计算机硬件系统

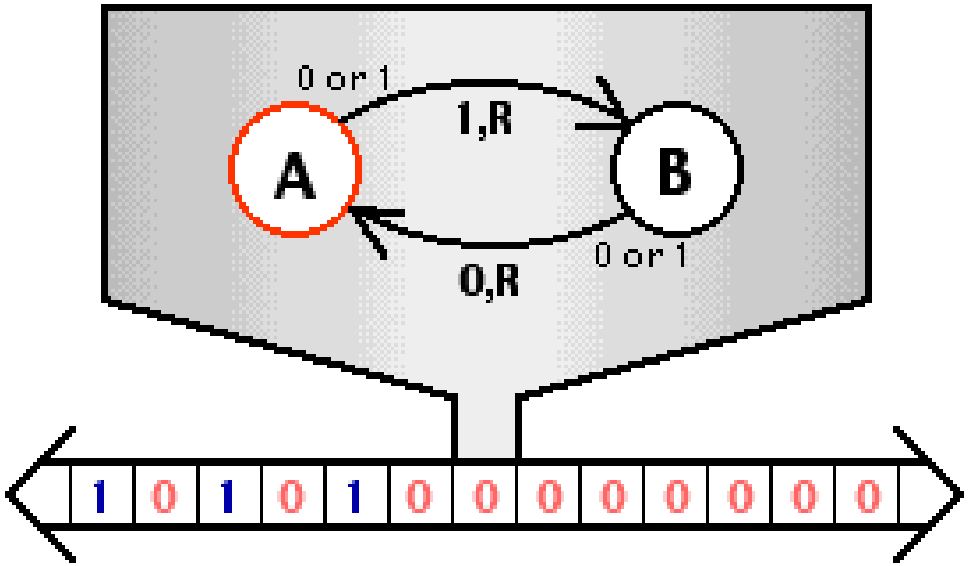
北京大学计算机系
代亚非

要了解的内容

- 冯·诺依曼结构如何工作？
  - 运算器
  - 控制器
  - 存储器
- 存储层次
- 程序的局部性原理

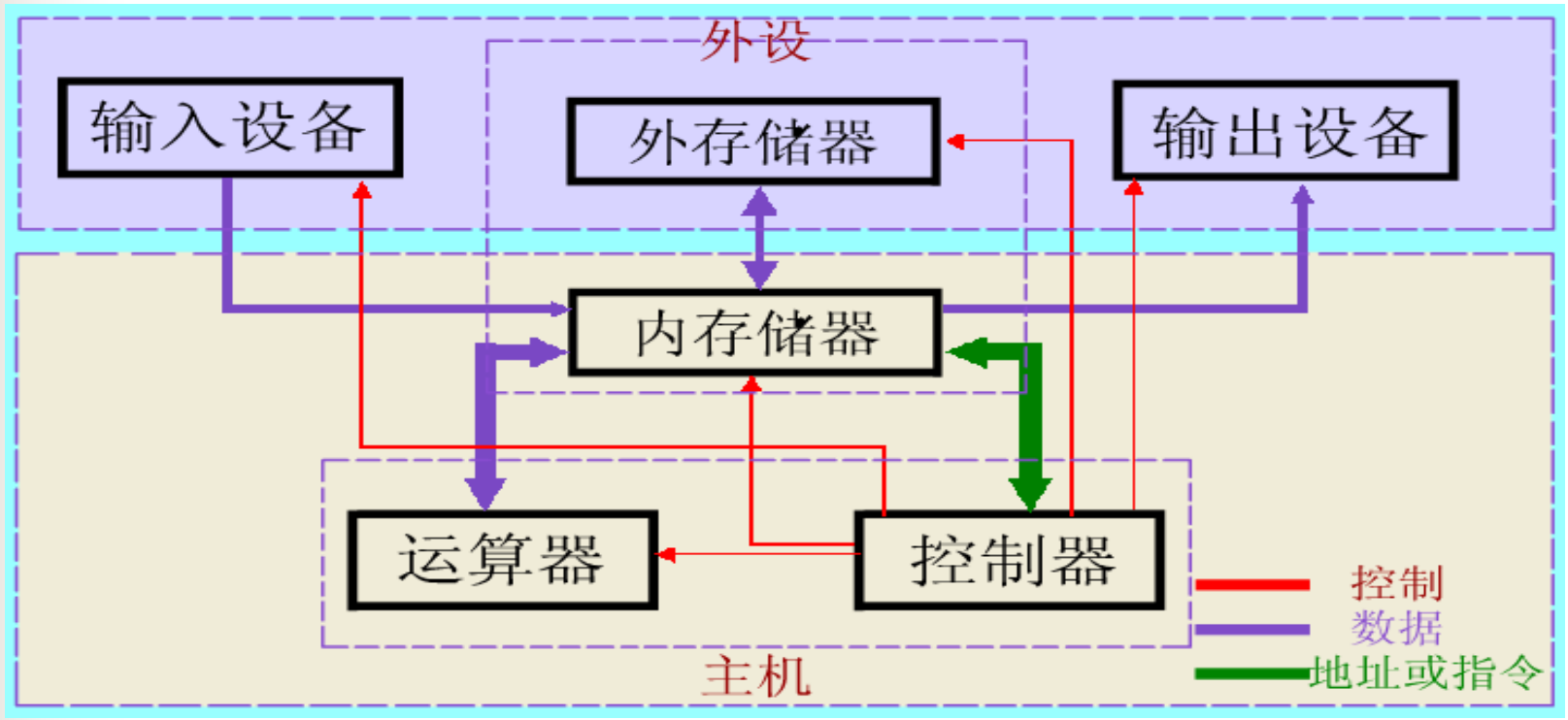
图灵机模型-回顾

- 图灵机定义的是一种具有能行性的用数学方法精确定义的计算模型，采用了算法的思想来研究计算的过程，并由此揭示可计算的概念。
- 由于算法的思想与今日计算机上运行的程序之间有着更密切的关系，从而使他的理论受到重视并被广泛使用。
- 图灵机的三个组成，对应了现代计算机的内存，处理器和程序。



现代计算机体系结构

- 冯.诺依曼(Von. Neumann, 1903～1957)：美国数学教授，提出了冯.诺依曼结构，奠定了现代计算机体系结构框架，被称为现代计算机之父。



冯.诺依曼结构

- 冯·诺依曼1945年提出存储存储程序原理的基本思想，奠定了计算机自动计算的基础--存储程序原理。

1. 计算机要自动完成解题任务，必须事先设计好用以描述计算机解题过程的程序。
2. 程序如同数据一样采用二进制形式存储在机器中。
3. 计算机在工作时自动高速地从机器中逐条取出指令加以执行。。

主机的硬件组成

DRAM

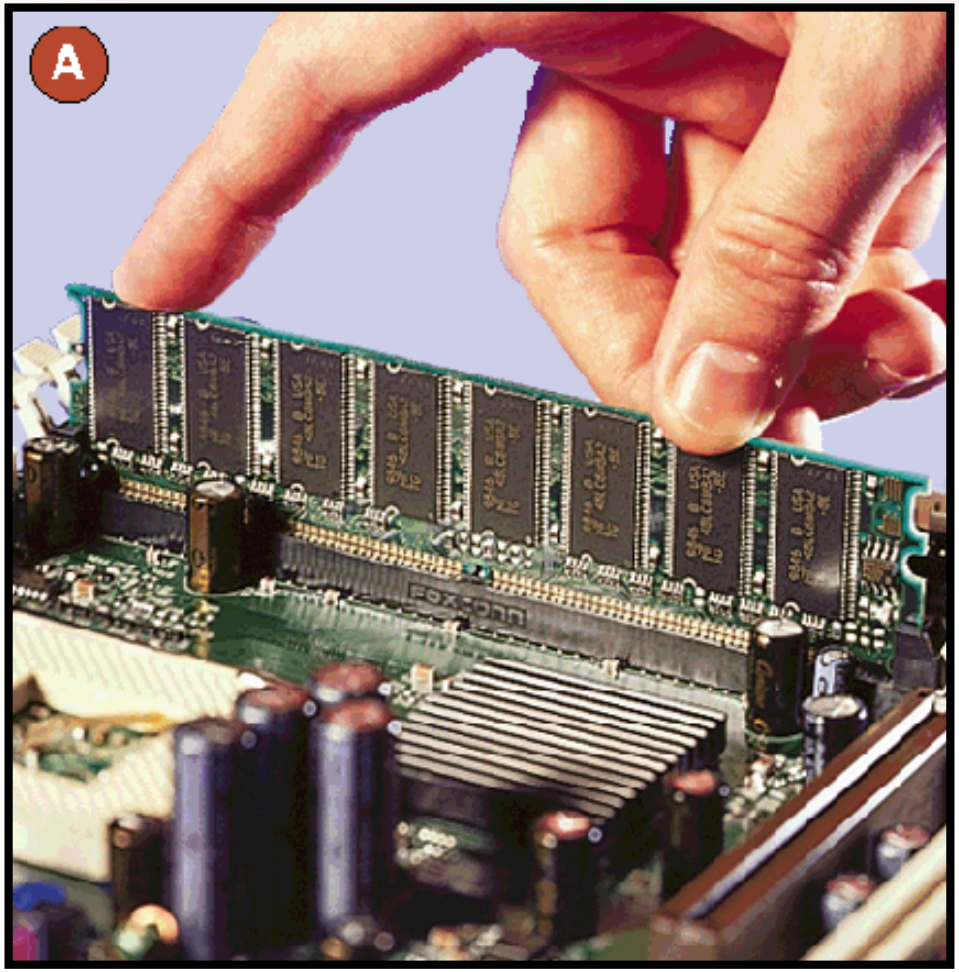
ROM

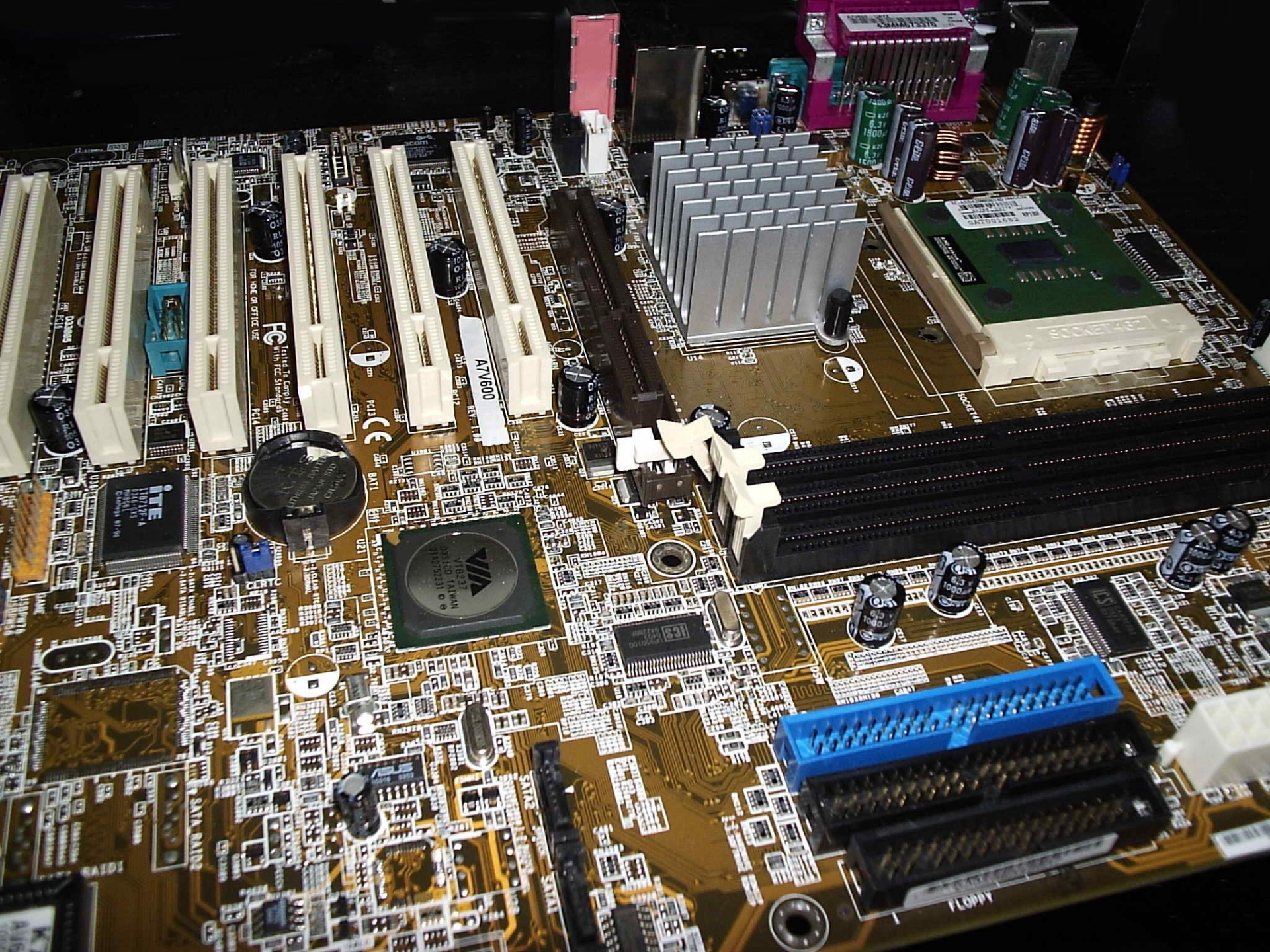
存储器

Central Processing Unit (CPU) or Microprocessor

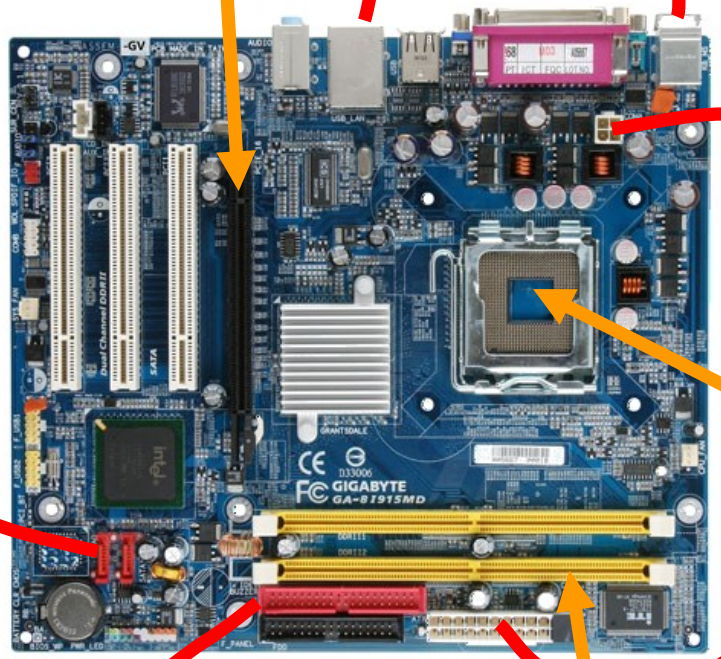
中央处理器（微处理器）

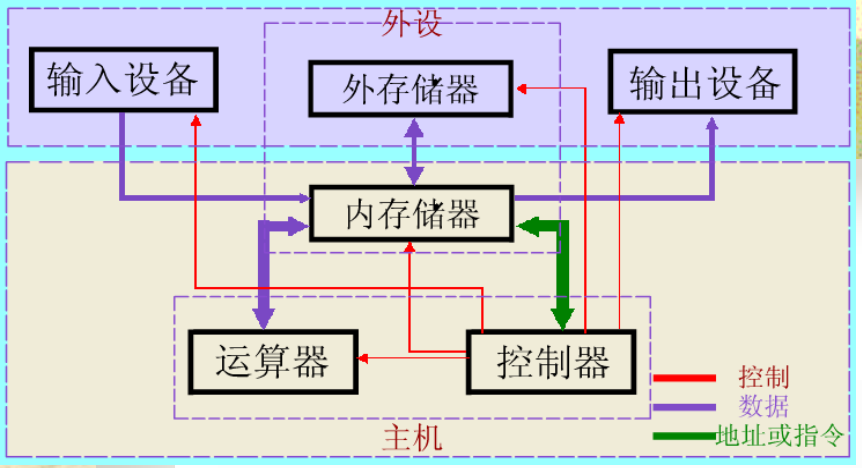
插入内存条

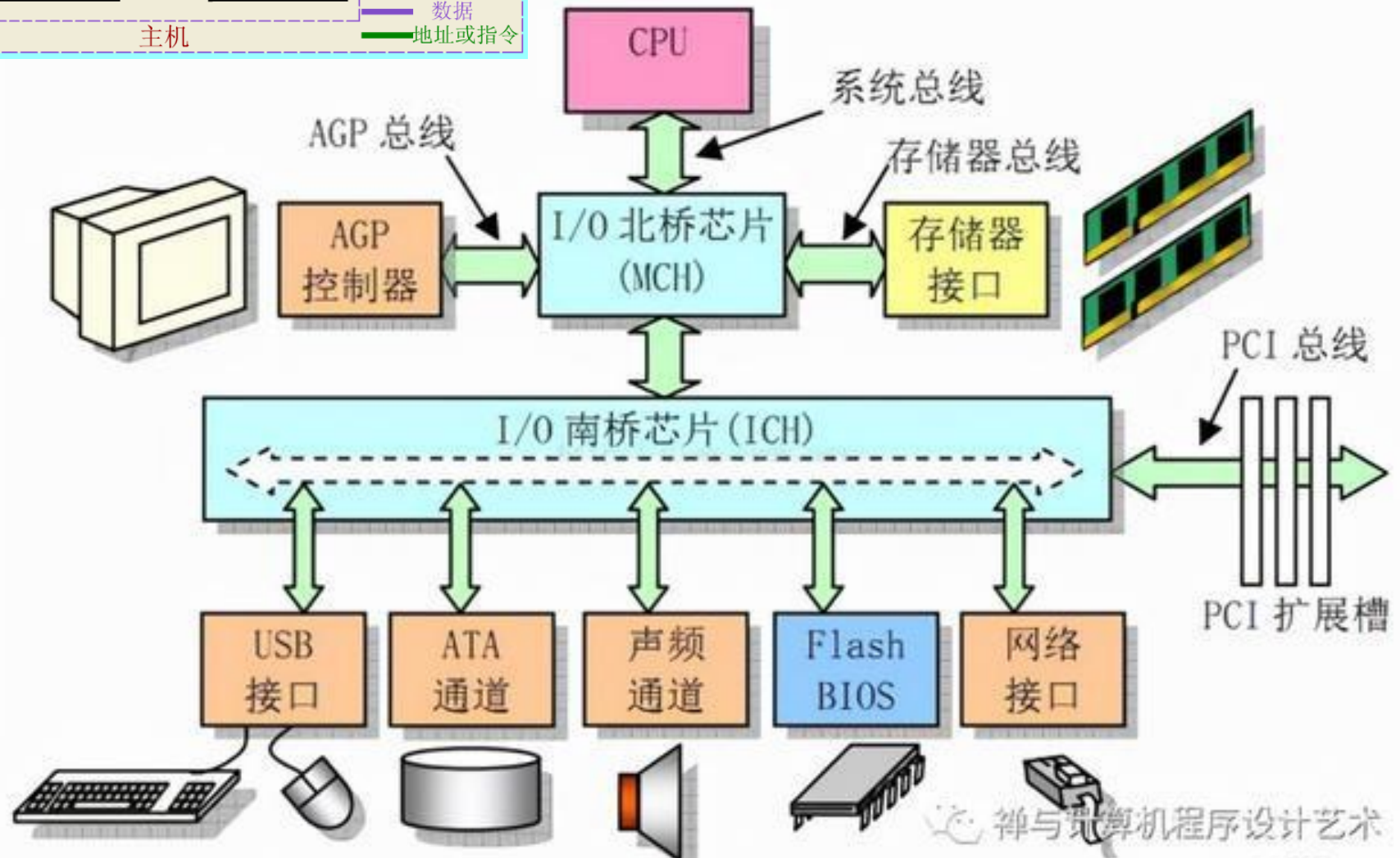



A7V600
SOCKET462
FLOPPY

主板

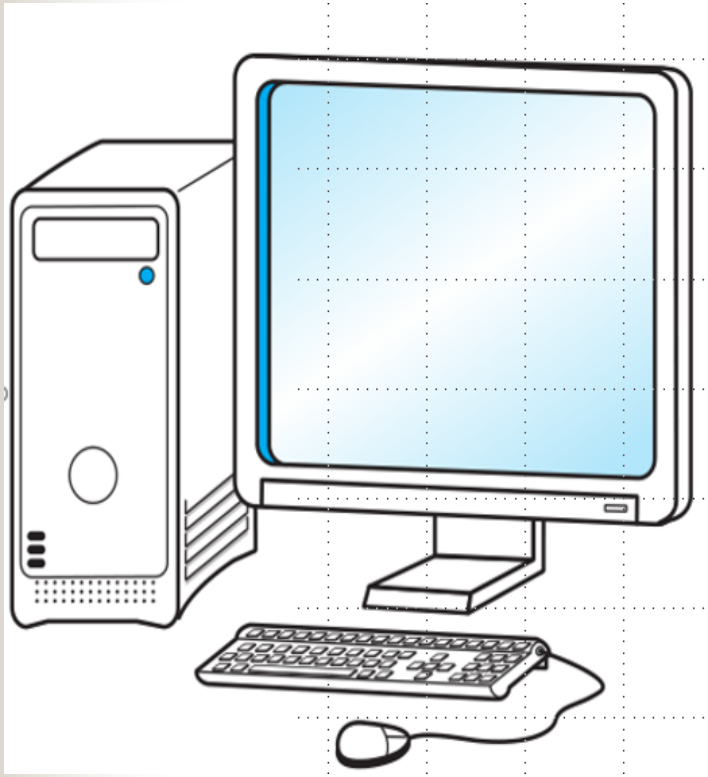






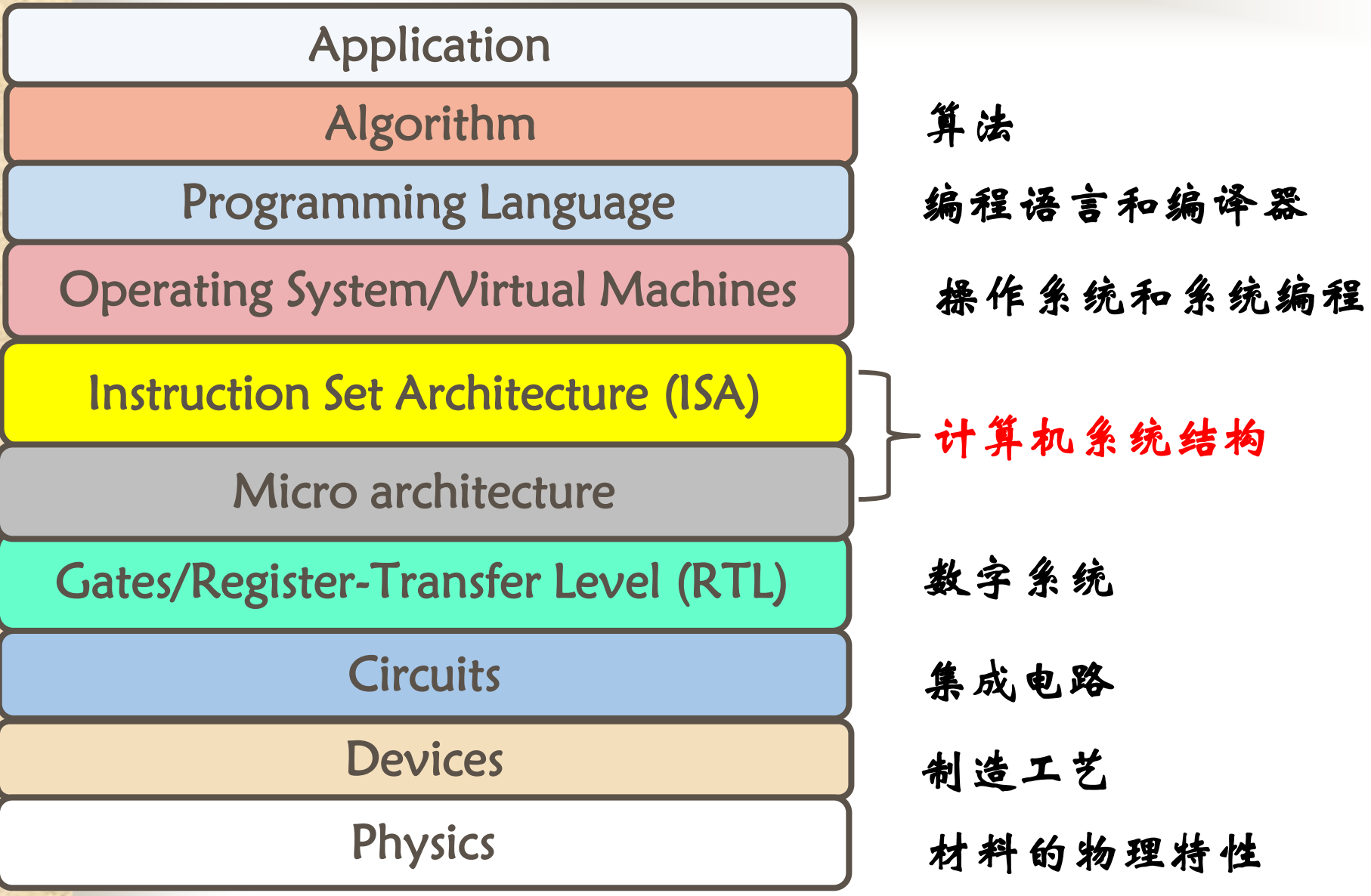
禅与计算机程序设计艺术

计算机的硬件指标

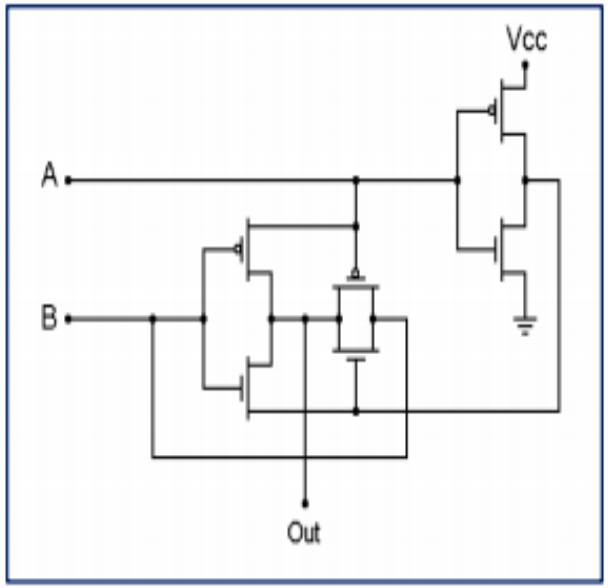


型号	Optiplex 7080MT
CPU	十代酷睿I7-10700处理器
内存	4G/8G/16G /32G DDR4内存
硬盘	2个M.2+1个3.5/2.5英寸
显卡	集显/2G/ 4G/5G/6G/RTX3070独显
系统	Windows 10家庭版
端口	1个PCI-32/1个PCIe x16/1个PCIe x1/1个PCIe 3.0x1

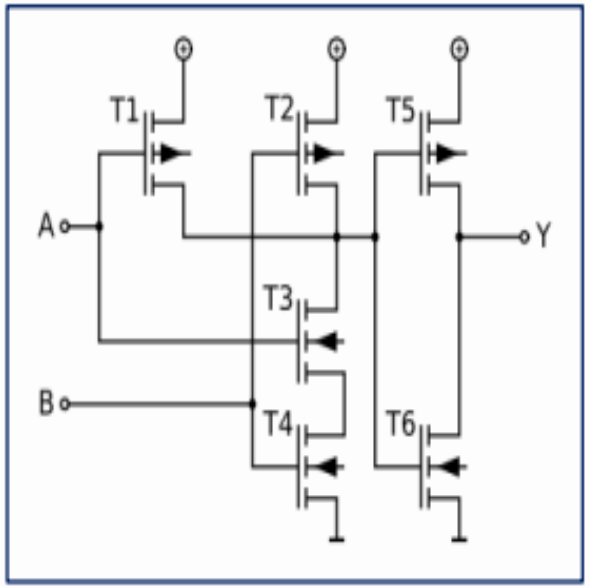
计算机系统结构


Application
Algorithm
Programming Language
Operating System/Virtual Machines
Instruction Set Architecture (ISA)
Micro architecture
Gates/Register-Transfer Level (RTL)
Circuits
Devices
Physics
算法
编程语言和编译器
操作系统和系统编程
计算机系统结构
数字系统
集成电路
制造工艺
材料的物理特性

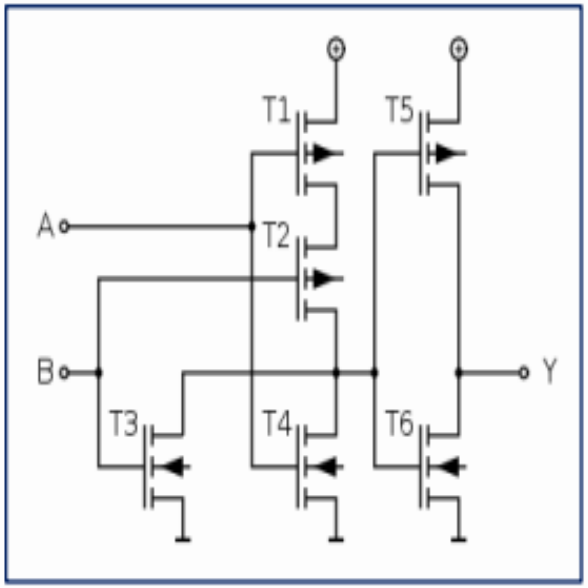
布尔代数与逻辑电路



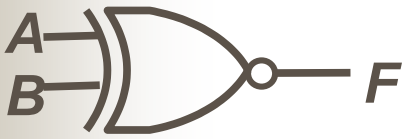
异或门



与门

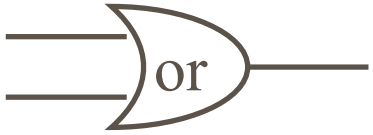


或门



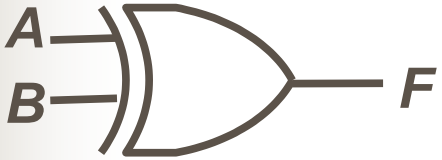
同或门

非门

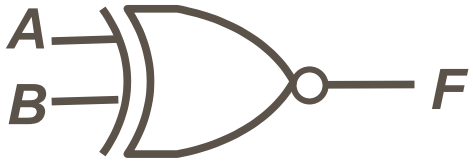


布尔代数与逻辑电路

- “异或”逻辑：F＝A⊕B
  - 输入二变量相异为“1”，相同为“0”，称为“异或”。
- “同或”逻辑：F＝A⊙B
  - 输入二变量相异为“0”，相同为“1”，称为“同或”。

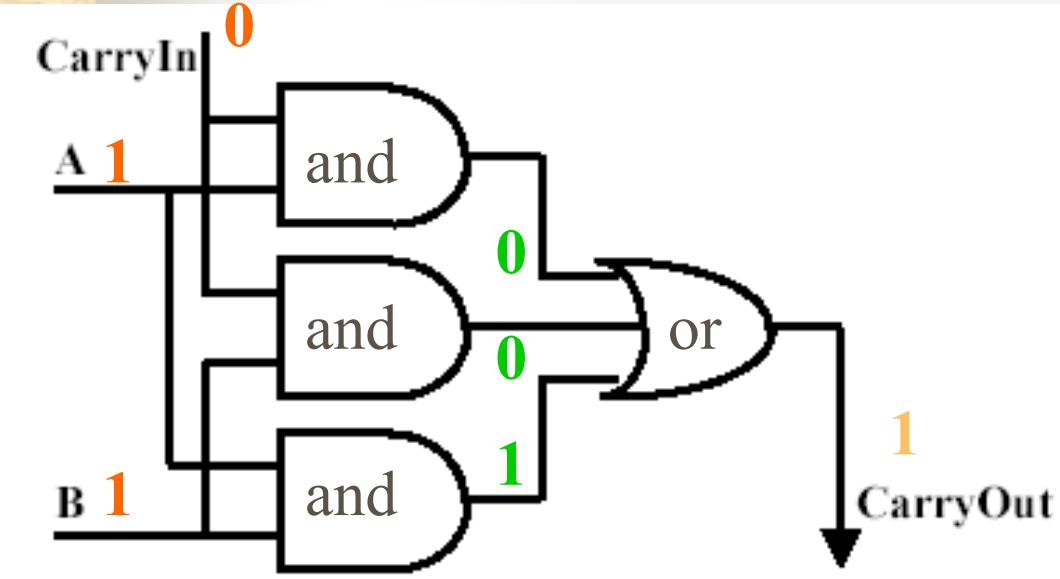


A B	F
0 0	0
0 1	1
1 0	1
1 1	0

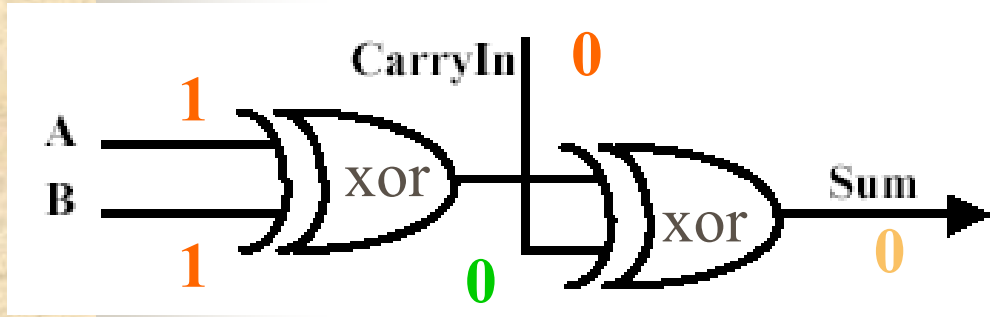


A B	F
0 0	1
0 1	0
1 0	0
1 1	1

布尔代数与逻辑电路



CarryIn	下一位进位
CarryOut	进位
A	加数
B	加数
Sum	和数



一位加法器的实现

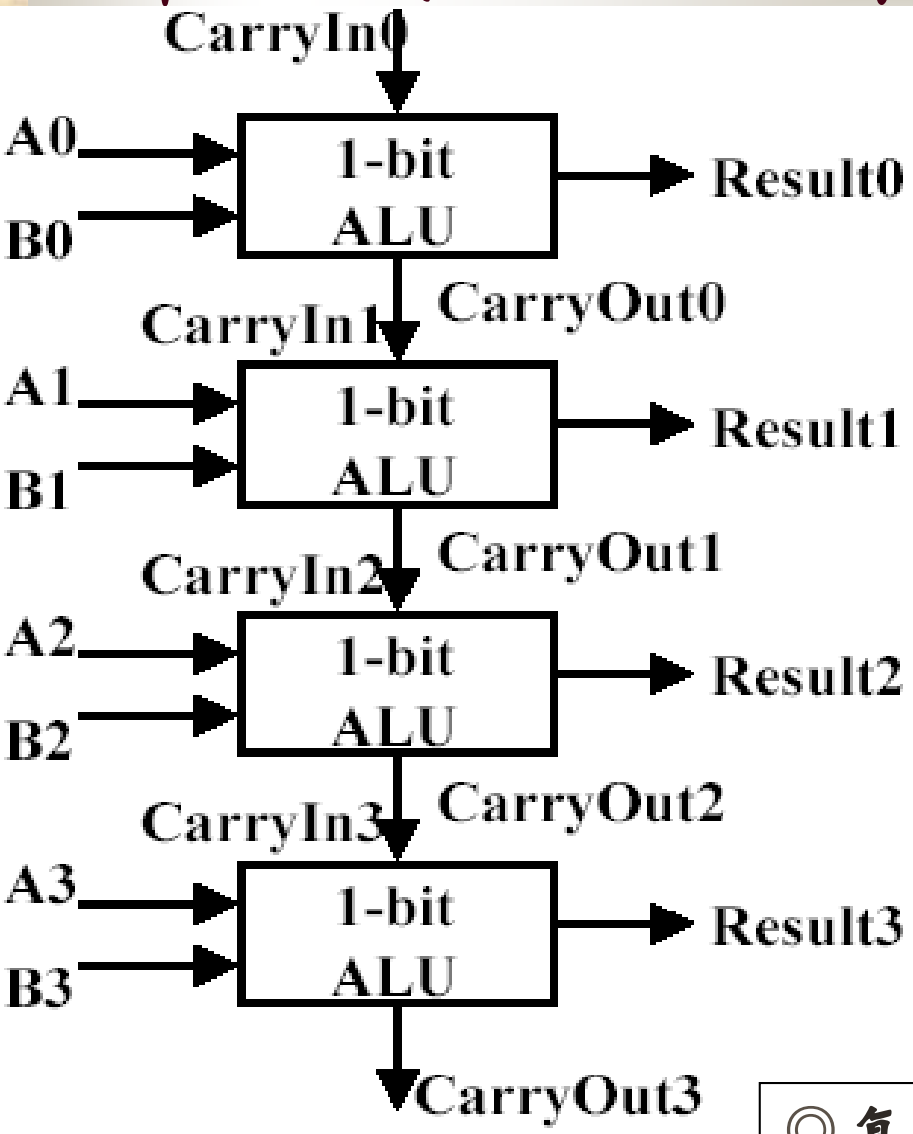
$$\begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}$$

本位加数

C :进位

S:本位和

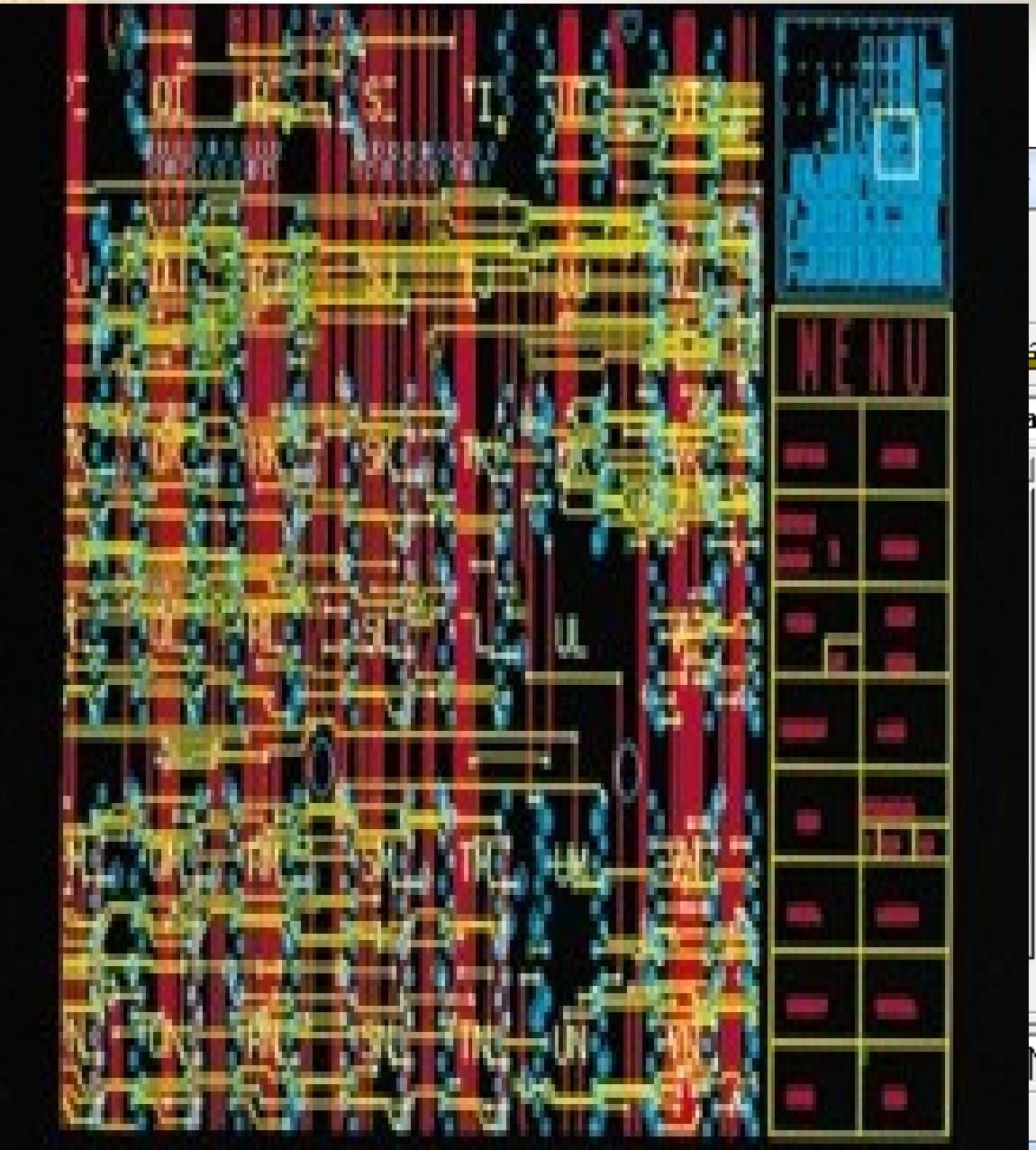
布尔代数与逻辑电路



四位全加器
从晶体管—
门电路—
运算的硬件逻辑

◎复杂的硬件由最基本的门逻辑组成

芯片内部



Microsoft Visual C++ - [main.cpp *]

Edit View Insert Project Build Tools Window

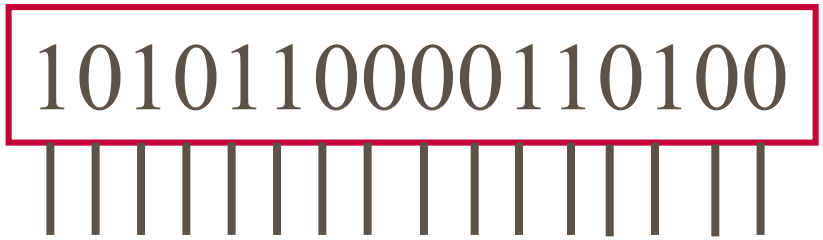
(All global members main

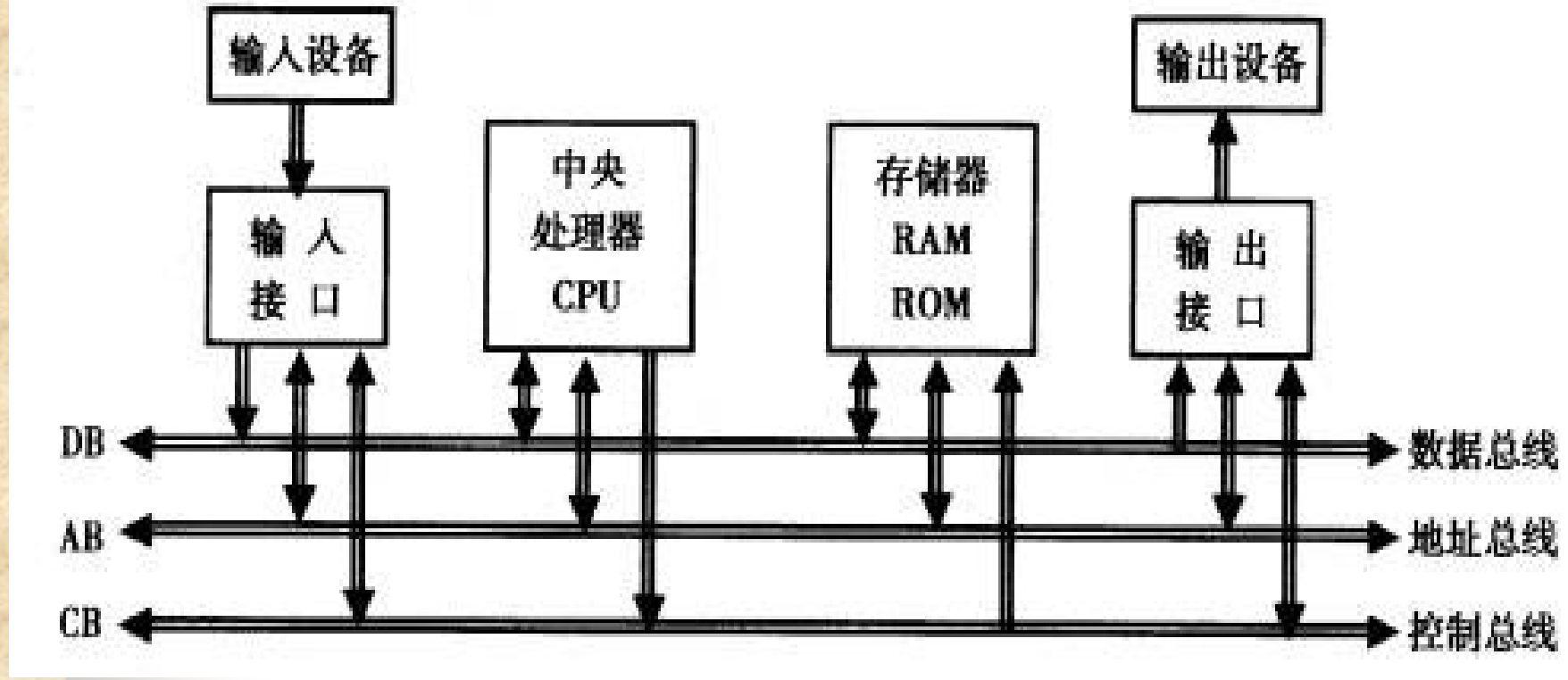
```
#include <iostream>

using namespace std;

int main()
{
    int a,b,c;
    a = 5;
    b = 4;
    c = a + b;
    cout << c << endl;

    return 0;

}
```

Build Debug Find in Files 1

总线

- 总线是计算机中数据传送的通路。
- 一共有三种总线
  - 数据总线：传输数据
  - 控制总线：传送指令
  - 地址总线：传送地址

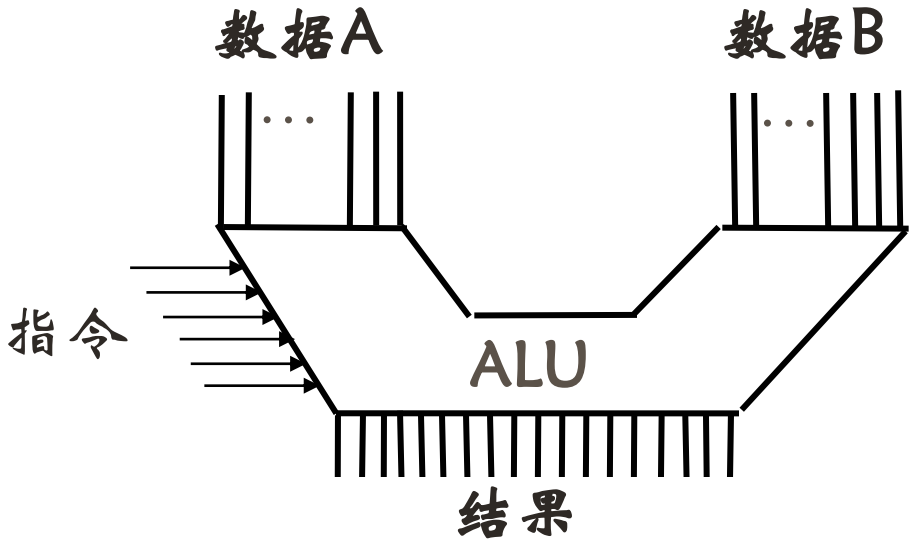




指令系统

- 计算机能执行哪些指令是由硬件决定，并且决定了指令的集合。假设：

1011000110100111是加法指令，它代表的是给做加法的硬件部件加电信号的一种命令。




```
int a = 5, b = 4, c;
c =  a + b;
cout << c << endl;
```

```
10001000000110000100000000
10001000000100000001000000
1010001000111000
```

- 无论多么复杂的应用，到了计算机硬件一层就是输入驱动硬件逻辑电路动作

指令系统

- 指令集合（指令系统）：
  - CPU所能够处理的全部指令的集合,是一个CPU的根本属性(计算机硬件能够直接执行的操作命令。)
  - 不同结构的计算机有不同的指令集合。
  - CISC---复杂指令集计算机
    - Complex Instruction Set Computer
    - 包含大量的指令以及相应的硬件逻辑。
    - RISC---精简指令集计算机
    - Reduced Instruction Set Computer
    - 只包含最基本的功能，其它复杂的功能由简单的执行组合而成。

指令系统

- X86是CISC指令系统
  - Intel和AMD的产品是X86系统
- ARM和POWER是RISC指令系统
  - ARM公司设计指令系统和处理器ip核，不出产芯片。
  - APPLE M1也是ARM指令系统。
  - IBM的POWER，基于开源的RISC-V。

机器语言

编译程序（翻译）

人理解的高级语言符号化表示

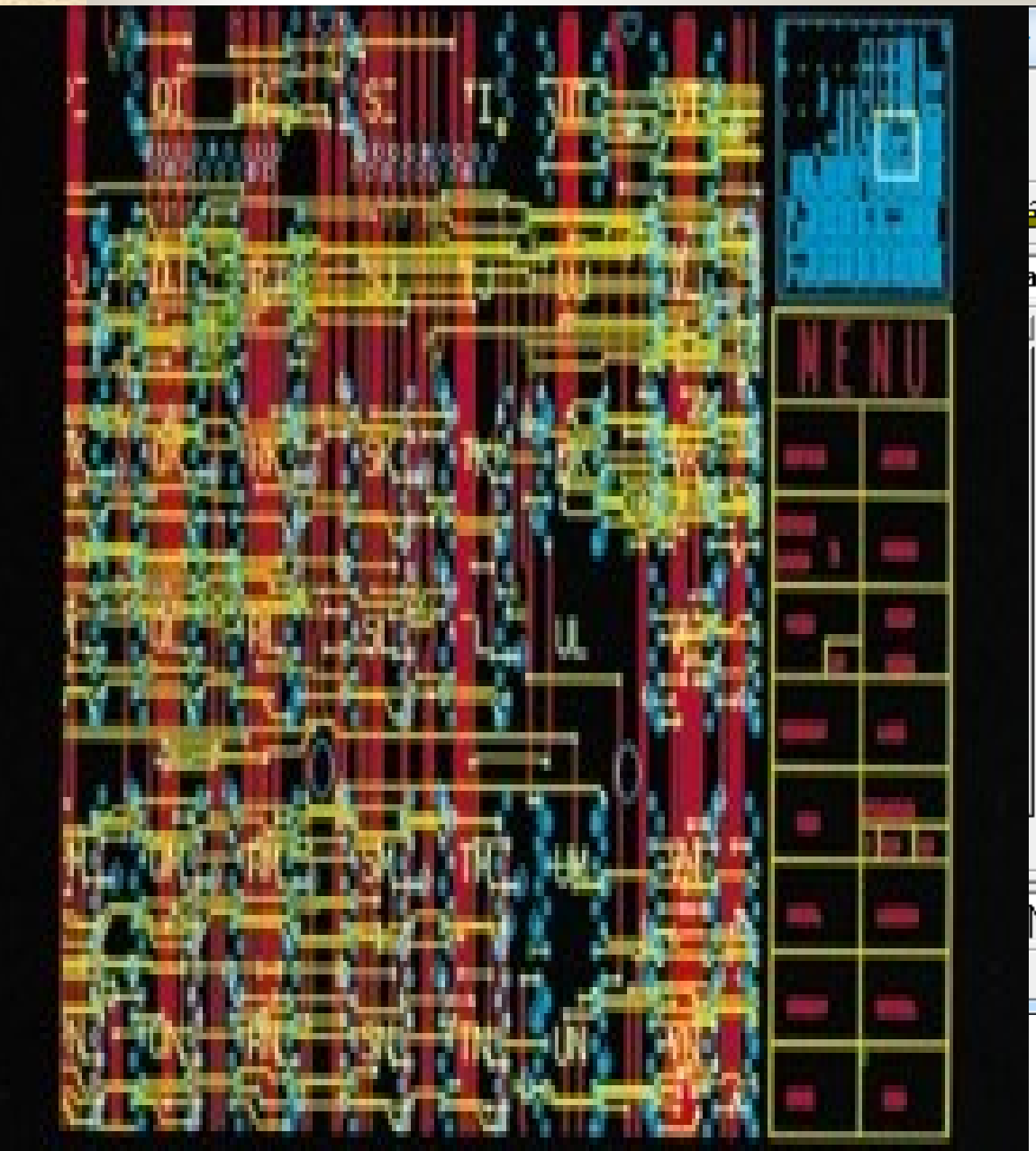
可执行程序机器语言
01代码
10001000000110000100000000
10001000000100000001000000
10100010001110000

```
int a = 5, b = 4, c;
c =  a + b;
cout << c << endl;
```

- ARM系统的机器语言。
- AMD系统的机器语言。
- Intel系统的机器语言。
- 苹果系统的机器语言。

可执行程序机器语言01代码，在不统的因概念系统下是不同的。

芯片内部



Microsoft Visual C++ - [main.cpp *]

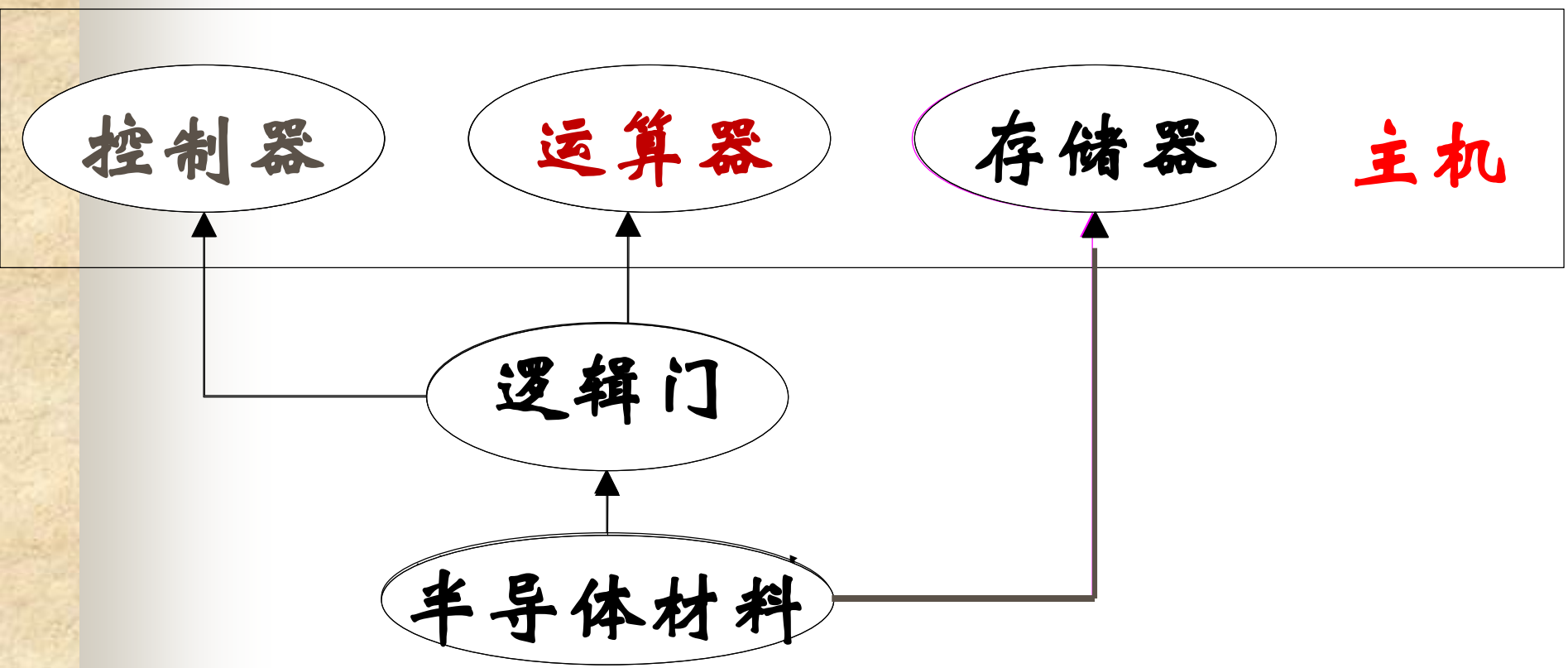
Edit View Insert Project Build Tools Window

(All global members main

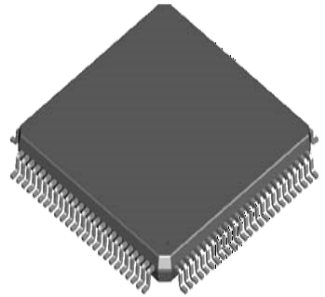
```
#include <iostream>

using namespace std;

int main()
{
    int a,b,c;
    a = 5;
    b = 4;
    c = a + b;
    cout << c << endl;

    return 0;

}
```

Build Debug Find in Files 1

知识结构



中央处理器之运算器



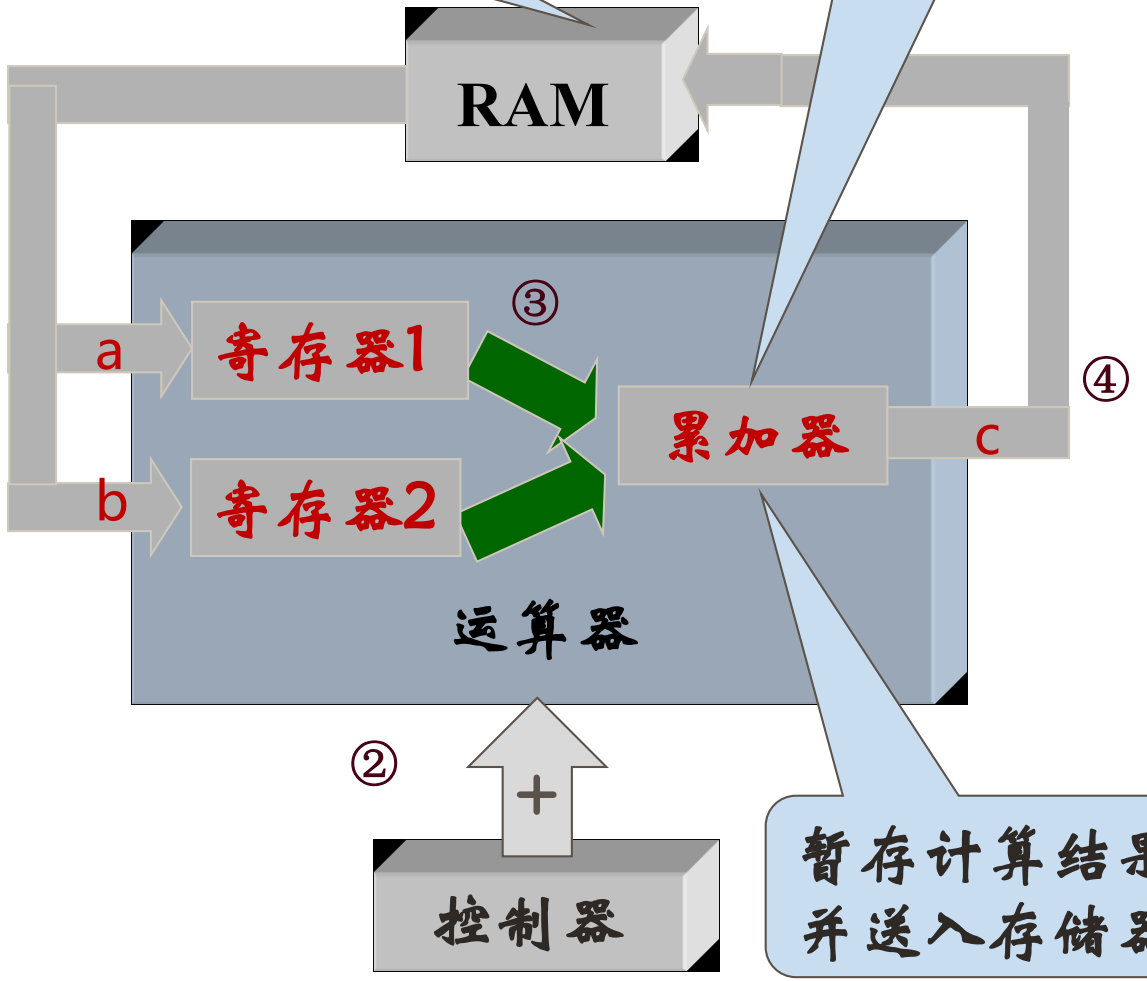
- 中央处理单元(CPU)的结构
  - Central Processing Unit
  - CPU由运算器和控制器组成，它们被集成在一个芯片上，也称微处理器.
  - 运算器(ALU)
    - 对各种数据或信息进行算术运算或逻辑运算的主要部件。
    - 运算器包含寄存器（暂存等待处理的数据）和累加器（暂存计算结果）。

中央处理单元—运算器

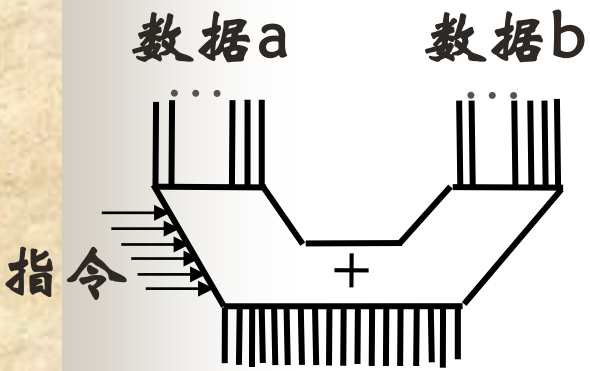
- 运算器的工作流程

主存储器，存放计算数据和程序

寄存器用于暂存待处理数据

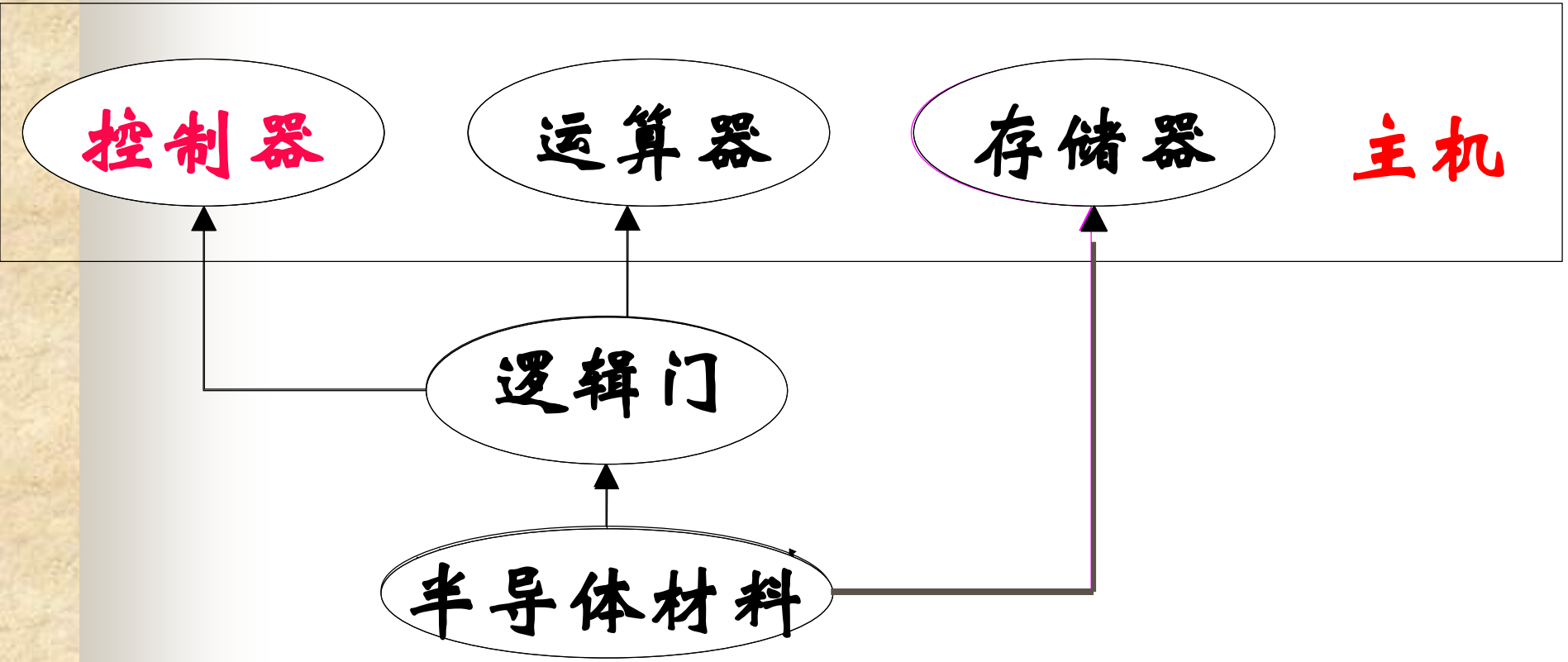


暂存计算结果，并送入存储器




```
int a, b, c;
c= a + b;
```

知识结构



中央处理单元--CPU

- 控制器
  - 进程：
    - 正在执行处理数据的活跃的指令系列及其相关数据，或者理解为程序的一次执行。
  - 指令周期
    - 取指令、分析指令，执行指令。
    - 指令计数器加1，得到下一条指令的地址

指令计数器

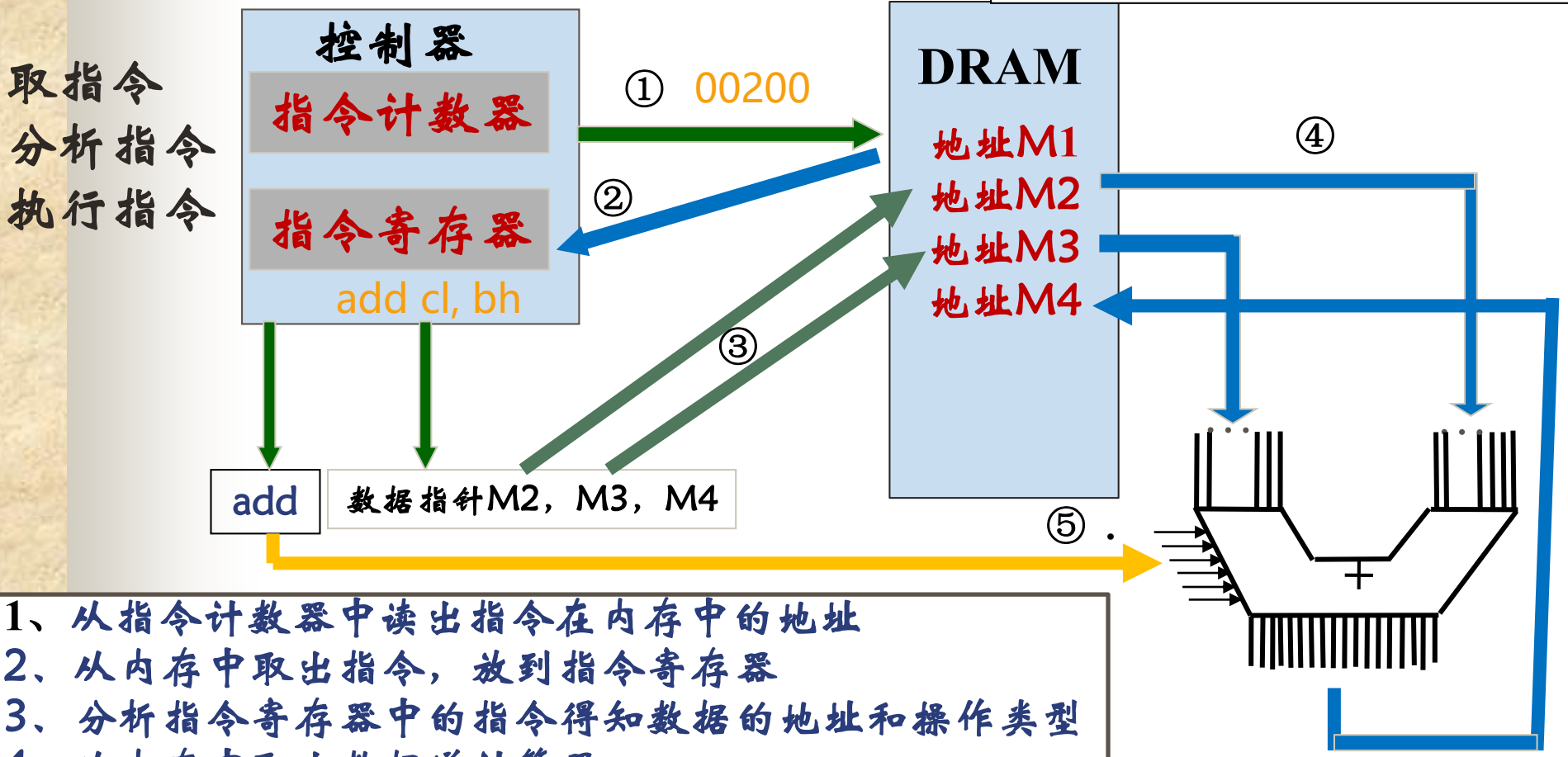
00200

00200 00000010 11001111 (add cl, bh)
00202 00000010 11111001 (add bh, cl)
00204 00000001 10010001 01000101 00100011
(add disp[bx][di])
00208 ……,

中央处理单元—控制器

- 控制器对其它部分进行协调控制，对输入输出设备的运行进行监控.

00200 00000010 11001111 (add cl, bh)
00202 00000010 11111001 (add bh, cl)
00204 00000001 10010001 01000101 0
(add disp[bx][di])
00208 ....., c = a + b;



1、从指令计数器中读出指令在内存中的地址
2、从内存中取出指令，放到指令寄存器
3、分析指令寄存器中的指令得知数据的地址和操作类型
4、从内存中取出数据送计算器
5、送加法命令给运算器

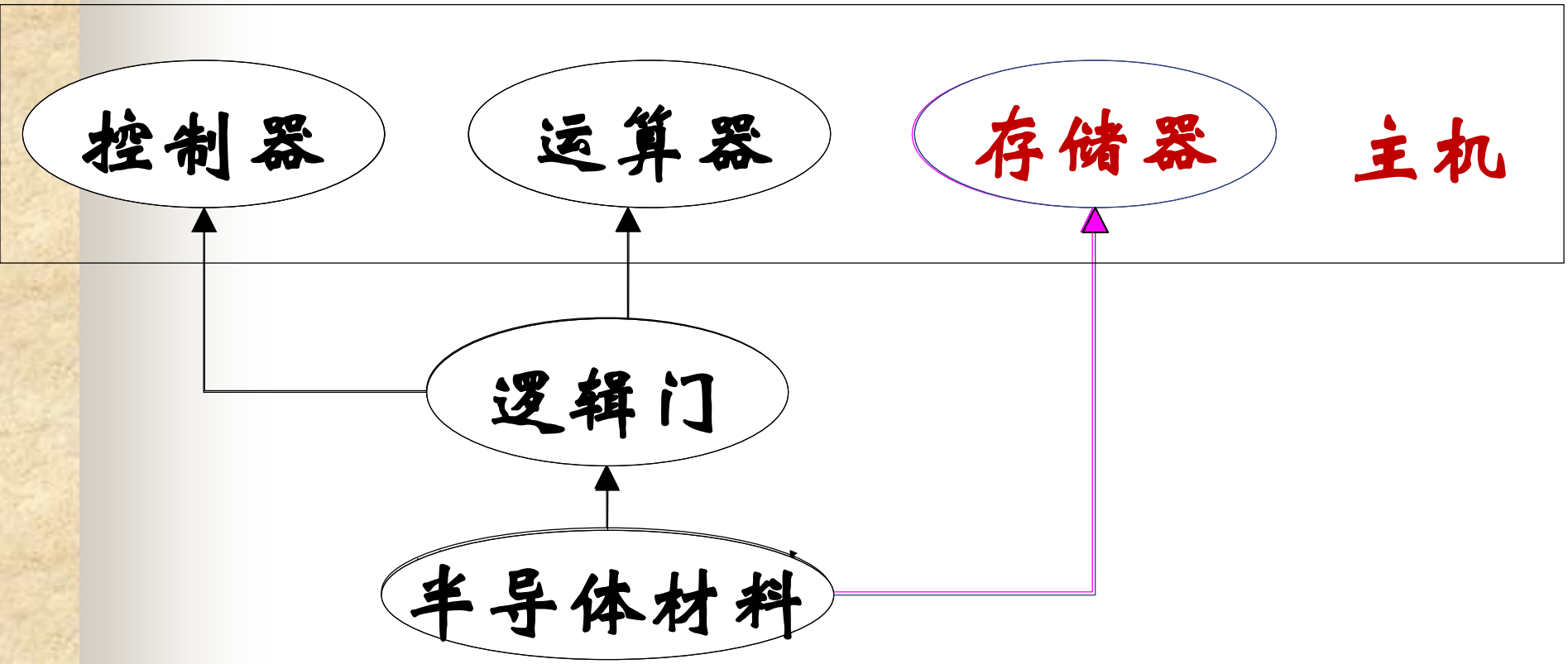
中央处理单元——控制器

- 控制器
  - 字长：是指中央处理器可以同时处理多少位数据。一般指寄存器的长度，总线的宽度。
  - 字长的演变：
    - 4bit→8bit→16bit→32bit→64bit。
  - 字长越长，能表示更多的存储单元的地址，能计算更大的数据。

中央处理单元——控制器

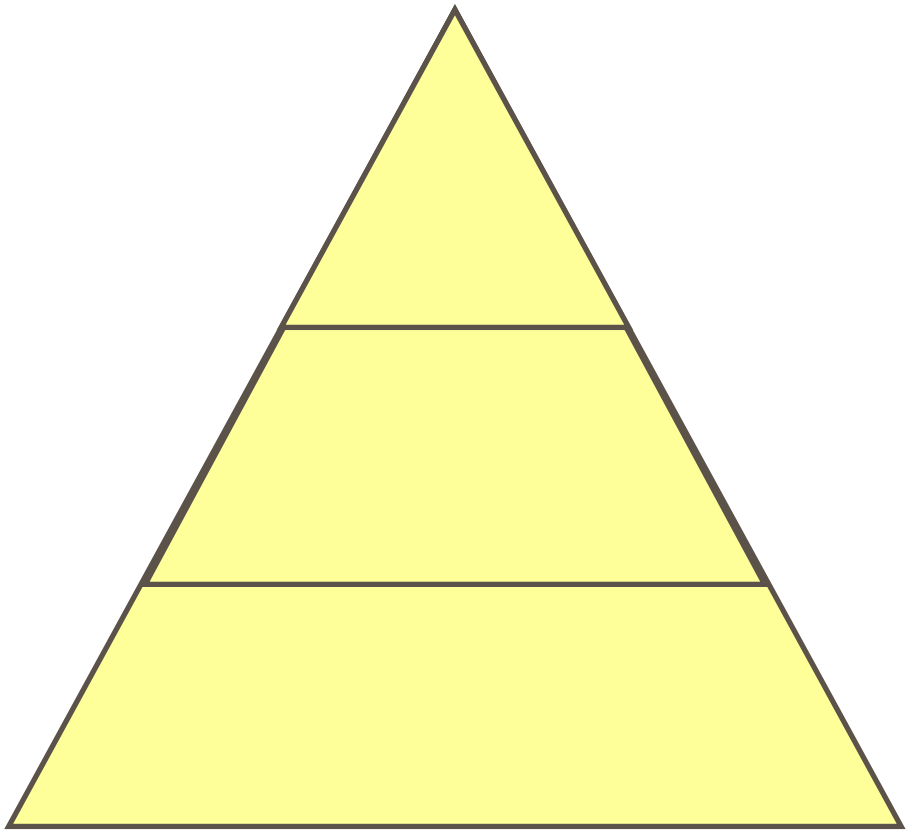
- 影响处理器运算效率的三个因素
  - 字长：
    - 字长不够时，需要靠多次执行来完成
  - 频率
    - 主频越快,CPU执行指令的节拍越快.
  - cache
    - 减少访问内存的次数，提高速度

计算机硬件系统

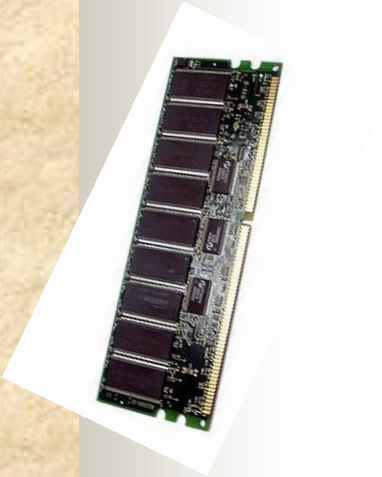


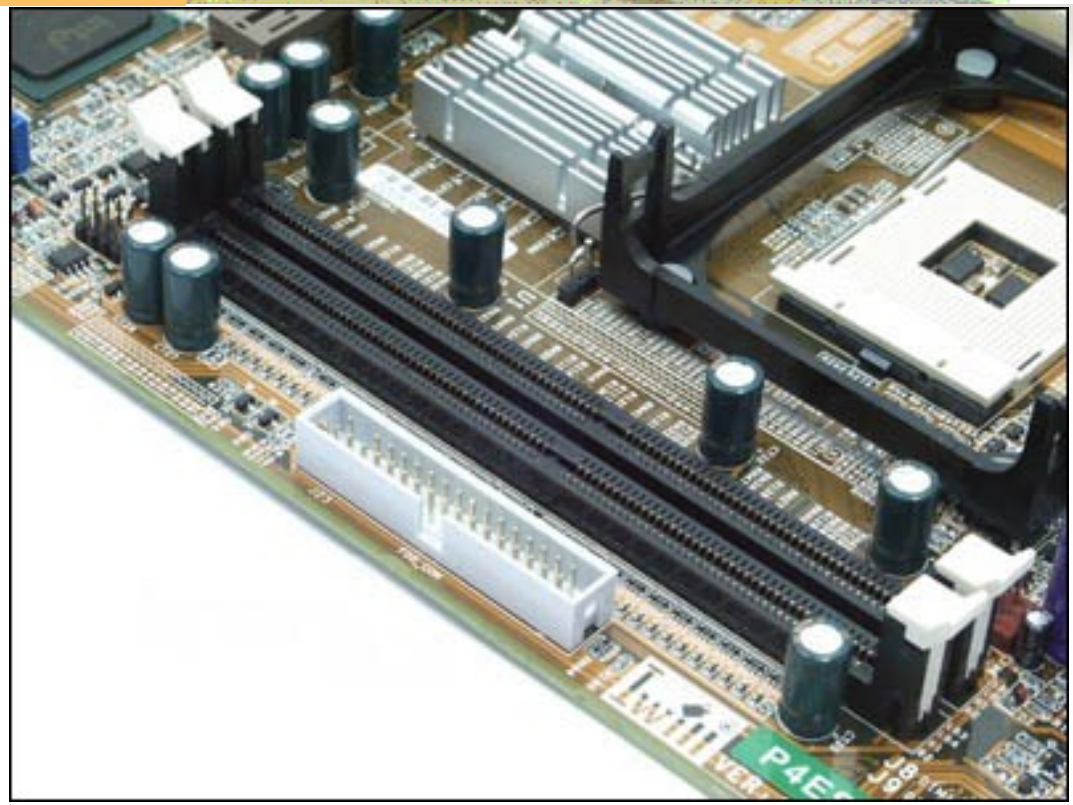
存储的层次结构

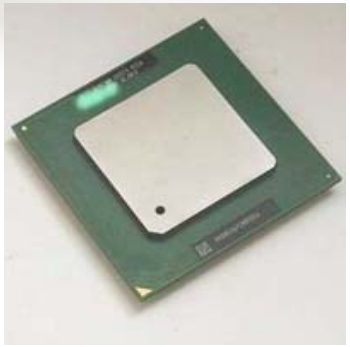
- 寄存器
- cache
- 主存DRAM

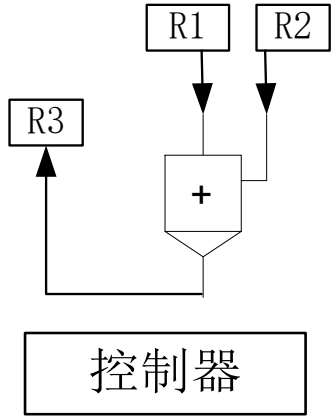


高速缓存cache（1）









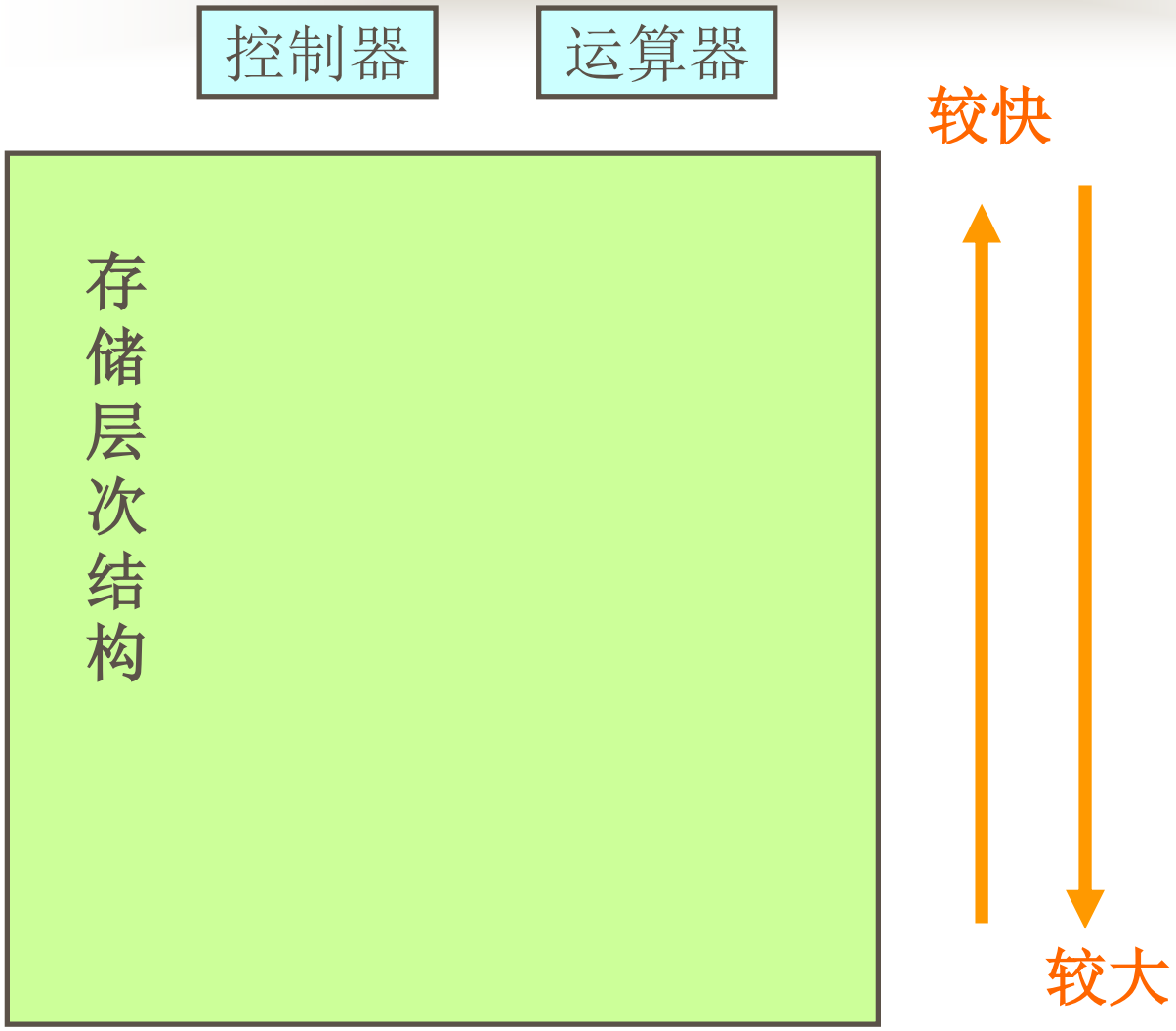
1000111011100 1001011101100 1000100101110 0010000111011 0010111101110 1101101111110
a b c

800MHz

?等待

3GHz

高速缓存cache

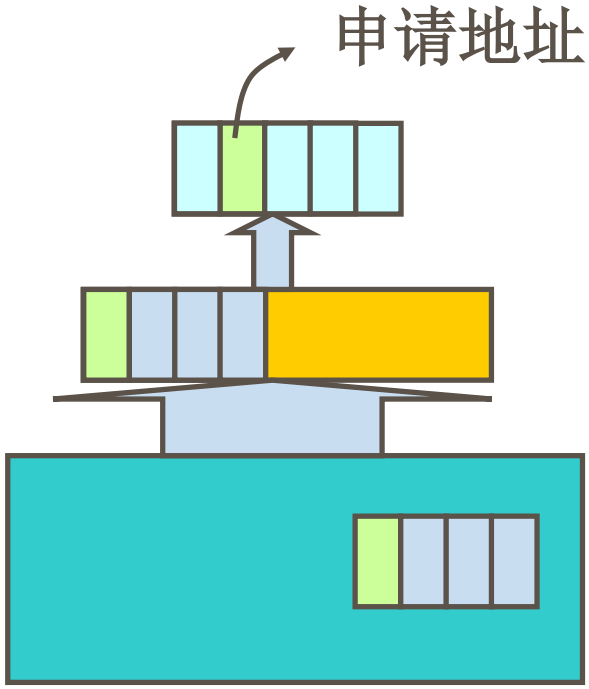


高速缓存cache

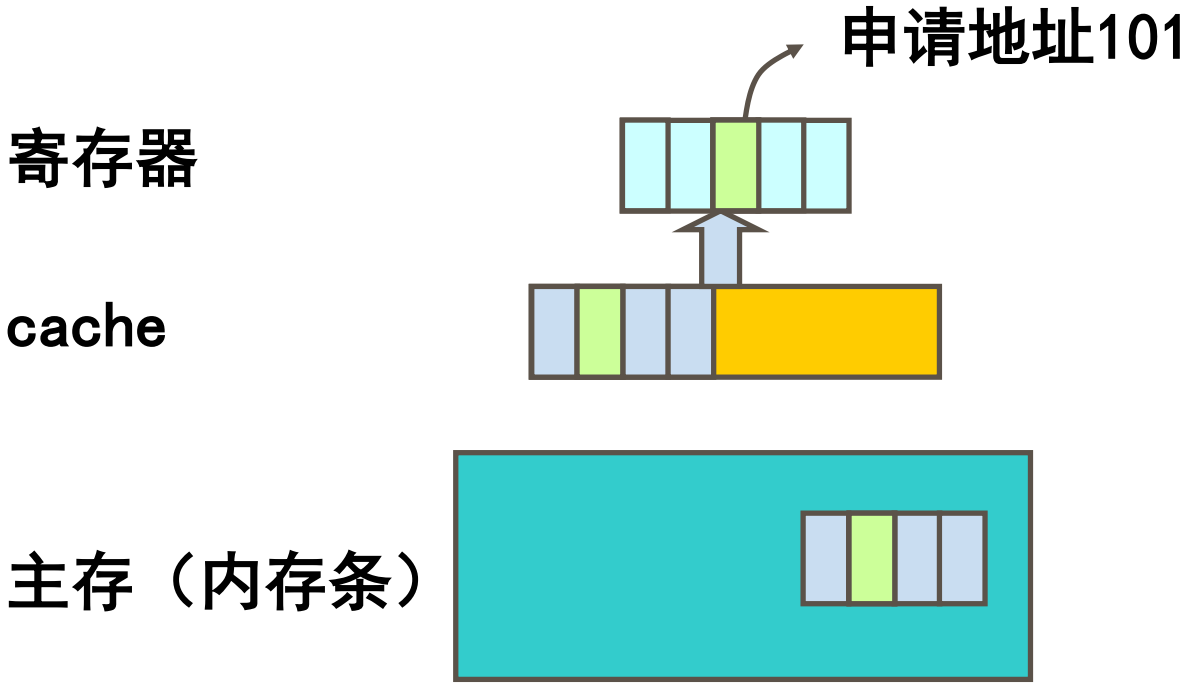
寄存器(CPU)

cache

主存（内存条）



高速缓存cache



cache也可以有多级，1级、2级至多级cache

高速缓存cache

- 程序的局部性原理
  - 程序在一定时间段内通常只会访问一个局部内存空间
  - 时间局部性
    - 如果一个内存地址正在被使用，那么在近期他很可能还会再次被访问
  - 空间局部性
    - 在最近的将来可能用到的信息很可能是与当前使用的信息相邻的

高速缓存cache

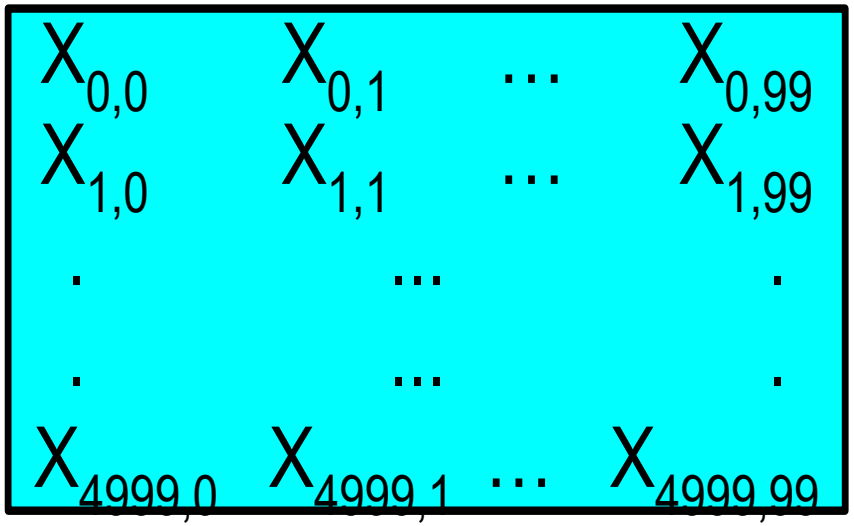
- 缓存命中率
  - CPU向cache读数据时，如果所需要的数据在cache中，则成为命中。
  - 如果没有命中，需要到内存中读取新的数据，替换掉cache中的一部分内容。
- 替换算法
  - 读取新的数据，需要一定的策略，替换掉老的数据，叫做替换算法。
  - FIFO(First Input First Output),先进先出。
  - LFU (least Frequently Used)最不经常使用。
  - 是看一定时间段内页面被使用的频率
  - LRU(Least Recently Used)最近最少使用算法
  - LRU是页面最后一次被使用到发生调度的时间长短。

一个简单编程问题

- 请将给定矩阵X[5000][100]的每个元素乘2

主存

（内存条）




```
for(j = 0; j < 100; j = j ++)
    for(i = 0; i < 5000; i = i ++)
        x[i][j] = 2 * x[i][j]
```

```
for(i = 0; i < 5000; i ++)
    for(j = 0; j < 100; j ++)
        x[i][j] = 2 * x[i][j]
```

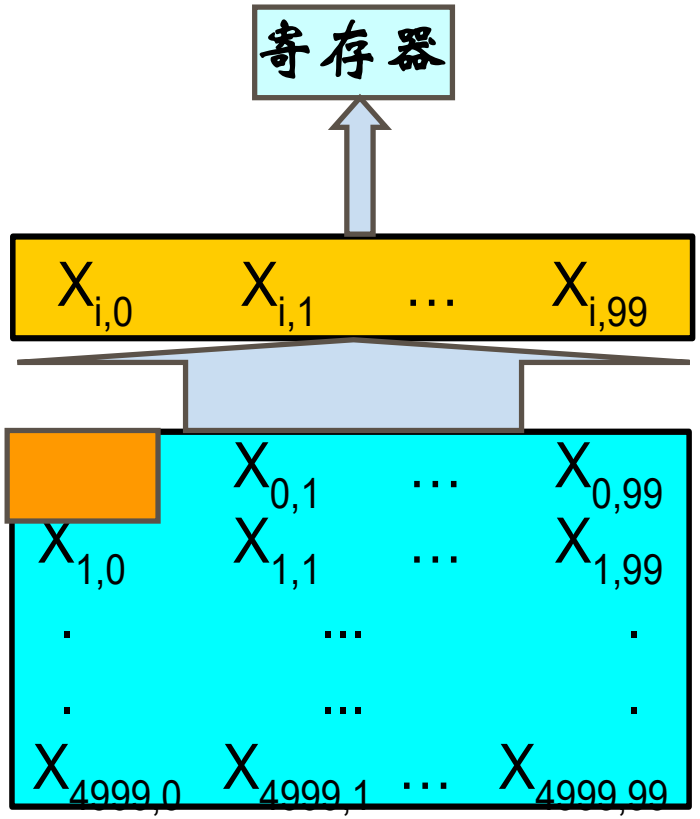
有什么区别？

一个简单编程问题（2）

假设取数据要10个周期，数据在Cache中，则3个周期能够完成乘法

```
for(j = 0; j < 100; j = j ++)
  for(i = 0; i < 5000; i = i ++)
    x[i][j] = 2 * x[i][j]
```

Cache



$5000 \times 100 \times (10 + 3)$

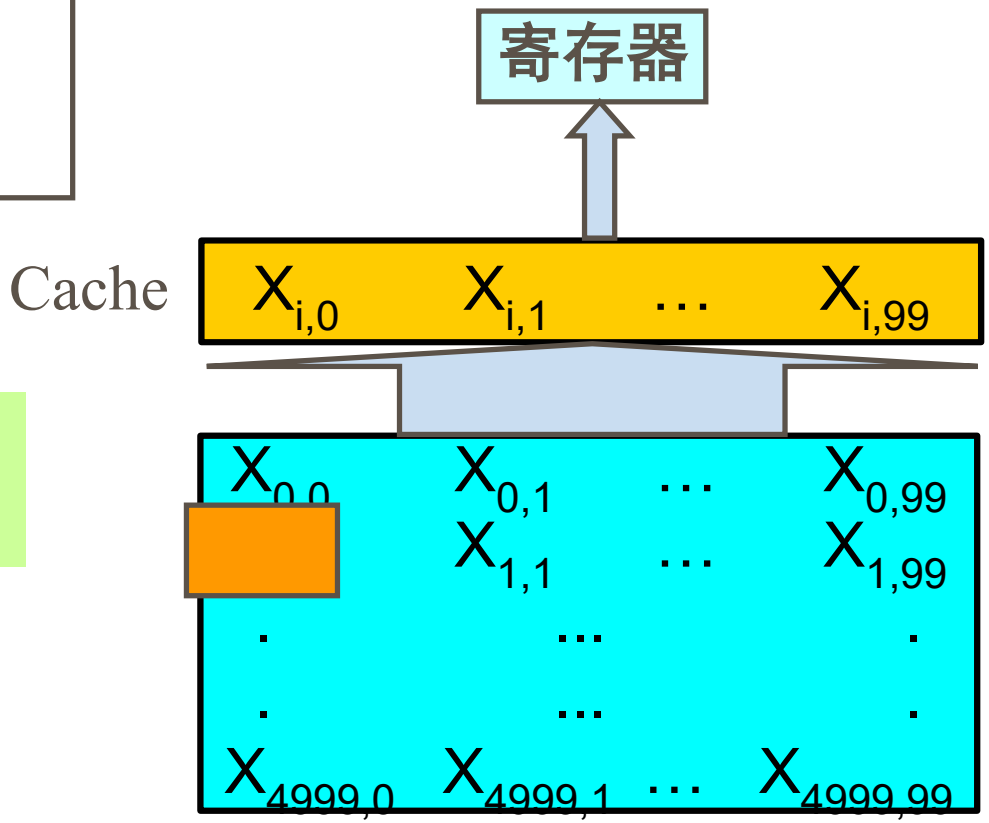
$= 6{,}500{,}000$

一个简单编程问题

```
for(i = 0; i < 5000; i ++)
    for(j = 0; j < 100; j ++)
        x[i][j] = 2 * x[i][j]
```

5000×10 + 5000×100×3
= 1,500,000

假设取数据要10个周期，数据在Cache中，则3个周期能够完成乘法



主存储器

- 存储器指存储数据或信息的存储部件。
  - 存储器分为内存储器和外存储器。
  - 直接和处理器相连，存放处理器正在处理的数据的存储器是主存储器，它是暂时性，关掉电源后，数据消失。
  - 内存大致有三种类型：
    - 动态随机存储器DRAM
    - 静态随机存储器SRAM
    - 只读存储器ROM

主存

- DRAM （Dynamic Random Access Memory），
  - 随机指的是访问时间与地址无关。
  - 主要元器件是电容，周期性刷新充电，否则数据会消失.
  - 操作系统程序、用户的程序，数据等均在DRAM中运行，关掉电源，DRAM中内容将消失。
- SRAM(Static Random-Access Memory)
  - 静态随机存取存储器,不刷新电路即能保存数据，具有较高的性能，通常用作cache。
- SRAM的集成度较低，功耗比DRAM大,相同容量的DRAM可以设计为较小的体积，而SRAM则需要很大的体积。等面积的硅片可以做出更大容量的DRAM，因此SRAM更贵。

存储器

地址	内容
0000000000000000	00000000
0000000000000001	00000000
0000000000000002	00000000
0000000000000003	00000001
	XXXXXXXX
	XXXXXXXX
0000000073642504	00110011
0000000073642505	11100011
0000000073642506	01100100
0000000073642507	00111011
	XXXXXXXX
	XXXXXXXX
	XXXXXXXX
FFFFFFFFFFFFFFFF	XXXXXXXX

- 内存单元以字节为单位，64位机器的第一个单元的地址是
- 0000000000000000
- 最后一个单元的地址是FFFFFFFFFFFFFFFF
- 每个变量对应到一个地址，从这个地址开始放数据。
- 并不是一个单元存放一个数，不同的数据类型占用的单元数不同，因此变量都是有类型的。
- 通过内存单元的地址访问数据

存储器的性能指标

- 两个指标：容量和访问速度
  - 一般个人电脑标配是4G~32G，服务器则更大。
  - 访问速度在800MHz，1800MHZ，2500MHZ不等。
- 访问硬盘的速率
  - 笔记本硬盘5400转，50-90MB每秒
  - 台式机硬盘7200转，90-190MB每秒
  - SATA固态硬盘的读写速度可以达到500MB/s
  - PCIe x4 NVMe固态硬盘的读写速度可以达到3GB/s
- 容量直接影响到处理速度
  - 软件越来越大，
  - 对内处的要求越来越高。
  - 扩充内存

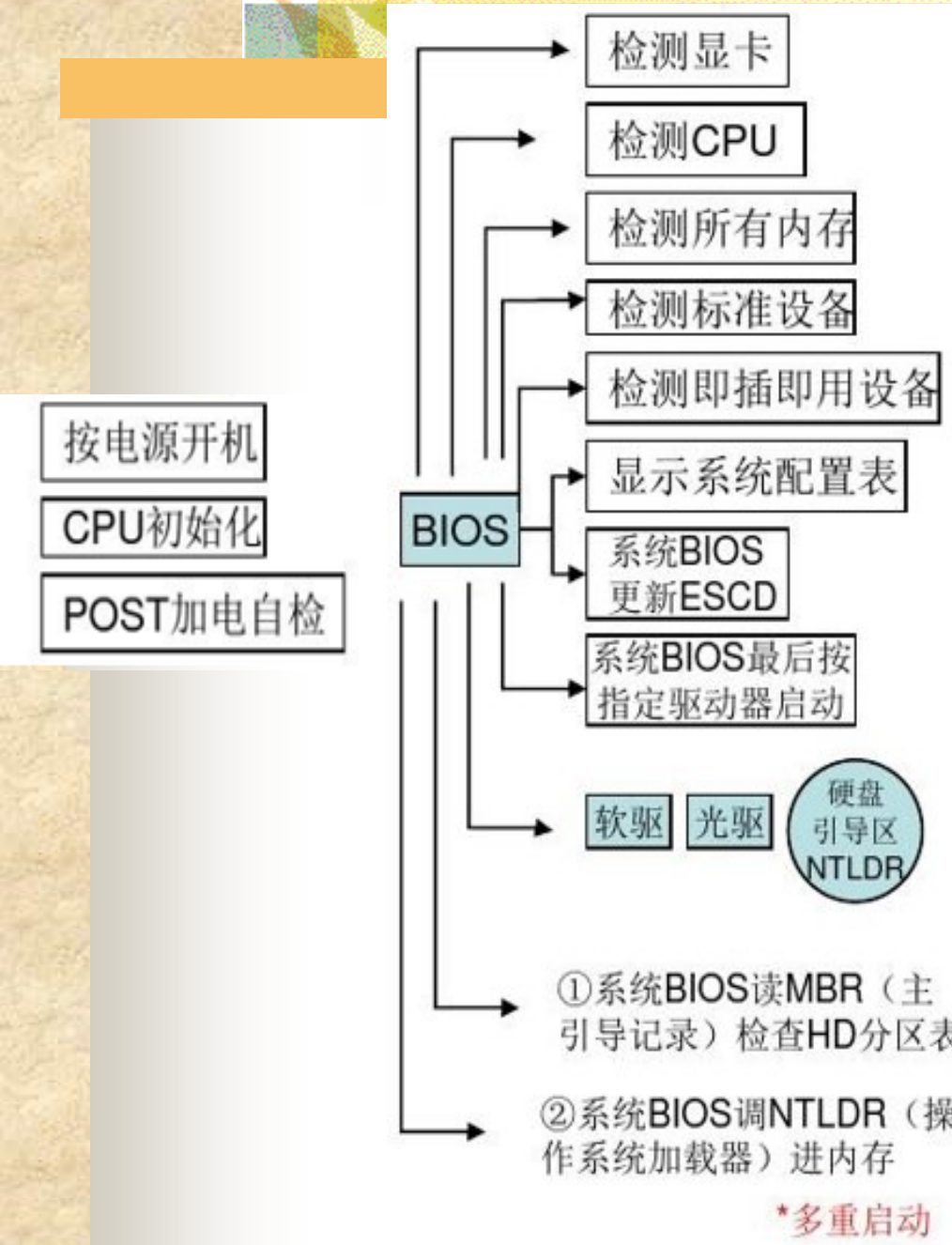
- 1bit, 1B, 1K, 1M, 1G
- 2的10次方是K
- 2的20次方是M
- 2的30次方是G
- 2的40次方是T

存储器-- ROM

- ROM（Read Only Memory）
  - ROM在出厂时，就将中断服务程序、启动程序固化在其中，因此也称固件，其中的内容是永久保存的，不能写信息。
  - 为什么需要永久保留在机器中指令？
    - 没有操作系统计算机不工作，计算机不工作不能将操作系统读如计算机---死锁。
    - ROM中的启动程序（小型的指令集合）将外存的操作系统读入内存。
  - BIOS（配置硬盘，读操作系统到内存，解压）

计算机要在操作系统指挥下才能工作，在操作系统进入内存之前谁指挥计算机工作？

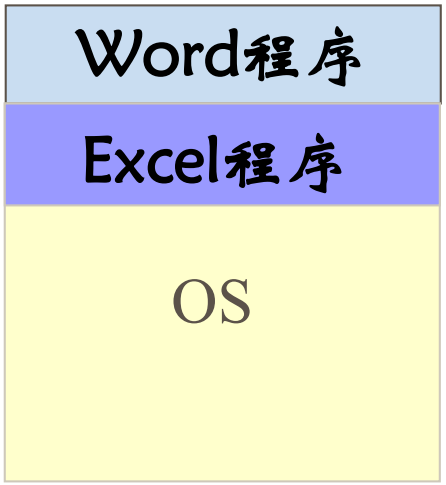
在刚刚接通电源的时候，整个系统由BIOS控制

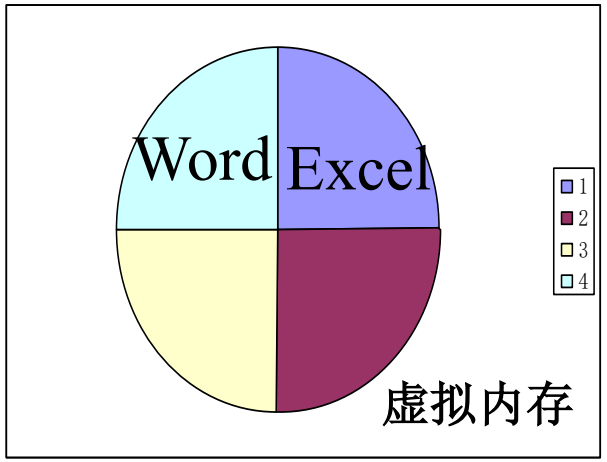


操作系统存储在外存储器中.BIOS引导操作系统加载器到硬盘的指定的位置来加载操作系统。

虚拟内存

- 虚拟内存
  - 利用计算机的硬盘来模拟内存的技术称为虚拟内存
  - 实现虚拟内存的功能是操作系统完成的。

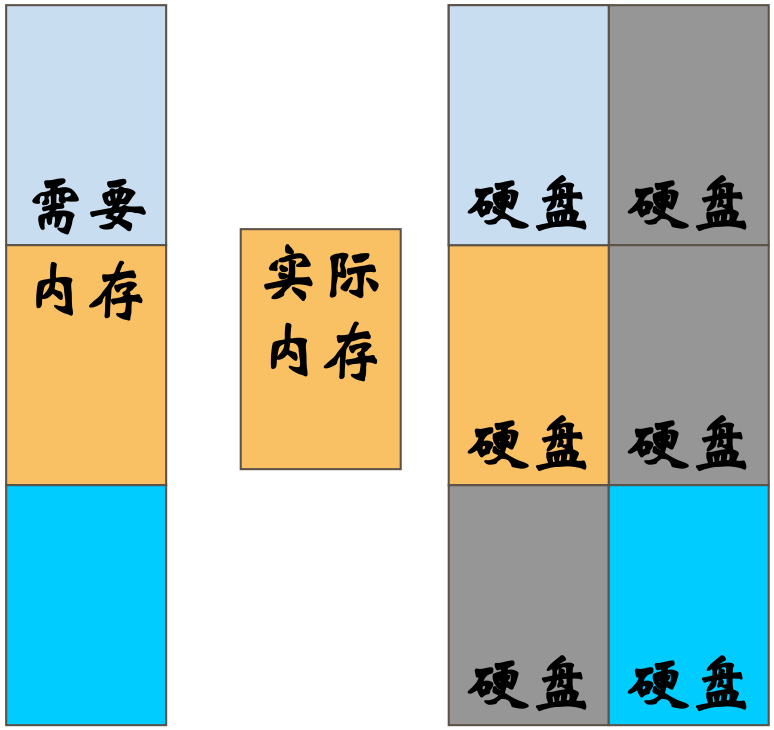




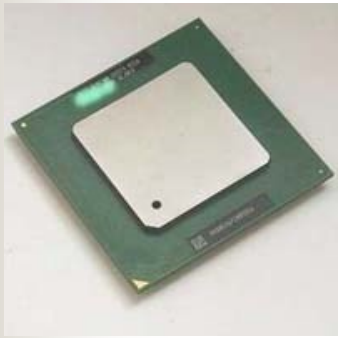
  - 虚拟内存的东西并不永久保留。

存储器--虚拟内存

- 虚拟内存 （实际是外存）
  - 利用计算机的硬盘来模拟内存的技术称为虚拟内存
  - 实现虚拟内存的功能是操作系统完成的。
  - 虚拟内存的东西并不永久保留。



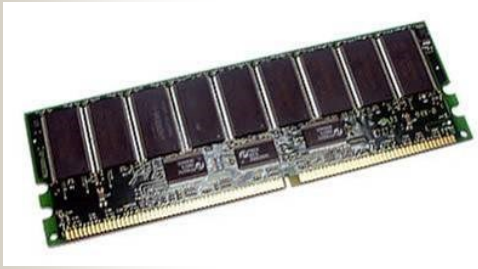
外部设备



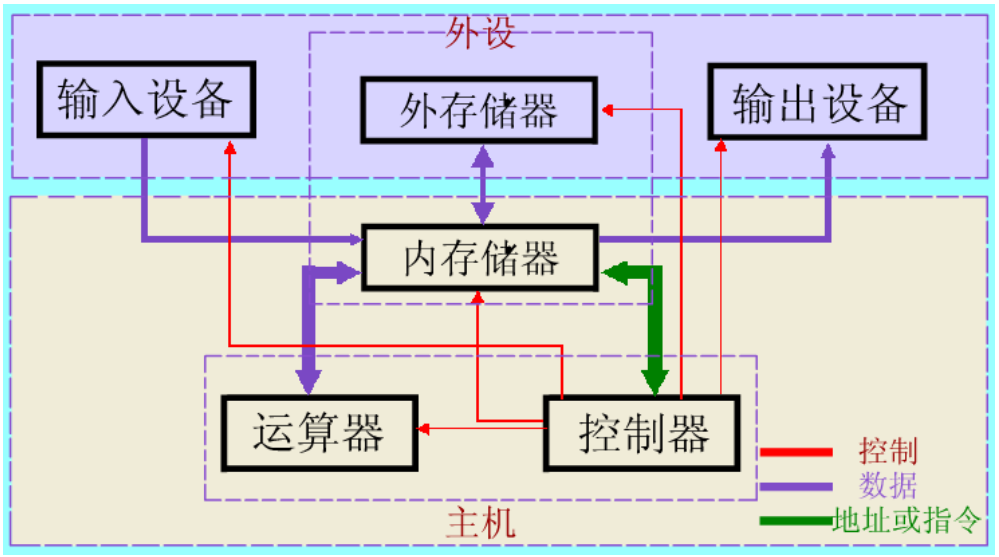
控制器

运算器

Cache

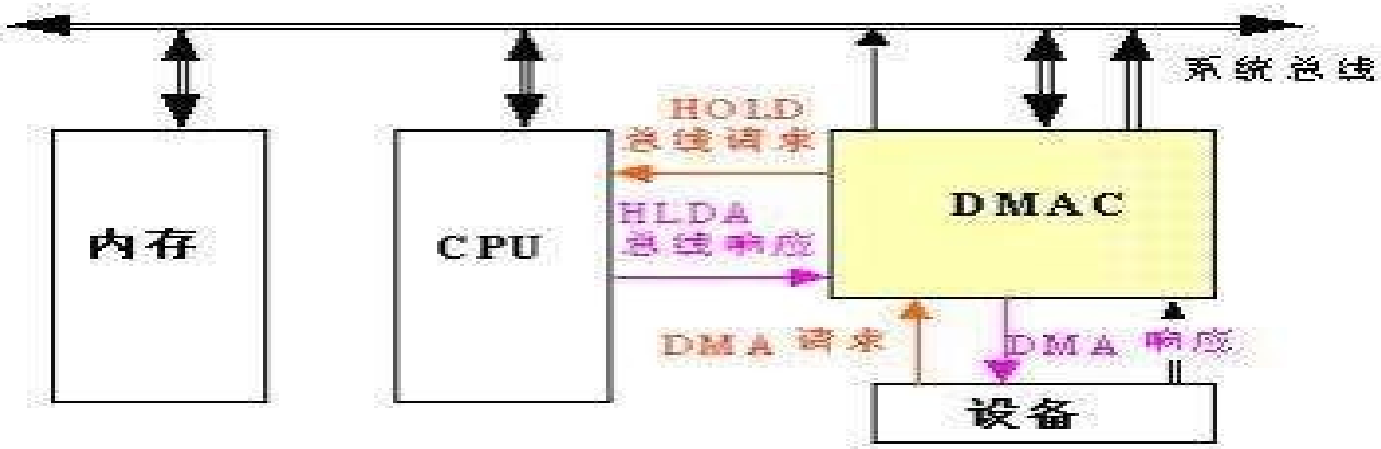


主存



外设(输入设备、输出设备)

I/O输入输出设备



1.早期的计算机采用等待方式

向外部设备发出请求，在外部设备准备期间，一直等待.

2.中断方式

向外部设备发出请求，在外部设备准备期间，干别的事情.

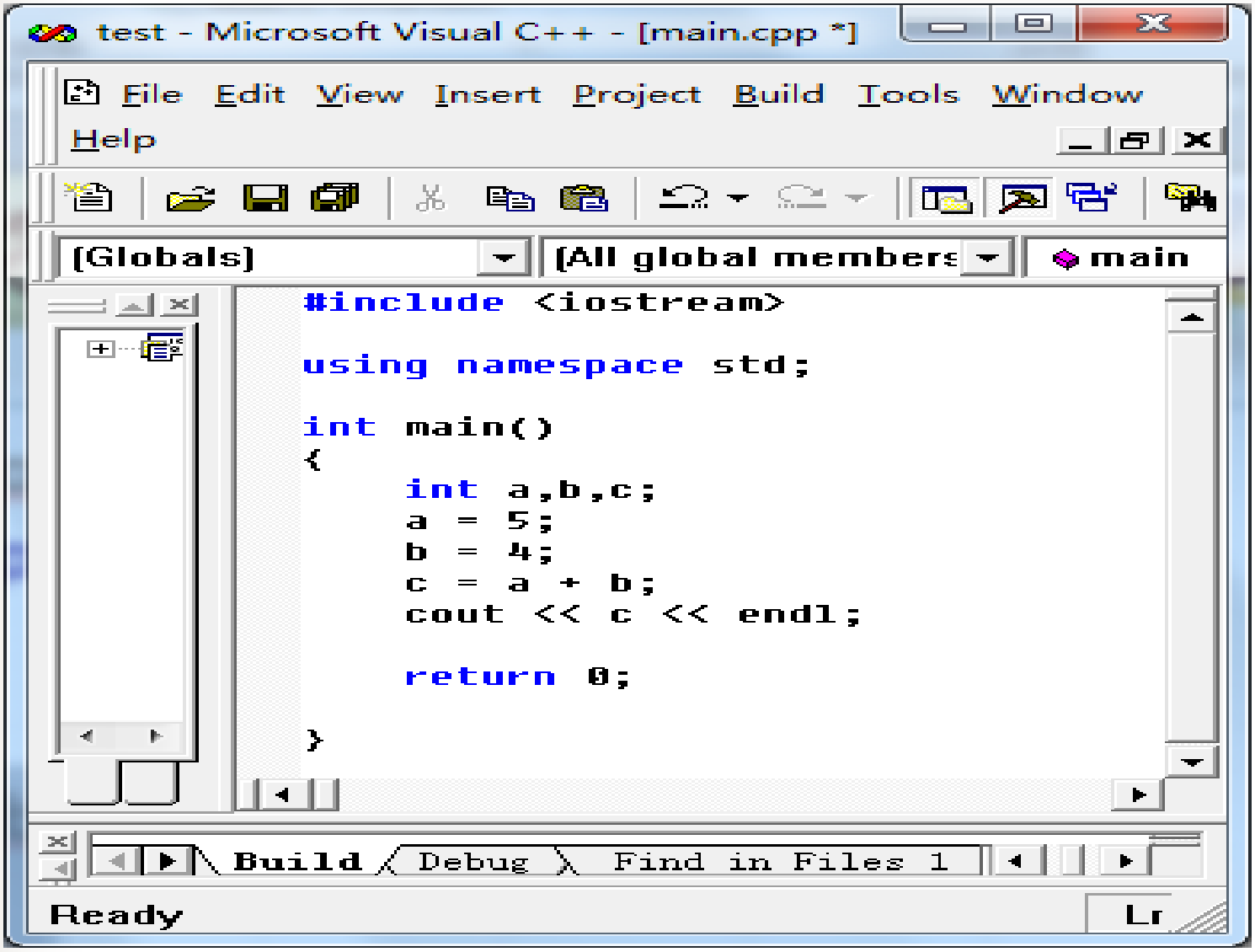
3.DMA方式

把数据发给一个直接访问控制器，CPU解放出来。

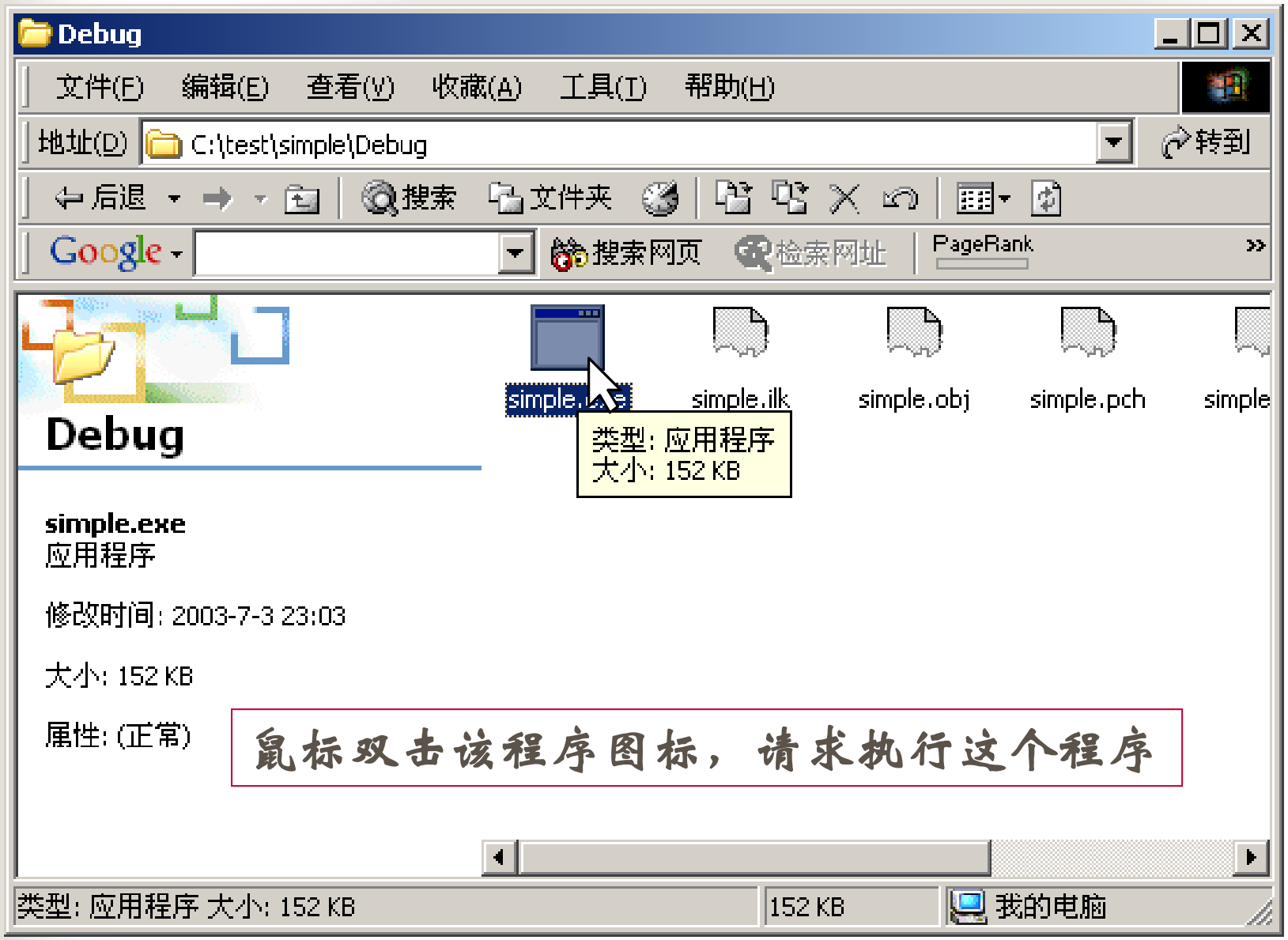
一段程序代码

```
#include <iostream>
using namespace std;
int main()
{
      int a,b,c;
      a = 5;
      b = 4;
      c = a + b;
      cout << c << endl;
      return 0;
}
```

在VC下编辑，编译


test - Microsoft Visual C++ - [main.cpp *]
File Edit View Insert Project Build Tools Window
Help
[Globals]
[All global members
main
#include <iostream>
using namespace std;
int main()
{
int a,b,c;
a = 5;
b = 4;
c = a + b;
cout << c << endl;
return 0;
}
Build Debug Find in Files 1
Ready

生成的可执行文件保存在硬盘中



含义	状态
空	0
系统留用	1
文件结束	999
簇指针	其他数字

文件名	起始簇
Bio.txt	3
Jordan.wks	7
Pick.wps	9

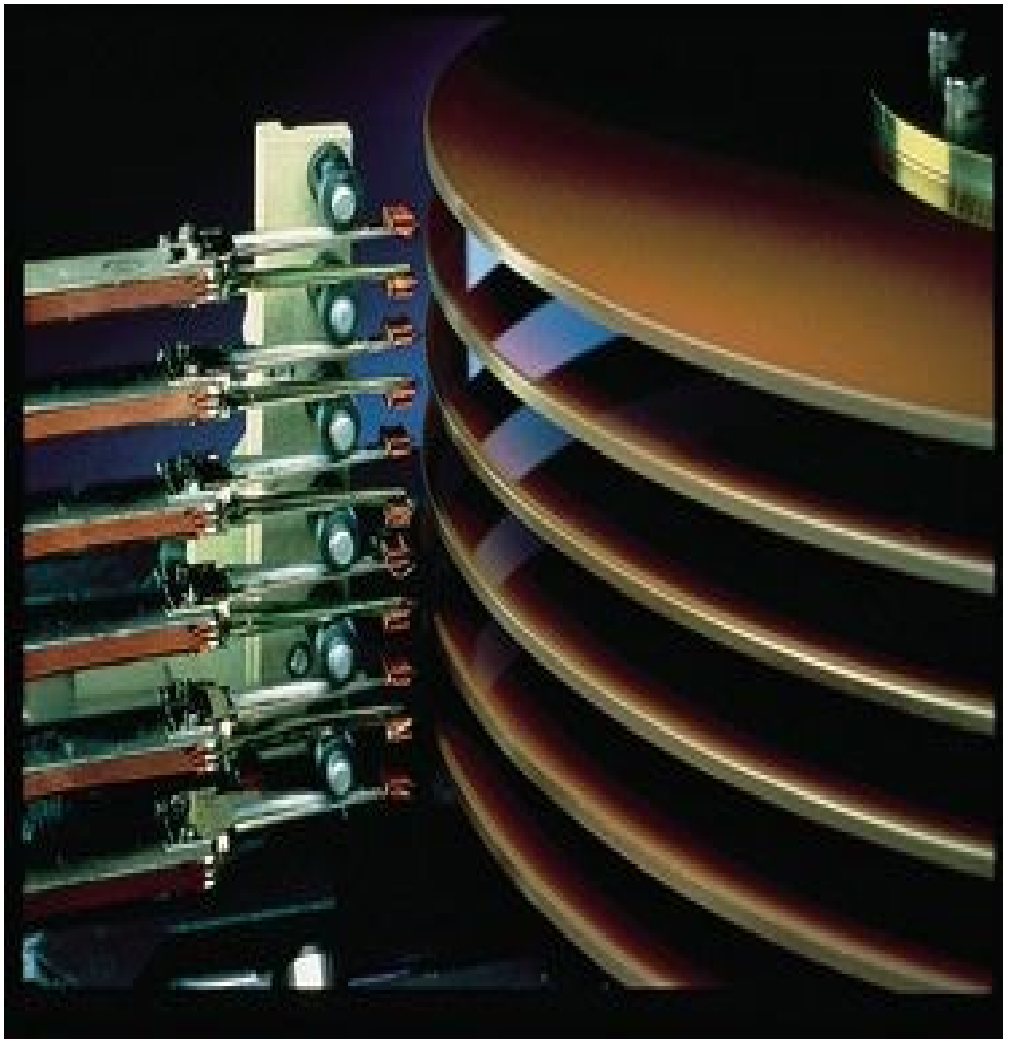
操作系统根据文件名找到存放该文件的第一个磁盘块

簇	状态	注释
1	1	操作系统保留
2	1	操作系统保留
3	4	Bio.txt的起始簇,指向下一簇
4	999	Bio.txt的最后一簇。
5	0	
6	0	
7	8	Jordan的起始簇，指向簇8
8	10	Jordan后续数据，第10簇
9	999	Pick.wps起始簇和最后一簇
10	999	。Jordan.wks的最后一簇

一个简单程序的运行（4）

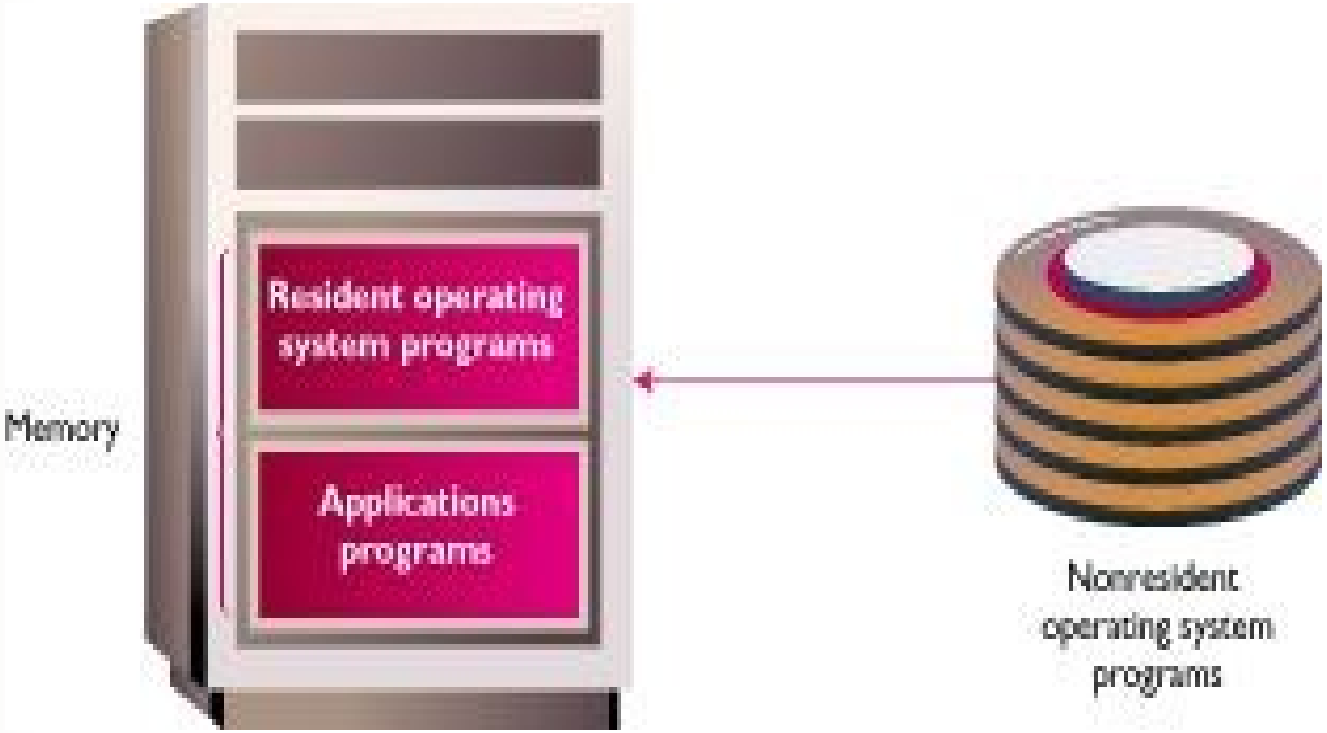
操作系统管理文件

通过FAT表，找到程序在磁盘上的位置，并控制磁盘旋转，开始读取程序

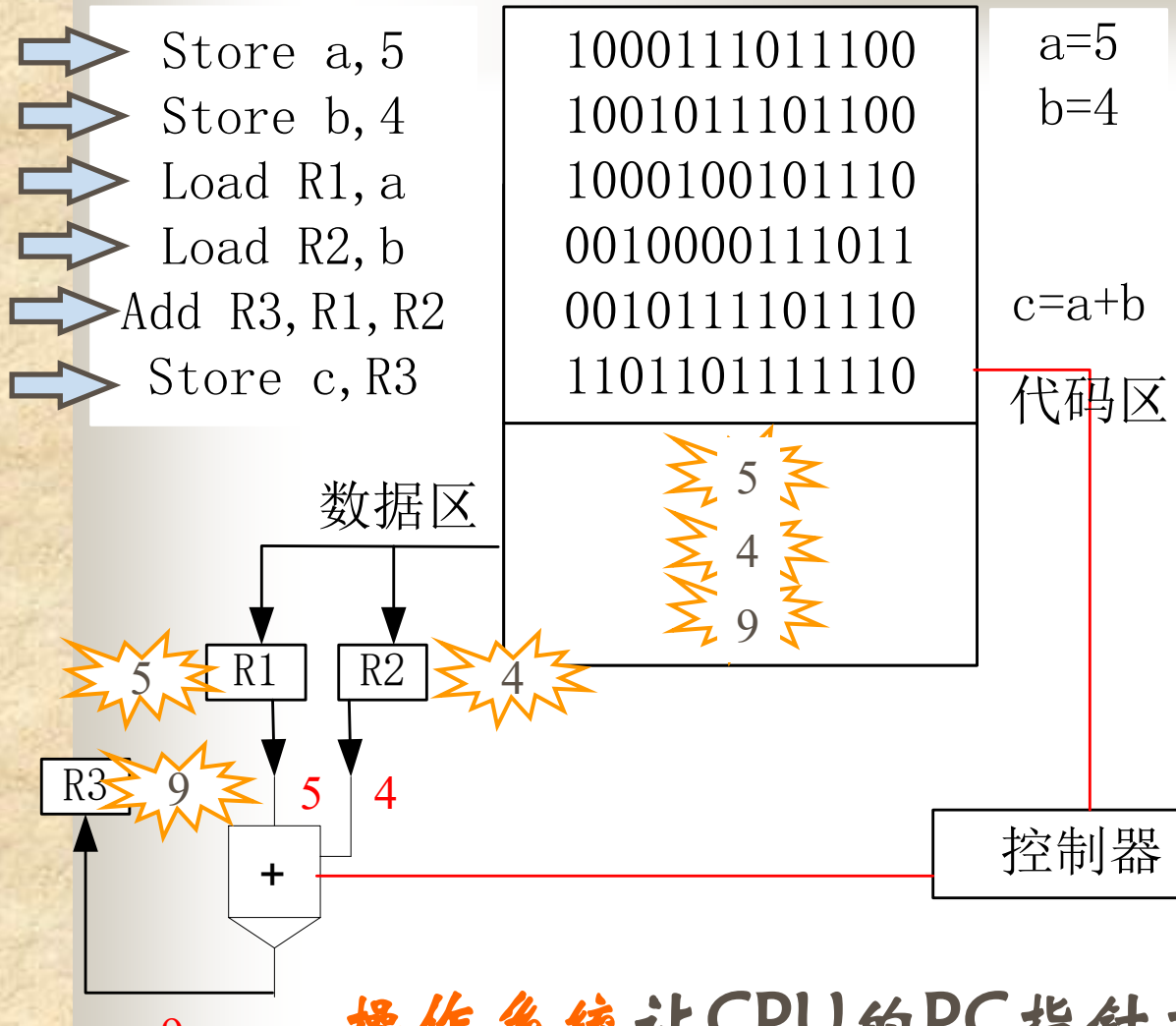


一个简单程序的运行（5）

- 操作系统在内存中创建进程结构，并将代码从磁盘copy到内存。



寄存器




```
#include <iostream>
using namespace std;
int main()
{
        int a,b,c;
        a = 5;
        b = 4;
        c = a + b;
        cout << c << endl;
    return 0;
}
```

编译系统讲高级语言转变汇编语言

操作系统让CPU的PC指针指向该程序的第一条指令，并执行

一个简单程序的运行（1）

```
#include <iostream>
using namespace std;

int main()
{
   int a, b, c;
   a = 5;
   b = 4;
   c = a + b;
   cout << "c "  = << c << endl;
   return 0;
}
```

1.鼠标双击该程序图标

2.操作系统根据文件名找到存放该文件的第一个磁盘块

3.操作系统在内存中创建进程结构，并将代码从磁盘copy到内存

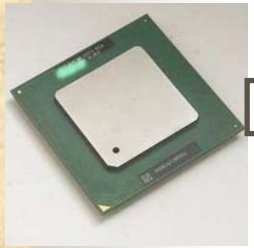
4.操作系统让CPU的PC指针指向该程序的第一条指令，并执行

5.操作系统向显示器设备输出字符串”c=9”

6.显示输出c=9

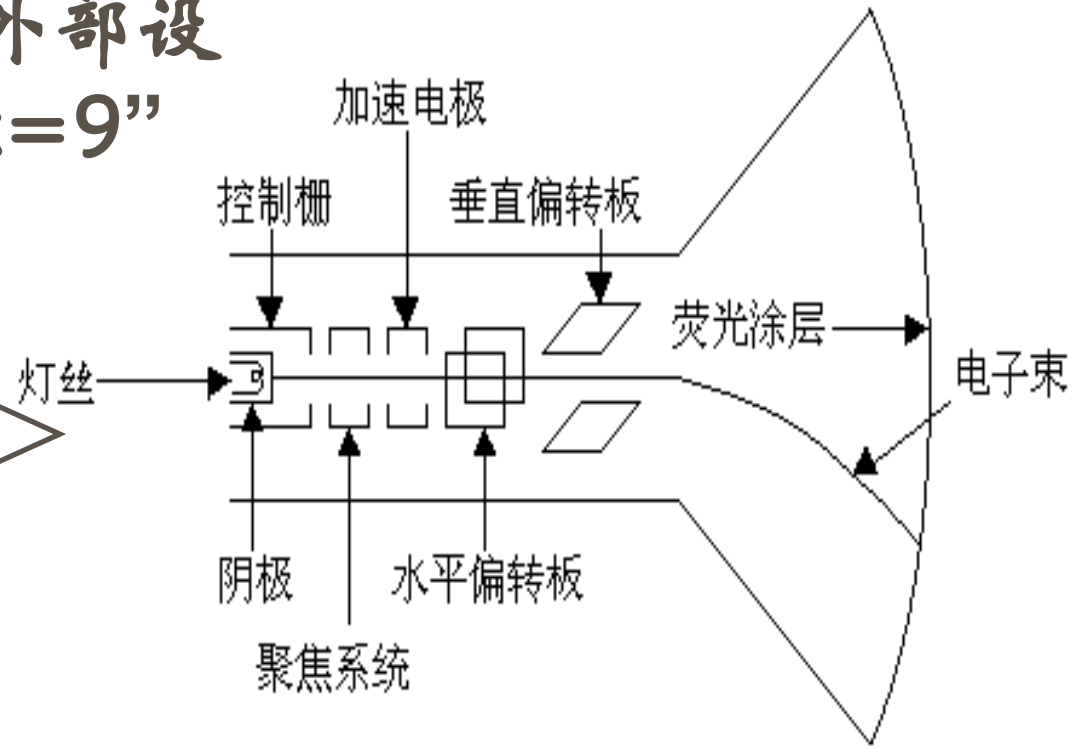
一个简单程序的运行（7）

操作系统向显示器设备发出命令，控制外部设备输出字符串”c=9”



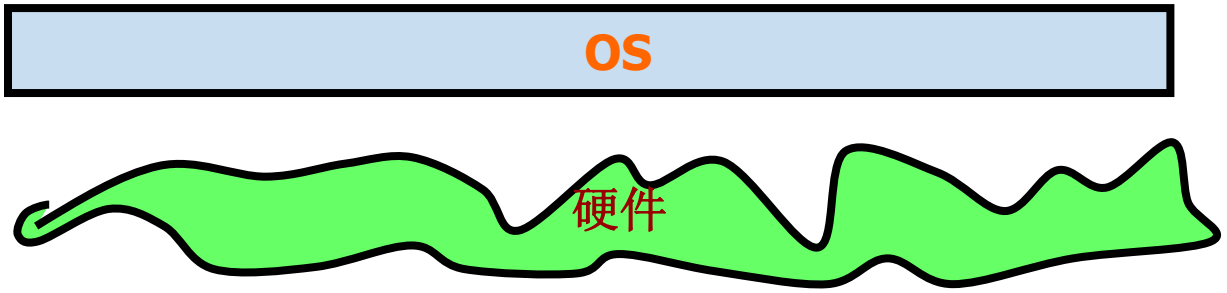
c＝9

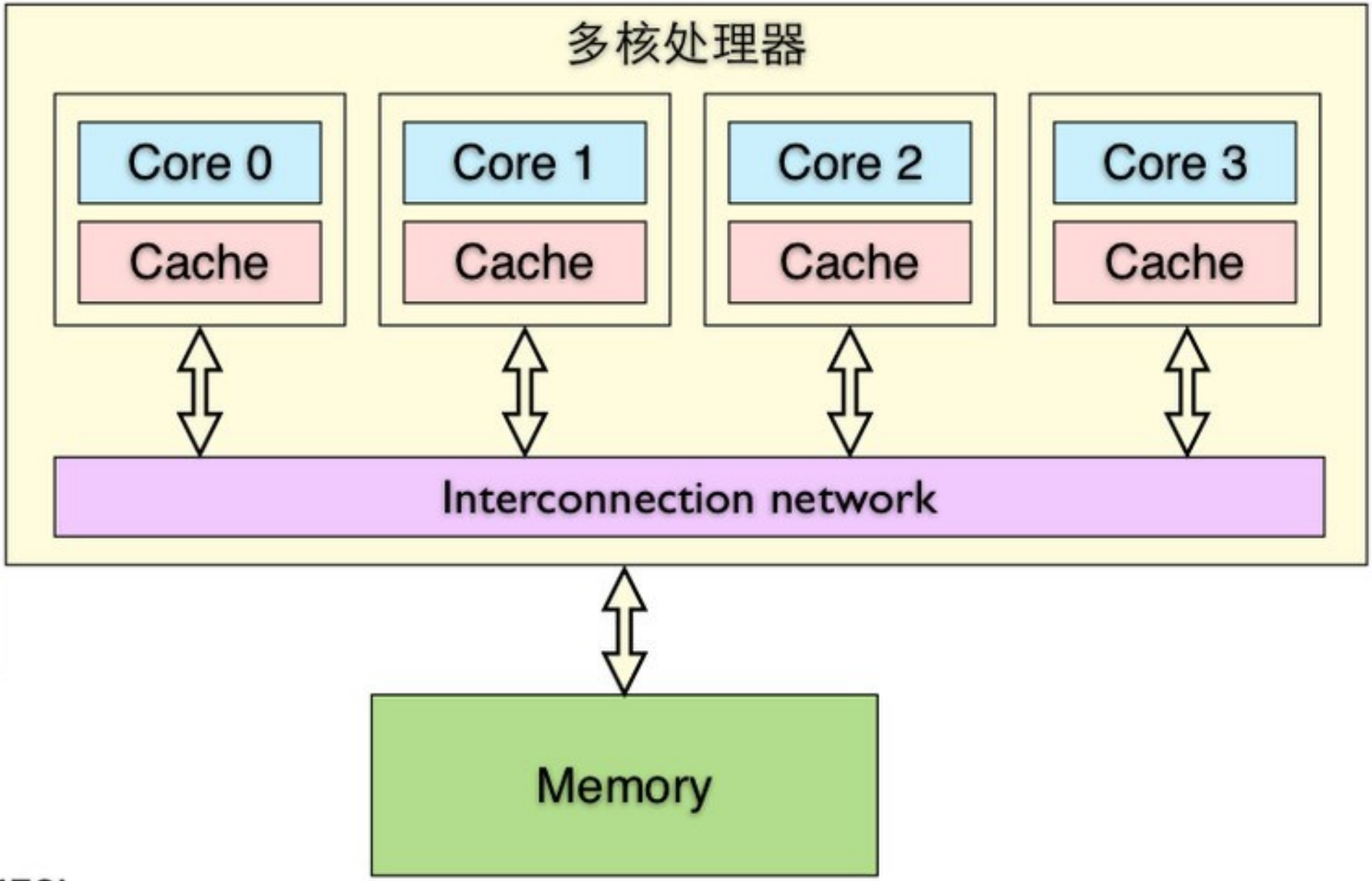
控制显示



操作系统

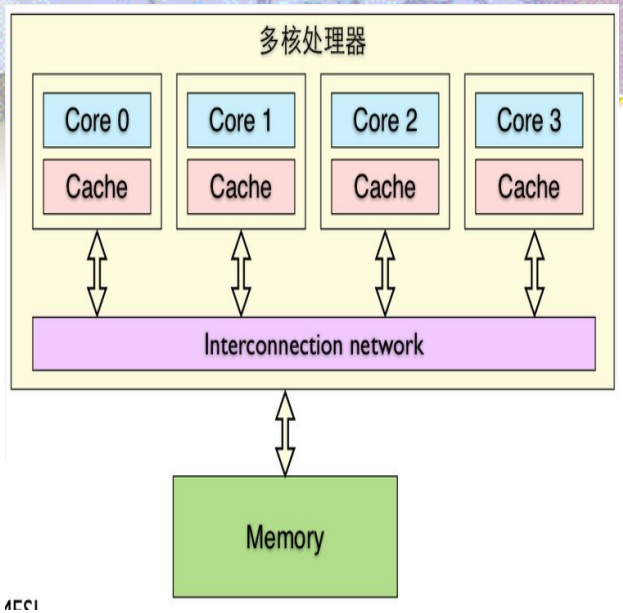
- 我们看到，这个简单的程序中有很多对设备的操作。而这些操作的控制并不是在我们的程序中完成的，看起来很像是自动的，这就是操作系统帮我们做的。
  - 就好像驾车的复杂，和雇佣司机的简单性（这是计算机科学中重要的原则，既屏蔽复杂，提供简单）




多核处理器
Core 0
Cache
Core 1
Cache
Core 2
Cache
Core 3
Cache
Interconnection network
Memory


SM
Instruction Cache
Instruction Buffer
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Core
DP Unit
LD/ST
SFU
Instruction Buffer
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Texture / L1 Cache
Tex
Tex
Tex
Tex
64KB Shared Memory

多核处理器



- 多核处理器带来的好处：
- 横向扩展，通过划分任务，线程应用能够充分利用多个执行内核，并可在特定的时间内执行更多任务。

- 程序：计算机程序是指以某些程序设计语言编写的指令序列。
- 进程：是一个应用程序在处理机上的一次执行过程，它是一个动态的概念。
- 线程是进程中的一部分，进程包含多个线程在运行

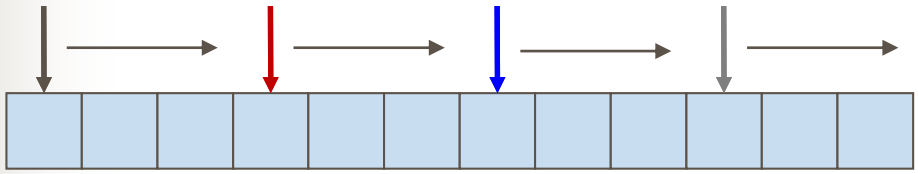
```
void one_function()
{
    int array[100];
    #pragma omp parallel for
    for (int i = 0; i < 100; i++)
    {
        array[i] = i * 10;
    }
}
```

多核情况下的执行方式

多个处理器核心并行计算,每个核心负责处理一部分数据.

每个核心都有独立的一级缓存和二级缓存.

三级缓存被所有核心共享,可用于核心之间的同步,通信.



数据中心





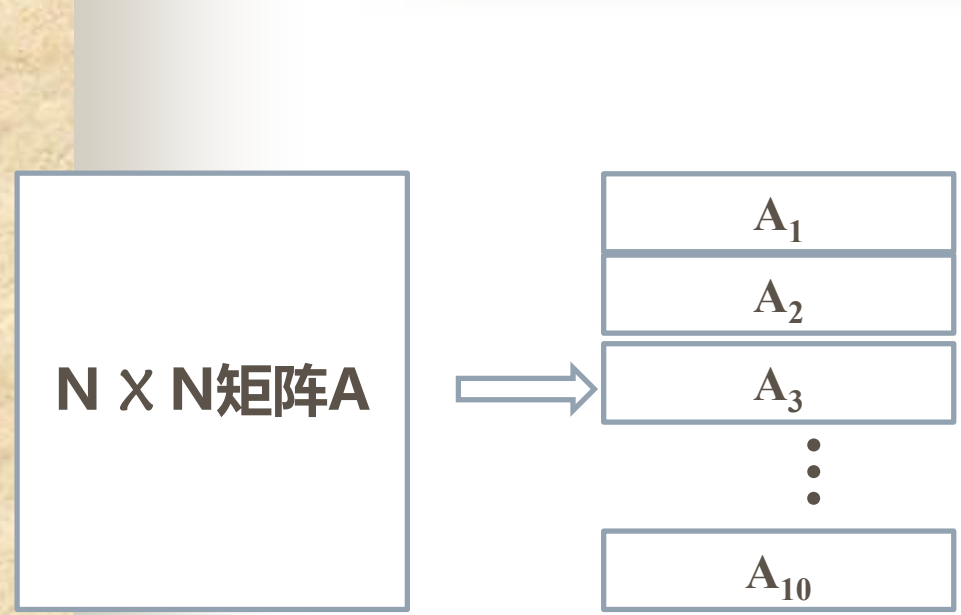

分治执行

$$\mathbf{AB}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

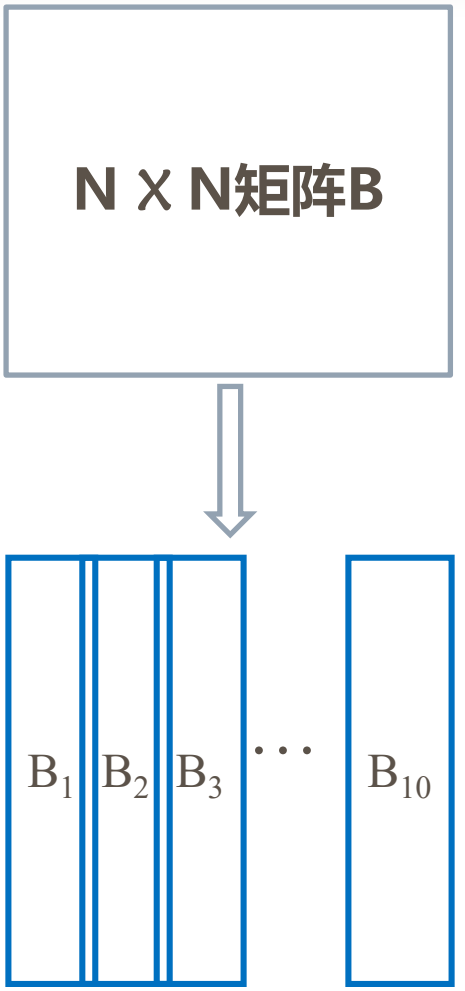
$$=\begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} & a_{11}b_{13}+a_{12}b_{23}+a_{13}b_{33} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} & a_{21}b_{11}+a_{22}b_{23}+a_{23}b_{33} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32} & a_{31}b_{13}+a_{32}b_{23}+a_{33}b_{33} \end{bmatrix}$$

公式 7.6　　3×3 矩阵乘法

分治执行



A_i：N/10 X N 的矩阵



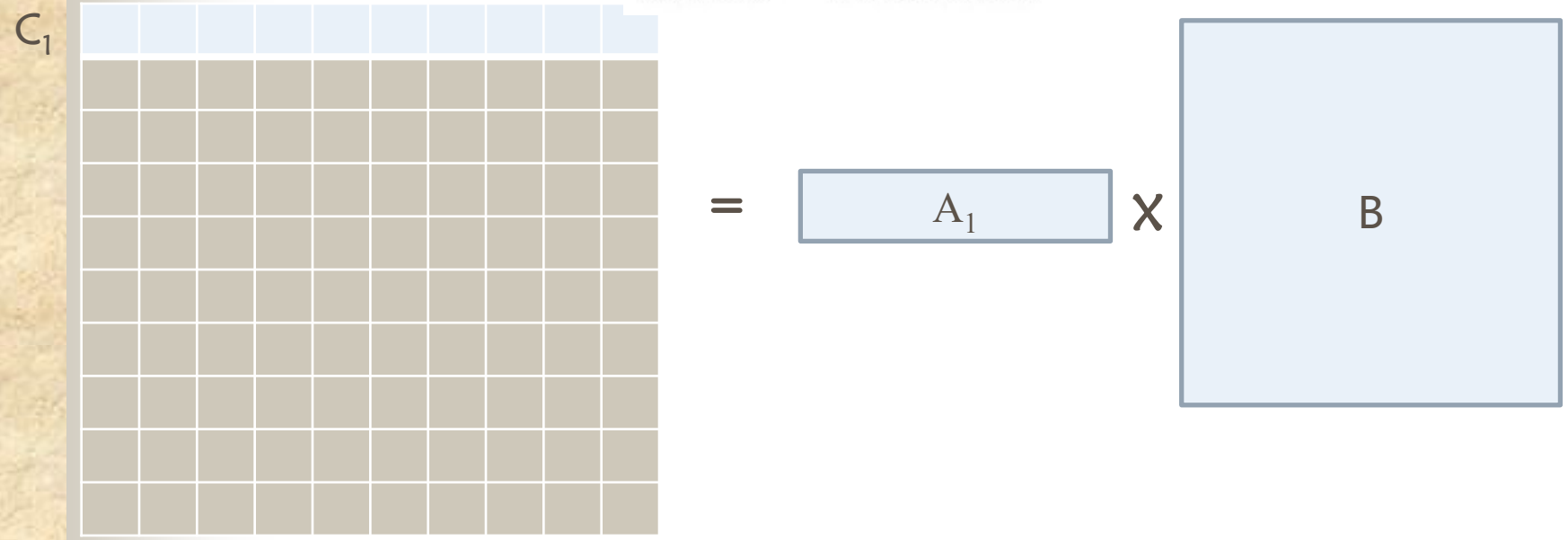
B_i：N X N/10 的矩阵

分治执行

$$AB=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix}\begin{bmatrix}b_{11} & b_{12} & b_{13}\\ b_{21} & b_{22} & b_{23}\\ b_{31} & b_{32} & b_{33}\end{bmatrix}$$

$$=\begin{bmatrix}a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} & a_{11}b_{13}+a_{12}b_{23}+a_{13}b_{33}\\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} & a_{21}b_{11}+a_{22}b_{23}+a_{23}b_{33}\\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32} & a_{31}b_{13}+a_{32}b_{23}+a_{33}b_{33}\end{bmatrix}$$

公式 7.6　　3×3 矩阵乘法



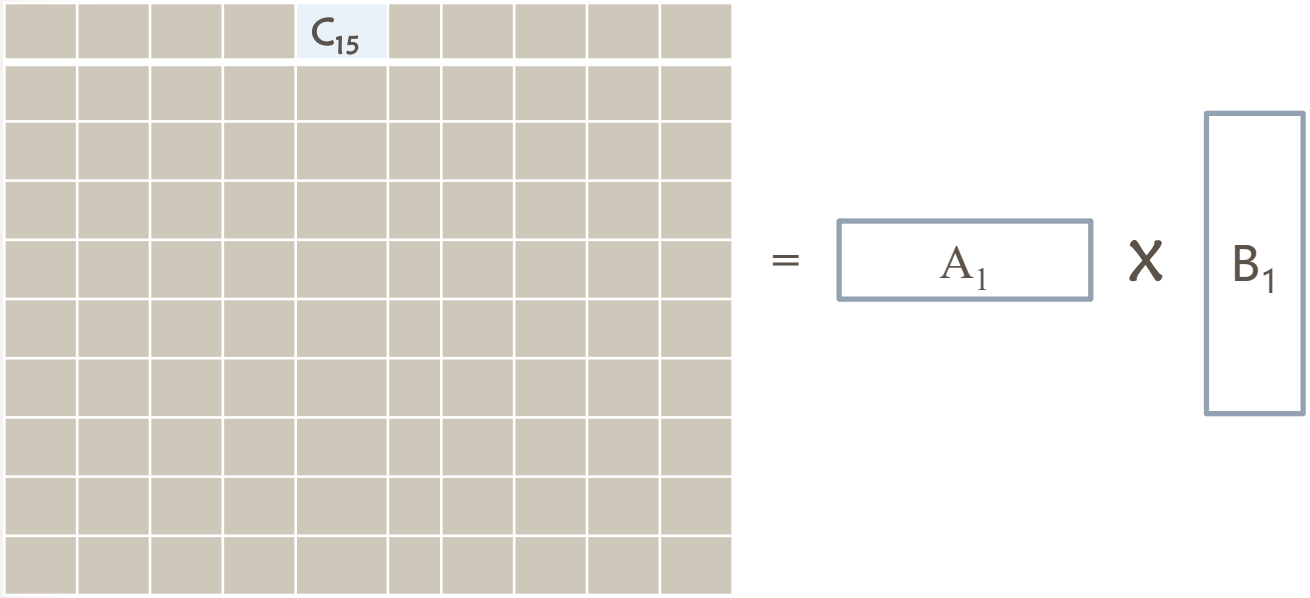
- 第1台服务器完成前十分之一行元素，既C_1
- 同理，有第2台…第10台服务器分别计算C_2…C_{10}

分治执行

$$\mathbf{AB}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$=\begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} & a_{11}b_{13}+a_{12}b_{23}+a_{13}b_{33} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} & a_{21}b_{11}+a_{22}b_{23}+a_{23}b_{33} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32} & a_{31}b_{13}+a_{32}b_{23}+a_{33}b_{33} \end{bmatrix}$$

公式 7.6　　3×3 矩阵乘法

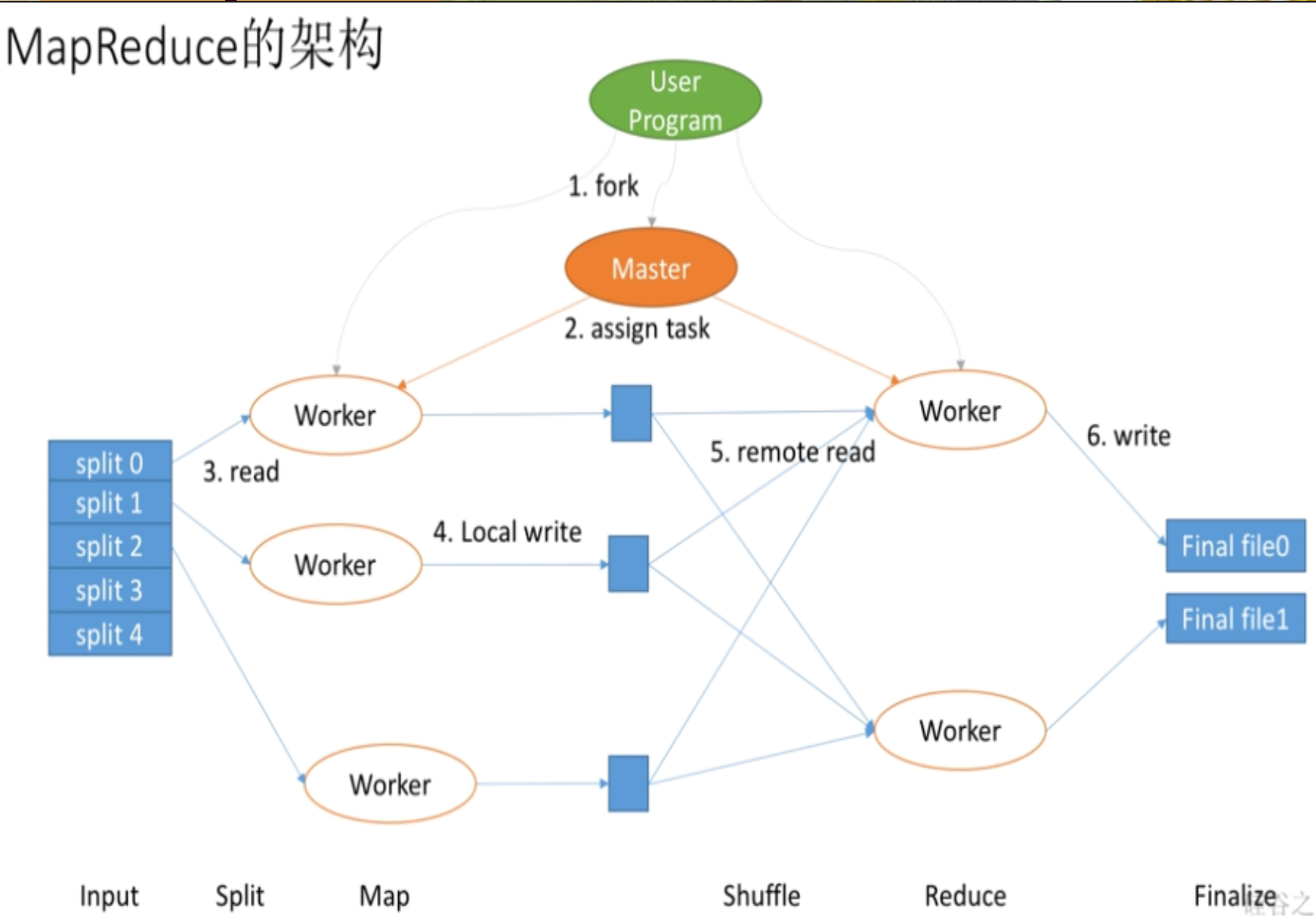


- 还可以，100台机器，第1台服务器完成前百分之一，既C_{11}

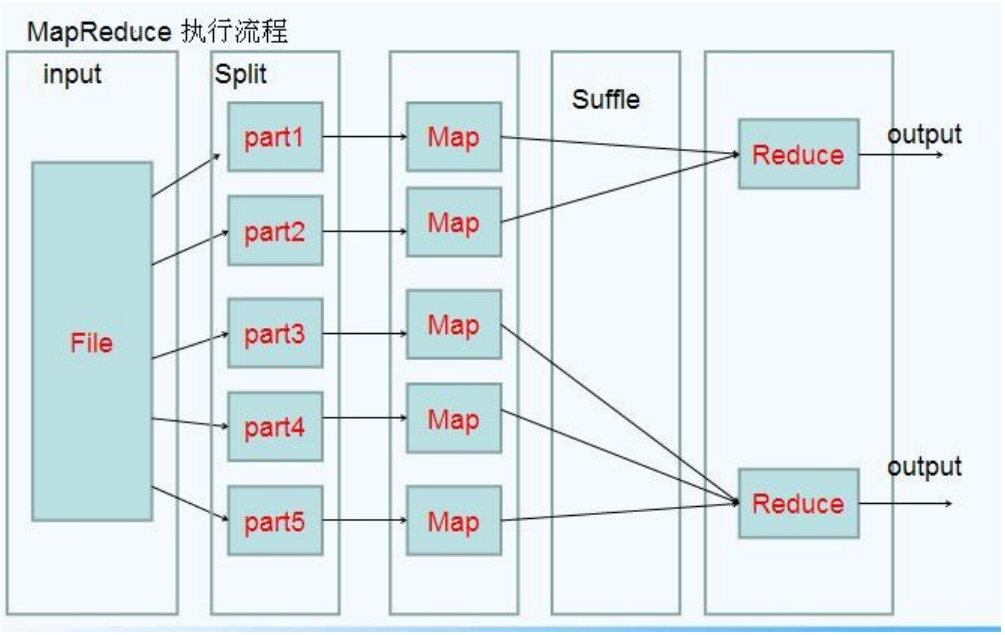
什么是Mapreduce，

- 如何把一个非常大的计算问题，自动分解到许多计算能力不是很强大的计算机上共同完成？
- Google给出的解决方案叫做Divide-and-Conquer。
- Google最著名的解决工具叫做Map-reduce。

MapReduce系统构架



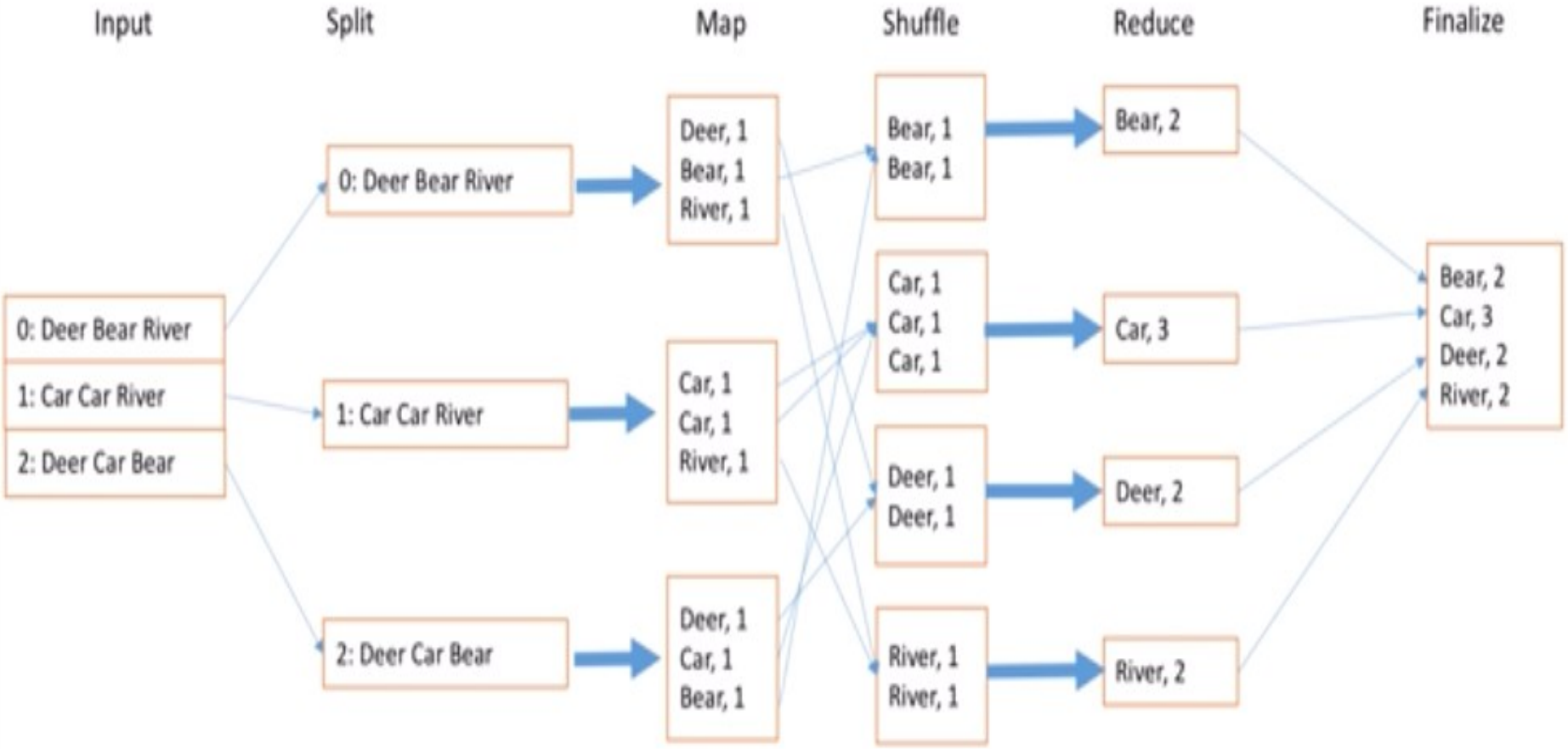
BitTiger.io


MapReduce 执行流程
input
Split
Suffle
File
part1
part2
part3
part4
part5
Map
Map
Map
Map
Map
Reduce
Reduce
output
output

单词统计的例子

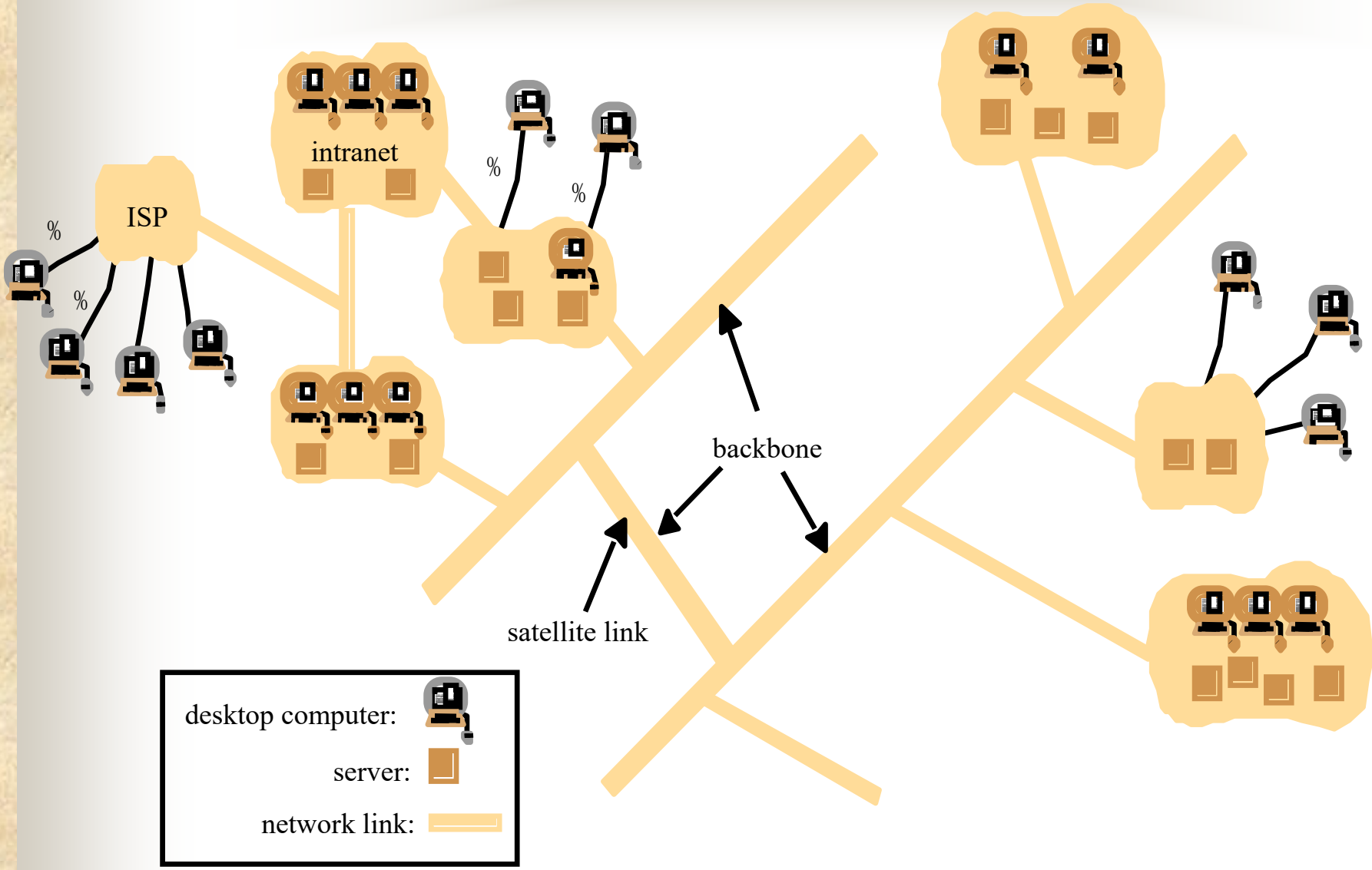
- 统计一个文档中的单词出现的频数
- 一般流程：
  - 打开文件→读取文件→存成字符串→统计单词
- 新的场景
  - 很大很大的文件，或者很多很多和文件
- 如何解决？
  - 将一个文件分成多个块，有多个计算机同时处理
- 问题：
  - 如何将最后的结果合并？

统计单词出现数



1x 1.5x 2x

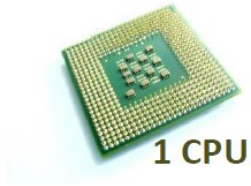
分布式系统

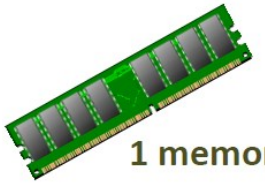

intranet
ISP
backbone
satellite link
desktop computer:
server:
network link:

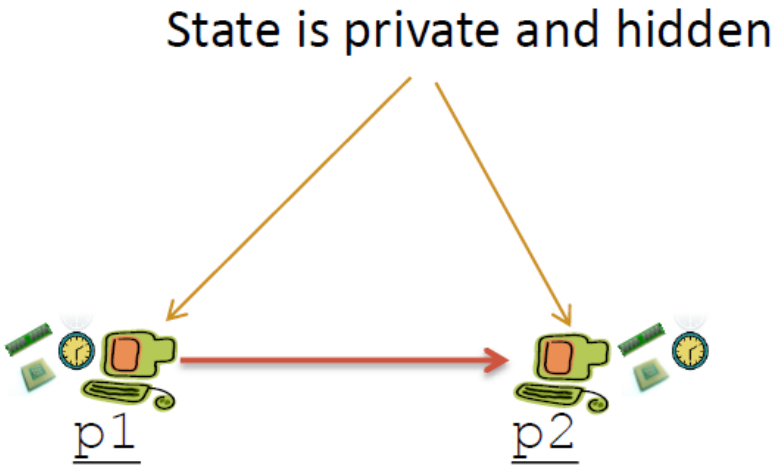
单机和分布式

Single process

```
int i;
i=i+1;
   …
```








```
int i;          int a;
i=i+1;          …
send(i,p2);     receive(p1,a);
…               …
```

以一个聊天室为例

- 应用逻辑：
  - 把每个人发的消息（图片、文字，音视频）转发给所有聊天室里的人
- 用界图形界面
  - 图形界面（MFC，Java…）
- 网络技术
  - 组播，需要聊天室成员的网络ID。
- 一次学生的作业
  - 在一台机器上开多个窗口，模拟多个人聊天
  - 这完全不是分布式系统，因为回避了分布式要解决的核心问题。

对分布式系统的“理想”假设

- 发出的信息对方一定能收到
- 发出去的消息到达的时间可预测
- 先发出去的消息先到达
- 每个用户都用windows操作系统
- 每个机器上的时钟都是精准的
- 机器不会宕机，进程不会出故障。

- 如果转发节点宕机了怎么办？
- 怎么保证消息能到达每一个人？
- 两个机器要进行消息传递，怎么知道对方是否收到了的信息？我在等待一个回复，可是它迟迟不到，怎么办？
- 怎么保证消息到达的顺序是正确的？
- 操作进行到一半，突然宕机了怎么办。
- 如果人数不断扩大，如何应付百万、千万的请求？
- 参与者使用的计算机操作系统不一样怎么办？
- 如何保证消息的私密性、完整性？

问题的背后——起因

- 互联网在IP协议是一种尽力而为的服务，不保证每个消息都能送达，数据会丢失。
- 在IP网络上，消息被打成若干信息包，不一定走的是不同的路径，消息的传递时间是不能准确预计的。
- 在分布式系统中没有全局时钟(global clock)，每个机器都有各自的时间，没有办法做到统一，程序间的协调靠交换消息。
- 系统的结构会影响效率
- 多个程序（进程，线程）并发执行，共享资源，会导致不一致的问题出现。
- 一些进程出现故障，并不能保证其它进程都能知道

系统结构的目的

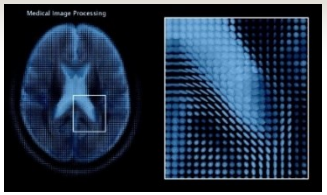
- 如何更好的和系统软件相结合，更好的利用硬件资源，提高计算能力和效率。
- 如何更好的支撑应用软件和算法。
- 保证软件可靠性和安全性。
- 屏蔽底层硬件异构

以深度学习为例，理解计算机系统结构的工作

Self-driving

Surveillance detection

Translation



Medical diagnostics

Game

Personal assistant

It is the Age of Deep Learning

Art

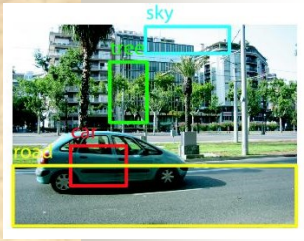
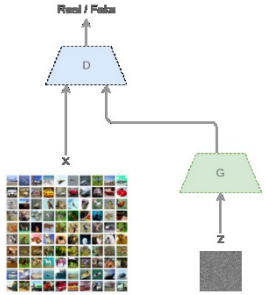


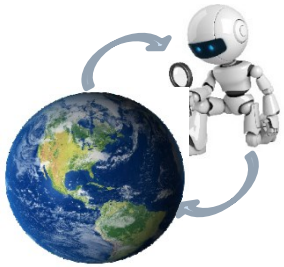
Image recognition

Speech recognition

Natural language

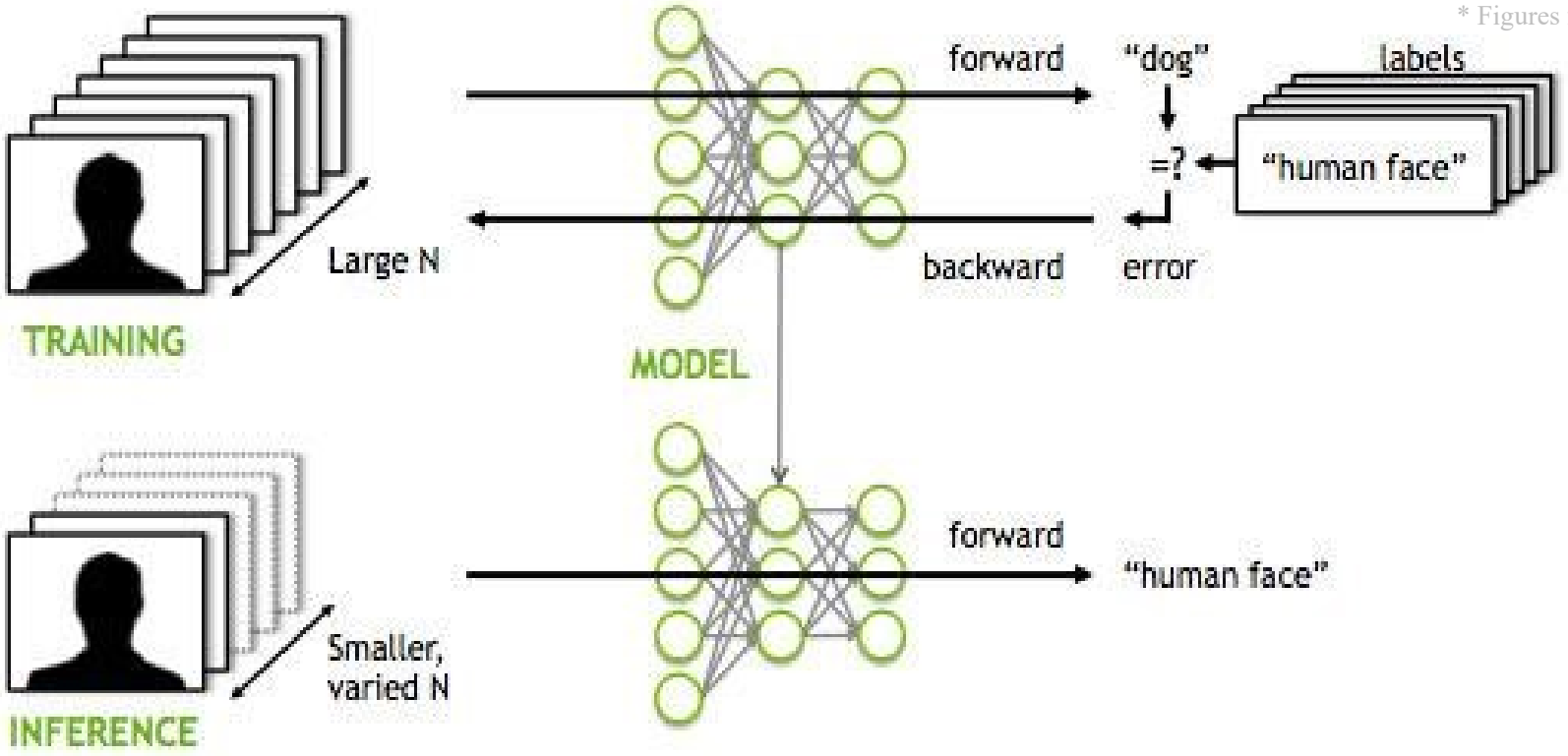


Generative model



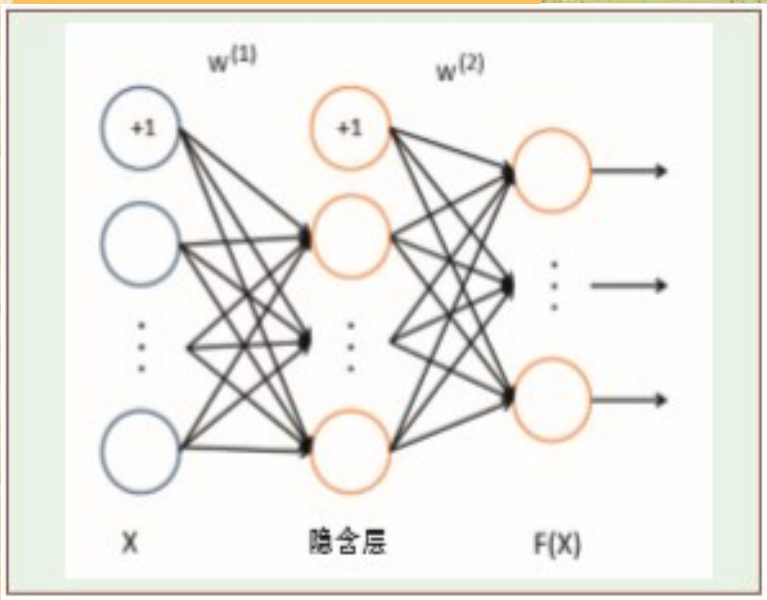
Reinforcement learning

深度学习周期

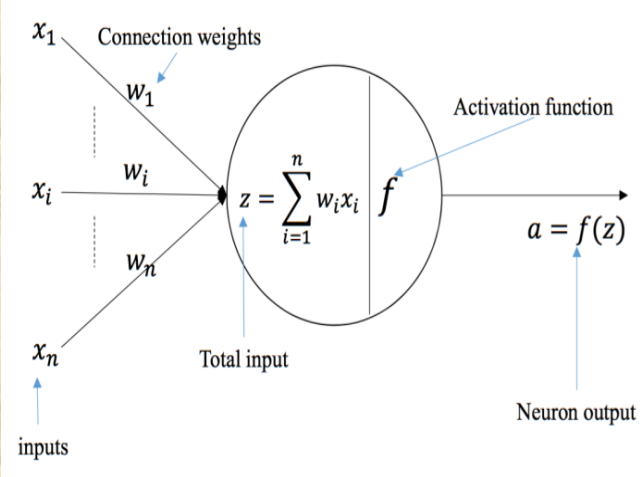


1. 大量人工标注的数据
2. 训练模型
3. 推理

一个深度学习神经元网络的概念图



- 一个深度学习的例子。
- 用计算机模拟400个输入节点x，10个输出节点，具有25个隐含层节点的神经网络，
- 表示成一个从400维到10维实数值向量的函数。
- 其中神经元联接的权重和阈值W共有10285个，它们是函数的可调参数。
- 机器用它学习识别手写体字。

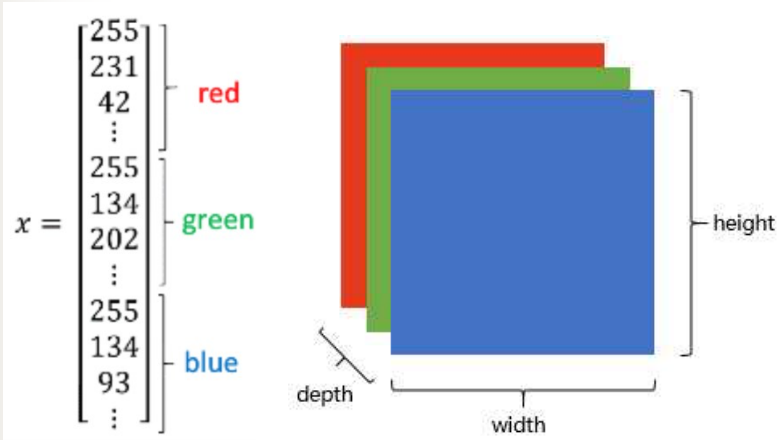


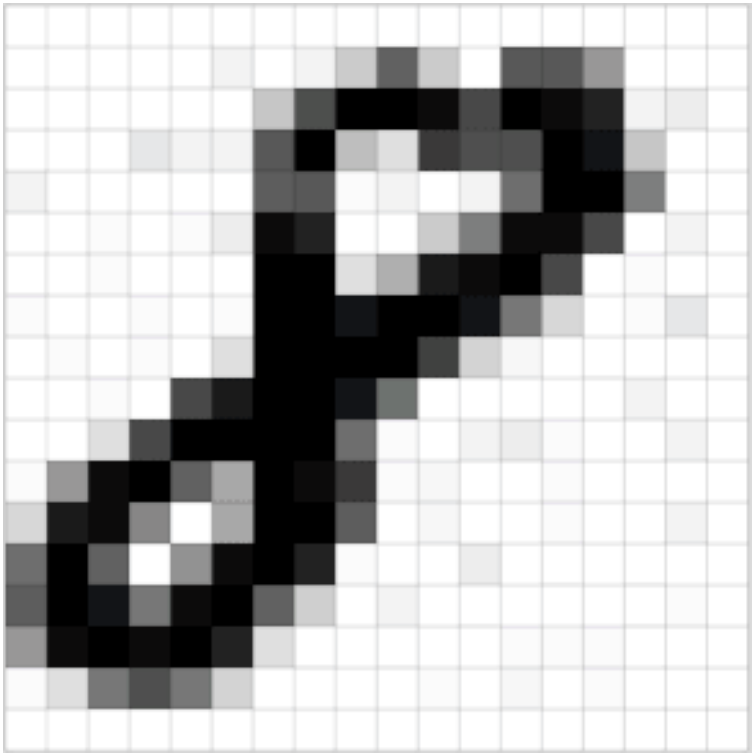
$$f\left(\begin{pmatrix}\square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square\end{pmatrix}\begin{pmatrix}\square \\ \square \\ \square\end{pmatrix}+\begin{pmatrix}\square \\ \square \\ \square \\ \square\end{pmatrix}\right)$$

$$f(W\vec{x}+\vec{b})$$

如果识别图片

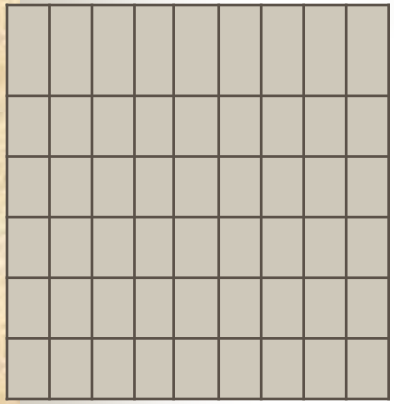
- 基本处理单元
  - 数字0（暗黑）到255(亮白)
  - 矩阵(保留宽高)或向量
  - 灰度（黑白，单通道）或彩色(RGB，三通道)





要处理的数据剧增

为了便于理解，省略了激活函数sigmoid函数，不影响理解

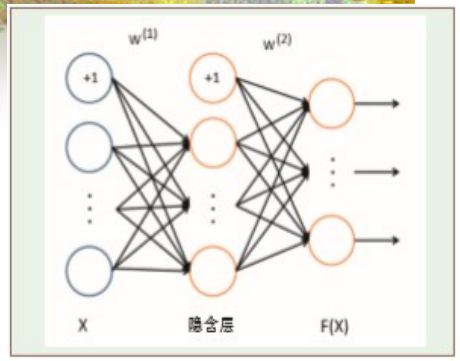


$$\times \begin{bmatrix} x_0 \\ x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_{399} \end{bmatrix} + \begin{bmatrix} w^1_0 \\ w^1_1 \\ \cdot \\ \cdot \\ \cdot \\ w^1_{24} \end{bmatrix} = \begin{bmatrix} t_0 \\ t_1 \\ \cdot \\ \cdot \\ \cdot \\ t_{24} \end{bmatrix}$$

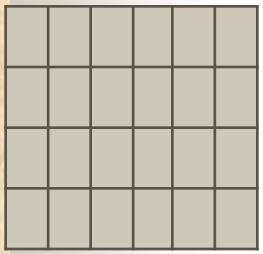
参数矩阵W^1
10000个参数

特征向量

参数向量
25个参数



共有10285个参数



$$\times \begin{bmatrix} t_0 \\ t_1 \\ \cdot \\ \cdot \\ \cdot \\ t_{24} \end{bmatrix} + \begin{bmatrix} w^2_0 \\ w^2_1 \\ \cdot \\ \cdot \\ w^2_9 \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ p_9 \end{bmatrix}$$

参数矩阵W^2
250个参数

参数向量
10个参数

10个数值
的概率

训练集：5000个各种形态图片的像素矩阵作为样本数据，每个样本是一张400个像素的手写数字图片，图片中手写所指的数字称为样本的标记。

- 每个像素对应着一个输入节点，每个输出节点代表输入图片被辨识的一个类别，输出值表示图片属于这个节点类别的相似度。
- 机器的学习过程是用反向传播误差算法(backpropagation)，来迭代调整这10285个参数，这个算法由最小二乘法导出，使得输入样本的标记与对应的输出节点类别判断的平均误差最小。
- 推理：用这个训练好的神经网络，来辨识手写数字的图片。

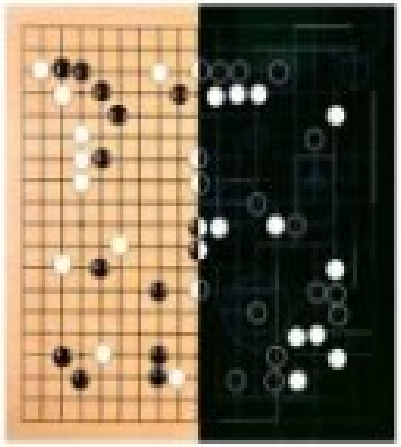
深度学习兴起的条件

- 真实的应用中，例如：最先进的自然语言处理模型XLNet，约有4亿个模型参数，训练需要数百个GPU，三天的时间
- 商汤科技的图像识别冠军模型有1207层
- 内存放不下整个矩阵
- 运算力不足。

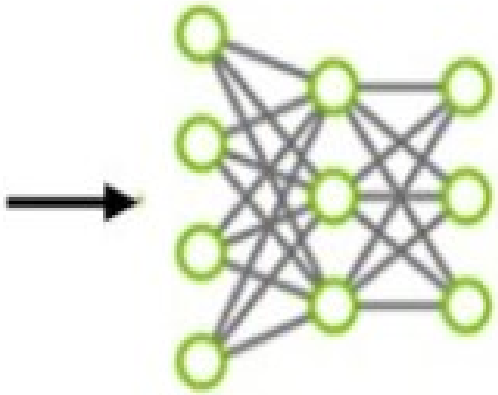
阿里提供给线下算法工程师开发用的集群是60000个CPU核，8000个GPU，几千台机器。其中训练模型用了200块tesla gpu，400亿样本，训练时间需要2天左右。

- 硬件的支持：
- RDMA：通过加速集群间的通信速度，提高计算速度
- GPU，通过并行计算来提高速度
- TPU：专门针对深度学习运算(张量运算),用硬件计算。
- 数据流：改变冯诺依曼结构。
- 分布式系统：让数千，数万台机器协同计算。

机器学习算法：
无监督学习和监督学习
线性回归
逻辑回归
支持向量机
CNN，RNN等



棋盘信息

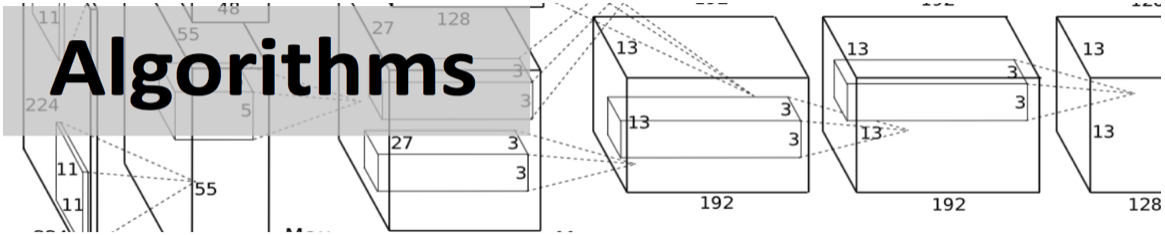


神经网络

落子选择

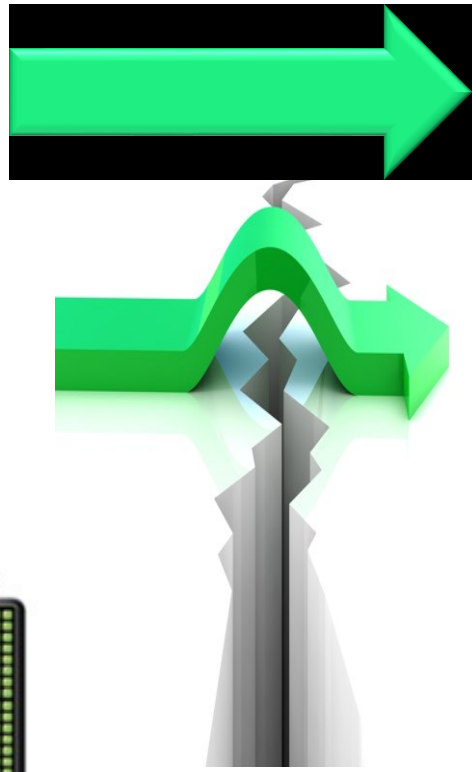
大数据、人工智能兴起的条件

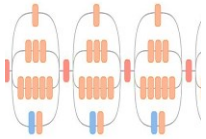
- 大量的数据
  - Internet
  - BigData
- 合理的算法
  - CNN
  - RNN
  - DNN
- 足够的算力
  - 硬件及系统



系统层对算法和应用的支持

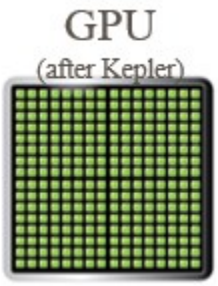
强大的计算能力



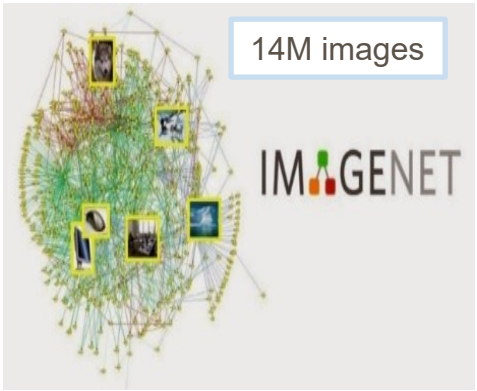


更复杂的模型

Fast communication



Fast computation



海量的标注的数据集

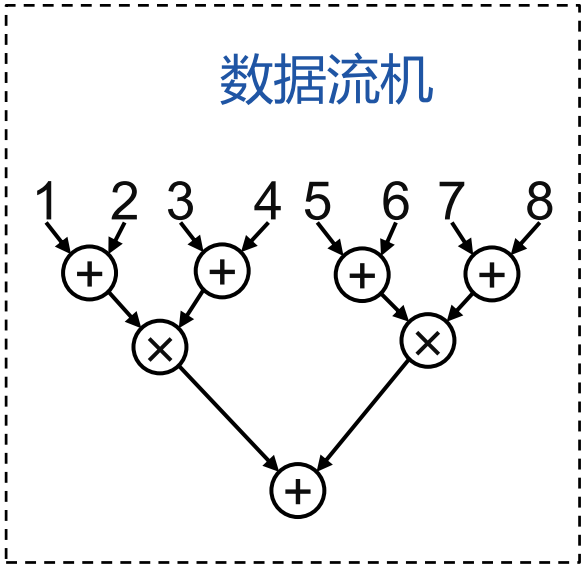
数据流机——非冯诺依曼结构

数据流与控制流的对比——什么是数据流

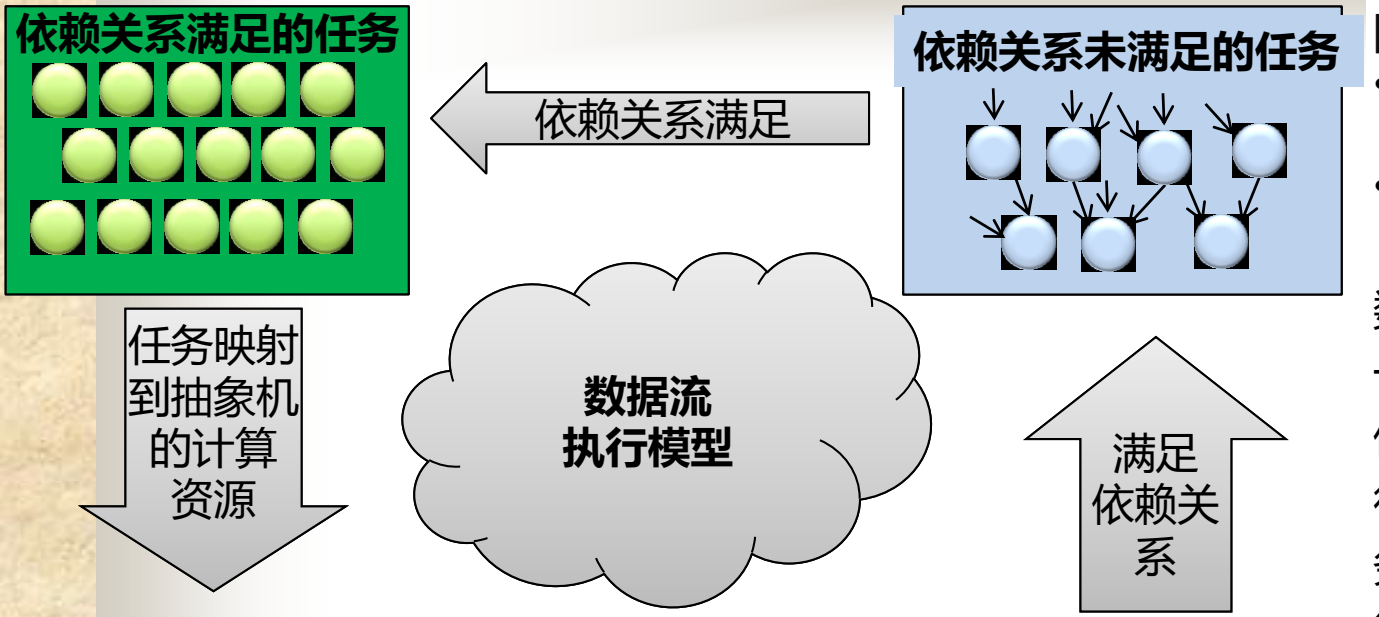
以（1 + 2）*（3 + 4）+（5 + 6）*（7 + 8）为例

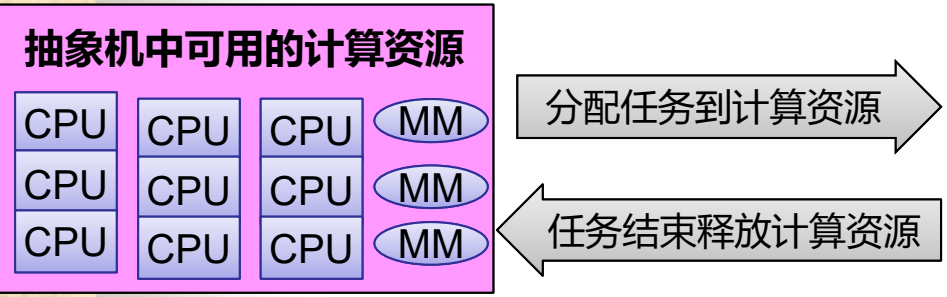
冯·诺依曼机

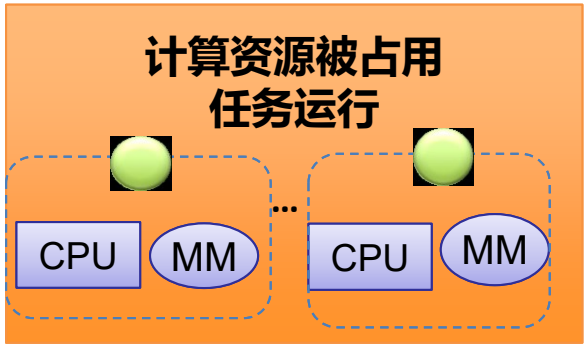
ADD 1, 2, R1
ADD 3, 4, R2
ADD 5, 6, R3
ADD 7, 8, R4
MUL R1, R2, R5
MUL R3, R4, R6
ADD R5, R6, R7



数据流执行模型——核心：Firing Rule







数据流最基础的 Firing Rule

同时满足以下两个条件：

- **任务的数据依赖关系得到满足**
- **存在任务所需的空闲计算资源**

数据流在计算过程中，任何一个任务的计算均会导致其他任务的某些数据依赖关系得到满足，从而触发其他任务满足Firing Rule，进入计算状态，进而进一步触发其它任务的计算，周而反复，直至停机。

数据流的停机条件

- 无正在计算的任务
- 依赖关系满足的任务池为空
- 依赖关系未满足的任务池为空

数据流的历史与今天

杰克·丹尼斯

数据流创始人

美国工程院院士

高光荣

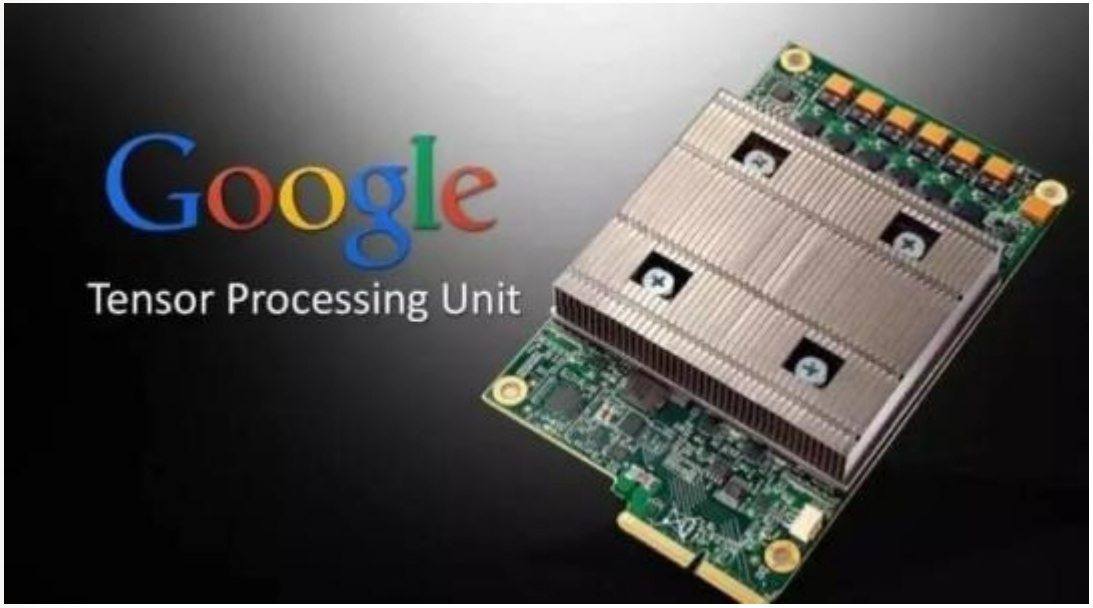
数据流泰斗

ACM、IEEE Fellow

- 数据流基础理论由MIT的Jack Dennis教授于上世纪七十、八十年代提出。
- Jack Dennis教授是美国工程院院士，由于其在数据流理论的贡献获得2012年的IEEE 冯诺依曼奖章。
- 美籍华裔科学家高光荣教授跟随Jack，坚持发展30余年，并成为全球数据流技术的主要代表人物。

Tensorflow

- TensorFlow是谷歌研发的第二代人工智能学习系统。Tensor意味着N维数组，Flow意味着基于数据流图的计算，TensorFlow为Tensor从流图的一端流动到另一端计算过程。TensorFlow是将复杂的数据结构传输至人工智能神经网中进行分析和处理过程的系统。



TPU

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g., MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

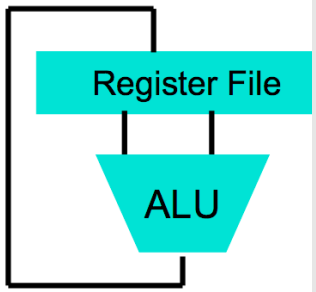
Logic Circuit Description (Circuit Schematic Diagrams)

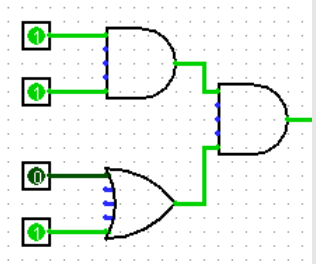
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

Anything can be represented as a *number*, i.e., data or instructions

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```





Application
Algorithm
Programming Language
Operating System/Virtual Machines
Instruction Set Architecture (ISA)
Micro architecture
Gates/Register-Transfer Level (RTL)
Circuits
Devices
Physics

构成计算机的一些基本的要素

- 没有计算模型,就不可能有通用的计算机
- 计算机更新换代，但是冯.诺依曼结构没有变化
- 存储程序原理是现代通用计算机的核心思想。
- 二进制是电子数字计算机电子元件的最佳选择
- 布尔代数奠定了数字逻辑的理论基础
- 硬件逻辑与指令系统是互为决定的。
- 机器语言--〉汇编语言--〉高级语言是计算科学发展的必然产物。
- 算法决定一个问题的能行问题。
- 好的软件取决于好的程序设计方法和新的软件工程方法在构成计算机的一些基本的要素。
- 好的系统结构是支撑算法和应用的基本保障。