计算机硬件系统底层能够**为软件所感知的部分**
- 硬件的功能通过ISA提供出来，软件通过ISA规定的方式使用硬件



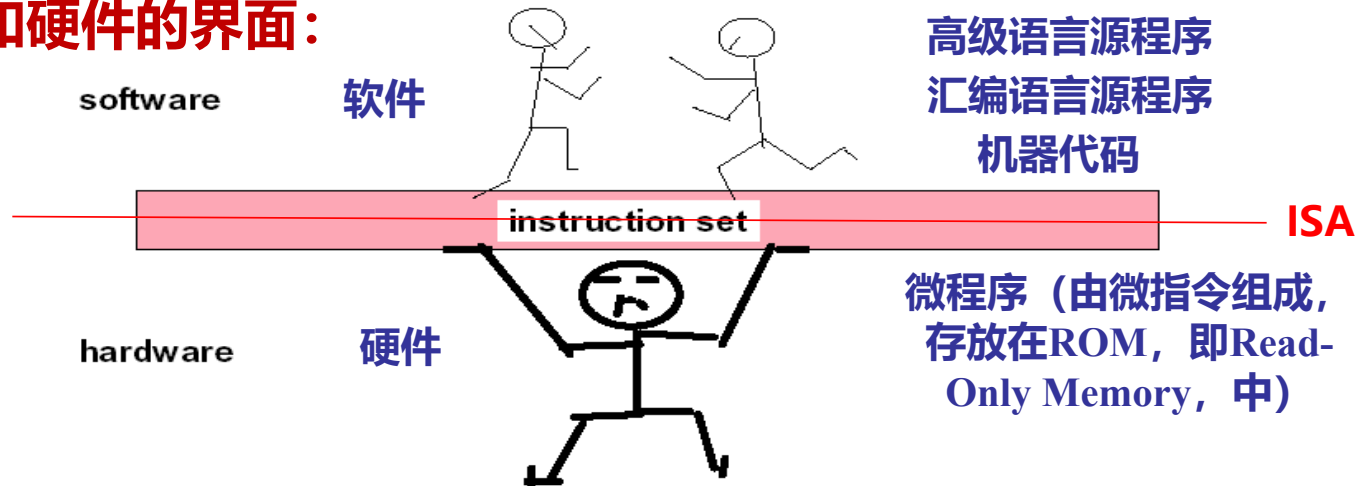
ISA可以看做是计算机提供的所有功能函数的“函数声明”；而微程序就是各“功能函数”的实现

指令集体系结构 (ISA)

- ISA是一种规范 (Specification) , 它规定了机器可以执行的所有指令的集合以及各机器指令格式和行为
 - 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中各寄存器的名称、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。

为什么ISA是计算机系统中最重要的抽象层

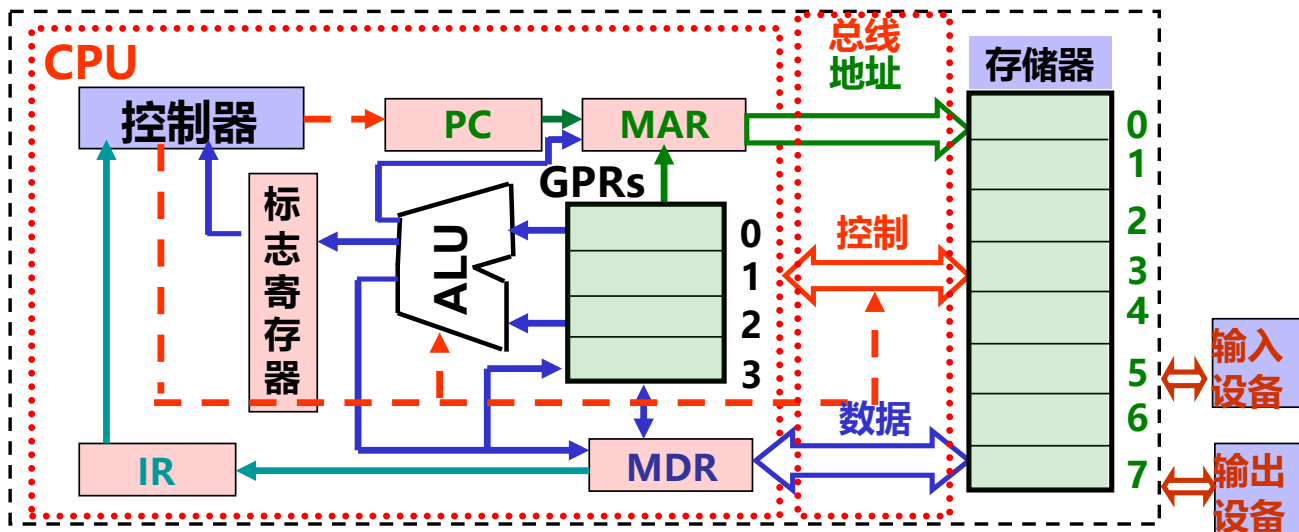
ISA (Instruction Set Architecture) 指令集体系结构：软件和硬件的界面：



- 因为，没有ISA，软件无法使用计算机硬件！（语言处理系统和操作系统都可以没有，ISA不能没有）开放性思考：实现ISA规范的微程序能不放在硬件层吗？为什么不交给上层程序员去开发？
- ISA规定了机器可以执行的所有指令的集合（最基本算子的集合），也决定了计算机能支持的所有功能的集合。在设计计算机时，如果ISA定义的不好，将导致一台计算机不能称为“通用计算机” 注：目前已有多种ISA，它们规定的指令集不同，如，IA-32 (CISC: Complex Instruction Set Computer)、ARM (RISC: Reduced Instruction Set Computer)

ISA和微结构、器件之间的关系

- **ISA**是计算机提供的所有功能函数的“函数声明”。在实现时，需要设计微体系结构（器件组织方式，比如采用2层cache还是3层cache），使用器件（如提供GPR、标志、运算电路等）和微程序实现“这些功能函数”
- 同一种ISA可以有不同的微体系结构和器件实现，如乘法指令可用不同的乘法器实现，甚至器件可以采用光器件等



- 可以看到，ISA解耦合了软件开发和底层硬件设计。例如：底层CPU不论是集成电路芯片（或者采用不同的微体系结构等）还是光芯片还是其它芯片，只要CPU实现时达到了ISA规范，然后程序员只要遵循ISA规范进行编程，都能运行在该CPU上。这很好地促进了硬件和软件开发商的分工和独自发展

主要内容

- 计算机基本工作原理
- 程序的开发与执行
- 计算机系统的层次结构
- 计算机系统性能评价（自学）

1.4 计算机系统性能评价

• 如何评价计算机的性能？

– 通常，对于不同的计算机，完成同样的工作量所需时间最短的那台计算机性能最好

– 在一般情况下，程序的真正执行时间 \neq 结束时间 - 开始时间。

例如：用户交互时间不能算进程序真正执行时间。

例如：现在系统上，通常有多个程序“分时”地轮流使用处理器。也就是说，在一个用户程序执行过程中，可能同时还会有其它用户程序和操作系统在执行。

– 也就是说，一个程序从开始到结束所需的时间，除了程序包含的指令在CPU上执行所用的时间外，还包括磁盘访问时间、输入输出操作所需时间以及操

• 因此，通常将用户能感觉到的用户程序执行时间分为两部分：

– CPU时间：指CPU用于本程序执行的时间，包含以下两部分：

用户CPU时间，指真正用于运行用户程序代码的时间

系统CPU时间，指为了执行用户程序而需要CPU运行操作系统程序的时间

– 其他时间：指等待I/O操作完成的时间或CPU用于执行其他用户程序的时间

不考虑应用背景，计算机系统的性能评价主要考虑的是：用户CPU时间（即CPU性能）

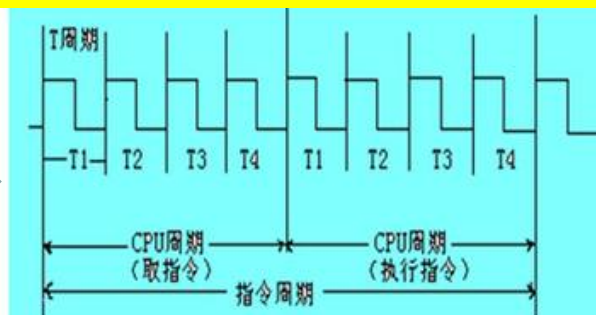
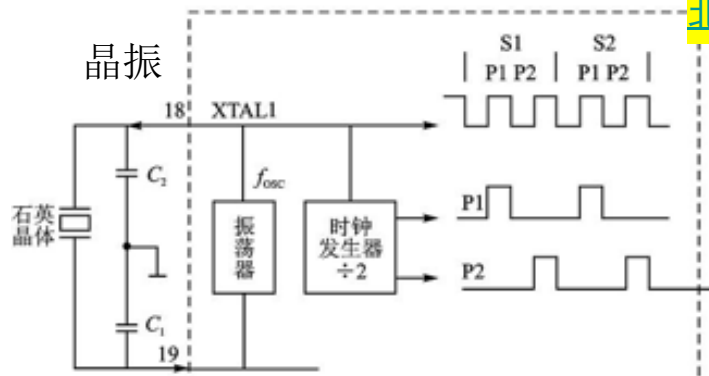
在计算用户CPU时间时，所需基本概念

● 时钟周期 (clock cycle, tick, clock tick, clock)

- 计算机执行一条指令的过程被分为若干步骤和相应的动作来完成，如取指令、取操作数、执行。
- 每一步动作都要有相应的控制信号进行控制，这些控制信号何时发出、作用时间多长，都要有相应的定时信号进行同步。
- 因此CPU必须能够产生**同步的时钟定时信号**
——这就是CPU的主脉冲信号，其宽度称为**时钟周期**。

脉冲信号是怎么产生的呢？

晶振是基准，是计算机内部CPU、存储器等部件之间传输和操作数据的“基准口令”。就如同生活中要用[北京时间](#)，这样全中国的人在生活中才不会出问题。



在计算用户CPU时间时，所需基本概念

- 时钟频率

- CPU的**主频**（CPU主脉冲信号的时钟频率）

= CPU时钟周期的倒数

一般来说，CPU主频越高，计算机速度越快，但CPU主频和计算机速度并不等价

- 因为一条**机器指令**往往包含多步，**需要多个时钟周期**才能执行完

- **不同计算机的指令集**所包含的**指令**在执行时所需**平均时钟周期数**可能不同

- CPI（Cycles Per Instruction）

- 执行一条指令所需的**时钟周期数**

- 不同指令由于功能不同，它的CPI也不同

- CPI是在**ISA设计时制定**的，对于各指令是一个确定值

用户CPU时间的计算

● 用户CPU时间与时钟周期、时钟频率、CPI之间的关系

用户CPU时间 = 程序总时钟周期数 \times 时钟周期

= 程序总时钟周期数 \div 时钟频率

= 程序总指令条数 \times 程序综合CPI \times 时钟周期

程序综合CPI：指该程序中所有指令执行所需的平均时钟周期数

假定 CPI_i 和 C_i 分别为第 i 类指令的CPI和指令条数，则程序总时钟周期数为：

程序总时钟周期数 = $\sum_{i=1}^n CPI_i \times C_i$ 所以，CPU时间 = 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，则程序综合CPI为：

$$CPI_{\text{综合}} = \sum_{i=1}^n CPI_i \times F_i \quad \text{其中} \quad F_i = \frac{C_i}{\text{程序总指令条数}}$$

评价计算机性能时，不能仅考虑单个因素，而必须应该同时考虑时钟周期、指令条数和CPI这三个因素

例子：假设计算机M的指令集中包含A、B、C三类指令，其CPI分别为1、2、4。某个程序P在M上被编译成两个不同的目标代码P1和P2，P1含A、B、C三类指令的条数分别为8、2、2，P2含A、B、C指令的条数分别为2、5、3，问哪个代码总指令数少？哪个执行的速度快？它们的CPI分别是多少？

Answer:

P1和P2的总指令数分别为 $8+2+2=12$ 和 $2+5+3=10$ ，所以P2的总指令数少。

P1的总时钟周期数： $8 \times 1 + 2 \times 2 + 2 \times 4 = 20$

P2的总时钟周期数： $2 \times 1 + 5 \times 2 + 3 \times 4 = 24$

因为两个指令序列在同一台机器上运行，所以时钟周期一样，故P1比P2快

P1的CPI= $20/12=1.67$ ，P2的CPI= $24/10=2.4$

可以看到，P2的总指令条数少，但是P1更快

时钟周期、指令条数和CPI的相互制约关系

● 时钟周期、指令条数和CPI与哪些方面有关？

	指令条数	CPI	时钟周期
Algorithm	X	X	
Compiler	X	X	
ISA	X	X	
Organization		X	X
Technology			X

● 时钟周期、指令条数、CPI的相互制约关系

- ✓ 时钟周期短（主频高）→运算速度快？
- ✓ CPI小（指令周期短）→程序执行得快？
- ✓ 程序指令条数少→程序执行得快？

例如：更改指令集可以减少程序总指令条数，但是，同时可能引起CPU结构的调整，从而可能会增加时钟周期的宽度（即降低时钟频率）

时钟周期、指令条数和CPI的相互制约关系

例如：程序P在机器A上运行需10 s， 机器A的时钟频率为2GHz。现在要设计一台机器B，希望该程序在B上运行只需6 s。

但是，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1.5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: 因为， $\text{CPU时间}_A = \text{时钟周期数}_A \times \text{时钟周期}_A$
 $= \text{时钟周期数}_A / \text{时钟频率}_A$

所以， $\text{时钟周期数}_A = \text{CPU时间}_A \times \text{时钟频率}_A = 10 \text{ sec} \times 2\text{GHz} = 20\text{G个}$

同理， $\text{时钟频率}_B = \text{时钟周期数}_B / \text{CPU时间}_B$
 $= 20\text{G} \times 1.5 / 6 \text{ sec} = 5\text{GHz}$

这意味着，机器B的频率是A的2.5倍，但机器B的速度并不是A的2.5倍！

用指令执行速度进行性能评估

- 其实，最早用来衡量计算机性能的指标是**每秒完成单个运算指令的条数**

单位：MIPS（ Million Instructions Per Second）

即，程序指令条数 / （CPU执行时间 $\times 10^6$ ），每秒执行多少百万条指令

统计对象：单字长定点指令，如加法运算

缺陷：用MIPS来对不同的机器进行性能比较，有时不准确或不客观。

- 不同的机器的指令集不同
- 不同的机器上，同样的**指令条数**所完成的功能也可能完全不同；

MIPS不可靠的例子

例子：假定某程序P编译后生成的目标代码有A、B、C、D四类指令组成，它们在程序中所占的比例分别是43%、21%、12%、24%，已知它们的CPI分别为1、2、2、2。现在对P进行编译优化，生成的新目标代码中A类指令条数减少了50%，其他指令的条数不变。问：1) 编译优化前后程序的CPI各是多少？

2) 假定程序在一台主频为50MHz的计算机上运行，则优化前后的MIPS各是多少？

优化前

Op	Freq	CPI
----	------	-----

A	43%	1
---	-----	---

B	21%	2
---	-----	---

C	12%	2
---	-----	---

D	24%	2
---	-----	---

$$21.5 / (21.5 + 21 + 12 + 24) = 27\%$$

$$21 / (21.5 + 21 + 12 + 24) = 27\%$$

$$12 / (21.5 + 21 + 12 + 24) = 15\%$$

$$24 / (21.5 + 21 + 12 + 24) = 31\%$$

优化后

Op	NewFreq	CPI
----	---------	-----

A	27%	1
---	-----	---

B	27%	2
---	-----	---

C	15%	2
---	-----	---

D	31%	2
---	-----	---

$$\text{优化前程序CPI} = 43\% \times 1 + 21\% \times 2 + 12\% \times 2 + 24\% \times 2 = 1.57$$

$$50\text{M} / 1.57 = 31.8\text{MIPS}$$

$$\text{同理，优化后程序CPI} = 1.73$$

$$50\text{M} / 1.73 = 28.9\text{MIPS}$$

可以看到，因为优化后减少了A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。

用指令执行速度进行性能评估

- 浮点操作速度的指标：

单位： MFLOPS (Million Floating-point operations Per Second)

每秒执行多少百万次浮点运算。——按浮点运算的次数来衡量。

其它单位： GFLOPS (10^9) 、 TFLOPS (10^{12}) 、 PFLOPS (10^{15}) 、 EFLOPS (10^{18}) 等

注：这个指标目前还在用，特别是GPU性能（因为使用GPU的应用常需要浮点运算，所以GPU常以自己浮点运算能力多强做卖点）

选择性能评价程序（Benchmarks）

- 用基准程序来评测计算机的性能（例如：全球超级计算机TOP500排名）
 - 基准测试程序是专门用来进行性能评价的一组程序
 - 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
- 基准程序的缺陷是什么？
 - 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，在运行SPECint时，用了公司内部使用的特殊编译器，使其性能表现得很高

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，使性能提升729.8倍！

本章小结

- 存储程序思想：

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，计算机能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

- 冯.诺依曼结构：

冯.诺依曼结构计算机由存储器、控制器、运算器、输入设备和输出设备五个基本部件组成

- 冯.诺依曼机运作机制：

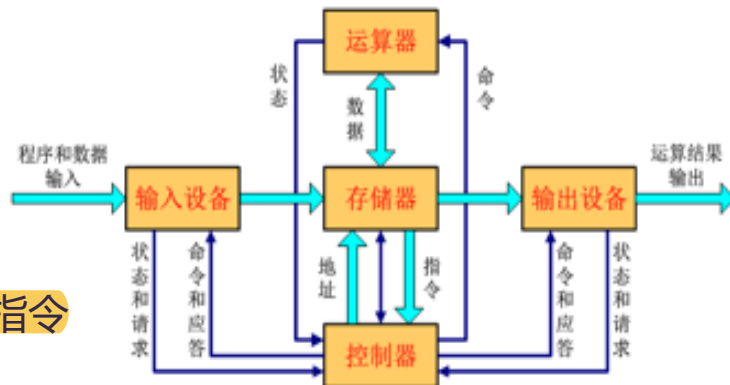
第一步：取指令（根据PC）

第二步：指令译码（确定操作类型）

第三步：PC增量，使其指向下一条要执行的指令

第四步：取操作数并执行

第五步：送结果，结果写入寄存器或内存单元



本章小结

- 现代传统计算机系统都采用层次结构：



其中，ISA（Instruction Set Architecture）指令集体系结构是软件和硬件的界面，是计算机系统中最重要的一层

课后作业

- 课后习题（P27）：

1. 名词解释：**ALU**、数据通路、程序计数器、指令寄存器、控制器、机器指令、**ISA**

2、（1）（2）（3）

5