



# 计算机系统基础

# Introduction to Computer Systems

王晶

jwang@ruc.edu.cn  
信息楼124

2024年9月



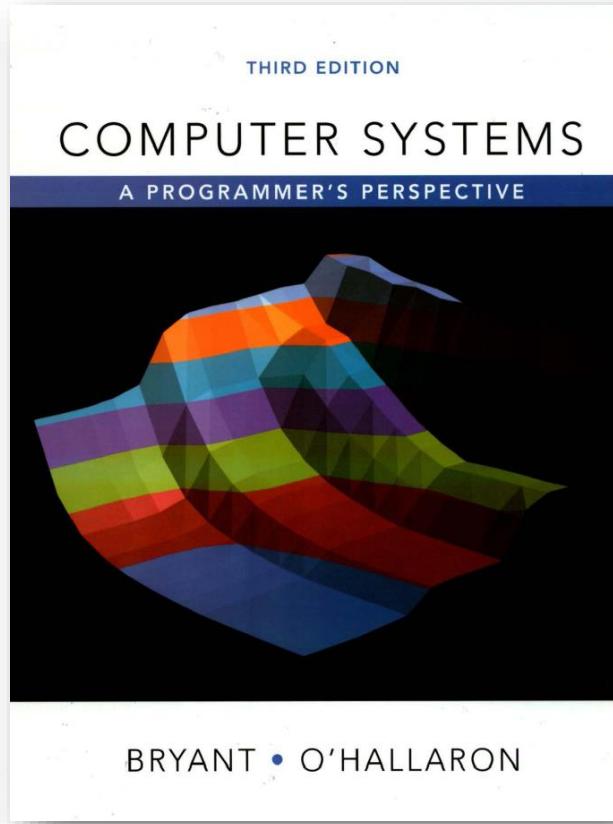
# 目录

- 课程基本信息 
- 课程学习动机
- 有趣的四个现实
- 人工智能与系统
- 计算机系统概述



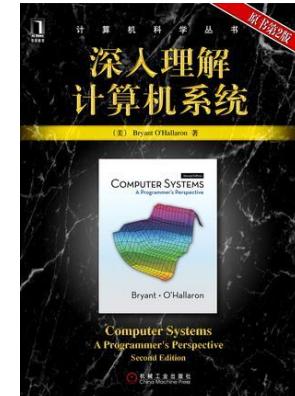
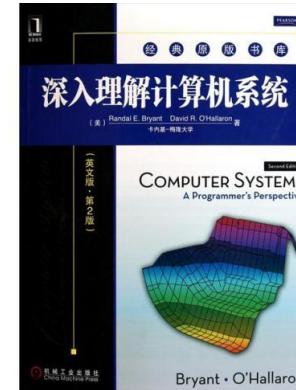
# 课程教材

- Computer Systems: A Programmer's Perspective (3<sup>rd</sup> Edition)  
深入理解计算机系统（英文版·第3版）
- 英文版作者: (美) Randal E.Bryant / David O'Hallaron



教材第3版

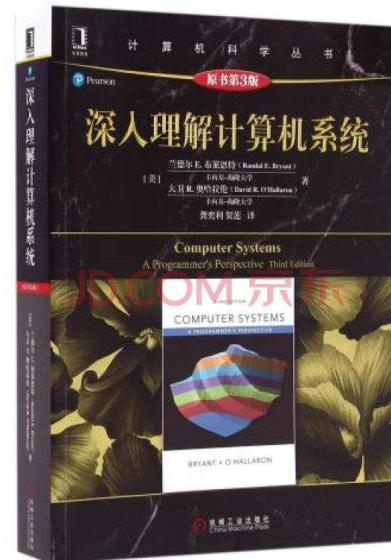
这些是第2版





# CS:APP

- Computer Systems: A Programmer's Perspective
  - 理解C语言下面的内容
  - 数字的表示和运算方法
  - 汇编语言（与C的对应，性能优化）
  - CPU的工作原理
  - 程序性能优化
  - 操作系统（硬件虚拟化、持久存储、并发）
    - + OS:TEP
  - 网络（TCP/IP）





## 第一学期

- 1. 计算机系统概述
- 2. 信息的表示和处理
- 3. 程序的机器级表示
- 期中考试
- 4. 存储层次结构
- 5. 链接与加载
- 6. 程序性能优化
- 期末考试

## 第二学期

- 7. 进程和CPU调度
- 8. 虚拟内存
- 9. I/O和文件系统
- 10. 计算机网络
- 11. 并发编程



# 成绩构成

- 平时成绩(70%)
  - 期中考试 (50%)
  - 实验 (40%)
    - Don't cheat
    - Start early
  - 作业 (10%)
- 期末考试(30%)
- 课程资料和作业提交
  - <http://obe.ruc.edu.cn/>





# 上机安排

- 周三11-13节：下午6:00~8:30，理工配楼二层机房
  - 上机实验
  - 测试
  - 习题讲解/答疑
- 实验内容
  - Lab1: DataLab
  - Lab2: BombLab
  - Lab3: CacheLab
  - Lab4: LinkLab

## 第一学期

- 1. 计算机系统概述
- 2. 信息的表示和处理
- 3. 程序的机器级表示
- 4. 存储层次结构
- 5. 链接与加载
- 6. 程序性能优化



# 授课教师



王晶

Email: jwang@ruc.edu.cn

电话: 13126606010

微信手机同号

办公室: 信息楼124

群聊: 24秋 ICS 课程群-2班



该二维码 7 天内 (9月15日前) 有效, 重新进入将更新



# 2024-2025学年助教



李知非

微信: andylizf

邮箱:

ZhifeiLi@ruc.edu.cn



吴豪宇

微信: 18908033529

邮箱:

wuhaoyu556@ruc.edu.cn

潘俊达

微信: Jarden1234

邮箱:

1747366367@qq.com

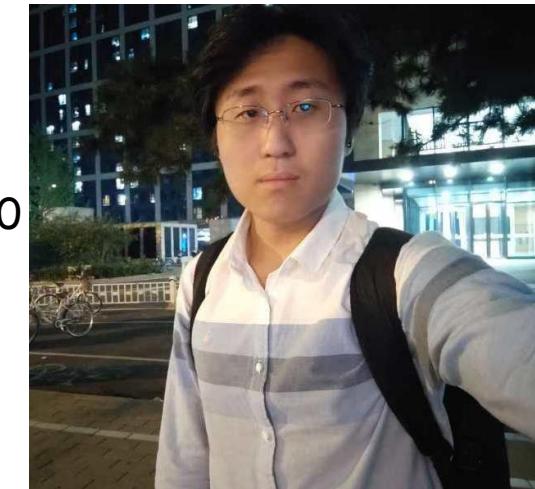


徐少轩

微信: xsx13671322580

邮箱:

xushaoxuan20040225  
@ruc.edu.cn





# 2024-2025学年助教



王宇航

微信: ruc\_cheems0112

邮箱:

wyh040112@ruc.edu.cn



张鑫恺

微信: ruczxk

邮箱: 2022201482@ruc.edu.cn

彭文博

微信号: starry\_dream04

邮箱: i@huanchengfly.top





# 目录

- 课程基本信息
- 课程学习动机
- 有趣的四个现实
- 人工智能与系统
- 计算机系统概述





# 学习动机

- 为什么学习？学习什么？
- 从课程题目《计算机系统基础》出发
  - 计算机
    - 你是否了解计算机的前世今生？
    - 你是否了解未来将要从事的这个行业的发展状况？
  - 计算机系统
    - 什么是计算机系统
    - 计算机系统包括哪些内容



## 单选题 1分

设置

# 世界上第一台电子计算机是？

- A ENIAC
- B ABC
- C A和B都不是
- D A和B都是

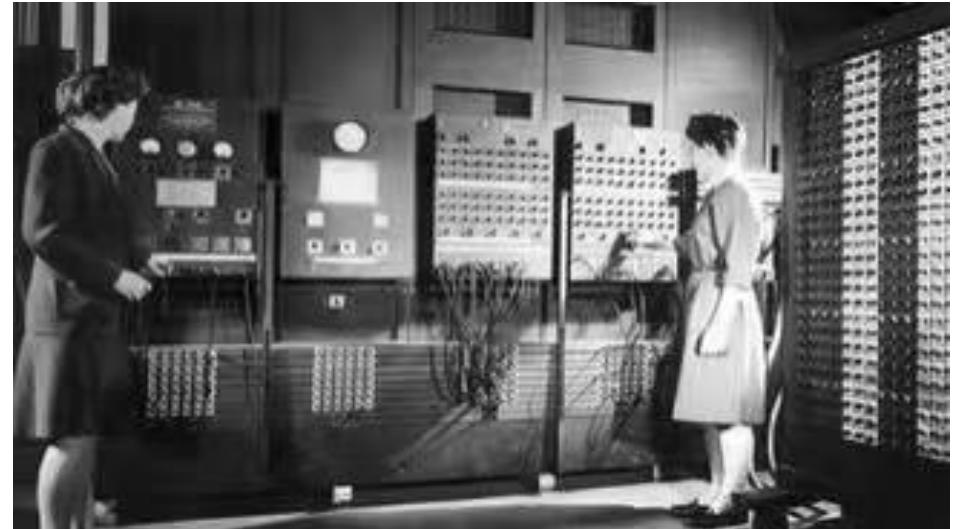
提交

13



# ENIAC：世界上第一台电子计算机

- 时间：1946年2月14日
- 地点：宾夕法尼亚大学
- 功率：150千瓦
- 占地：170平方米
- 重量：30吨
- 成本：40万美元





# ENIAC研制背景：第二次世界大战

- 需求：每天6张弹道表
- 现实：200人2个月完成一张





# ENIAC的主要设计者



约翰·莫克利  
John Mauchly  
1907~1980

约翰·埃克特  
John Presper Eckert Jr.  
1919~1995





# 第一台电子计算机之争

- 1967年，Honeywell公司起诉Sperry Rand公司专利违法
- 1973年，美国明尼苏达州法院裁定ENIAC专利无效



约翰·阿塔纳索夫  
John Vincent Atanasoff  
1903~1995



1939年，美国艾奥瓦州立大学



克利夫·贝里  
Clifford Edward Berry  
1918~1963



# 全球企业股票市值 (2014.2)

1	Apple	4635.49 (亿美元)	美国
2	Google	3954.22	美国
3	Exxon Mobil	3926.64	美国
4	Microsoft	3034.48	美国
5	Berkshire Hathaway	2778.29	美国
6	General Electric	2548.57	美国
7	Johnson & Johnson	2540.42	美国
8	Roche Holding A.G.	2388.18	瑞士
9	Nestle S.A.	2387.19	瑞士
10	Wal-Mart Stores	2386.38	美国
12	PetroChina	2266.94	中国
22	IBM	1861.13	美国
36	Alibaba	1680.00	中国



# 全球企业股票市值(2024.4)

排行	公司名称	行业	市值 (亿美元)
1	微软	软件	31300
2	苹果	科技	26500
3	英伟达	半导体	22600
4	沙特阿美	能源	19820
5	谷歌	互联网	16590
6	亚马逊	电子商务	6850
7	脸书	互联网	7800
8	伯克希尔哈撒韦公司	投资	11400
9	礼来	制药	6050
10	台积电	半导体	4590
11	博通	半导体	5410
21	三星	韩国	3420
26	腾讯	中国	2950



# 目录

- 课程基本信息
- 课程学习动机
- 有趣的四个现实 
- 人工智能与系统
- 计算机系统概述



# 为什么要学习计算机系统？

- 从Programmer角度看

- 计算机是一个黑盒，接受指令（C, Java, Python, ...）
- 学会编程和算法就可以了，有些公司面试也只考这些(软工)
- 以上认识建立在一个假设基础上：
  - 计算机黑盒是完美的，高性能、很智能、不出错

- 但实际上计算机系统并不完美（Abstraction Is Good But Don't Forget Reality）

- 学习计算机系统课程，主要是要了解其局限性（程序出Bug、性能出问题等）
- 成为更好的Programmer
- **4个有趣的现实问题**



# 问题1：整型不是整数，浮点型不是实数

## Ints are not Integers, Floats are not Reals

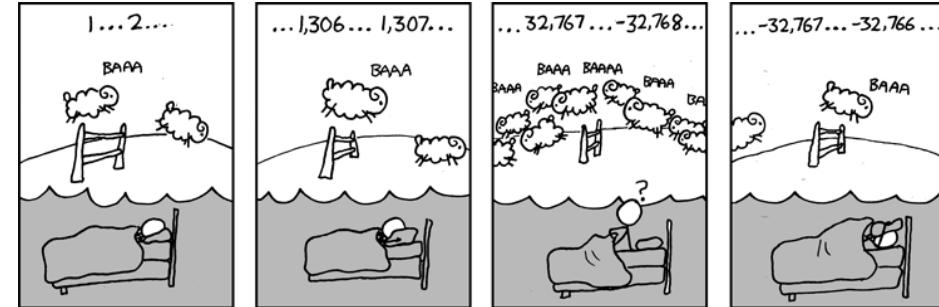
- 例1.1:  $x^2 \geq 0$ 永远成立吗?

- 如果  $x$  是浮点型, 成立

- 如果  $x$  是整型

- $40000 * 40000 \rightarrow 1600000000$

- $50000 * 50000 \rightarrow$  负数, 因为整型有上界溢出



- 例1.2: 是否满足结合律  $(x + y) + z = x + (y + z)$ ?

- 如果  $x, y, z$  是整型, 满足结合律

- 如果  $x, y, z$  是浮点型

- $(1e20 + -1e20) + 3.14 \rightarrow 3.14$

- $1e20 + (-1e20 + 3.14) \rightarrow 0$ , 因为浮点数精度不同不满足结合律



# 计算机系统中的算术 $\neq$ 数学中的算术(1/2)

- 整数性质
  - 交换律:  $a+b = b+a$
  - 结合律:  $(a+b)+c=a+(b+c)$
  - 分配律:  $a \cdot (b+c)=a \cdot b + a \cdot c$
  - 整型运算满足以上性质
- 实数性质
  - 单调性: if  $a \geq b, c \geq 0$ , then  $(a+c) \geq (b+c)$
  - 浮点型运算满足单调性



# 计算机系统中的算术 $\neq$ 数学中的算术(2/2)

- 有些性质在计算机系统中并不成立
  - 计算机系统只能表示“**有限大小的数**”：溢出问题（例1.1）
  - 浮点型不满足结合律：**舍入操作会造成精度误差**（例1.2）
  - 需要记住计算机中不同数据类型所满足的数学性质
  - 对编译器和科学计算程序员尤为重要：因为缺少一些数学性质会使得解决某些简单问题变得麻烦。
- 例1.3：
  - 两个整型a和b是否相等： $a == b$  
  - 两个浮点型a和b是否相等： $a == b$  
    - 因为两个数精度可能不同
    - 正确方法——作差取绝对值  
 $\text{fabs } (a-b) \leq \text{epsilon}$ , ( $\text{epsilon}$ 是很小的数，如0.00001)



## 单选题 1分

设置

# 这样一段程序输出什么？

```
int array[] = {1,2,3};  
#define TOTAL sizeof(array) /* unsigned int */  
void main() {  
    int d = -1;  
    if (d <= TOTAL)  
        printf("small\n");  
    else printf("large\n");  
}
```

A

small

B

large

提交

25



## 问题2：了解汇编 (1/4)

### You've Got to Know Assembly

可能你永远都不会去写汇编程序，但是.....

有助于了解机器层面的程序执行模型

- 帮助查找底层实现相关的程序错误 (bug)

- 例2.1：比较整型(int)、无符号整型(unsigned int)
  - $d = -1 < TOTAL = 12$ , 理应输出small, 但结果却是large
    - `sizeof()`的返回值是unsigned int;
    - if语句作比较时, 编译器认为-1是unsigned int (很大的整数)
  - 通过底层汇编代码/目标程序文件(二进制文件)查看 d 的数值

```
int array[] = {1,2,3};  
#define TOTAL sizeof(array) /* unsigned int */  
void main() {  
    int d = -1;  
    if (d <= TOTAL)  
        printf("small\n");  
    else printf("large\n");  
}
```



## 单选题 1分

设置

### 以下两个程序效率有什么差别

```
void fun1(int *x, int *y)
{
    *x += *y;
    *x += *y;
}
```

```
void fun2(int *x, int *y)
{
    *x += 2* (*y);
}
```

A

fun1效率高

B

fun2效率高

C

没差别

提交

27



## 问题2：了解汇编 (2/4)

### You've Got to Know Assembly

- 程序性能调优

- **例2.2：**尝试不同代码写法，分析比较不同的底层汇编代码效率
- 两个程序似乎有相同的行为。但是fun2的效率会更高
- 通过底层代码可以看出，fun1需要6次存储器引用，而fun2只需3次
- 编译器/人工智能的局限性
- 理解系统低效的原因

```
void fun1(int *x, int *y)
{
    *x += *y;
    *x += *y;
}
```

```
void fun2(int *x, int *y)
{
    *x += 2* (*y);
}
```



## 问题2：了解汇编 (3/4)

### You've Got to Know Assembly

- 系统软件或嵌入式软件开发

- 例如系统软件工程师往往要求写小段汇编代码
- **例2.3：**把小段汇编代码加入C代码，来访问硬件（处理器）上的周期计数器（cycle counter）。
- 系统软件不能只用高层抽象，要管理到具体的进程、线程、共享内存等系统底层的状态，甚至控制硬件Cache刷回内存（clflush, mfence）

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

/* Set *hi and *lo to the high and low order bits
   of the cycle counter.
*/
void access_counter(unsigned *hi, unsigned *lo){
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        :
        : "%edx", "%eax"); }
```



## 问题2：了解汇编 (4/4)

### You've Got to Know Assembly

- 防范恶意软件或分析第三方软件的安全性
  - 分析没有源代码的软件时，需要进行**反汇编**
  - 常见的安全漏洞包括：缓冲区溢出、内存泄露、非授权内存写入等
  - 对反汇编得到的代码进行**静态分析**，是一种找到已知安全漏洞代码的有效手段
  - 例2.4：**定位 **gets()** 这样不安全函数对应的汇编代码

```
void main{}  
{  
    char buf[1024];  
    gets(buf);  
    /*用户输入不做限制，缓冲区溢出*/  
}
```

```
#define BUFSIZE 1024  
void main{}  
{  
    char buf[BUFSIZE];  
    fgets(buf, BUFSIZE, stdin);  
    /*限制输入大小的参数*/  
}
```



# 问题3：内存对程序性能的影响至关重要

## Memory Matters

**Random Access Memory Is  
an Unphysical Abstraction**

- 内存是有限的
  - 必须合理地分配和管理内存
  - 很多程序受限于内存
- 内存引用错误尤为严重
  - 错误的危害因时间、空间而异
- 内存性能并不是始终如一的
  - 高速缓存和虚拟内存极大地影响程序性能
  - 根据存储系统的特点，可以对程序进行调优(见问题4)



# 内存引用错误 (1/3)

```
typedef struct {
    int a[2];
    double d;
} struct_t;

double fun(int i) {
    volatile struct_t s;
    s.d = 3.14;
    s.a[i] = 1073741824; /* Possibly out of bounds */
    return s.d;
}
```

fun(0)	→	3.14
fun(1)	→	3.14
fun(2)	→	3.1399998664856
fun(3)	→	2.00000061035156
fun(4)	→	3.14
fun(6)	→	segmentation fault

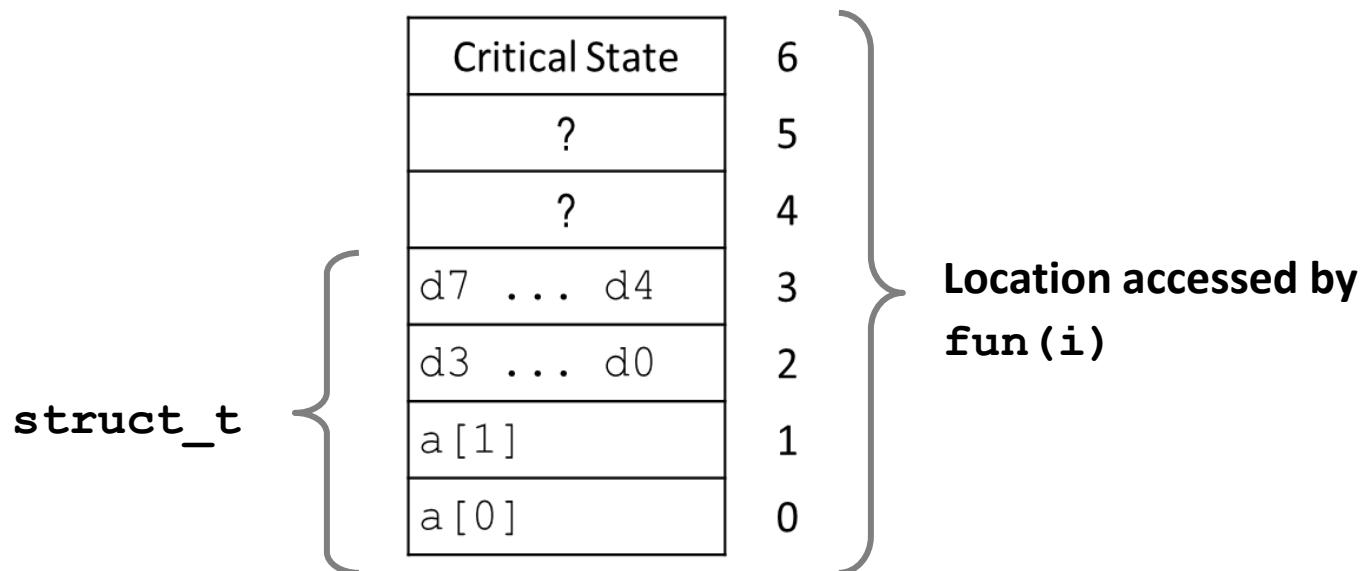


# 内存引用错误 (2/3)

```
typedef struct {  
    int a[2];  
    double d;  
} struct_t;
```

fun(0) →	3.14
fun(1) →	3.14
fun(2) →	3.1399998664856
fun(3) →	2.00000061035156
fun(4) →	3.14
fun(6) →	segmentation fault

## Explanation:





# 内存引用错误 (3/3)

- C 和 C++ 并没有提供对此类错误的防范机制，比如：
  - 数组越界错误
  - 指针错误
  - 滥用 malloc/free 函数
- 应对措施
  - 用其他语言编程，例如 Java, Ruby, Python, ML
  - 使用工具来检测此类内存错误



# 问题4：算法性能分析结果 ≠ 实际程序性能

**There's more to performance than asymptotic complexity**

- 代码写的好坏与否，可能导致程序性能的**数量级差别**
- 程序性能优化有多个层面
  - 算法，数据表达，过程，循环
- 只有理解了系统实现才能做到有效优化
  - 衡量程序性能的指标：执行时间、内存占用、能耗等。找出瓶颈（perf, DTrace等）
  - 了解程序的编译、执行过程中的细节，如内存访问模式
    - 例：内存访问模式影响程序性能
  - 理解如何在不破坏代码模块性和通用性的前提下优化性能



# 内存性能影响程序性能

```
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];  
}
```

4.3ms

```
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];  
}
```

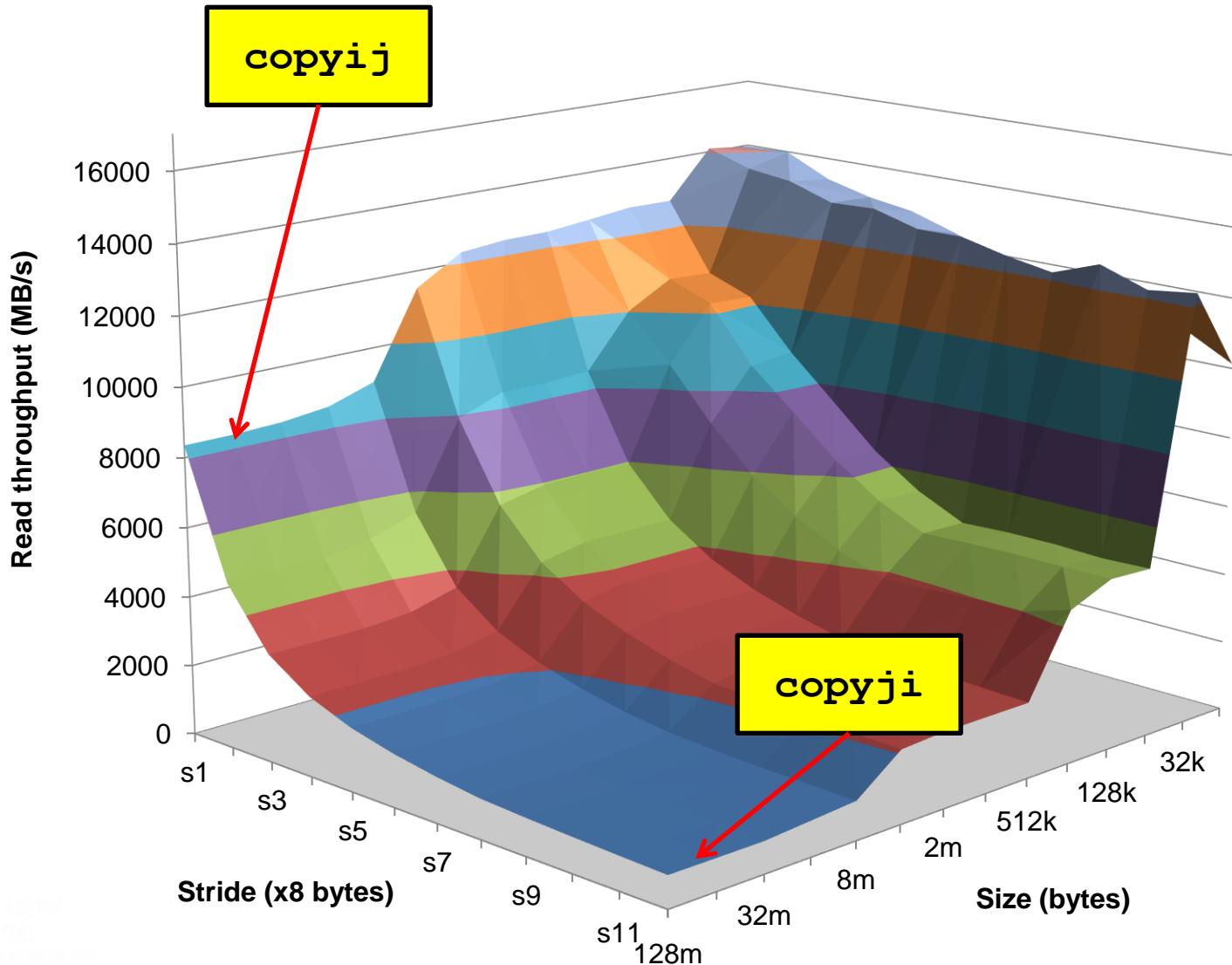
81.8ms

2.0 GHz Intel Core i7 Haswell

- 内存是分层组织的
- 程序性能取决于内存访问模式
  - 例如：如何访问内存中的二维数组、多维数组



# 为什么性能有这些差别





# 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，它永远对！你的感觉不可靠！**
  - 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，并不是那么简单

**学完本年度的课程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“系统思维”。  
即：站在“计算机系统”的角度考虑问题！**



# 什么是计算机系统?

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

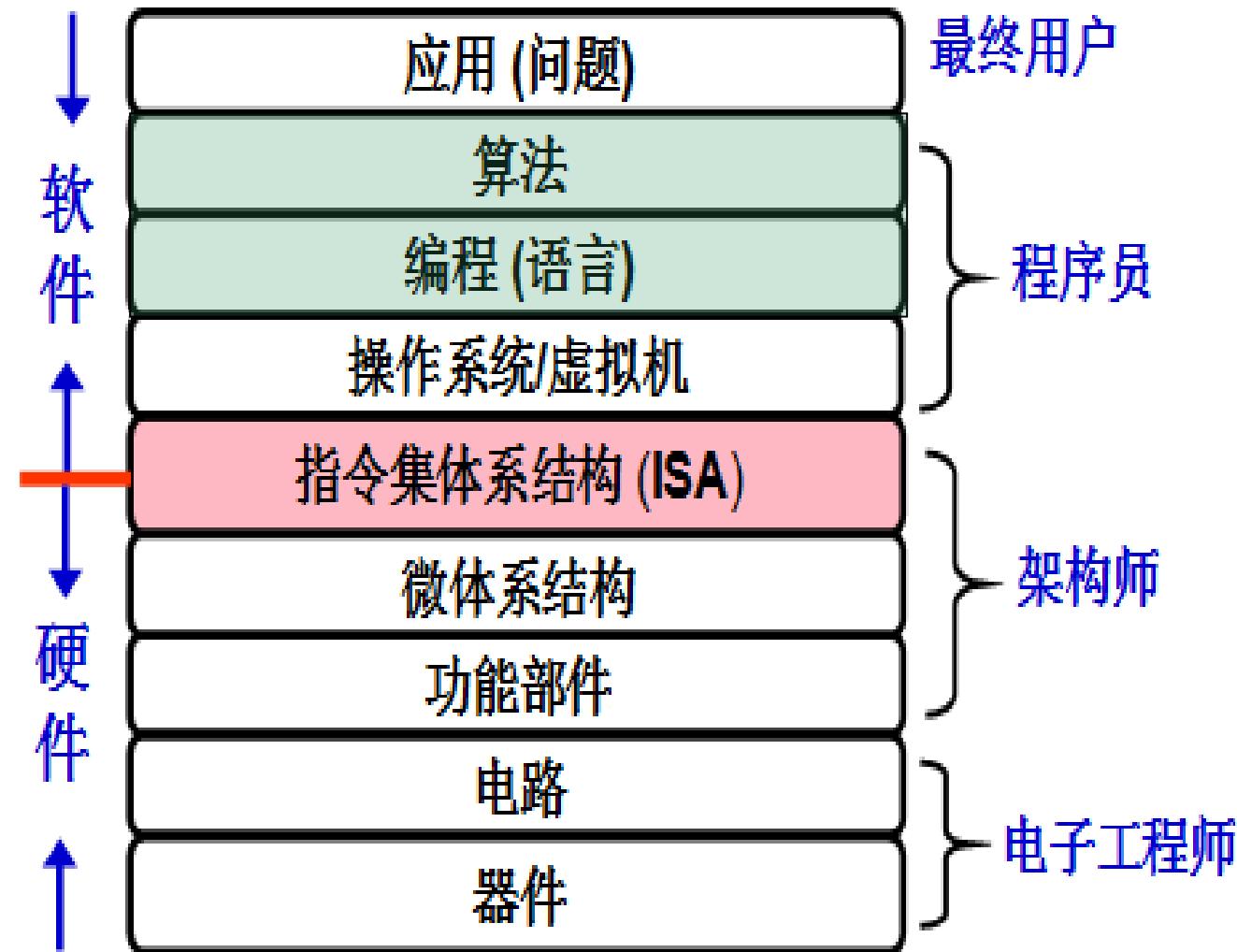
ISA

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

## 计算机系统抽象层的转换





## 单选题 1分

设置

# 计算机系统中，从上层应用到底层实现具体的顺序是什么？

A

应用（问题）->算法->操作系统->编程语言->指令集体系结构->微体系结构->硬件

B

应用（问题）->算法->编程语言->操作系统->指令集体系结构->微体系结构->硬件

C

应用（问题）->编程语言->算法->操作系统->指令集体系结构->微体系结构->硬件

D

应用（问题）->编程语言->算法->操作系统->指令集体系结构->微体系结构->硬件

提交



# 指令系统体系结构





## 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)
}
```

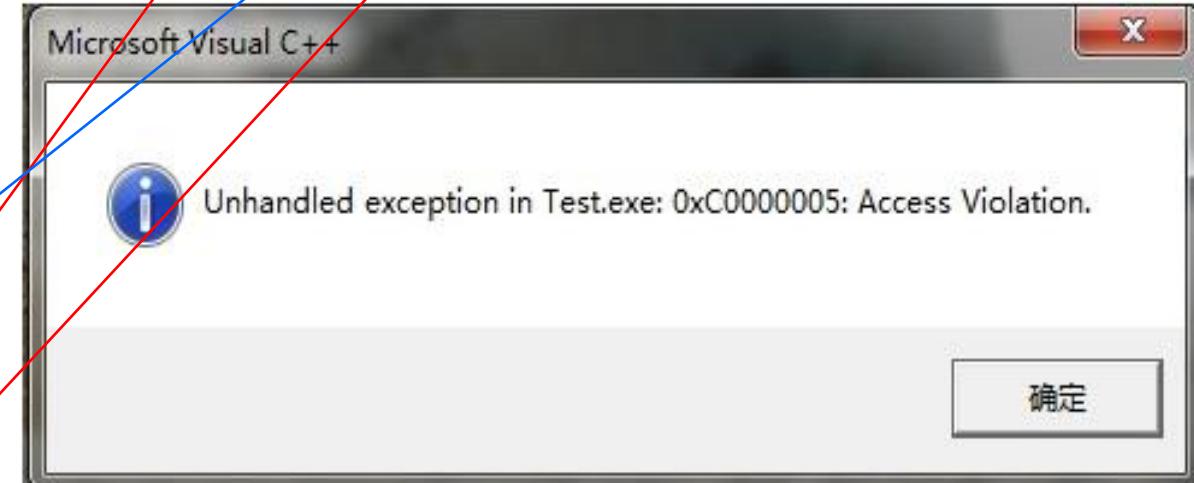
# 数据的表示

## 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



## 程序执行（存储器访问）



# 目录

- 课程基本信息
- 课程学习动机
- 有趣的四个现实
- 人工智能与系统 
- 计算机系统概述



# 我们应该关注哪些指标？

- 精度
  - 结果的质量
- 吞吐量
  - 海量数据的分析
  - 实时处理性能（视频）
- 延迟
  - 交互系统（如自动驾驶）
- 能耗
  - 嵌入式设备电池供电
  - 数据中心冷却问题
- 硬件成本
  - ¥ ¥ \$ \$

MNIST

3 6 8 1 7 9 6 6 9 1  
6 7 5 7 8 6 3 4 8 5  
2 1 7 9 7 1 2 1 1 5  
4 8 1 9 0 1 8 8 9 4  
7 6 1 8 6 4 1 5 6 0  
7 5 9 2 6 5 8 1 9 7  
1 2 2 2 2 3 4 4 8 0  
0 2 3 8 0 7 3 8 5 7  
0 1 4 6 4 6 0 2 4 3  
7 1 2 8 1 6 9 8 6 1

CIFAR-10



ImageNet



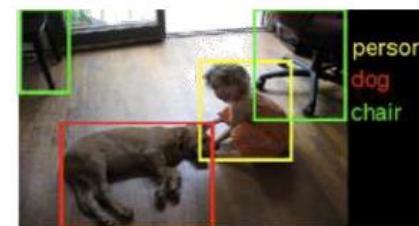
Embedded Device



Data Center



Computer Vision



Speech Recognition



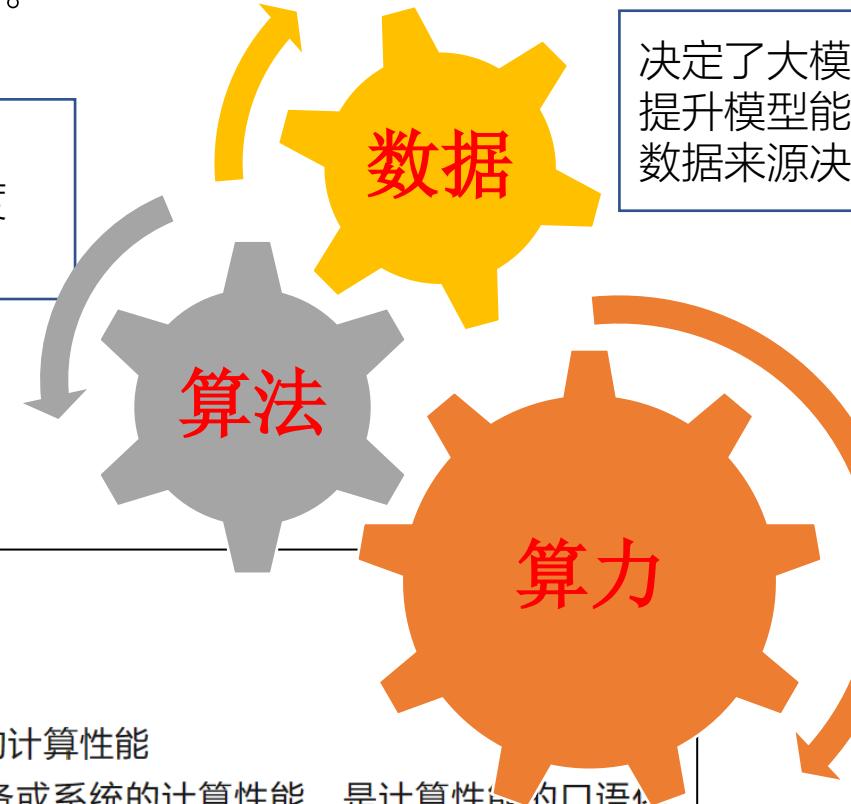
[Sze, CICC 2017]



# 大模型的三驾马车

- 算力已经成为科技创新和发展的重要支柱，算力经济成为数字经济社会的重要支柱，算力的规模也正在成为衡量一个国家经济增长幅度的一个重要指标。

大模型底层的运作原理方法从理论到应用有工程部属难度需要持续迭代优化



中文名：算力

英文名：Computility

学科：计算机系统结构

定义：算力是体现为用户实际效用的计算性能

实质：“算力”的本义是表示某个设备或系统的计算性能，是计算性能的口语化表达。随着智能时代的到来，智能计算的三要素——算力、算法、数据，逐渐成为社会的信息基础设施的重要组成部分，“算力”的内涵进一步扩大到用户能获得的体现为用户实际效用的计算性能。

决定了大模型的生成结构  
提升模型能力与准确度的前提  
数据来源决定了模型的适用范围





# AI的算力需求



和李世石下一盘围棋电费数千美元的AlphaGo

1.6万个CPU核学一周识别猫脸的谷歌大脑

- 今年1月平均每天约有1300万独立访客使用ChatGPT，对应芯片需求为3万多片英伟达A100GPU，初始投入成本约为8亿美元，每日电费在5万美元左右。
- GPT-3训练一次的成本约为140万美元，对于一些更大的LLM模型，训练成本介于200万美元至1200万美元之间。



# 算力已经成为瓶颈

- 黄仁勋在2023年2月财报会中表示“过去十年，通过提出新处理器、新系统、新互连、新框架和算法，并与数据科学家、AI 研究人员合作开发新模型，已使大语言模型的**处理速度提高了100万倍。**”
- 大规模深度学习模型的参数和数据量大幅提升，庞大的模型如果没有先进算力基础，训练耗时和成本将成为不可承受之重。

图：英伟达最近三代AI服务器芯片主要参数对比

	V100	A100	H100
FP64	7TFLOPS	9.7TFLOPS	26 TFLOPS
FP32	14TFLOPS	19.5TFLOPS	51 TFLOPS
GPU 显存	32/16GB	80GB	80GB
GPU 显存带宽	900 GB/s	1935GB/s	2TB/s
最大设计功耗	250W	300W	300-350W
Interconnect	NVLink: 300 GB/s PCIe: 32 GB/s	NVLink: 600GB/s PCIe: 64GB/s	NVLink: 600GB/s PCIe: 128GB/s

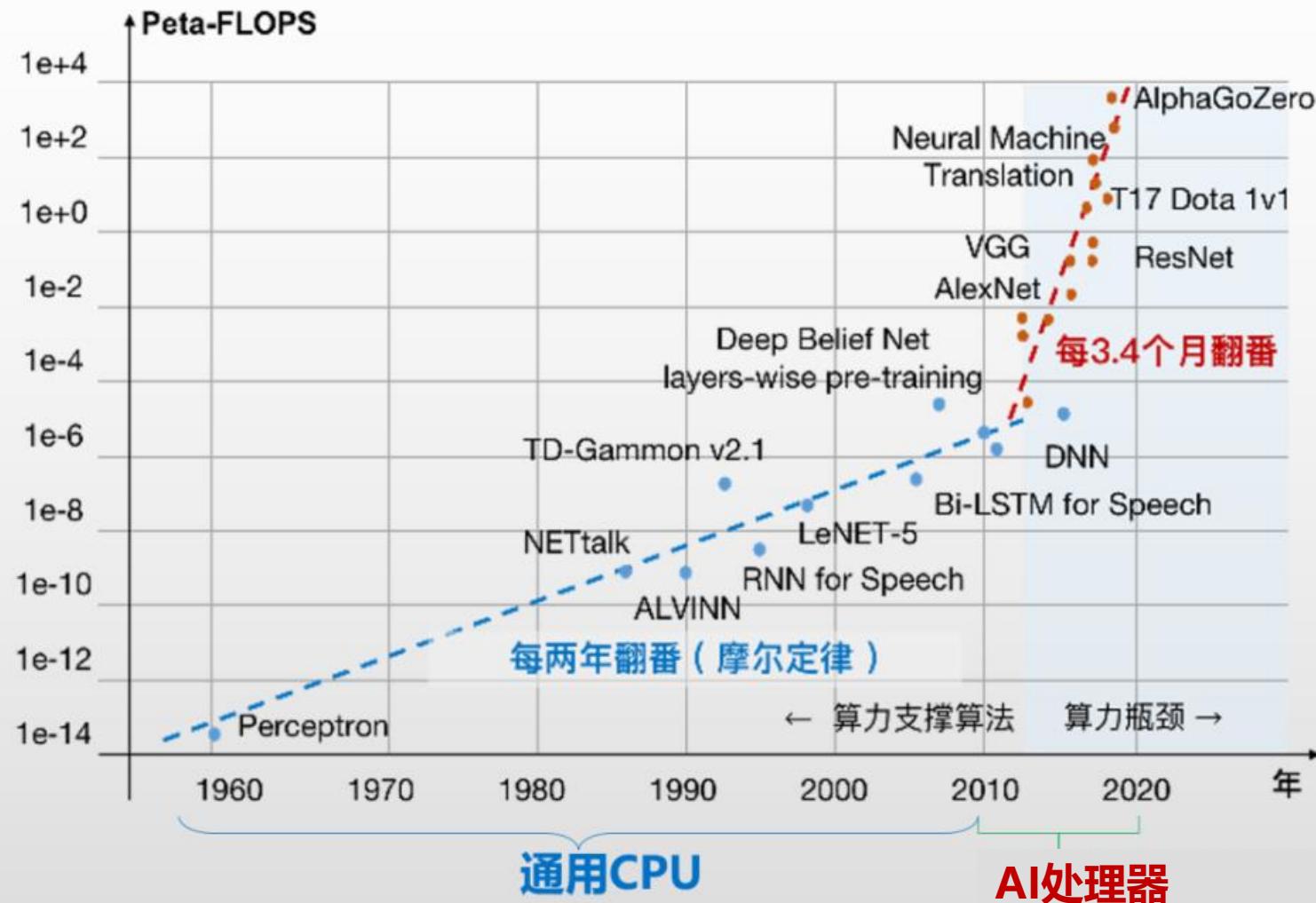
图：近年主要模型参数规模情况

时间	机构	模型名称	模型规模	数据规模	使用单块V100的训练时间
2018.6	OpenAI	GPT-1	110M	4GB	3天
2018.10	Google	BERT	330M	16GB	50天
2019.2	OpenAI	GPT-2	1.5B	40GB	200天
2019.7	Facebook	RoBERTa	330M	160GB	3年
2019.10	Google	T5	11B	800GB	66年
2020.6	OpenAI	GPT-3	175B	2TB	355年



# 算力已经成为瓶颈

- 算力需求每3.4个月翻番





# 算力的理想与现实





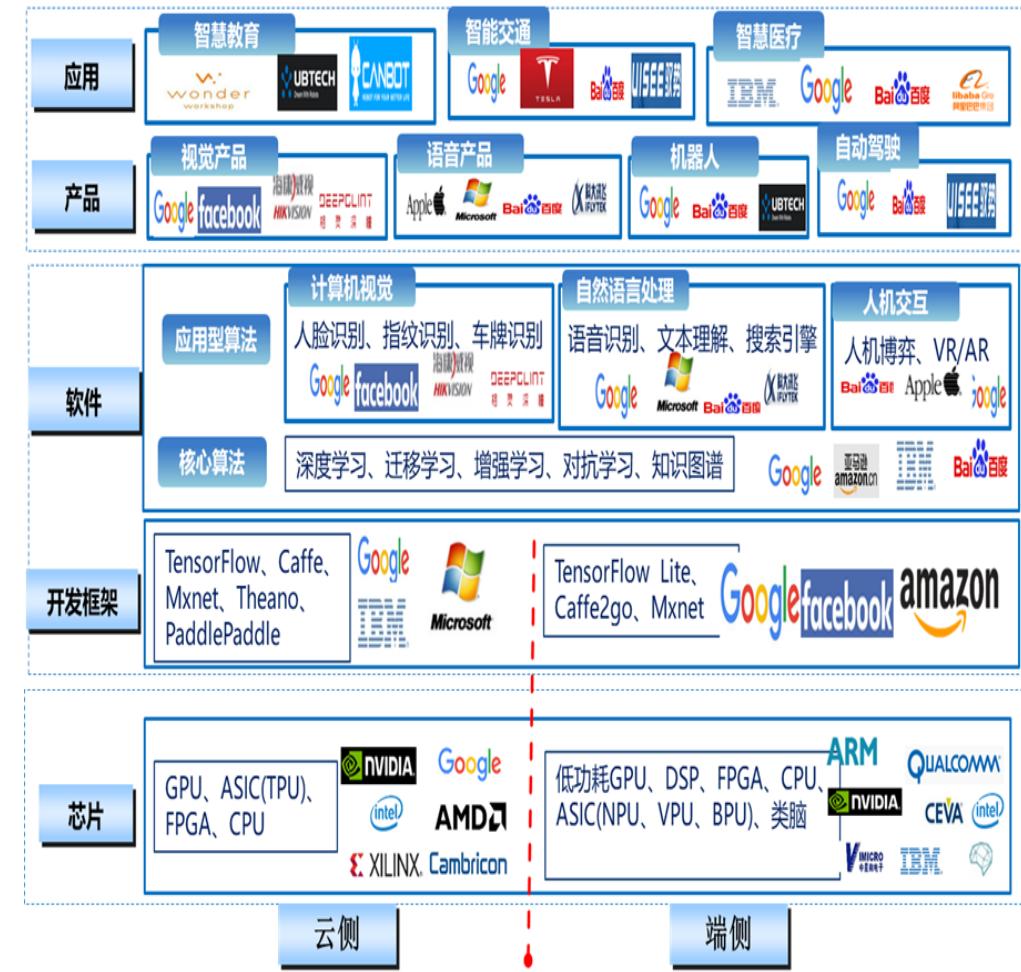
# 人工智能技术分层

应用层

算法层

系统层

芯片层





# 中美人工智能竞争态势分析

- 我国人工智能产业发展潜力被国际公认与美国相当，但中美差距仍不乐观。

应用层

算法层

系统层

芯片层

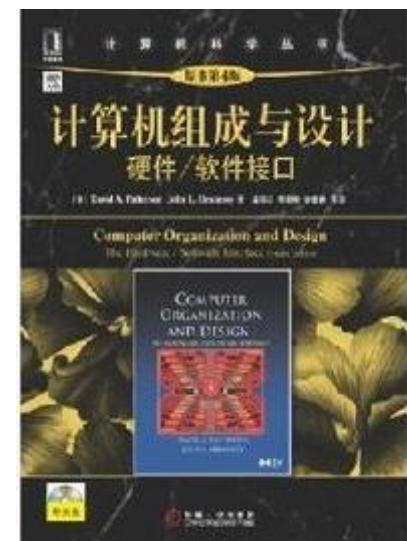
	中国	落后5年	美国
兴起时间	<input type="checkbox"/> 早期人工智能企业创办于1996年		<input type="checkbox"/> 早期人工智能企业创办于1991年
企业数量	<input type="checkbox"/> 国内人工智能企业数量约占全球的23%	相差45%	<input type="checkbox"/> 美国人工智能企业数量约占全球的42%
投融资情况	<input type="checkbox"/> 国内投融资总额占全球的33%	相差34%	<input type="checkbox"/> 美国投融资总额占全球的50%
基础支撑产业	<input type="checkbox"/> AI芯片处于起步阶段，少数定制化领域有局部突破	短板严重	<input type="checkbox"/> 全面领先，尤其是GPU市场基本垄断
软件算法产业	<input type="checkbox"/> 基础平台起步晚、生态弱，应用软件快速发展具散点优势	生态不足	<input type="checkbox"/> 基础平台形成完整生态，应用软件起步早，前沿领域具领先优势。

人工智能底层科技的缺失可能使得我国智能产业成为空中楼阁



# 业界在系统方面的投入

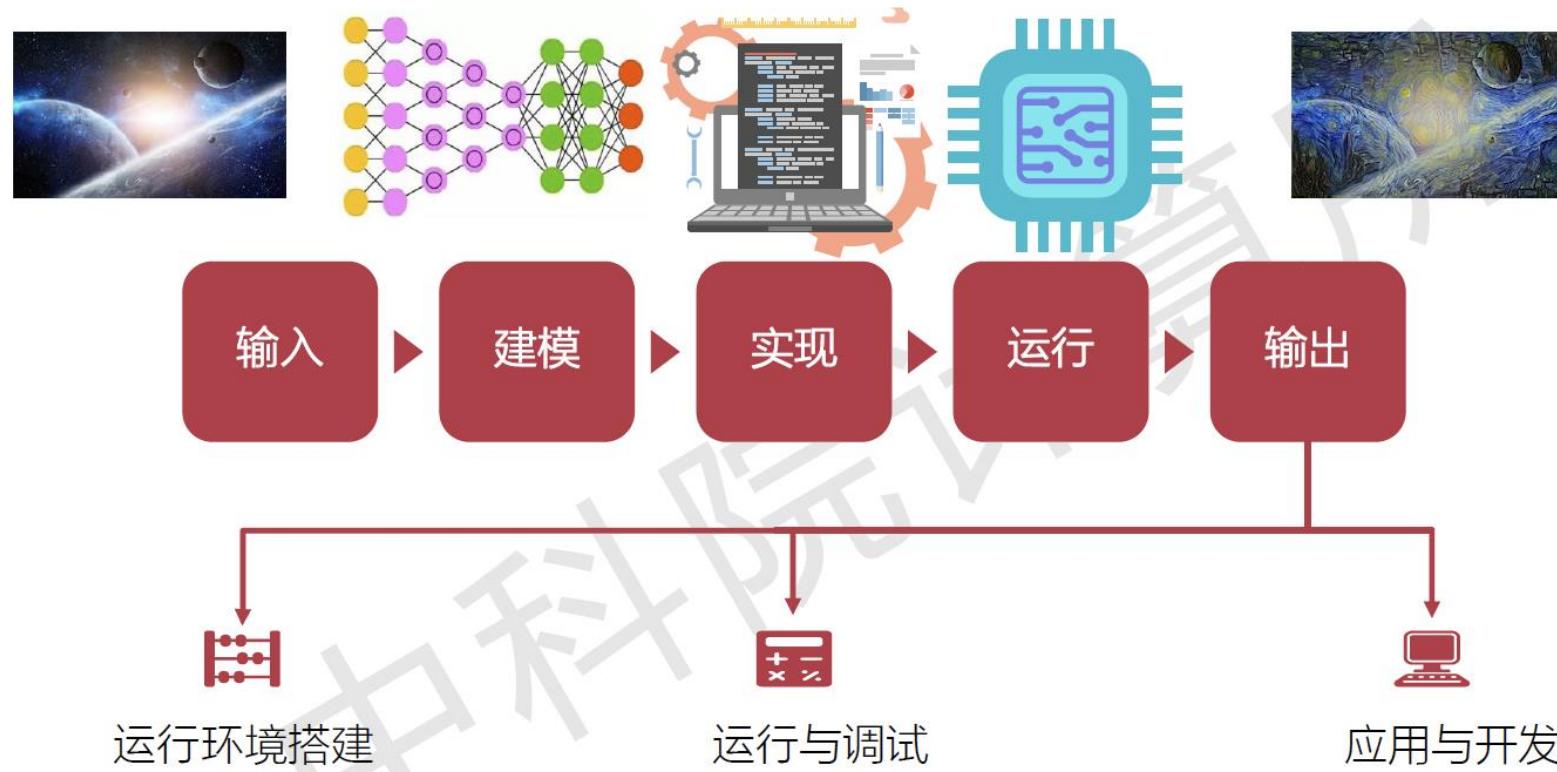
- 谷歌是世界上最大的AI算法研究团队，然而
- ✓ 谷歌董事长Jonh Hennessy是计算机系统结构科学家，图领奖得主。同David Patterson共同创建RISC架构和经典MIPS, RISC V处理器
- ✓ 谷歌AI的总领导Jeff Dean 是计算机系统结构研究者（Map Reduce）
- ✓ 谷歌最令人瞩目的三个进展都是系统（TensorFlow, AlphaGo, TPU），而不仅仅是某个特定算法，算法是系统的一个环节





# 智能计算系统

- 国科大、中科大、北大，北航
- 人大2023年春季开课
- 2025年春季人工智能必修，信息学院三选二





# AI Systems 海外课程

- UCB
  - <https://ucbrise.github.io/cs294-ai-sys-sp19/>
- MIT
  - [MIT 6.5940 Fall 2023 TinyML and Efficient Deep Learning Computing](#)
- CMU
  - <https://catalyst.cs.cmu.edu/15-884-mlsys-sp21/>
- Microsoft
  - <https://github.com/microsoft/AI-System>



# 目录

- 课程基本信息
- 课程学习动机
- 有趣的四个现实
- 人工智能与系统
- 计算机系统概述





# 从高级语言到硬件执行

- 例子：最简单的C语言程序

```
#include <stdio.h>
```

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



# 从高级语言到硬件执行

## • 信息的存储和传输格式

- 计算机中其他的文件基本都是二进制文件，即直接用二进制的0、1串来表示信息
- 包括磁盘文件、存储器中的程序、存储器中存放的用户数据，以及网络上传输的数据
- 字符在计算机中是以ASCII码的形式来表示的
  - 每个字符用一个0~255的数字来表示，在计算机中占用一个字节（每个字节固定为8个二进制位）

```
# i n c l u d e <sp> < 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 <sp> m a i n ( ) { \n
104 62 10 10 105 110 116 32       109 97 105 110 40 41 123 10
<sp><sp><sp>p r i n t f ( " h e l l o , \n
32 32 32 . /hello 运行程序之前发生了什么? 108 111 44
<sp>w o
32 119 111 114 108 100 33 92 110 34 41 59 10 125
```



## Hello world程序：从C语言到机器语言

## 标准输入输出函数库

## 函数printf在标准输入 输出函数库中定义

```
.data
msg:    db "hello, world",10
len:    equ $-msg
...
main:
        mov     edx,len
        mov     ecx,msg
        mov     ebx,1
        mov     eax,4
```

# 汇编语言

```
#include <stdio.h>
```

```
int main ()
```

1

```
→ printf ( "hello, world\n" );  
return 0;
```

C语言

00000000	7F	45	4C	46	01	01	01	00	00	00	00	00	00	00	00
00000010	02	00	03	00	01	00	00	00	80	80	04	08	34	00	00
00000020	F8	00	00	00	00	00	00	00	34	00	20	00	02	00	28
00000030	05	00	04	00	01	00	00	00	00	00	00	00	00	80	04
00000040	00	80	04	08	A2	00	00	00	A2	00	00	00	05	00	00
00000050	00	10	00	00	01	00	00	00	A4	00	00	00	A4	90	04
00000060	A4	90	04	08	09	00	00	00	09	00	00	00	06	00	00
00000070	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00
00000080	BA	09	00	00	00	B9	A4	90	00	00	00	00	00	00	B8
00000090	04	00	00	00	CD	80	BB	00	00	00	00	00	00	00	00
000000A0	CD	80	00	00	48	69	20	57	6F	72	6C	64	0A	00	00

# 机器语言



# 从高级语言源代码到可执行程序文件

高级语言  
源程序 *helloworld.c*

↓ *helloworld.i*

高级语言  
程序

Pre-processor  
预处理器(#开头)

编译过程

↓ *helloworld.s*

汇编语言  
中间代码

Compiler 编译器

↓ *helloworld.o*

目标文件  
(机器语言)

Assembler 汇编器

↓ *helloworld.exe*

可执行文件  
(机器语言)

Linker 链接器

汇编过程

*printf.o*

库文件  
(机器语言)

反汇编



## 单选题 1分

设置

下列关于高级语言的描述错误的是：

- A 高级语言与具体的计算机结构无关
- B 一条高级语言的语句会对应很多条机器语言语句
- C 高级语言程序必须被翻译成机器语言才能被执行
- D 高级语言程序只要转换成机器级目标代码就能执行

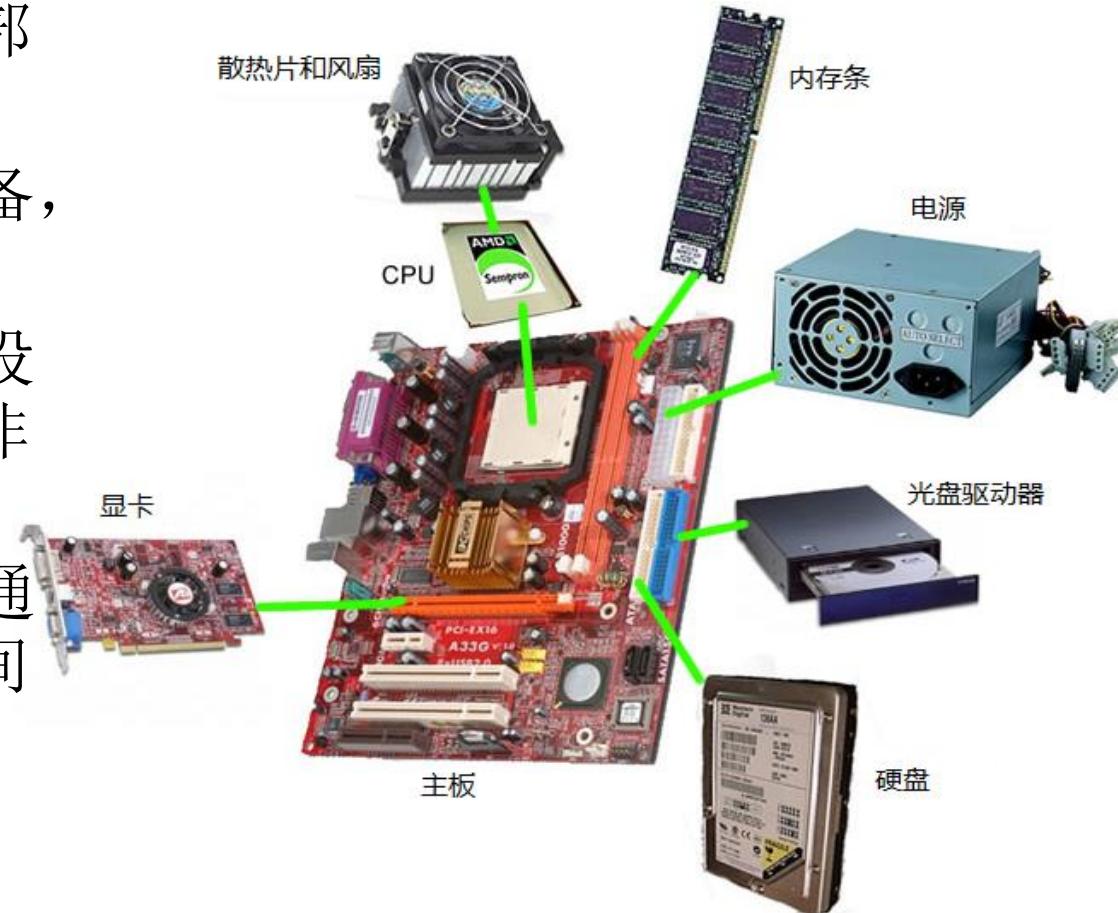
提交



# 从高级语言到硬件执行

- 计算机的主要硬件结构

- 1) CPU是计算机的核心部件，解释或执行指令
- 2) 主存储器临时存储设备，存储数据和指令
- 3) 磁盘是一种慢速存储设备，容量大，价格低，非易失
- 4) 总线连接各个设备的通道，负责在各个设备之间传输数据





# 现代电子计算机之父：冯·诺依曼

- 美籍匈牙利数学家
  - 1926年，获匈牙利布达佩斯大学数学博士
  - 1931年，成为普林斯顿大学终身教授
  - 1933年，转入普林斯顿高等研究院，与爱因斯坦等人成为该院最初的四位教授之一
  - 1944年，参加原子弹的研究工作
  - 1951~1953，任美国数学会主席
  - 1954年，任美国原子能委员会委员

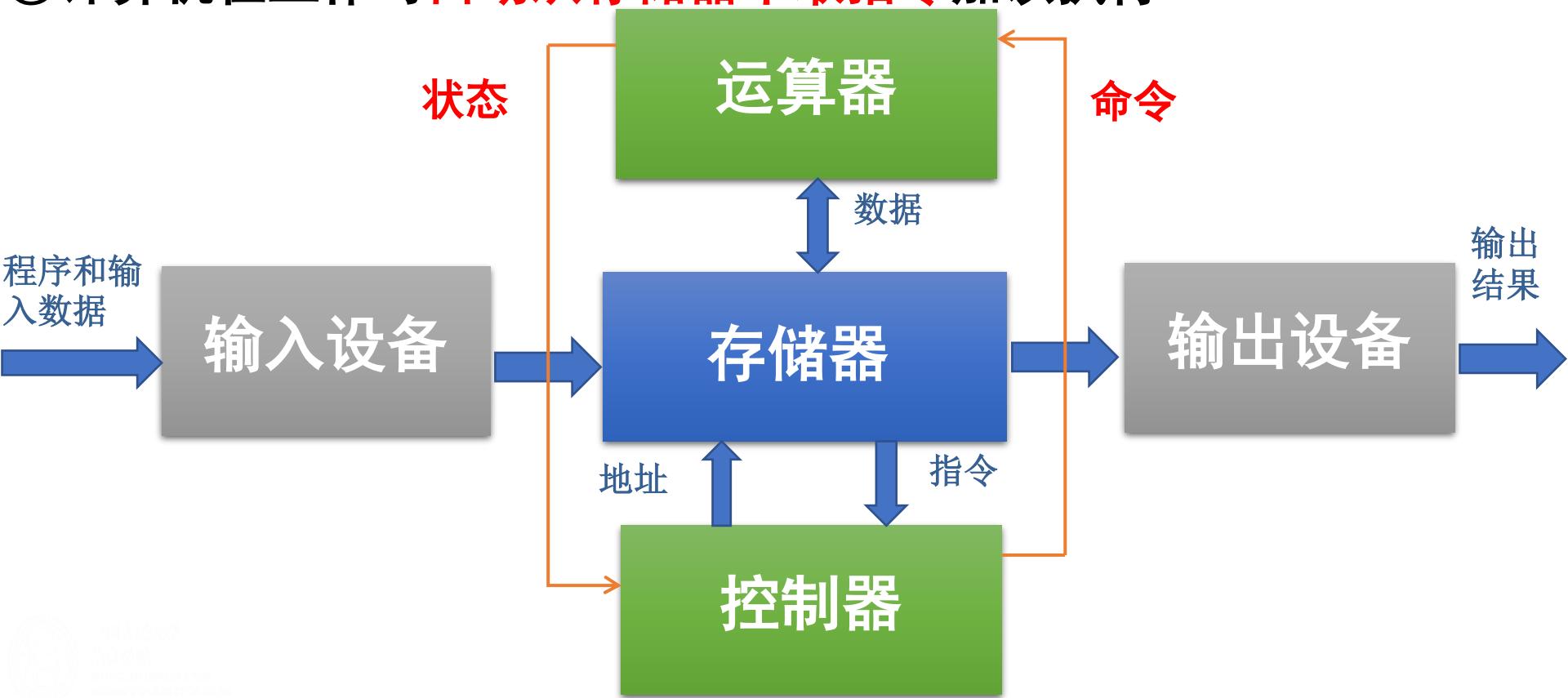


约翰·冯·诺依曼  
John Von Neumann  
1903~1957



# 冯·诺依曼结构

- ①计算机应由运算器，控制器，存储器，输入设备和输出设备共5个部分组成
- ②程序和数据均以二进制形式存放在存储器中，存放位置由存储器地址指定
- ③计算机在工作时自动从存储器中取指令加以执行





# 计算机的硬件

下一任务单位位置

当前任务单

CPU

IR

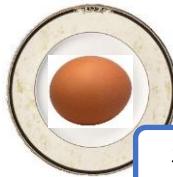
PC  
1

控制器



运算器

A



寄存器

B

1



餐馆—计算机

任务  
(指令)

主存

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

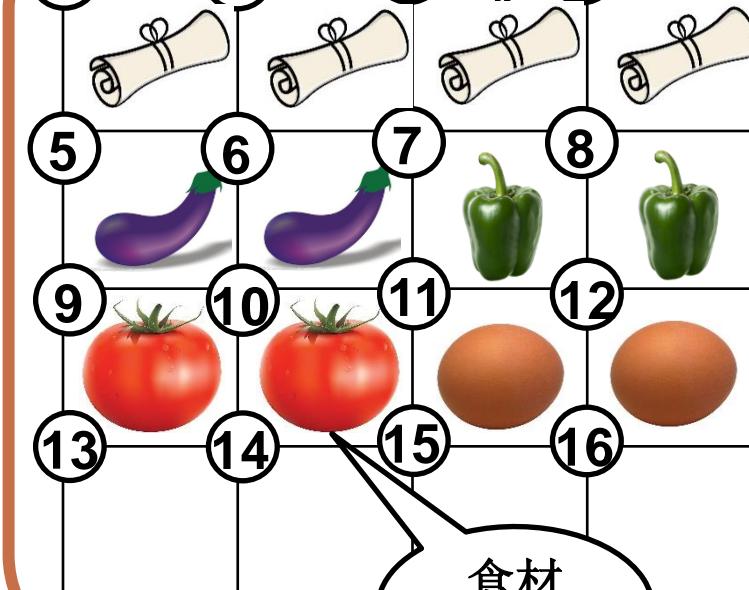
13

14

15

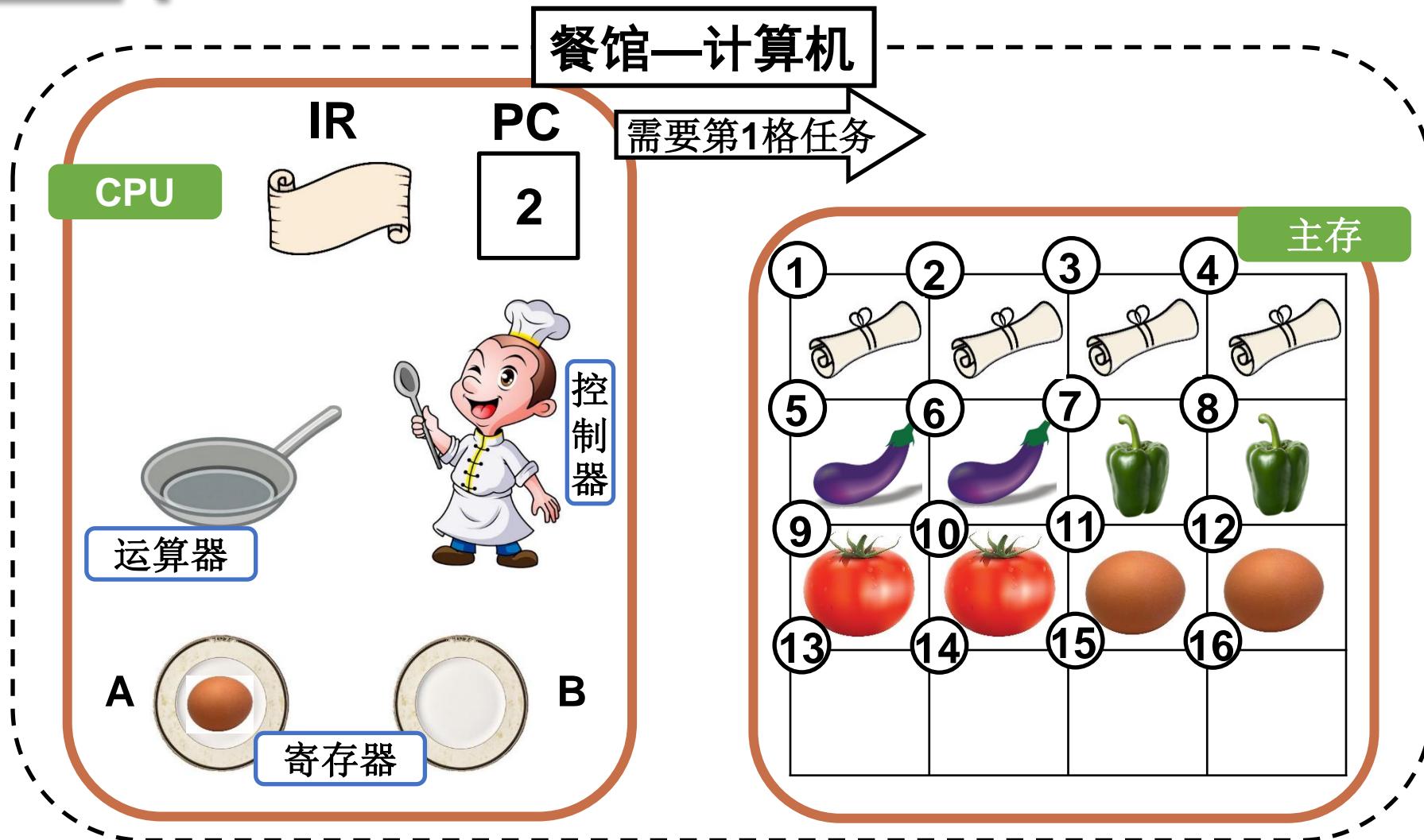
16

食材  
(数据)





# 第一步：取任务单





# 第二步：分析任务

餐馆—计算机

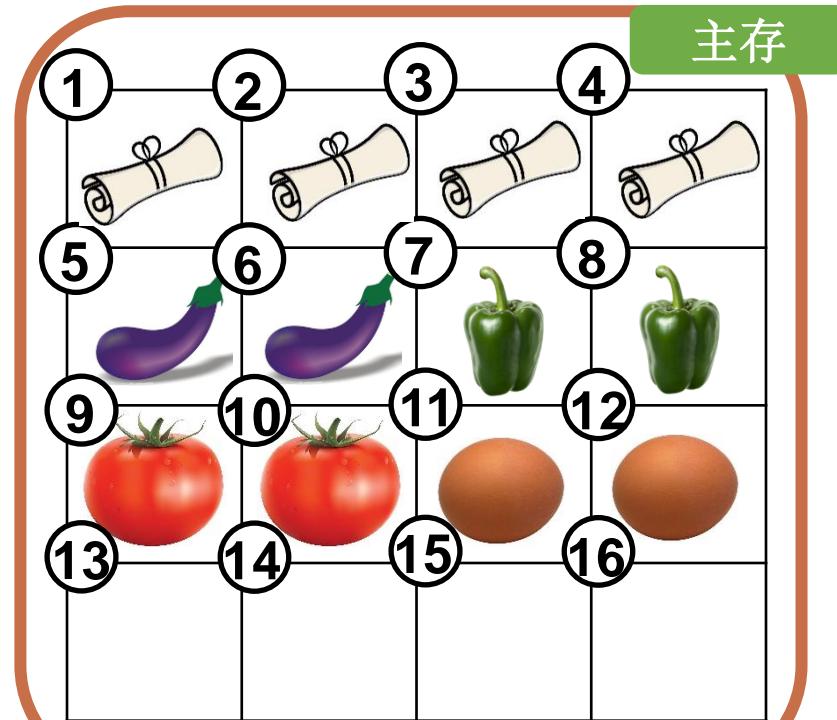
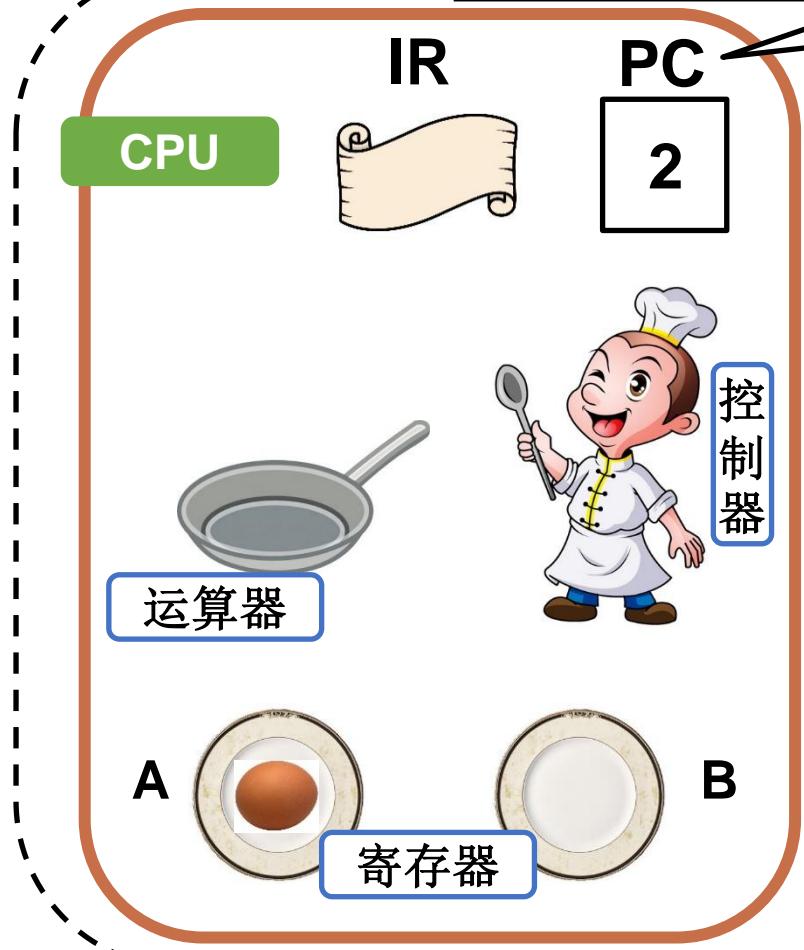
制作方法:

原料位

存放:

寄存器A

- ① 取第9格食材
- ② 取寄存器A食材
- ③ 炒一分钟
- ④ 存放到寄存器A

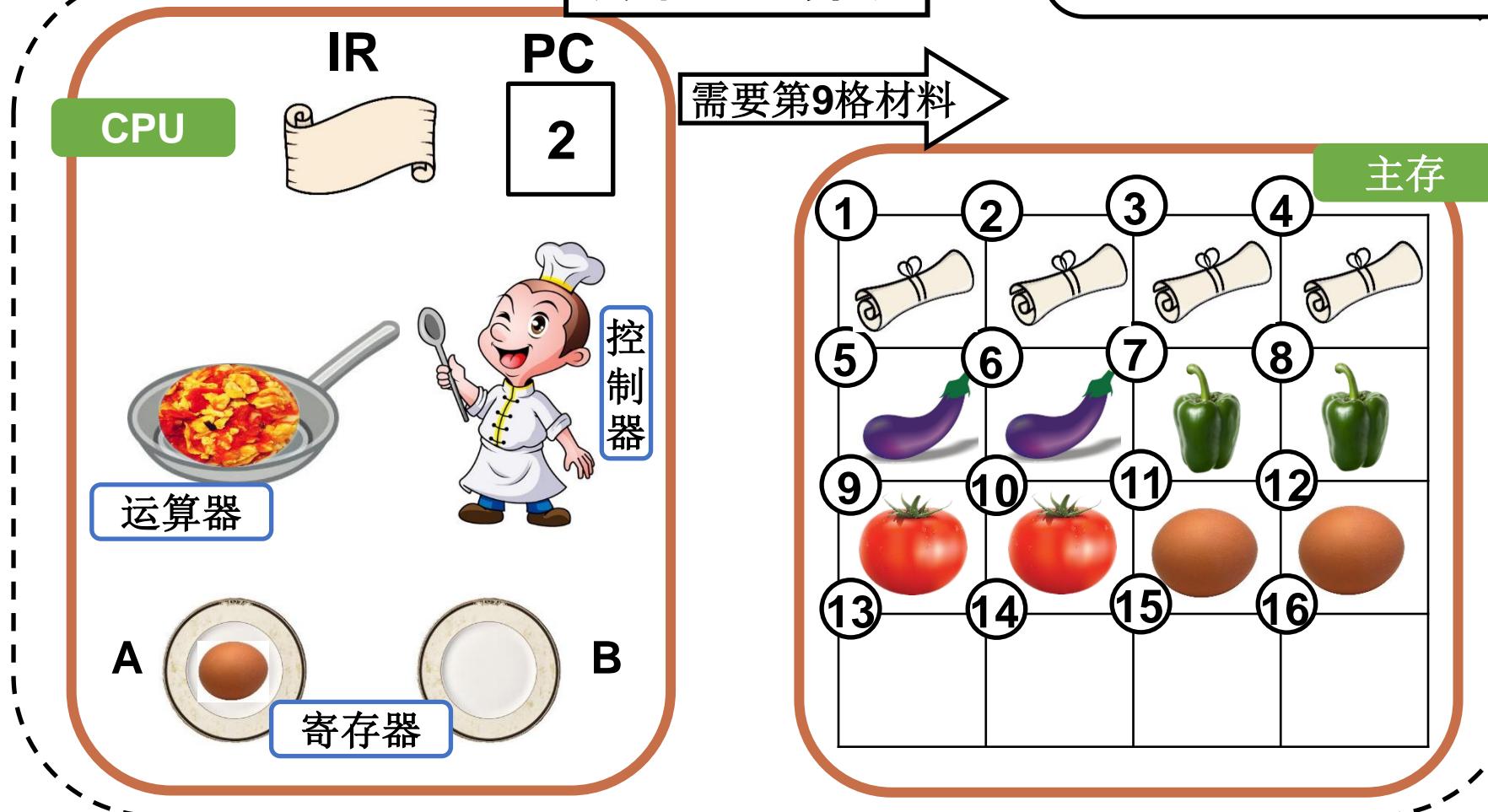




# 第三步：执行任务

- ① 取第9格食材
- ② 取寄存器A食材
- ③ 炒一分钟
- ④ 存放到寄存器A

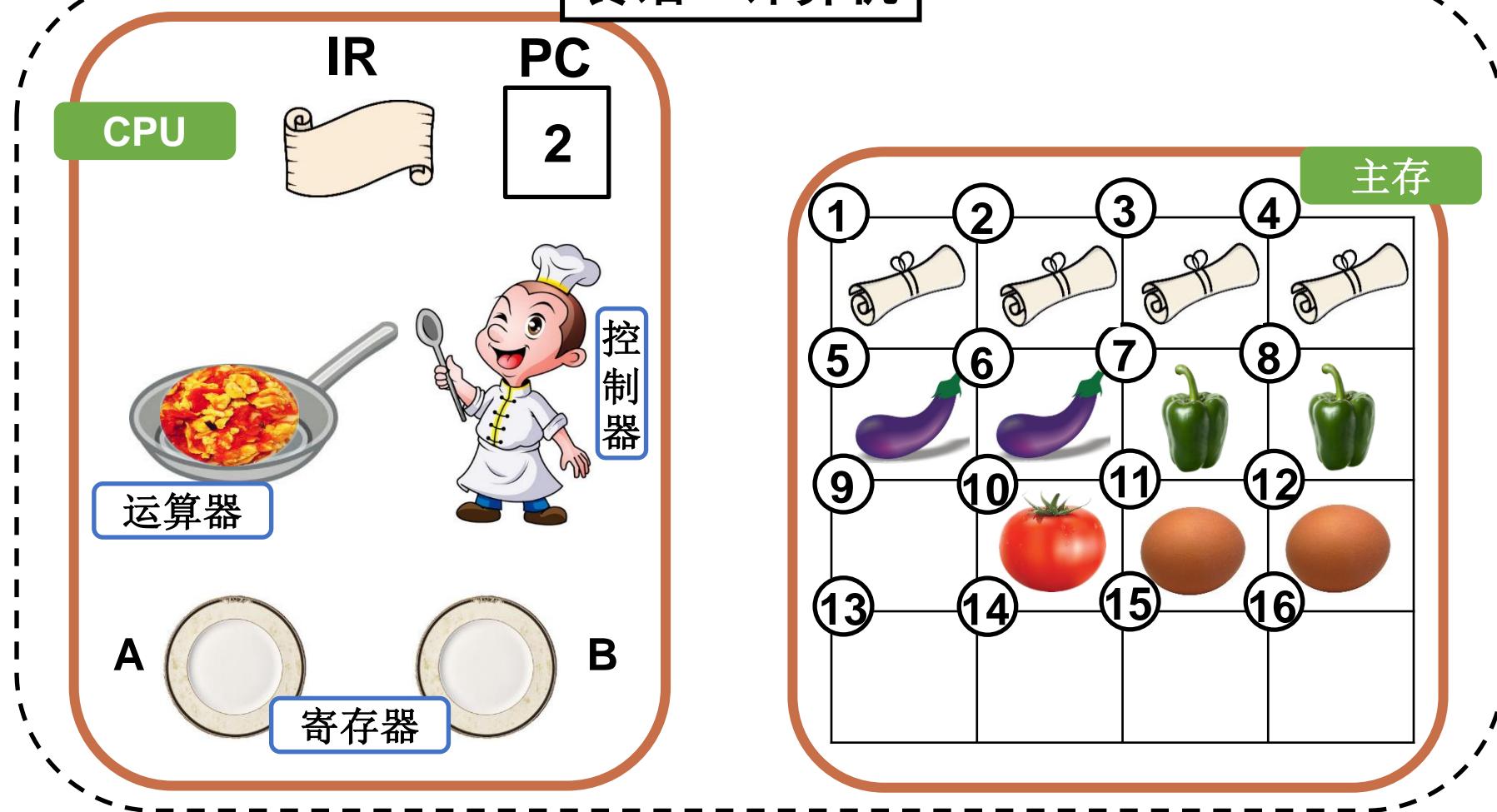
餐馆—计算机





# 第四步：保存结果

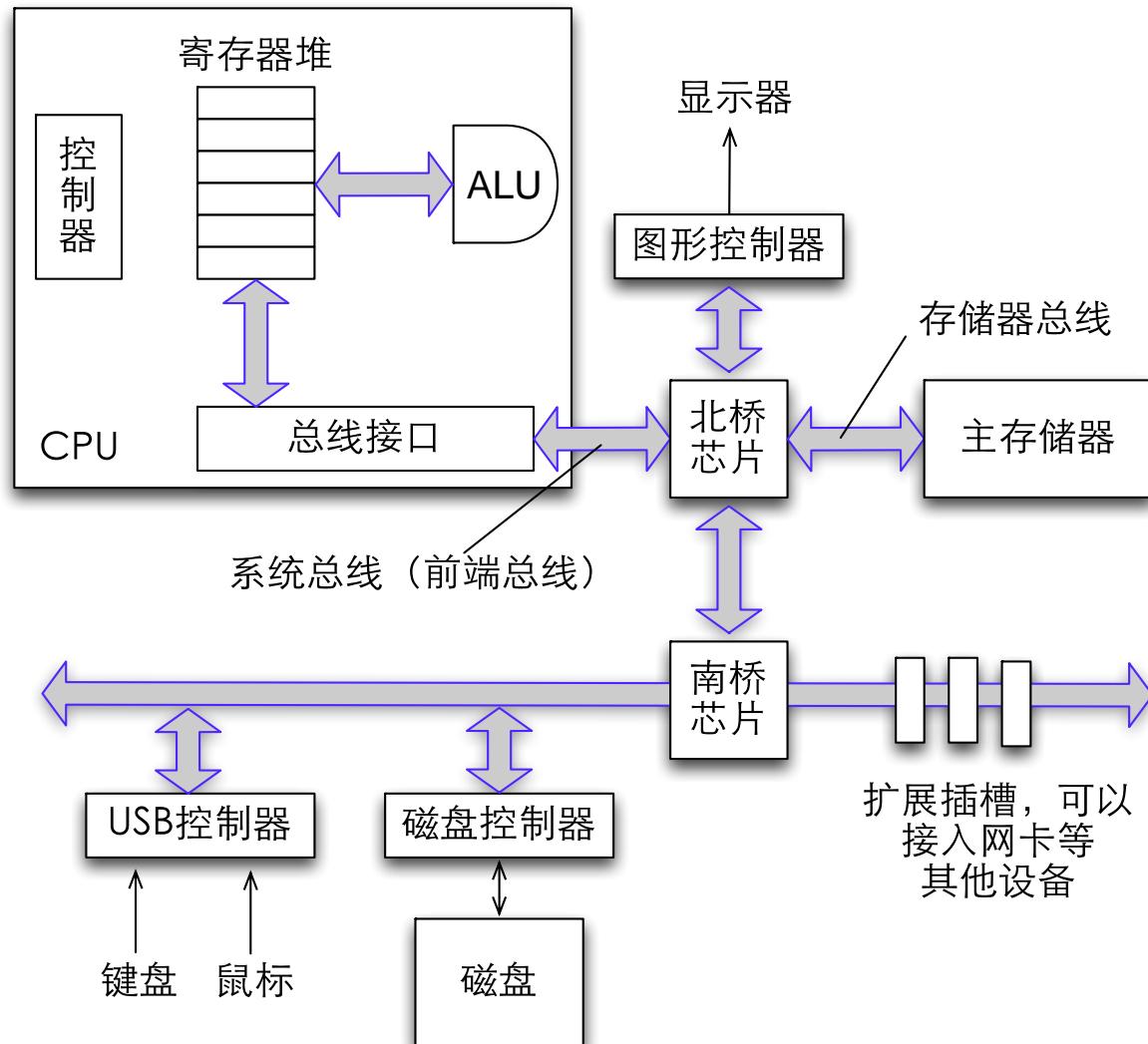
餐馆—计算机





# 从高级语言到硬件执行

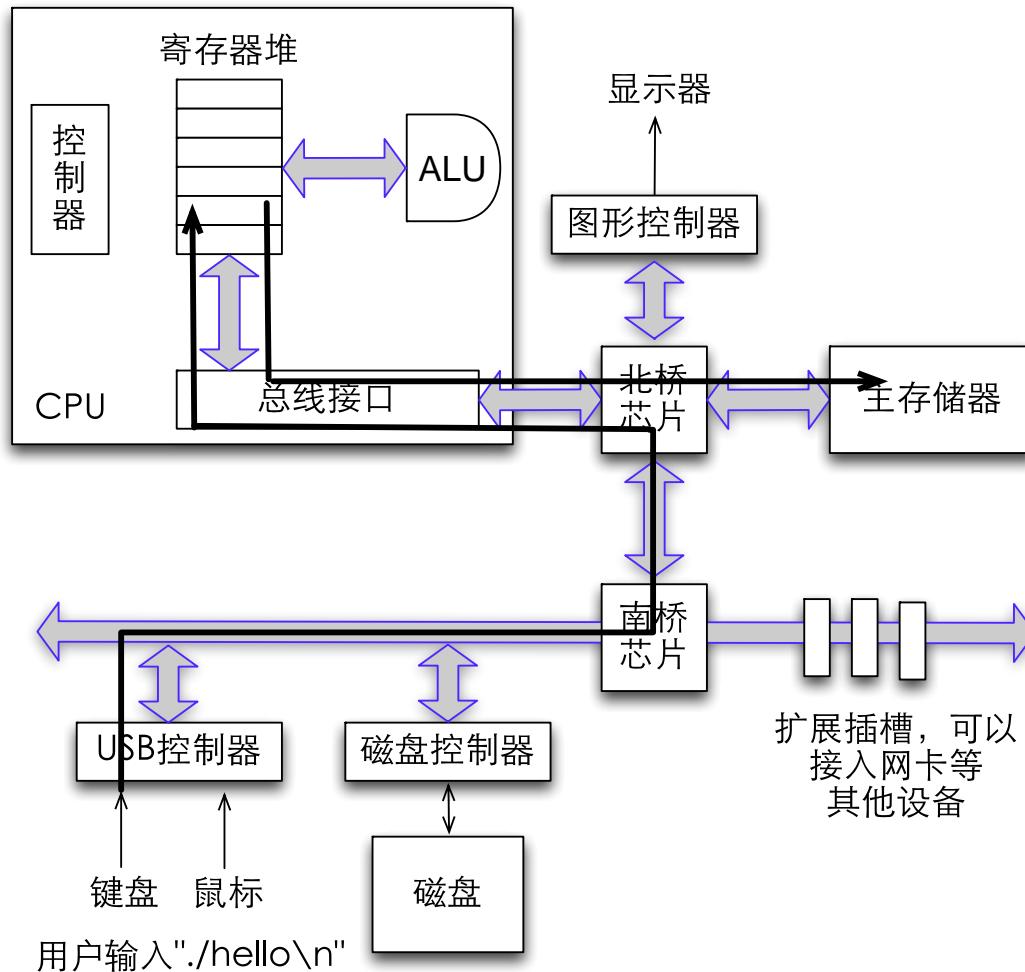
- 计算机的主要硬件结构





# 从高级语言到硬件执行

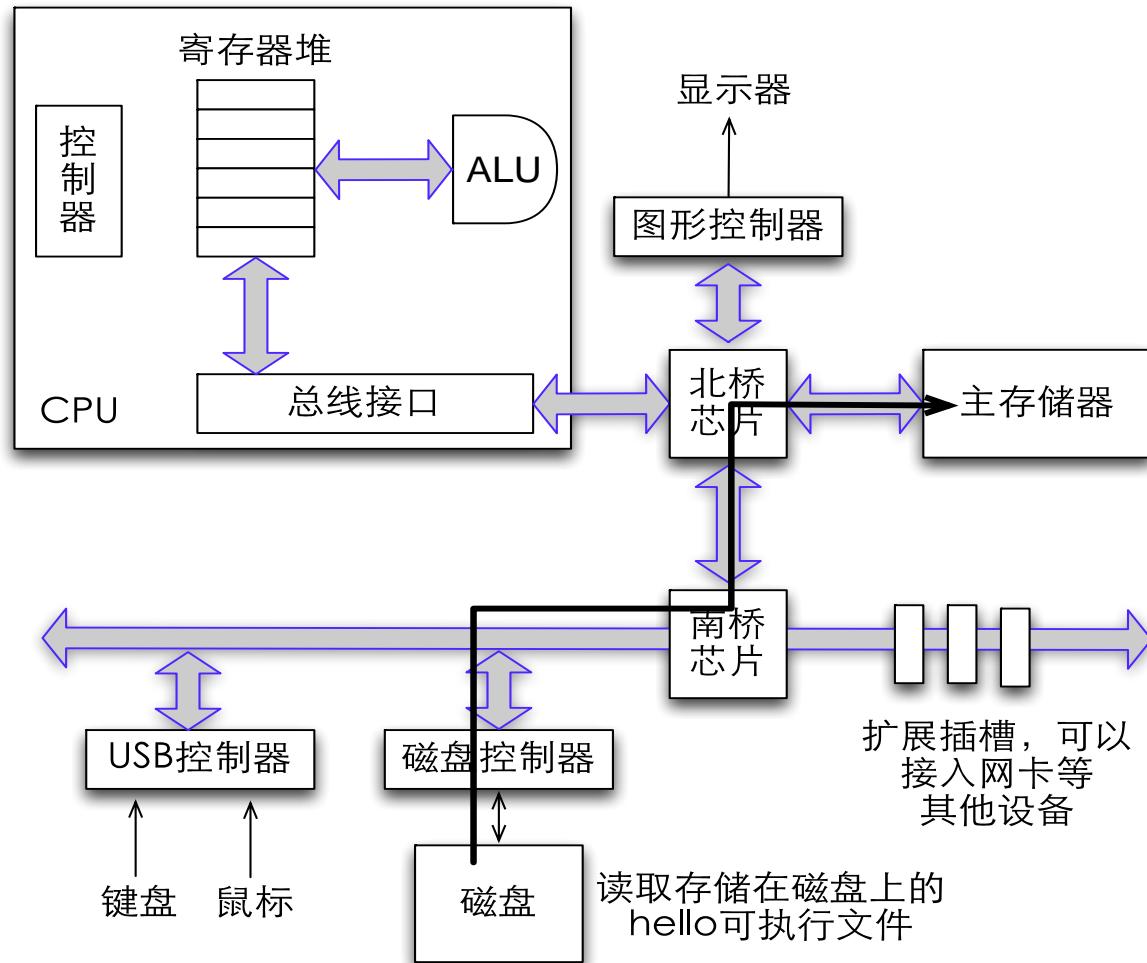
- **hello world**程序在计算机硬件上的执行
  - 从键盘读入这些信息，并把它存放到主存储器中





# 从高级语言到硬件执行

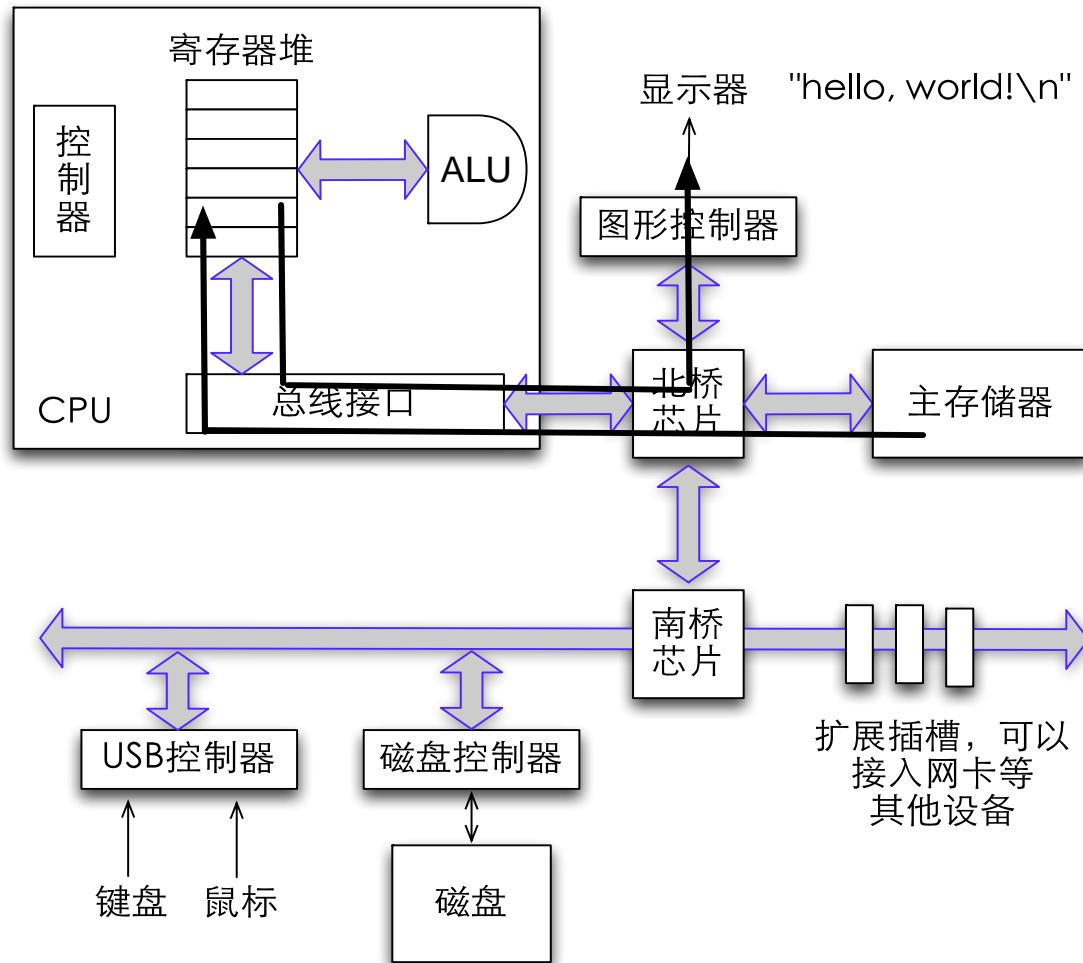
- hello world程序在计算机硬件上的执行
  - 从磁盘加载可执行程序文件hello到主存储器





# 从高级语言到硬件执行

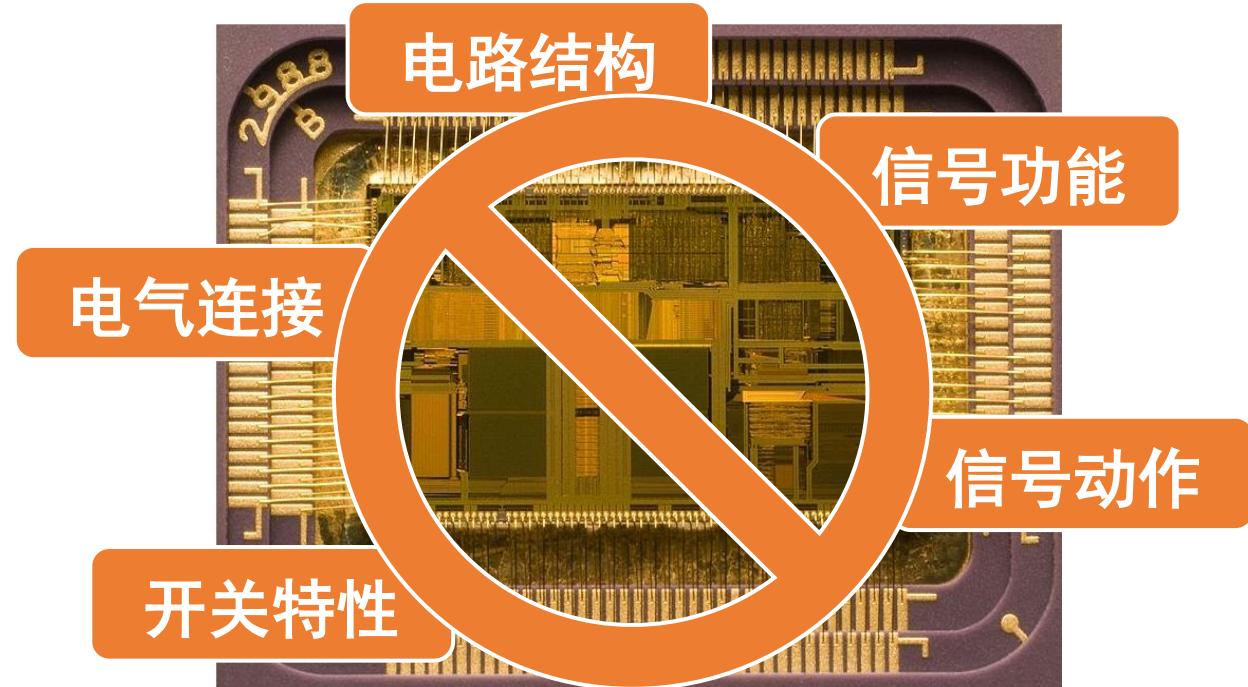
- hello world程序在计算机硬件上的执行
  - 将输出字符串从内存写到显示器





# 处理器的编程结构

- 处理器的编程结构的作用
  - 便于从软件视角了解计算机系统的操作和运行
  - 编程人员可以不必知道微处理器复杂的内部结构和引脚信号





# 处理器的编程结构

- 编程人员看到的处理器的软件结构模型

处理器内部：

- ① 寄存器的功能、操作和限制
- ② 程序设计中如何使用寄存器

处理器外部：

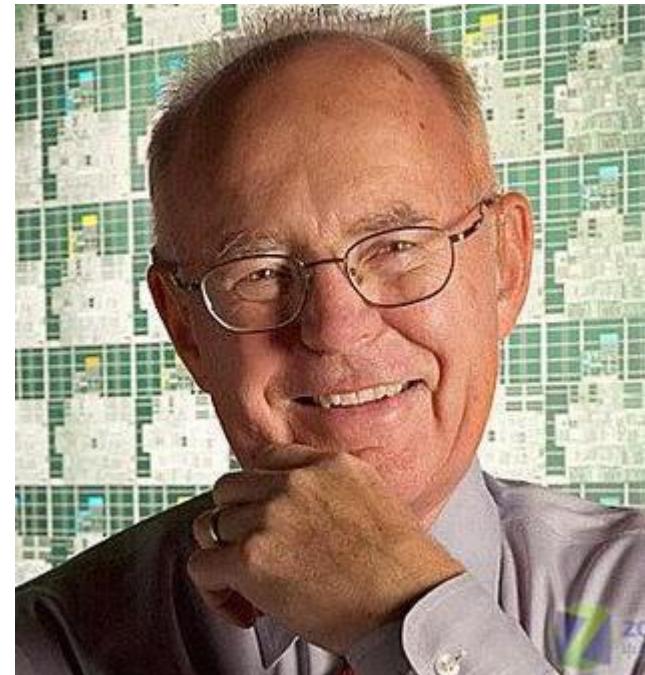
- ① 存储器中数据如何组织
- ② 如何从存储器中取指令和数据



# 摩尔定律 (Moore's Law)

“当价格不变时，集成电路  
上可容纳的晶体管数目，约每  
隔18个月便会增加一倍，性能  
也将提升一倍。”

- ① 1965年，戈登·摩尔提出了摩尔定律的原型（“最低元件价格下的复杂性每年大约增加一倍”），后经多次修正
- ② 摩尔定律并非数学、物理定律，而是对发展趋势的一种分析预测



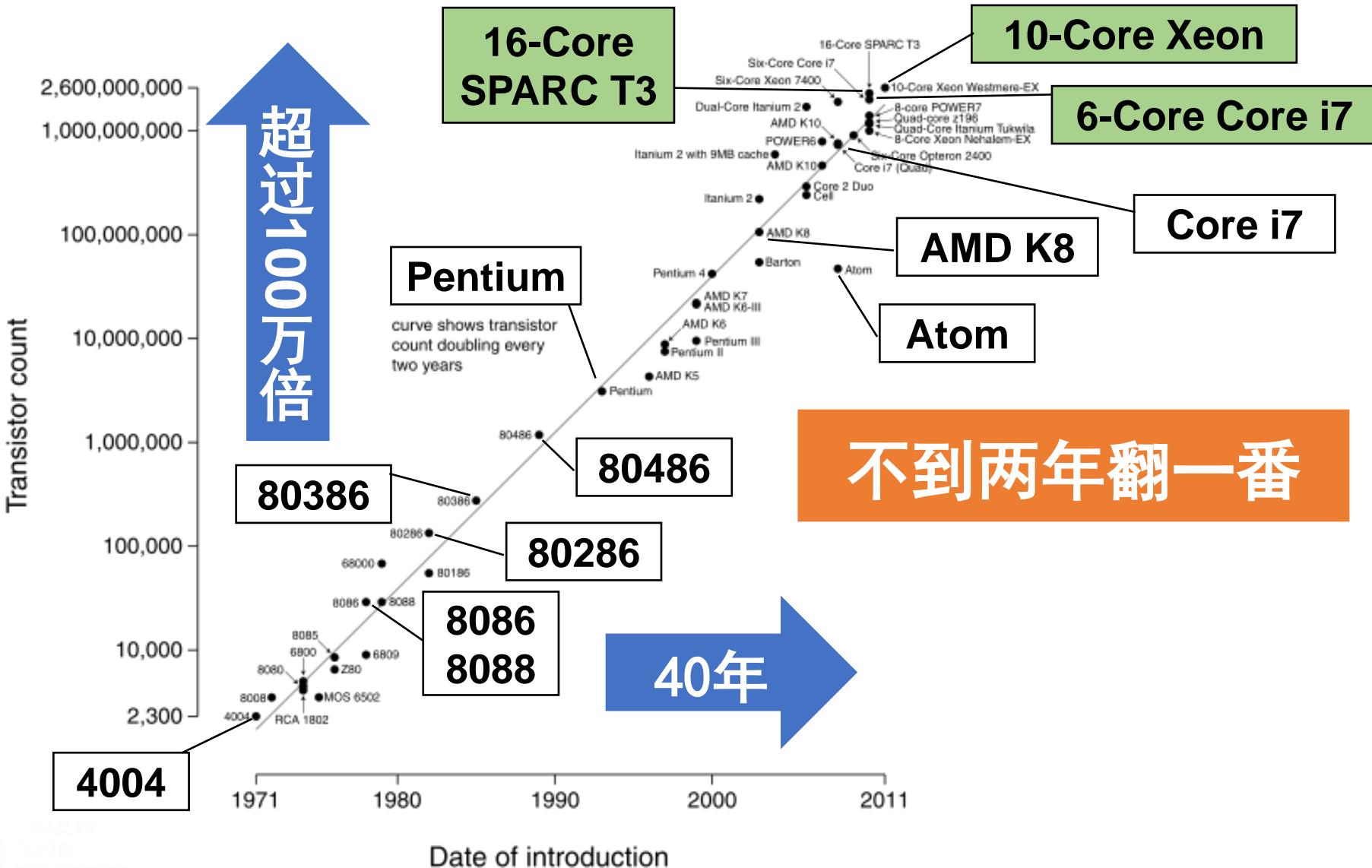
戈登·摩尔  
**Gordon Moore**  
1929~

与罗伯特·诺宜斯创建了以INTEGRATED Electronics的前缀组成的“Intel”公司



# 微处理器晶体管数增长曲线

Microprocessor Transistor Counts 1971-2011 & Moore's Law





# 阿姆达尔定律

阿姆达尔：

- 改变世界的25人”之一；
- 主持IBM360的设计工作，他提出：360应是一条向下兼容的生产线，可使软件和设备用在这一家族的每个成员身上；
- 量化研究方法
  - 软硬件优化、加速
  - 并行计算、并行编程
- 阿姆达尔定律（Amdahl）



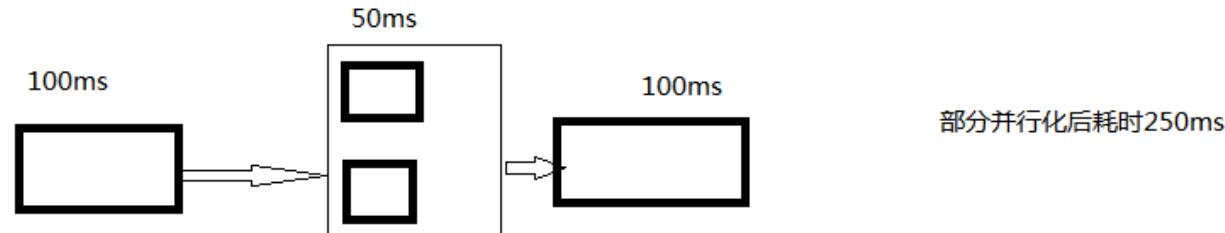
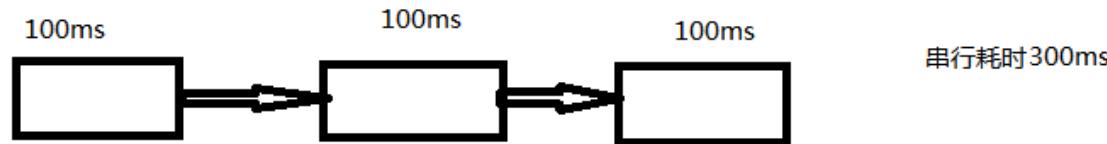


# 阿姆达尔定律

- 对计算机系统的某一部分加速的时候,该加速部分对系统整体性能的影响取决于该部分的重要性和加速程度
  - 加速比 = 优化之前系统耗时 / 优化后系统耗时

$$= \frac{1}{(1-p) + \frac{p}{s}}$$

$p$ 是被加速部分原始时间  
 $s$ 是加速技术的并行度





## 单选题 1分

设置

图形处理器中经常需要求平方根。浮点（FP）平方根的实现  
在性能方面有很大差异，特别是在为图形设计的处理器中，  
尤为明显。假设FP平方根（FPSQR）占用一项关键图形基准  
测试中20%的执行时间。

1. 提议1：升级FPSQR硬件，使这一运算速度提高到原来的10倍。
2. 提议2：让图形处理器中所有FP指令的运行速度提高到原来的1.6倍，FP指令占用该应用程序一半的执行时间。

请问两种方法哪个更好？

A

第一种好

B

第二种好

提交

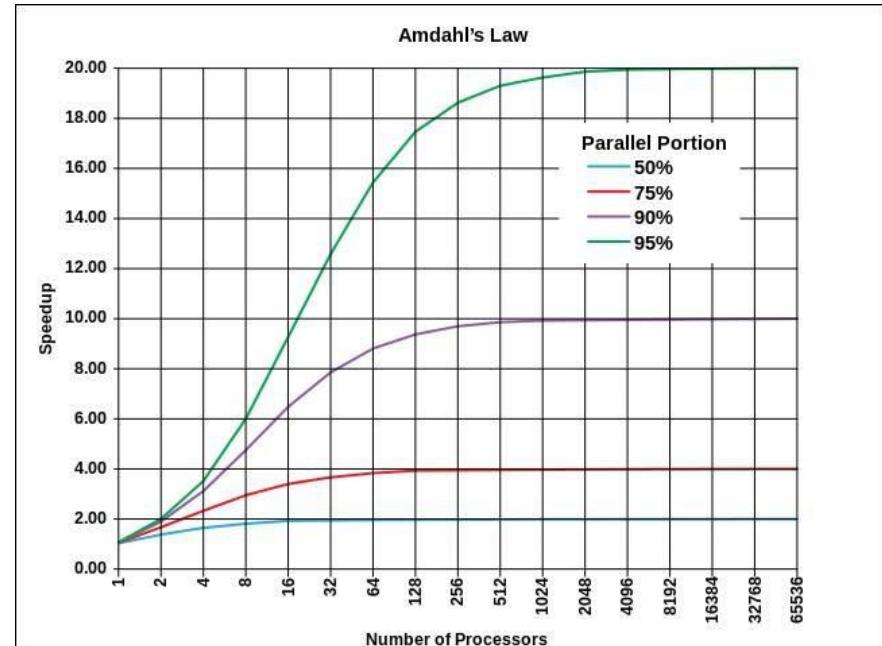


# 课堂练习（答案）

- 第二种方法要好一点

$$\text{加速比}_{\text{FPSQR}} = \frac{1}{(1 - 0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{加速比}_{\text{FP}} = \frac{1}{(1 - 0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$





# 课后练习

- 阅读教材第1章
- 练习题1.1, 1.2