



南京大學  
NANJING UNIVERSITY



# C语言程序举例

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 用“系统思维”分析问题

---

ISO C90标准下，在32位系统上

以下C表达式的结果是什么？

`-2147483648 < 2147483647`

`false`（与事实不符）！Why？

以下关系表达式结果呢？

`int i = -2147483648;`

`i < 2147483647`

`true`！Why？

`-2147483647-1 < 2147483647`，结果怎样？

理解该问题需要知道：

编译器如何处理字面量

高级语言中运算规则

高级语言与指令之间的对应

机器指令的执行过程

机器级数据的表示和运算

.....

# 用“系统思维”分析问题

```
sum(int a[ ], unsigned len)
{
    int i, sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

当参数len为0时，返回值应该是0，但是在机器上执行时，却发生访存异常。但当len为int型时则正常。Why?

访问违例地址为何是0xC0000005?

理解该问题需要知道：

高级语言中运算规则

机器指令的含义和执行

计算机内部的运算电路

异常的检测和处理

虚拟地址空间

.....

Exception in Test.exe: 0xC0000005: Access Violation.

确定

# 用“系统思维”分析问题

---

若x和y为int型，当 $x=65535$ 时， $y=x*x$ ；y的值为多少？

$y=-131071$ 。Why？

现实世界中， $x^2 \geq 0$ ，但在计算机世界并不一定成立。

对于任何int型变量x和y， $(x > y) == (-x < -y)$  总成立吗？

当 $x=-2147483648$ ，y任意（除-2147483648外）时不成立

Why？

在现实世界中成立，

但在计算机世界中并不一定成立。

理解该问题需要知道：  
机器级数据的表示  
机器指令的执行  
计算机内部的运算电路

# 用“系统思维”分析问题

main.c

```
int d=100;
int x=200;
int main()
{
    p1();
    printf ( "d=%d, x=%d\n" , d, x );
    return 0;
}
```

p1.c

```
double d;

void p1( )
{
    d=1.0;
}
```

**打印结果是什么？**

**d=0 , x=1 072 693 248**

**Why ?**

理解该问题需要知道：

机器级数据的表示

变量的存储空间分配

数据的大端/小端存储方式

链接器的符号解析规则

.....

# 用“系统思维”分析问题

/\* 复制数组到堆中，count为数组元素个数 \*/

```
int copy_array(int *array, int count) {
```

```
    int i;
```

```
    /* 在堆区申请一块内存 */
```

```
    int *myarray = (int *) malloc(count*sizeof(int));
```

```
    if (myarray == NULL)
```

```
        return -1;
```

```
    for (i = 0; i < count; i++)
```

```
        myarray[i] = array[i];
```

```
    return count;
```

```
}
```

当 $\text{count}=2^{30}+1$ 时，  
程序会发生什么情况？

理解该问题需要知道：

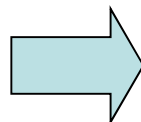
乘法运算及溢出

虚拟地址空间

存储空间映射

.....

当参数count很大时，则  
 $\text{count} \times \text{sizeof}(\text{int})$ 会溢出。  
如 $\text{count}=2^{30}+1$ 时，  
 $\text{count} \times \text{sizeof}(\text{int})=4$ 。



堆 ( heap ) 中大量  
数据被破坏！

# 用“系统思维”分析问题

代码段一：

```
int a = 0x80000000;
```

```
int b = a / -1;
```

```
printf("%d\n", b);
```

运行结果为-2147483648

代码段二：

```
int a = 0x80000000;
```

```
int b = -1;
```

```
int c = a / b;
```

```
printf("%d\n", c);
```

运行结果为“Floating point exception”，显然CPU检测到了溢出异常

为什么两者结果不同！

objdump  
反汇编代码，  
得知除以 -1  
被优化成取  
负指令neg，  
故未发生除  
法溢出

a/b用除法指令IDIV实现，但它不生成OF  
标志，那么如何判断溢出异常的呢？

实际上是“除法错”异常#DE（类型0）  
Linux中，对#DE类型发SIGFPE信号

理解该问题需要知道：

编译器如何优化

机器级数据的表示

机器指令的含义和执行

计算机内部的运算电路

除法错异常的处理

.....

# 用“系统思维”分析问题

---

以下是一段C语言代码：

```
#include <stdio.h>
main()
{
    double a = 10;
    printf("a = %d\n", a);
}
```

理解该问题需要知道：

IEEE 754 的表示

X87 FPU的体系结构

IA-32和x86-64中过程

调用的参数传递

计算机内部的运算电路

.....

在IA-32上运行时，打印结果为a=0

在x86-64上运行时，打印出来的a是一个不确定值

为什么？



# 用“系统思维”分析问题

```
double fun(int i)
{
    volatile double d[1] = {3.14};
    volatile long int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}
```

对于上述C语言函数， $i=0\sim 4$ 时， $\text{fun}(i)$ 分别返回什么值？

$\text{fun}(0) \rightarrow 3.14$   
 $\text{fun}(1) \rightarrow 3.14$   
 $\text{fun}(2) \rightarrow 3.1399998664856$   
 $\text{fun}(3) \rightarrow 2.00000061035156$   
 $\text{fun}(4) \rightarrow 3.14$ , 然后存储保护错

Why ?

理解该问题需要知道：

机器级数据的表示

过程调用机制

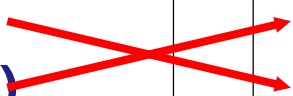
栈帧中数据的布局

.....

# 用“系统思维”分析问题

```
void copyij (int src[2048][2048],
             int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

```
void copyji (int src[2048][2048],
             int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```



以上两个程序功能完全一样，算法完全一样，因此，时间和空间复杂度完全一样，执行时间一样吗？

21 times slower  
(Pentium 4)  
**Why ?**

理解该问题需要知道：

数组的存放方式

Cache机制

访问局部性

.....

# 用“系统思维”分析问题

使用老版本gcc -O2编译时，程序一输出0，程序二输出却是1

Why ?

程序一：

```
#include <stdio.h>
double f(int x) {
    return 1.0 / x ;
}
void main() {
    double a, b;
    int i;
    a = f(10) ;
    b = f(10) ;
    i = a == b ;
    printf( "%d\n" , i ) ;
}
```

程序二：

```
#include <stdio.h>
double f(int x) {
    return 1.0 / x ;
}
void main() {
    double a, b, c;
    int i;
    a = f(10) ;
    b = f(10) ;
    c = f(10) ;
    i = a == b ;
    printf( "%d\n" , i ) ;
}
```

# 用“系统思维”分析问题

C/C++ code

```
1  #include "stdafx.h"
2  int main(int argc, char* argv[])
3  {
4      int a=10;
5      double *p=(double*)&a;
6      printf("%f\n", *p);           //结果为0.000000
7      printf("%f\n", (double)a);   //结果为10.000000
8
9      return 0;
10 }
11 为什么printf("%f", *p)和printf("%f", (double)a)结果不一样呢?
```

理解该问题需要知道：

**数据的表示**

**编译（程序的转换）**

**局部变量在栈中的位置**

.....

关键差别在于一条指令：

**fldl 和 fildl**

不都是强制类型转换吗？怎么会不一样

# 你在想什么？

---

- 看了前面的举例，你的感觉是什么呢？
  - 计算机好像不可靠                      从机器角度来说，它永远是对的！
  - 程序执行结果不仅依赖于高级语言语法和语义，还与其他好多方面有关
    - 一点不错！理解程序的执行结果要从系统层面考虑！
  - 本来以为学学编程和计算机基本原理就能当程序员，没想到还挺复杂的，计算机专业不好学
    - 学完“计算机系统基础”就会对计算机系统有清晰的认识，以后再学其他相关课程就容易多了。
  - 感觉要把很多概念和知识联系起来才能理解程序的执行结果
    - 你说对了！把许多概念和知识联系起来就是李国杰院士所提出的“系统思维”。      即：站在“计算机系统”的角度考虑问题！

# 系统能力基于“系统思维”

---

- 系统思维

- 从计算机系统角度出发分析问题和解决问题
- 首先取决于对计算机系统有多了解，“知其然并知其所以然”
  - 高级语言语句都要转换为机器指令才能在计算机上执行
  - 机器指令是一串0/1序列，能被机器直接理解并执行
  - 计算机系统是模运算系统，字长有限，高位被丢弃
  - 运算器不知道参加运算的是带符号数还是无符号数
  - 在计算机世界， $x*x$ 可能小于0， $(x+y)+z$ 不一定等于 $x+(y+z)$
  - 访问内存需几十到几百个时钟，而访问磁盘要几百万个时钟
  - 进程具有独立的逻辑控制流和独立的地址空间
  - 过程调用使用栈存放参数和局部变量等，递归过程有大量额外指令，增加时间开销，并可能发生栈溢出
  - .....

只有先理解系统，才能改革系统，并应用好系统!

# 什么是计算机系统？

## 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



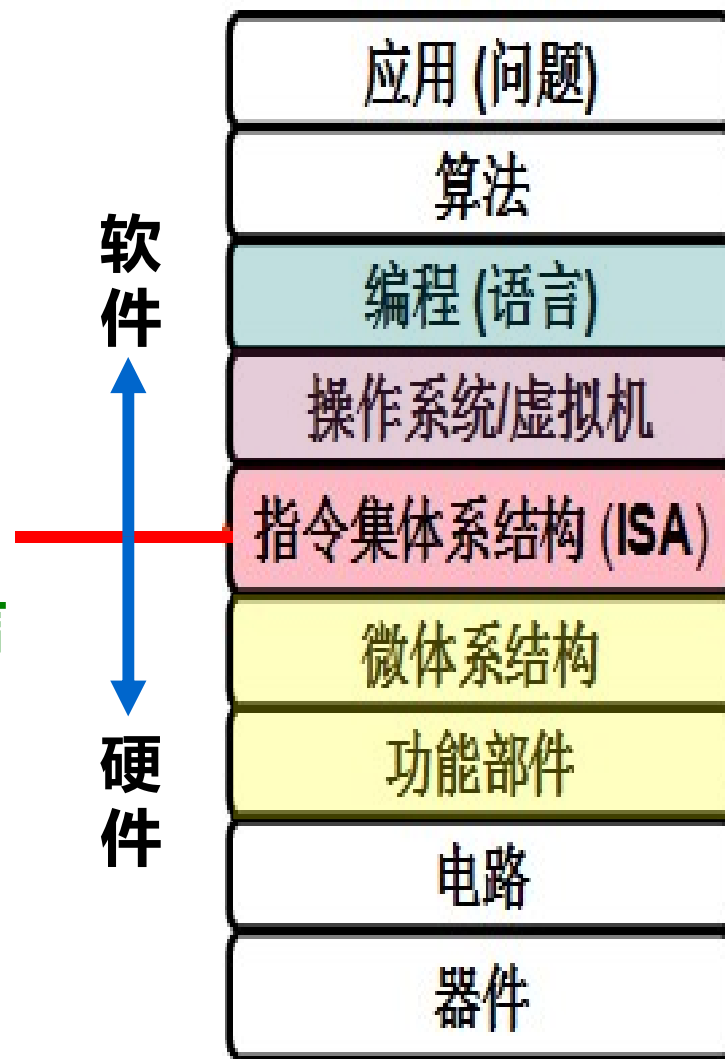
# “计算机系统基础” 内容提要

**课程目标：**使学生清楚理解计算机是如何生成和运行可执行文件的！

**重点在高级语言以下各抽象层**

- **C语言程序设计层**
  - 数据的机器级表示、运算
  - 语句和过程调用的机器级表示
- **操作系统、编译和链接的部分内容**
- **指令集体系结构 (ISA) 和汇编层**
  - 指令系统、机器代码、汇编语言
- **微体系结构及硬件层**
  - CPU的通用结构
  - 层次结构存储系统

## 计算机系统抽象层





# 为什么要学习“计算机系统基础”？

---

- 为什么要学习“计算机系统基础”呢？
  - 为了编程序时少出错
  - 为了在程序出错时很快找到出错的地方
  - 为了明白程序是怎样在计算机上执行的
  - 为了强化“系统思维”
  - 为了更好地理解计算机系统，从而编写出更好的程序
  - 为后续课程的学习打下良好基础
  - 为了编写出更快的程序
  - 为了更好地认识计算机系统
  - .....

**本课程属于计算机系统基础（一）**

**下面就开始本课程的学习吧！**



南京大学  
NANJING UNIVERSITY



# 计算机系统基本组成与基本功能

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 第一台通用电子计算机的诞生

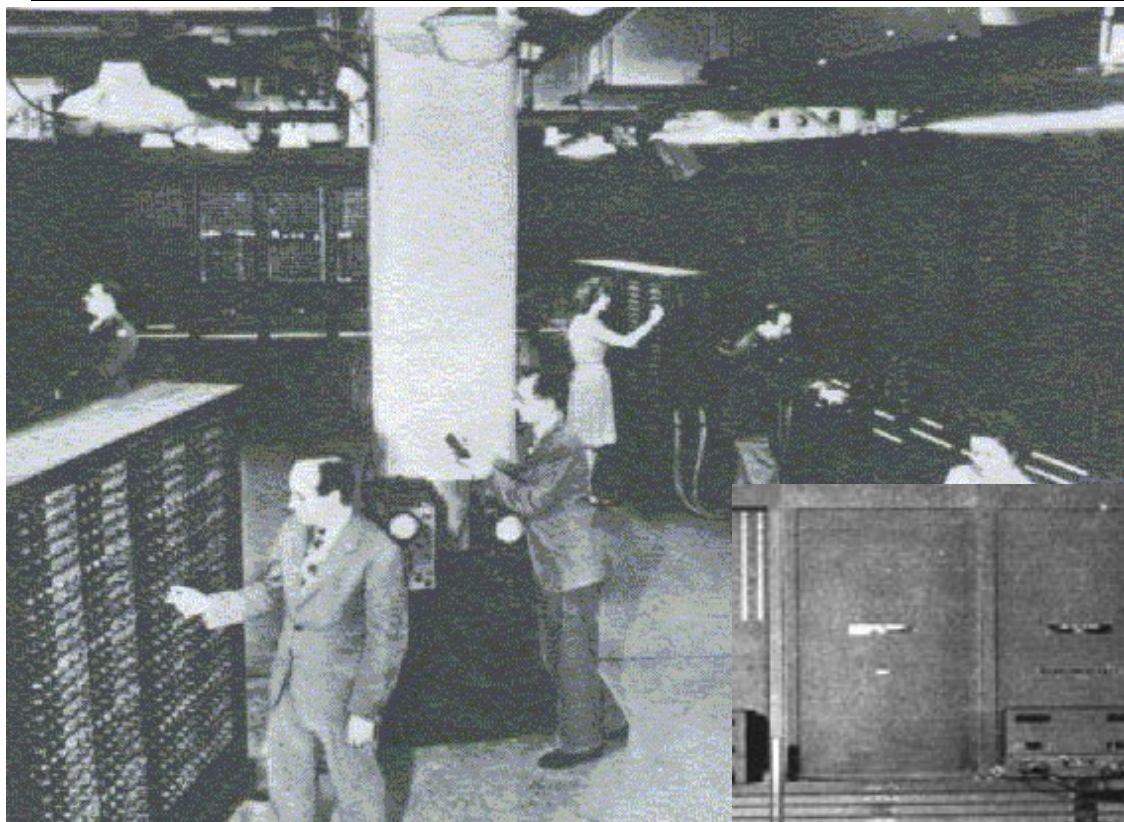
---

1946年，第1台**通用**电子计算机 **ENIAC**诞生

- 由电子真空管组成
- 美国宾夕法尼亚大学研制
- 用于解决复杂弹道计算问题
- 5000次加法/s
- 平方、立方、sin、cos等
- 用**十进制**表示信息并运算
- 采用**手动编程**，通过设置开关和插拔电缆来实现

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

# Electronic Numerical Integrator And Computer



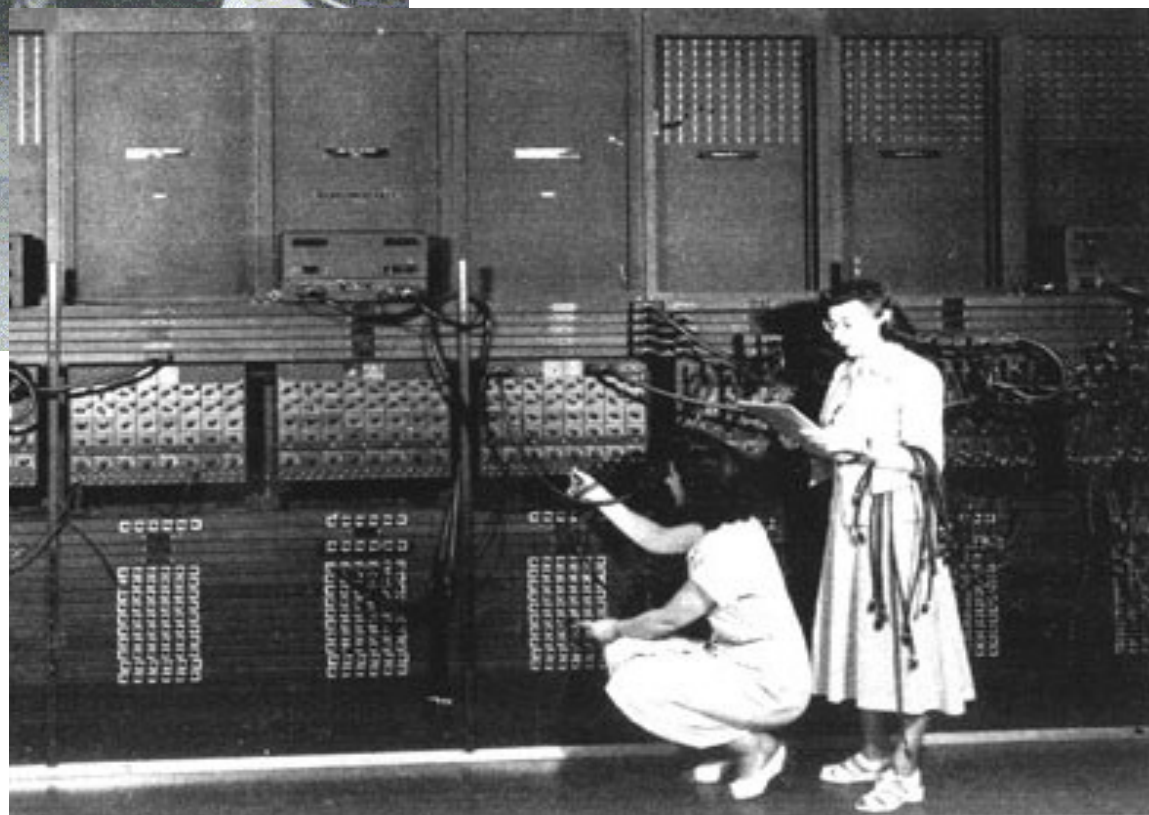
占地面积170平方米

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  
80 223个小时



# 冯·诺依曼的故事

- 1944年，冯·诺伊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，诺伊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，他们在共同讨论的基础上，冯·诺伊曼以“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**批准让冯·诺依曼建造计算机，其依据就是这份报告。



**E**lectronic  
**D**iscrete  
**V**ariable  
**A**utomatic  
**C**omputer

# 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，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·诺依曼结构最重要的思想是“**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。

- 冯·诺依曼结构计算机也称为**冯·诺依曼机器（ Von Neumann Machine ）**。
- 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。

冯·诺依曼结构的主要思想是什么呢？

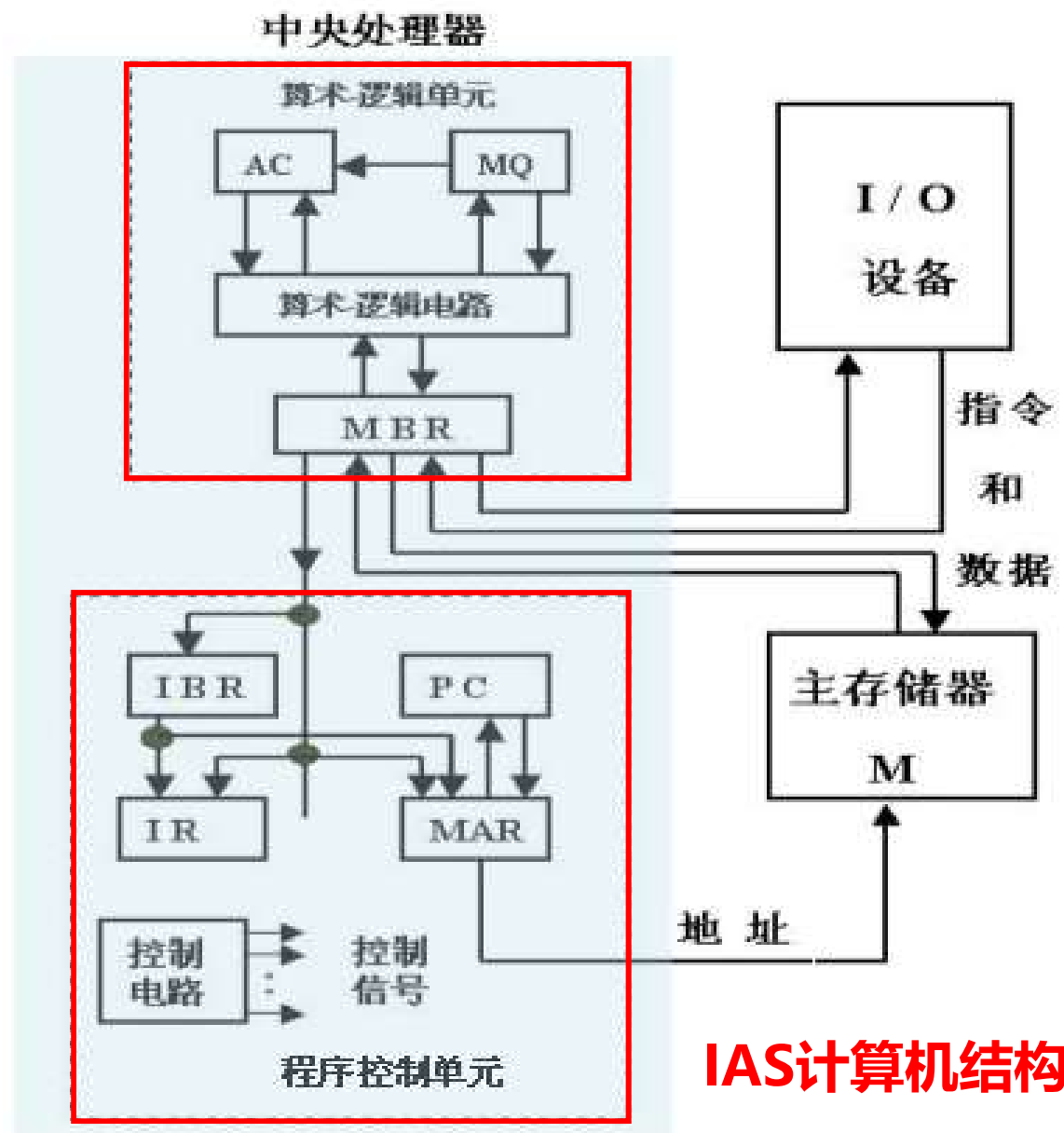


# 你认为冯·诺依曼结构是怎样的？

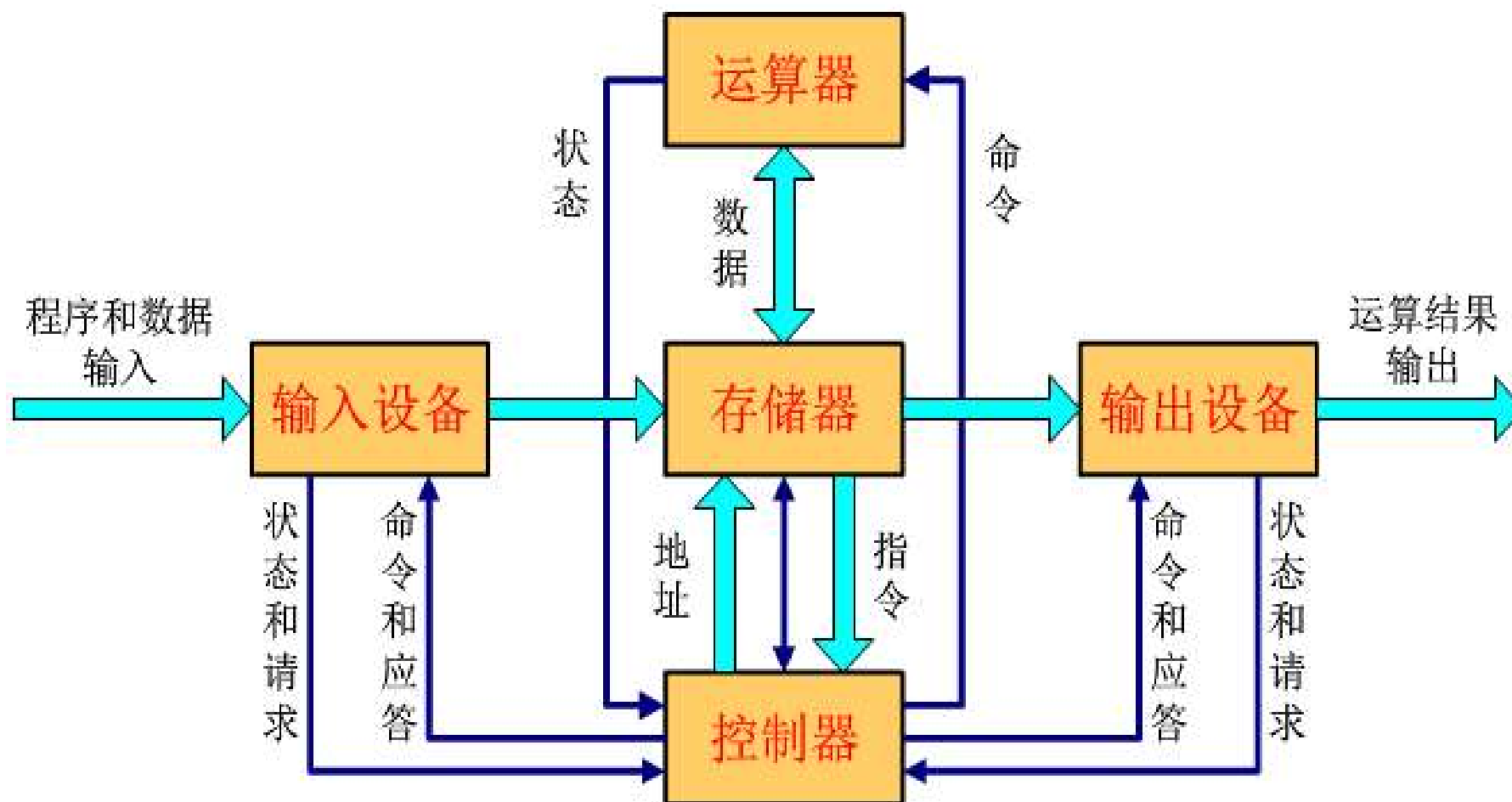
- 应该有个主存，用来存放程序和数据
- 应该有一个自动逐条取出指令的部件
- 还应该具体执行指令（即运算）的部件
- 程序由指令构成
- 指令描述如何对数据进行处理
- 应该有将程序和原始数据输入计算机的部件
- 应该有将运算结果输出计算机的部件

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



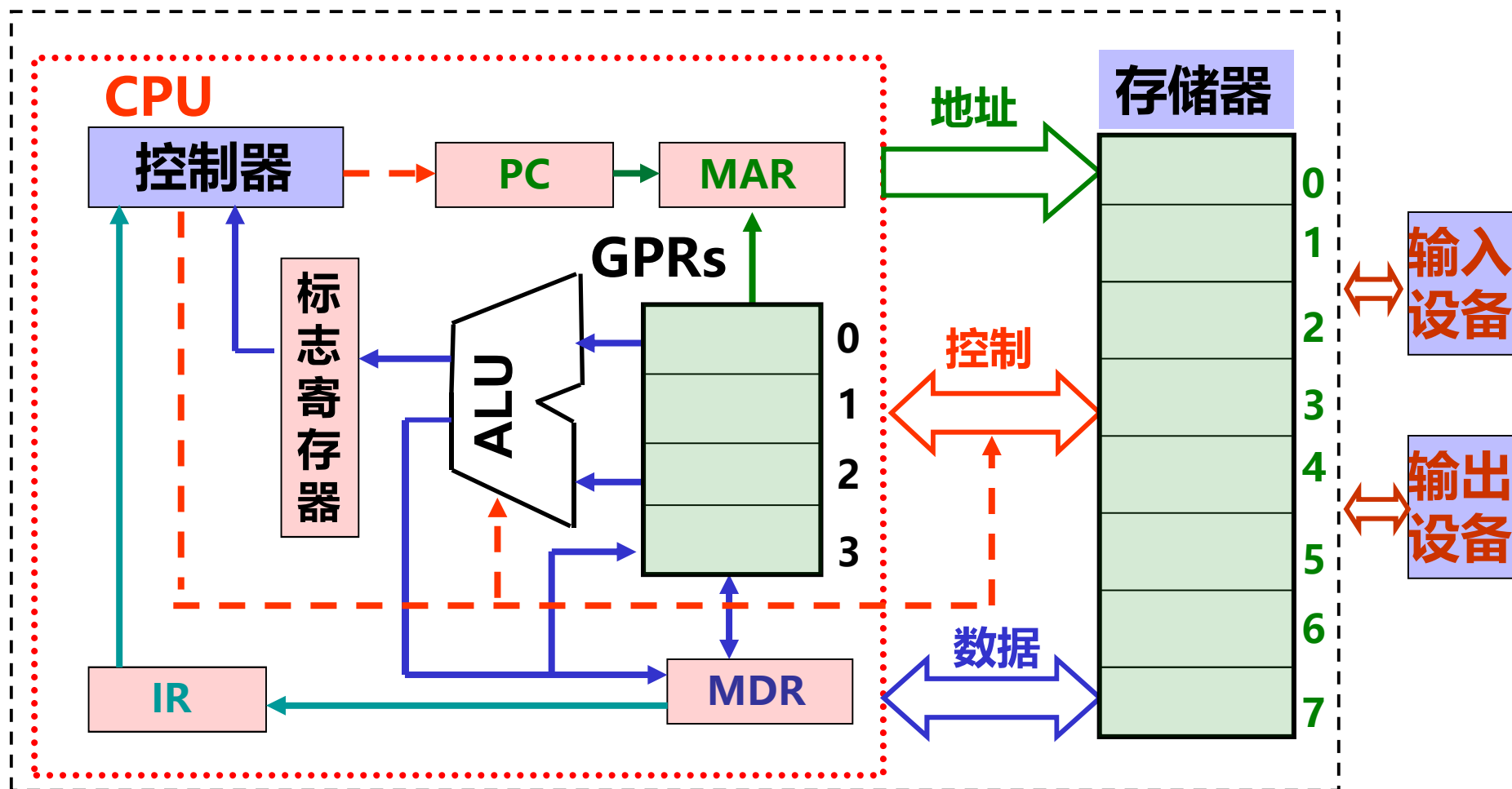
# 冯·诺依曼结构的主要思想

---

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**应能自动取出指令来执行；
  - **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。

# 现代计算机结构模型

你还记得冯.诺依曼计算机结构的特点吗？



你能想到计算机相当于现实生活中的什么呢？ 工厂、饭店？

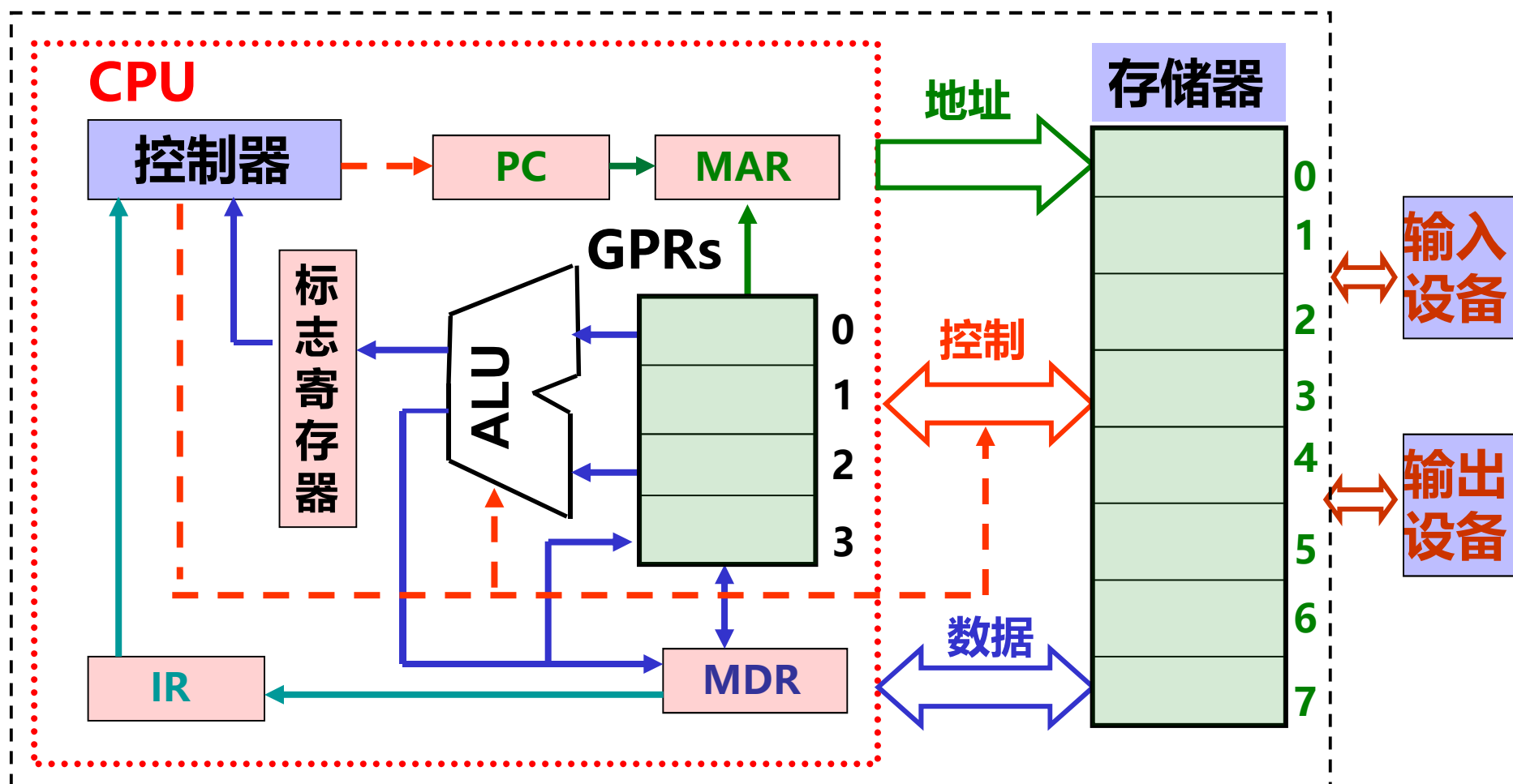
计算机是如何工作的呢？

# 认识计算机中最基本的部件

**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

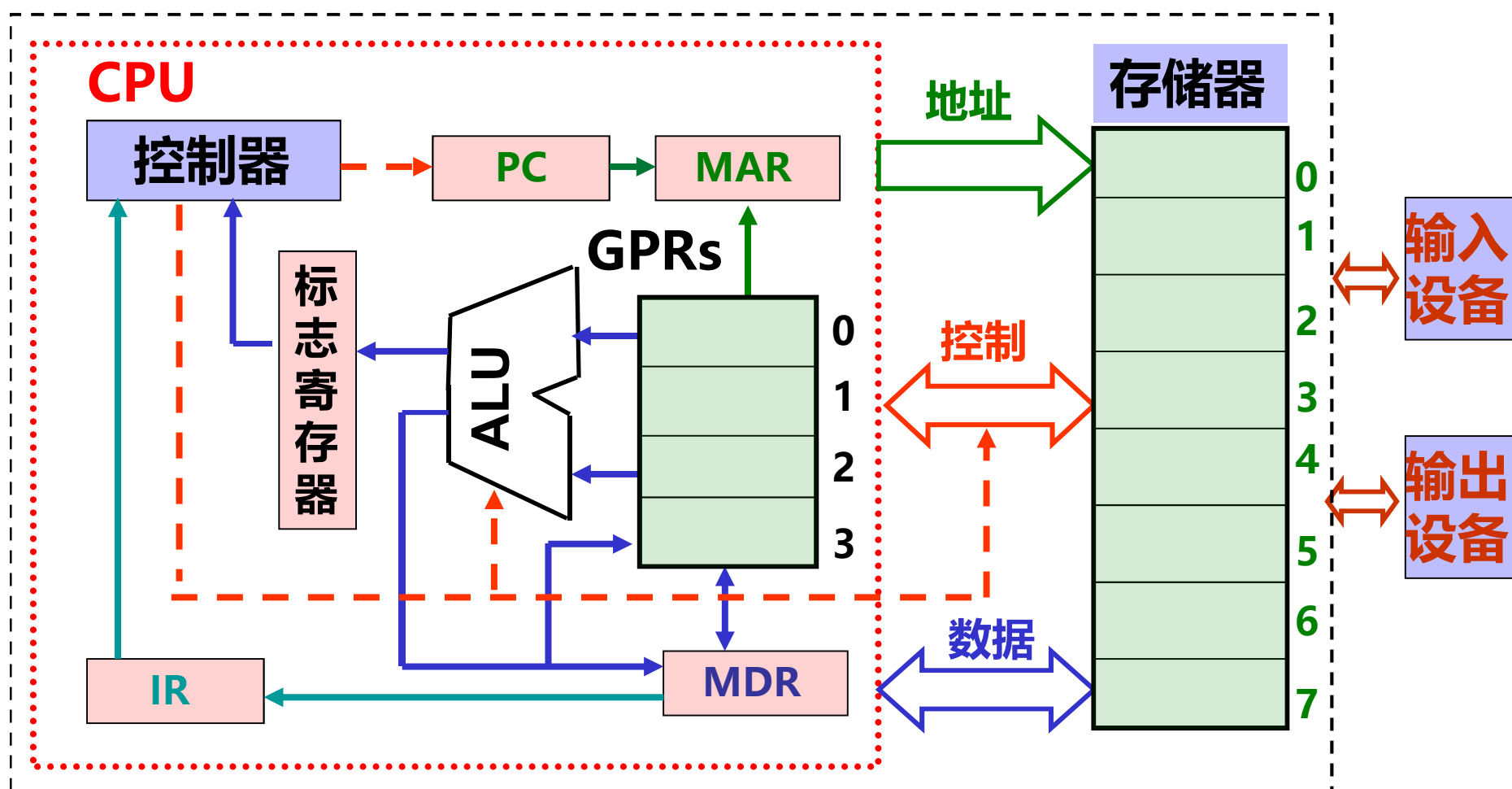
**GPRs**：通用寄存器组（由若干通用寄存器组成，早期就是累加器）



# 计算机是如何工作的？

先想象一下妈妈是怎样做一桌你喜欢（指定）的菜的？

厨房-CPU，你妈-控制器，盘-GPRs，锅灶等-ALU，架子-存储器



# 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，我告诉妈妈从第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）

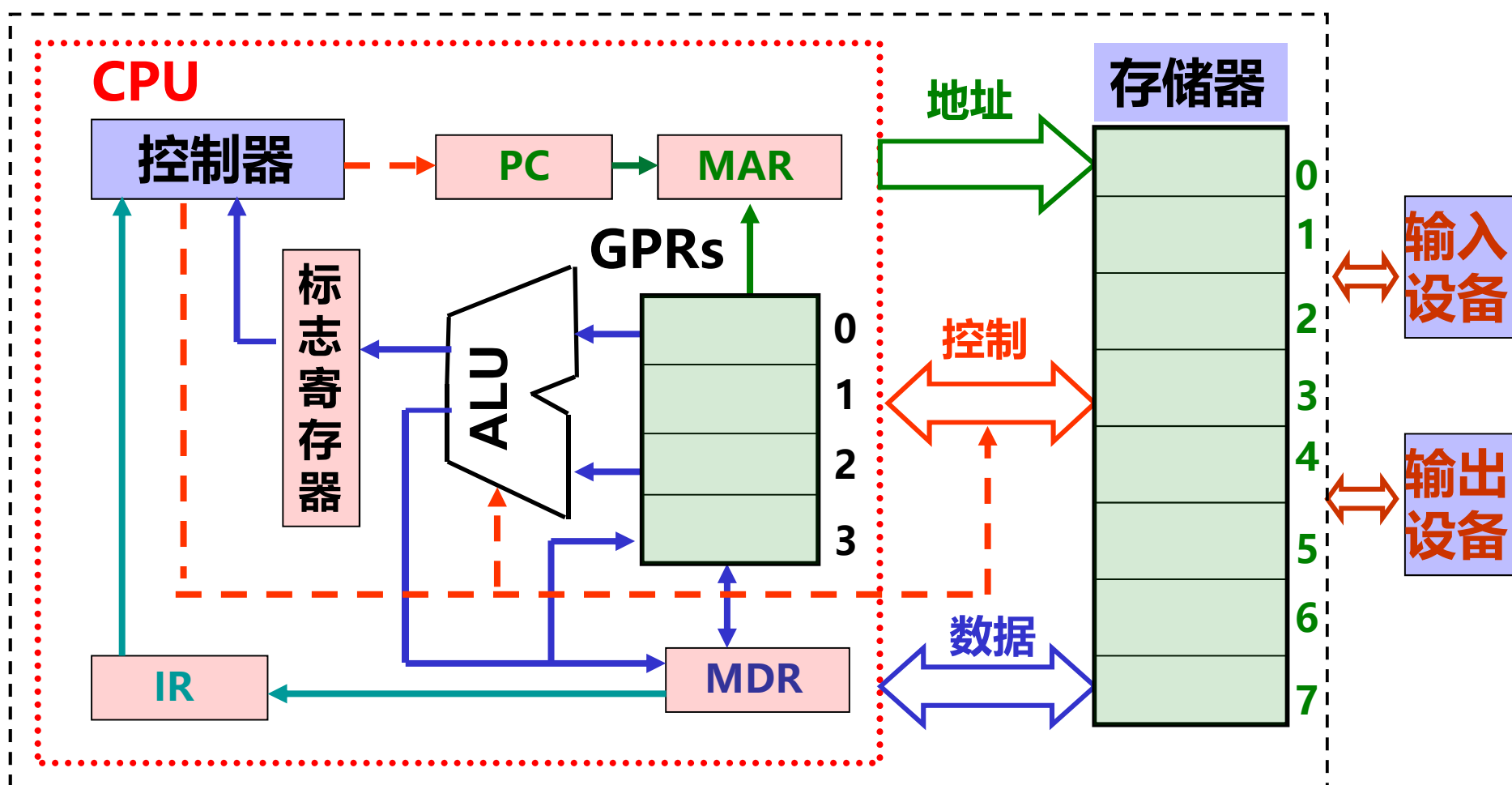
第六步：算出下一菜谱所在架子号 $6=5+1$ （修改PC的值）

继续做下一道菜（执行下一条指令）

# 计算机是如何工作的？

如果你知道你妈妈是如何做饭的，你就已经知道计算机是如何工作的！

你能告诉我计算机是如何工作的吗？“存储程序”工作方式！



# 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。妈妈从第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中，数据在GPR中。

**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！**



# 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、GPRs、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(MDR)、地址(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！



南京大學  
NANJING UNIVERSITY



# 程序开发和执行过程简介

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 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

输入：按钮、开关；所有信息都是0/1序列！  
输出：指示灯等

假设：0010-jc

0：0101 0110

1：0010 0100

2：.....

3：.....

4：0110 0111

5：.....

6：.....



太原始了，无法忍受，咋办？

用符号表示而不用0/1表示！

若在第4条指令前加入指令，则需重新计算地址码（如jxx的目标地址），然后重新打孔。不灵活！

书写、阅读困难！

# 用汇编语言开发程序

- 若用**符号**表示跳转位置和变量位置，是否简化了问题？

- 于是，汇编语言出现
  - 用**助记符**表示操作码
  - 用**标号**表示位置
  - 用**助记符**表示寄存器
  - .....

0 : 0101 0110

1 : 0010 0100

2 : .....

3 : .....

4 : 0110 0111

5 : .....

6 : .....

7 : .....

add B

jc L0

.....

.....

L0 : sub C

.....

B : .....

C : .....

你认为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他指令

不需记忆指令编码，编写方便

可读性比机器语言强

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  
add、jxx和sub指令中的地址码！

# 进一步认识机器级语言

- 汇编语言(源)程序由**汇编指令**构成
- 你能用一句话描述**什么是汇编指令**吗？
  - 用助记符和标号来表示的指令（与机器指令一一对应）
- **指令**又是什么呢？
  - 包含操作码和操作数或其地址码  
（**机器指令**用**二进制**表示，**汇编指令**用**符号**表示）
  - 只能描述：取（或存一个数）  
两个数加（或减、乘、除、与、或等）  
根据运算结果判断是否转移执行
- 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（sort）、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就不能运行！

```
add B
jc L0
.....
.....
L0 : sub C
.....
B : .....
C : .....
```



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**

**SKIP**

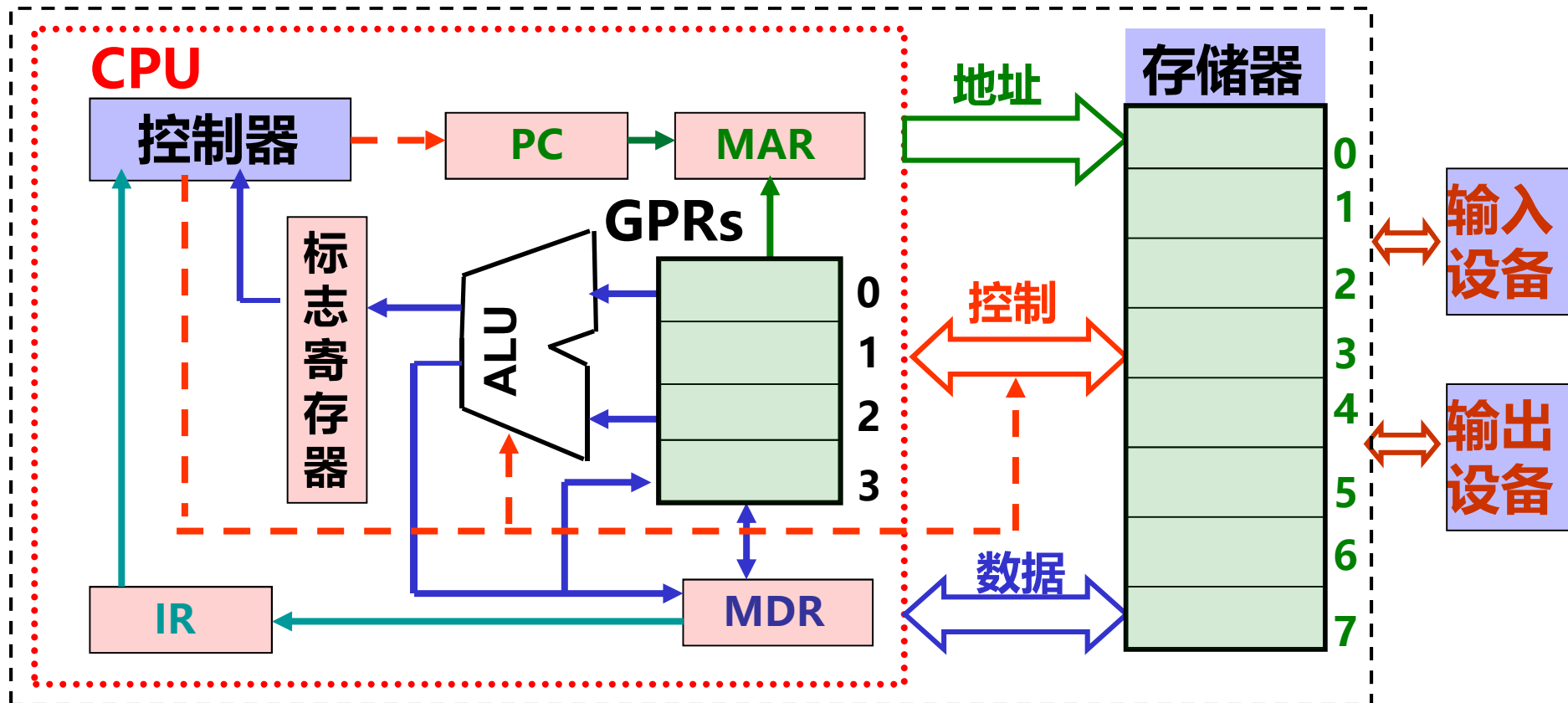
结论：用汇编语言比机器语言好，但是，还是很麻烦！

# 指令所能描述的功能

对于以下结构的机器，你能设计出几条指令吗？

[BACK](#)

Ld M# , R#     ( 将存储单元内容装入寄存器 )  
St R# , M#     ( 将寄存器内容装入存储单元 )  
Add R# , M#    ( 类似的还有Sub , Mul等 ; 操作数还可 “R# , R#” 等 )  
Jxx M#        ( 若满足条件 , 则转移到另一处执行 )  
.....



# 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
    - 面向算法描述，比机器级语言描述能力强得多
    - 高级语言中一条语句对应几条、几十条甚至几百条指令
    - 有“面向过程”和“面向对象”的语言之分
    - 处理逻辑分为三种结构
      - 顺序结构、选择结构、循环结构
    - 有两种转换方式：“编译”和“解释”
      - 编译程序(Compiler)：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)：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。
- 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转换为机器语言程序

# 一个典型程序的转换处理过程

经典的 “hello.c” C-源程序

```
#include <stdio.h>

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

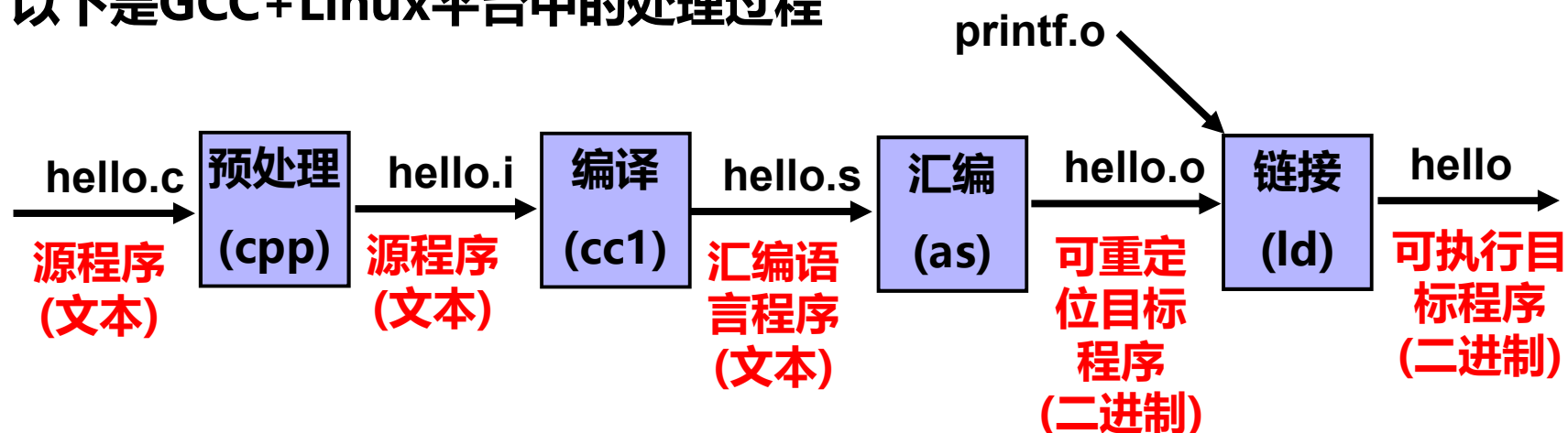
hello.c的ASCII文本表示

```
# 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
```

功能：输出 “hello,world”

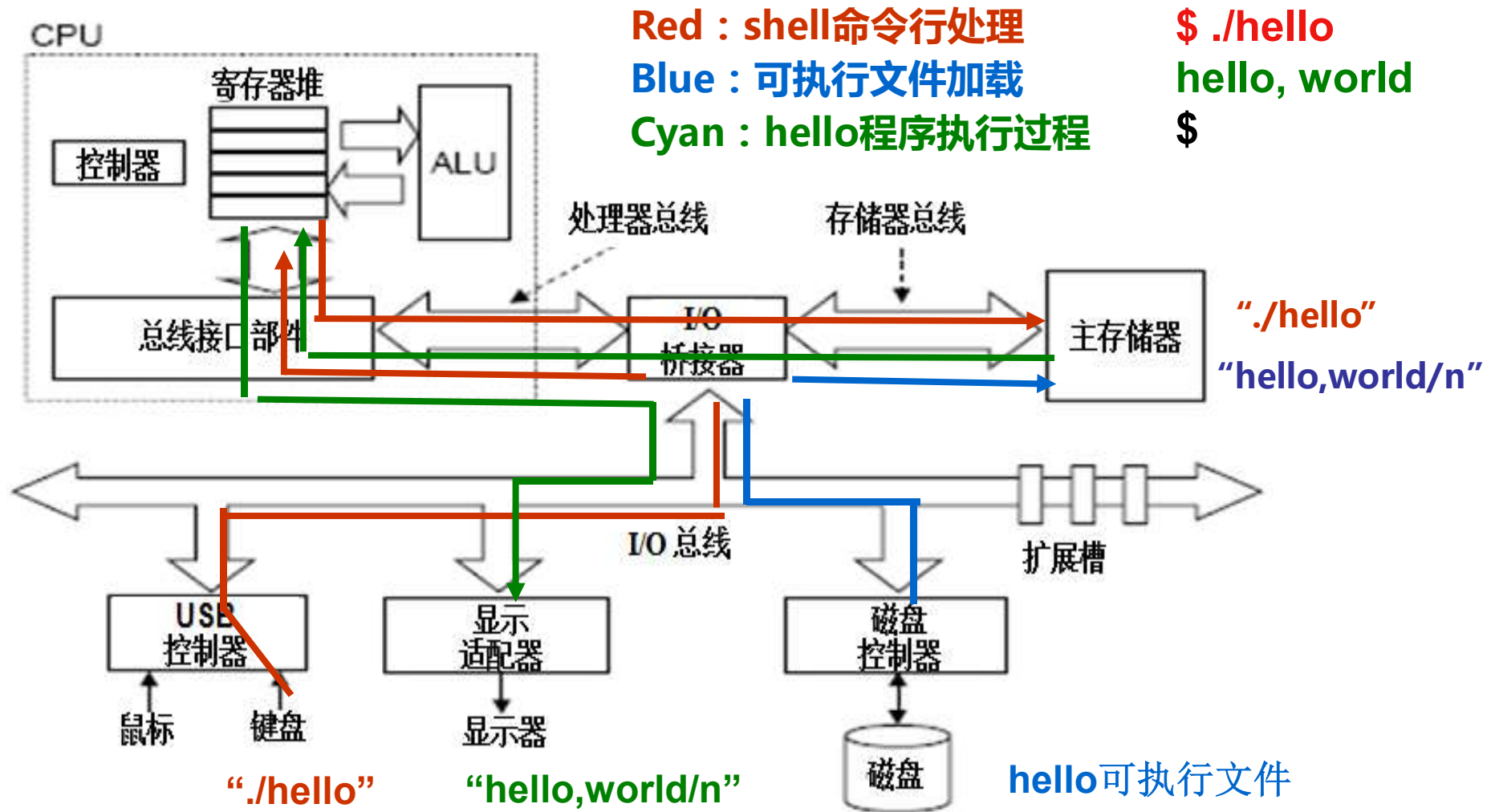
计算机不能直接执行hello.c！

以下是GCC+Linux平台中的处理过程





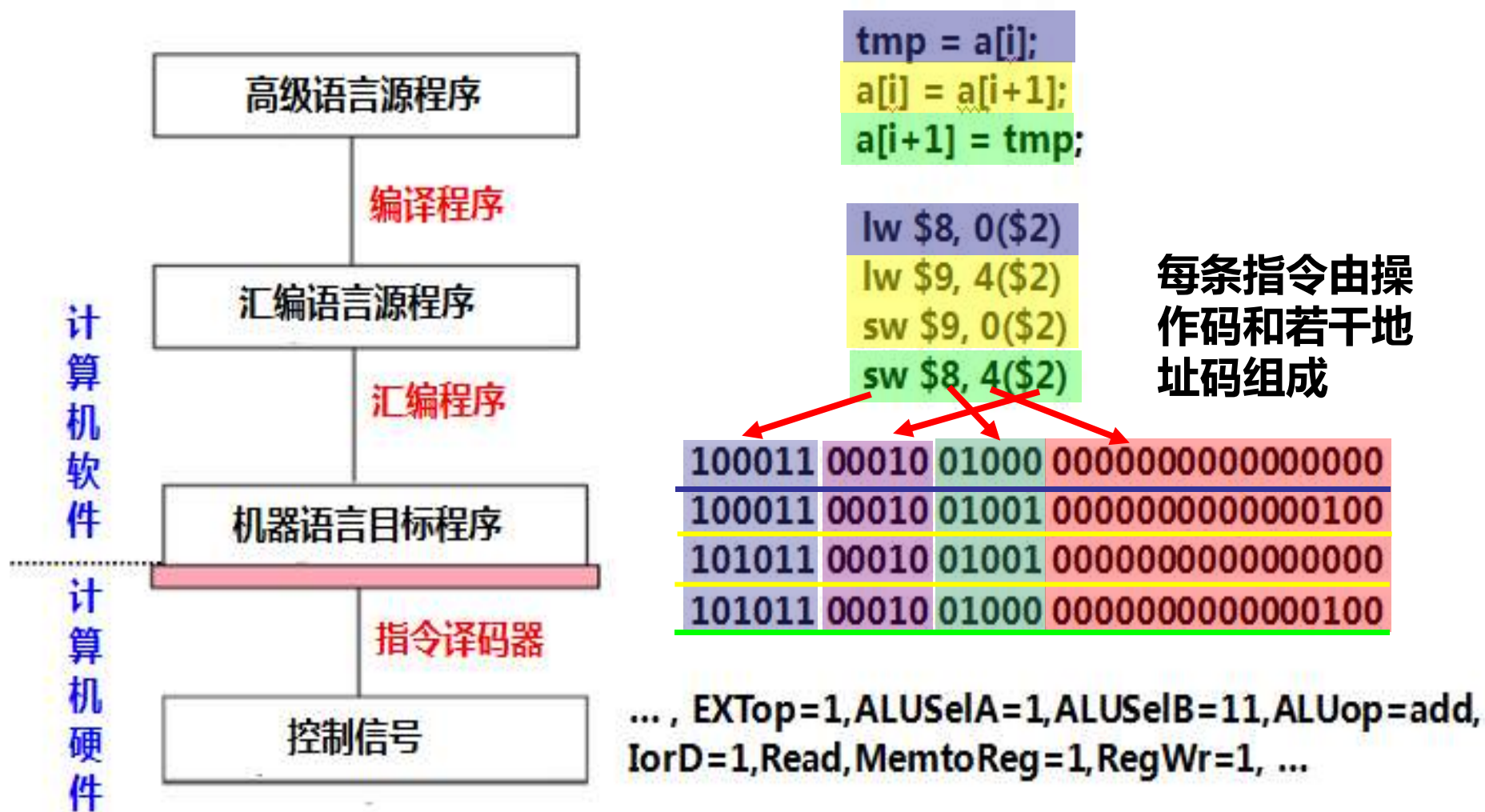
# Hello程序的数据流动过程



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。

# 不同层次语言之间的等价转换



**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
  - 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

语言  
处理  
程序

语言处理系统 +

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

人机  
接口

+  
操作  
系统

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



南京大学  
NANJING UNIVERSITY



# 计算机系统层次结构

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 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
  - 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

语言  
处理  
程序

语言处理系统 +

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

人机  
接口

+  
操作  
系统

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

# 早期计算机系统的层次

---

- 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 )

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

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

应用程序

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

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”，不需要描述具体的算法实现细节。

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



**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

# 回顾：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

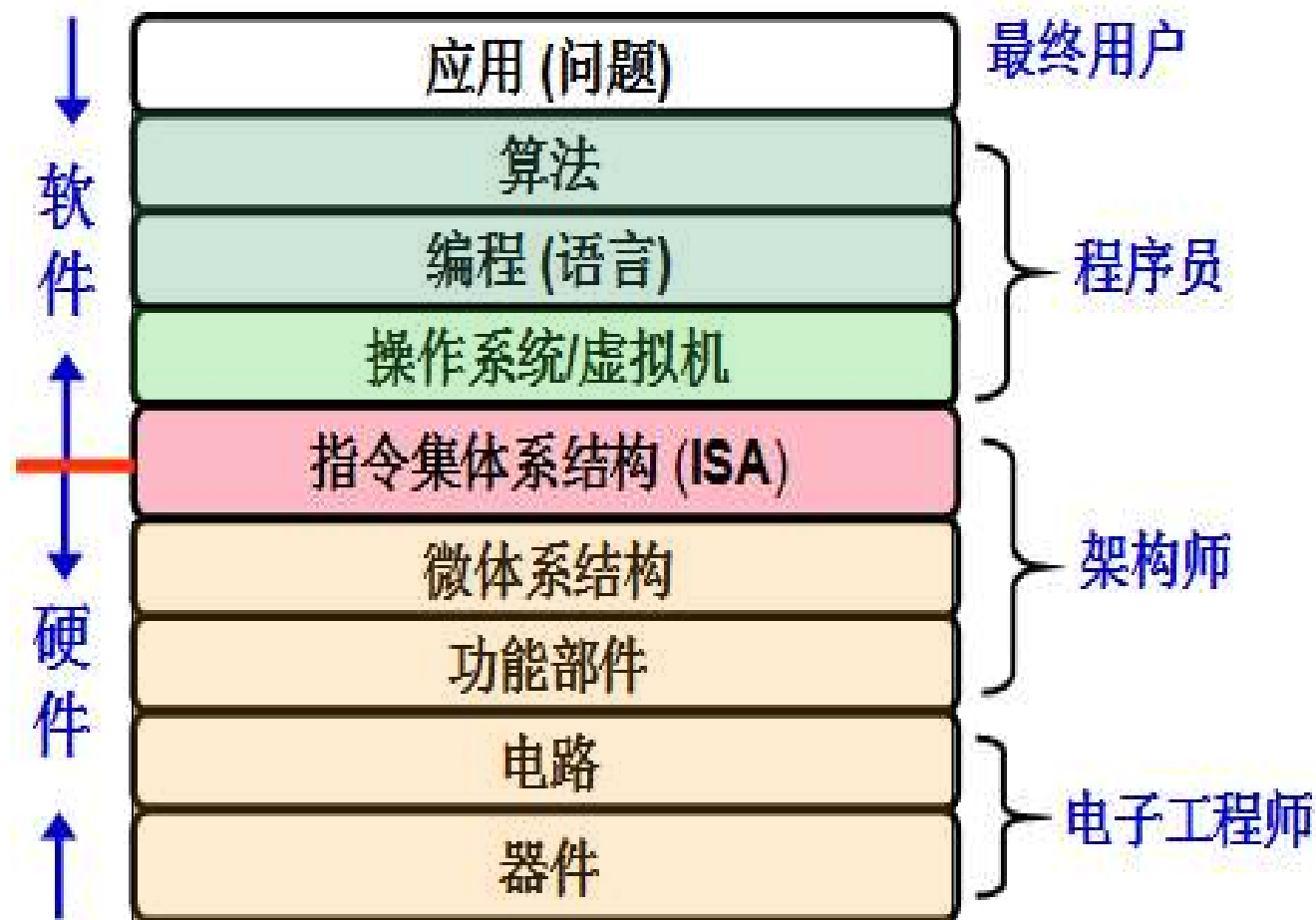
ISA

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

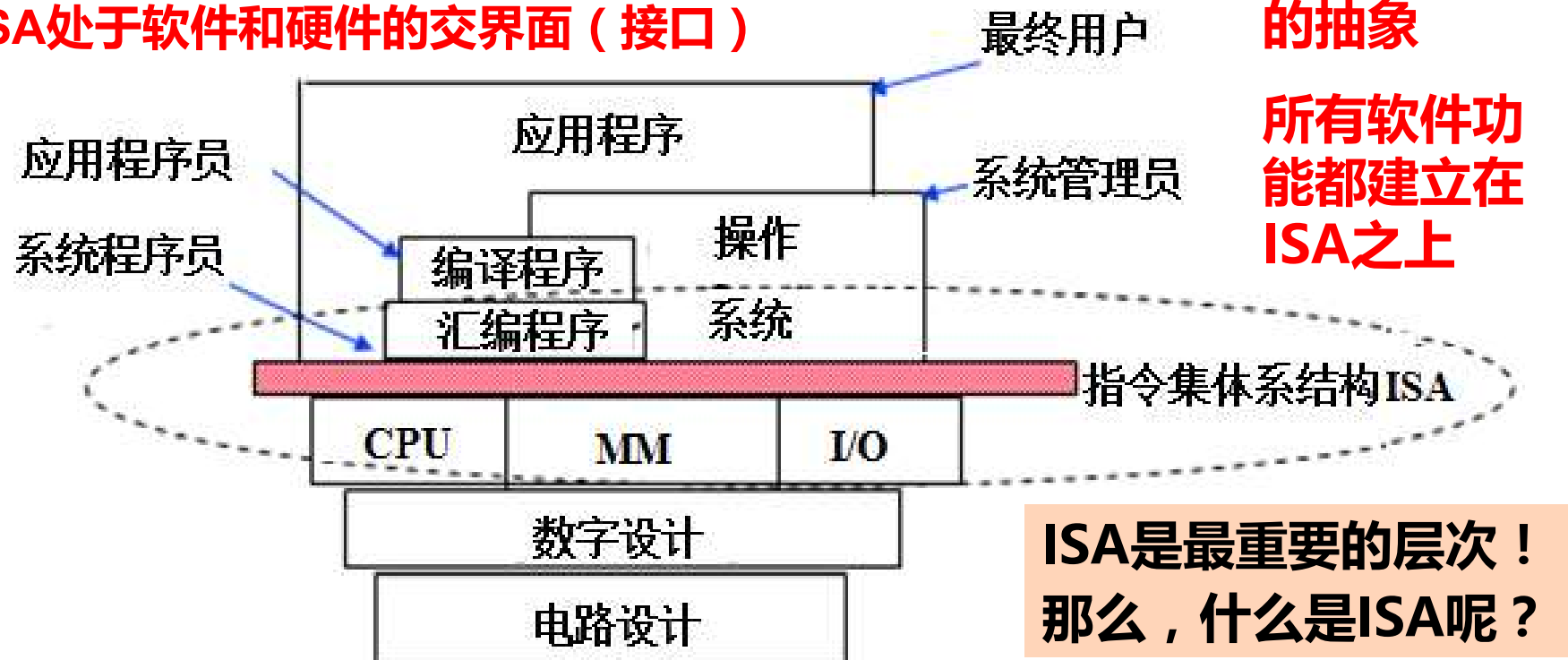
**语言处理系统**建立在**操作系统**之上

**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）**



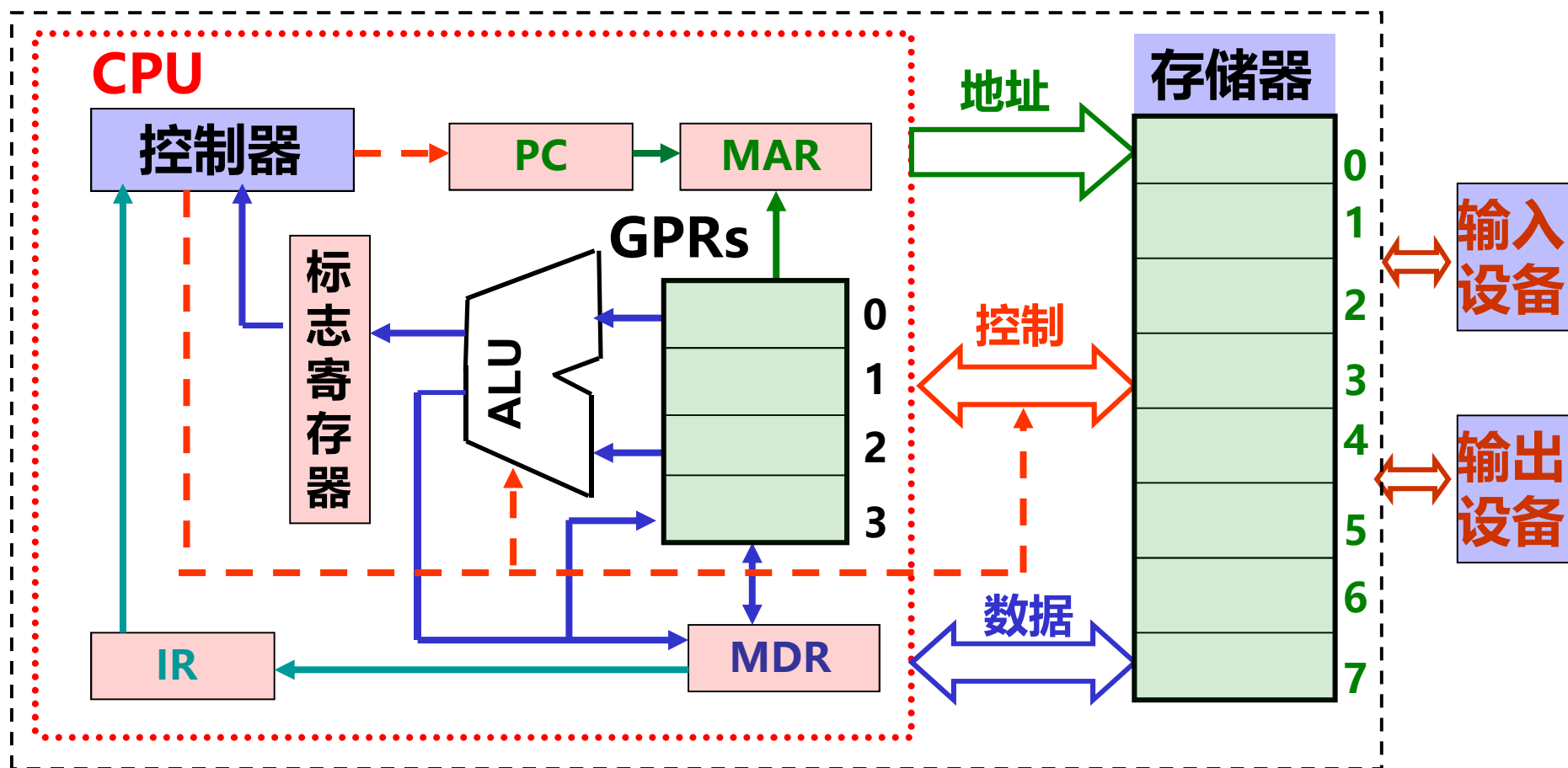
# 指令集体系结构（ISA）

- ISA指Instruction Set Architecture，即**指令集体系结构**，有时简称为**指令系统**
- 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（PC）、条件码定义等**。
- ISA在**通用**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”

微体系结构

**ISA和计算机组成（Organization，即MicroArchitecture）是何关系？**

# ISA和计算机组成（微结构）之间的关系



**ISA是计算机组成的抽象**

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



南京大學  
NANJING UNIVERSITY



# 本课程的主要学习内容

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 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



# 后PC时代计算机教学面临的挑战

---

## 后PC时代的几个特征

- 计算资源多样化，I/O设备无处不在，数据中心、PMD与PC等共存
- 软件和硬件协同设计（硬件、OS和编译器之间的关联更加密切）
- 对应用程序员的要求更高

- 编写高效程序必需了解计算机底层结构
- 必需掌握并行程序设计技术和工具
- 应用问题更复杂，领域更广

- 气象、生物、医药、地质、天文等领域的高性能计算
- Google、百度等互联网应用领域海量“大数据”处理
- 物联网（移动设备、信息家电等）嵌入式开发
- 银行、保险、证券等大型数据库系统开发和维护
- 游戏、多媒体等实时处理软件开发，。。。。。

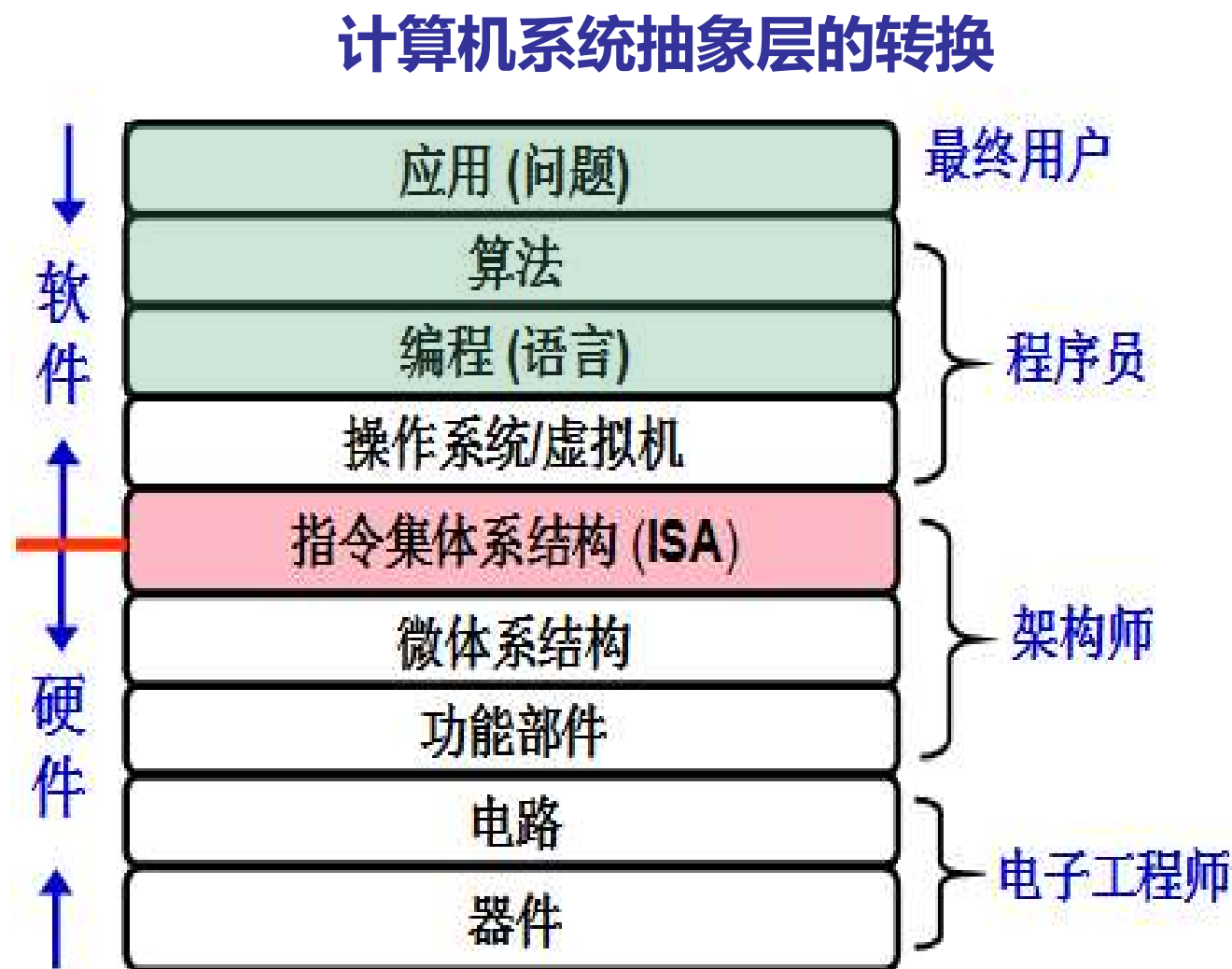
大规模  
分布式  
多粒度并行

“并行”成为重要主题、培养具有系统观的软/硬件贯通人才是关键！



# 计算机系统抽象层的转换

- 计算机学科主要研究的是**计算机系统各个不同抽象层的实现及其相互转换的机制**
- 计算机学科培养的应该主要是在**计算机系统或在系统某些层次上从事相关工作的人才**



所有计算机专业课程基本上围绕其中一个或多个不同抽象层开展教学！

本课程为**基础课程**，试图通过讲解程序开发和执行来构建计算机系统总体框架

# “计算机系统基础” 课程

---

## 从程序员角度出发来认识计算机系统

- 教学目标：

培养学生的**系统能力**，使其成为一个“**高效**”程序员，在程序调试、性能提升、程序移植和健壮性等方面成为高手；建立完整的计算机系统概念，为后续的OS、编译、体系结构等课程打下坚实基础

- 以 **IA-32+Linux+C+gcc** 为平台（开源项目平台）

- 与以下**MOOC课程**的想法类似

<https://www.coursera.org/course/hwswinterface>

**主要内容：描述程序执行的底层机制**

- 思路：

在程序与程序的执行机制之间**建立关联**，**强化理解**而不是记忆



# “计算机系统基础” 课程内容概要

```
/*---sum.c---*/
```

```
int sum(int a[ ], unsigned len)
```

```
{
```

```
    int  i, sum = 0;
```

```
    for (i = 0; i <= len-1; i++)
```

```
        sum += a[i];
```

```
    return sum;
```

```
}
```

```
/*---main.c---*/
```

```
int main()
```

```
{
```

```
    int  a[1]={100};
```

```
    int  sum;
```

```
    sum=sum(a,0);
```

```
    printf(“%d”,sum);
```

```
}
```

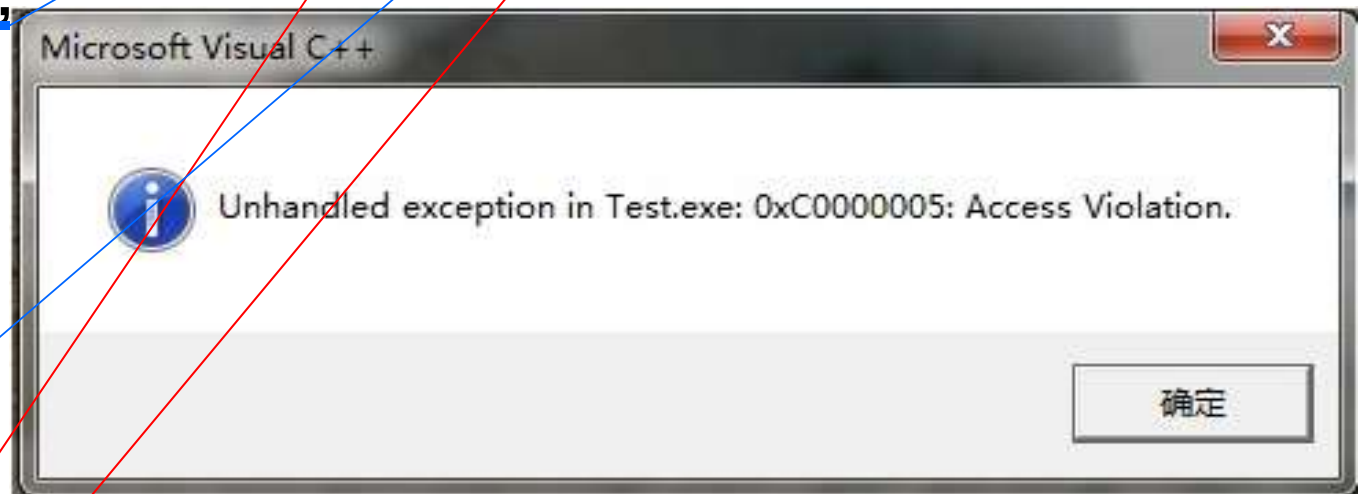
数据的表示

数据的运算

各类语句的转换与表示(指令)

各类复杂数据类型的转换表示

过程（函数）调用的转换表示



链接（linker）和加载

程序执行（存储器访问）

异常和中断处理

输入输出(I/O)

# “计算机系统基础” 课程内容概要

- 使学生清楚理解：

**计算机是如何生成和运行可执行文件的！**

“问题求解” 解决  
应用→算法（数据  
结构）→编程层

- 重点在高级语言以下各抽象层

- C语言程序设计层

- 数据的机器级表示、运算

- 语句和过程调用的机器级表示

- 指令集体系结构（ISA）和汇编层

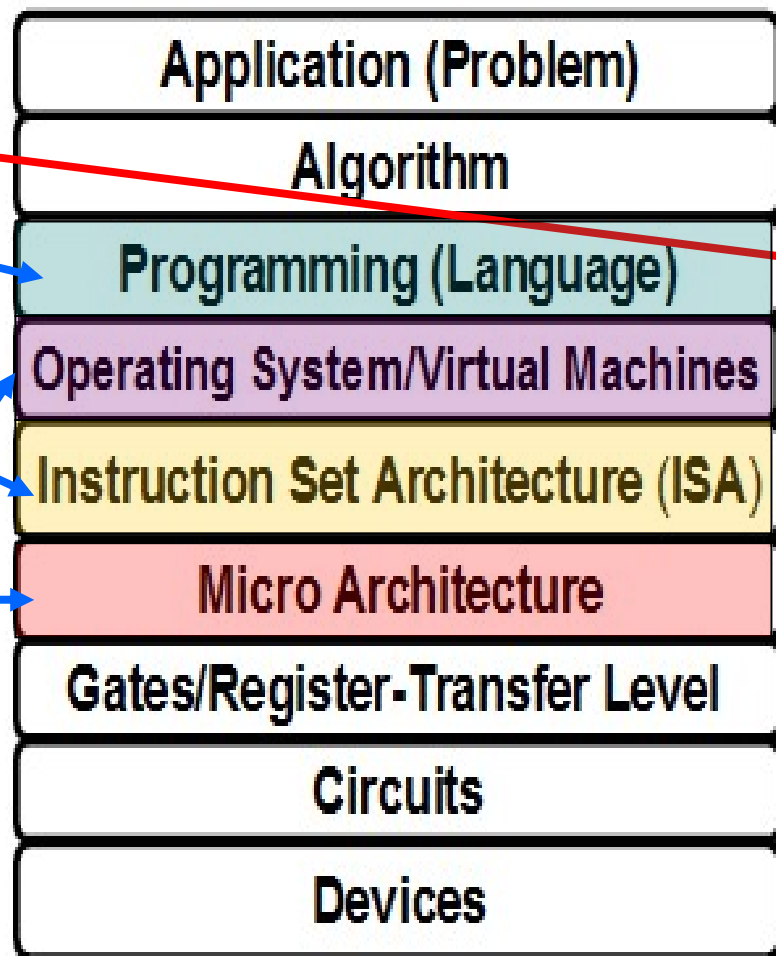
- 指令系统、机器代码、汇编语言

- 微体系结构及硬件层

- CPU的通用结构

- 层次结构存储系统

- 操作系统、编译和链接的部分内容



# “计算机系统基础” 课程内容概要

---

## 三个主题：

- 表示 ( Representation )
  - 不同数据类型 ( 包括带符号整数、无符号整数、浮点数、数组、结构等 ) 在寄存器或存储器中如何表示和存储？
  - 指令如何表示和编码 ( 译码 ) ？
  - 存储地址 ( 指针 ) 如何表示？如何生成复杂数据结构中数据元素的地址？
- 转换 ( Translation )
  - 高级语言程序对应的机器级代码是怎样的？如何转换并链接生成可执行文件？
- 执行控制流 ( Control flow )
  - 计算机能理解的“程序”是如何组织和控制的？
  - 如何在计算机中组织多个程序的并发执行？
  - 逻辑控制流中的异常事件及其处理
  - I/O操作的执行控制流 ( 用户态→内核态 )

# “计算机系统基础” 课程内容概要

---

**先导知识：**C语言程序设计

**内容组织：**分两门短课程

- **第一部分 可执行文件的生成（表示和转换）** 本阶段将学习  
第一门短课程
  - 计算机系统概述
  - 数据的机器级表示与处理
  - 程序的转换及机器级表示
  - 程序的链接

计算机系统基础(一) ----程  
序的表示、转换与链接
- **第二部分 可执行文件的运行（执行控制流）**
  - 程序的执行
  - 层次结构存储系统
  - 异常控制流
  - I/O操作的实现

计算机系统基础(二) ----程  
序的执行、异常及IO处理