# **相关书籍推荐**

读书的原则：不求甚解，观其大略

你如果进到庐山里头，二话不说，蹲下头来，弯下腰，就对着某棵树某棵小草猛研究而不是说先把庐山的整体脉络跟那研究清楚了，那么你的学习方法肯定效率巨低而且特别痛苦，最重要的还是慢慢地还打击你的积极性，说我的学习怎么那么不happy啊，怎么那么特没劲那，因为你的学习方法错了，大体读明白，先拿来用，用着用着，很多道理你就明白了

▪《编码：隐匿在计算机软硬件背后的语言》

▪《深入理解计算机系统》

▪语言：C JAVA K&R《C程序设计语言》《C Primer Plus》

▪ 数据结构与算法： -- 毕生的学习 leetCode

–《Java数据结构与算法》《算法》

–《算法导论》《计算机程序设计艺术》//难

▪操作系统：Linux内核源码解析 30天自制操作系统

▪网络：机工《TCP/IP详解》卷一 翻译一般

▪编译原理：机工 龙书 《编译原理》 《编程语言实现模式》马语

▪数据库：SQLite源码 Derby - JDK自带数据库

# **硬件基础知识**

## **CPU的制作过程**

Intel cpu的制作过程

<https://haokan.baidu.com/v?vid=11928468945249380709&pd=bjh&fr=bjhauthor&type=video>

CPU是如何制作的（文字描述）

<https://www.sohu.com/a/255397866_468626>

## **CPU的原理**

计算机需要解决的最根本问题：如何代表数字

晶体管是如何工作的：

<https://haokan.baidu.com/v?vid=16026741635006191272&pd=bjh&fr=bjhauthor&type=video>

晶体管的工作原理：

<https://www.bilibili.com/video/av47388949?p=2>

## **汇编语言（机器语言）的执行过程**

汇编语言的本质：机器语言的助记符 其实它就是机器语言

计算机通电 -> CPU读取内存中程序（电信号输入）

->时钟发生器不断震荡通断电 ->推动CPU内部一步一步执行

（执行多少步取决于指令需要的时钟周期）

->计算完成->写回（电信号）->写给显卡输出（sout，或者图形）

## **量子计算机**

量子比特，同时表示1 0

## **CPU的基本组成**

PC -> Program Counter 程序计数器 （记录当前指令地址）

Registers -> 暂时存储CPU计算需要用到的数据

ALU -> Arithmetic & Logic Unit 运算单元

CU -> Control Unit 控制单元

MMU -> Memory Management Unit 内存管理单元

cache

## **缓存**

一致性协议：<https://www.cnblogs.com/z00377750/p/9180644.html>

缓存行：

缓存行越大，局部性空间效率越高，但读取时间慢

缓存行越小，局部性空间效率越低，但读取时间快

取一个折中值，目前多用：

64字节

package com.mashibing.juc.c\_028\_FalseSharing;  
​  
public class T03\_CacheLinePadding {  
​  
    public static volatile long[] arr = new long[2];  
​  
    public static void main(String[] args) throws Exception {  
        Thread t1 = new Thread(()->{  
            for (long i = 0; i < 10000\_0000L; i++) {  
                arr[0] = i;  
           }  
       });  
​  
        Thread t2 = new Thread(()->{  
            for (long i = 0; i < 10000\_0000L; i++) {  
                arr[1] = i;  
           }  
       });  
​  
        final long start = System.nanoTime();  
        t1.start();  
        t2.start();  
        t1.join();  
        t2.join();  
        System.out.println((System.nanoTime() - start)/100\_0000);  
   }  
}  
​

package com.mashibing.juc.c\_028\_FalseSharing;  
​  
public class T04\_CacheLinePadding {  
​  
    public static volatile long[] arr = new long[16];  
​  
    public static void main(String[] args) throws Exception {  
        Thread t1 = new Thread(()->{  
            for (long i = 0; i < 10000\_0000L; i++) {  
                arr[0] = i;  
           }  
       });  
​  
        Thread t2 = new Thread(()->{  
            for (long i = 0; i < 10000\_0000L; i++) {  
                arr[8] = i;  
           }  
       });  
​  
        final long start = System.nanoTime();  
        t1.start();  
        t2.start();  
        t1.join();  
        t2.join();  
        System.out.println((System.nanoTime() - start)/100\_0000);  
   }  
}  
​

缓存行对齐：对于有些特别敏感的数字，会存在线程高竞争的访问，为了保证不发生伪共享，可以使用缓存航对齐的编程方式

JDK7中，很多采用long padding提高效率

JDK8，加入了@Contended注解（实验）需要加上：JVM -XX:-RestrictContended

# 第二课

## CPU的乱序执行

指令之间没有依赖关系，在CPU等待指令1去内存读数据的时候（这个速度比较慢 低于CPU计算100倍），可以优先执行其他指令，来提高效率。

