

《计算机科学导论》知识点梳理

Powered by Jaser Li

第一章 计算机科学概论

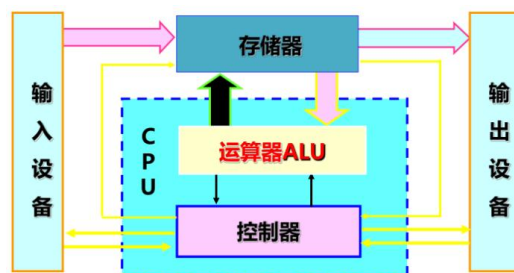
1. 硬件

输入设备：如键盘、鼠标；

存储设备：如内存，硬盘；

运算控制设备：如中央处理器 CPU；

输出设备：如显示器。



2. 输入数据在硬件上的逻辑流动过程中，涉及到的数据传输都是通过**总线**完成的。

3. 硬件无法自我完成和实现用户的需求

CPU 是计算机的大脑，**计算机语言**控制 CPU 按步骤执行任务

指示 CPU 进行操作的语言称为**程序语言**

人们预定的安排是通过一个指令序列来表达的，这个指令序列称为**程序（或软件）**

在程序执行的这个黑匣子中，软件**描述**了用户的需求，硬件则**实现**了用户的需求。

4. **软件并不能独立的控制硬件进行工作。**

计算机结构中，还需要一个层次来衔接软件与硬件、控制硬件工作、为软件提供服务等。这个特殊的层次就是**操作系统（OS）**。操作系统其实也是一组程序。

5. 计算机系统分为三个层次：**硬件层、操作系统层、软件层。**

（1）硬件层包括计算机的各个部件，**控制器、运算器、存储器、输入设备和输出设备**。这个存储程序计算机架构（**冯·诺依曼结构**）早在 1964 年，由美籍匈牙利科学家冯·诺依曼提出。目前，绝大多数计算机采用冯·诺依曼结构，**冯·诺依曼也因此被人们称为“计算机之父”**。

【重要】冯·诺依曼结构特点：

■ 计算机的数制采用**二进制**，数据和指令均采用 1 和 0 组成的**二进制代码**表示。

■ 计算机指令和数据存储在**同一记忆装置中**，即**存储器**。

■ 计算机按照预先编制的程序顺序执行，即**程序控制**。

（2）软件层包括**由汇编以及高级语言（C/C++，Java，Python 等）等开发出的应用程序**。对通用型计算机而言，功能的实现需要软硬件的无缝配合。要使硬件 CPU 发挥计算功能，需要控制指令来完成。

（3）操作系统层是连接硬件和软件的中间桥梁，操作系统的种类繁多，根据不同的应用要求，有不同的操作系统。生活中最常见的操作系统有微软的 Windows 系列产品、Ubuntu、Fedora 等 Linux 系统，苹果 Mac OS 系列产品，以及智能手机中所使用的 Android、IOS 系统等等。

6. 操作系统的主要职能

■ 管理**文件系统**，管理**各种硬件资源**，例如 U 盘、网络、键盘等

■ 管理**程序共享的资源**，例如 CPU、主存等

■ 管理和调度**多个程序的执行**

■ 提供**程序和硬件的衔接**，提供**各种系统的服务和接口**

■ 设法**维护系统的安全**，尽量防止病毒（恶意软件）有意或无意的侵入

7. 软件层的实现主要由**程序设计语言**完成。程序设计语言是计算机能够理解和识别用户操作意图的一种交互体系，它按照特定规则组织计算机指令，使计算机能够自动进行各种运算处理。**按照程序设计语言规则组织起来的一组计算机指令称为计算机程序**。程序设计语言包括：**机器语言、汇编语言和高级语言**。

(1) 机器语言：一种二进制语言，它直接使用**二进制代码**表达指令，是计算机硬件可以直接识别和执行的程序设计语言。

(2) 汇编语言：使用**助记符**与机器语言中的指令进行一一对应，在计算机发展早期帮助程序员提高编程效率。

机器语言和汇编语言都直接操作计算机硬件并基于此设计，所以它们统称为**低级语言**。

(3) 高级语言：高级语言区别于低级语言在于，高级语言是接近自然语言的一种计算机程序设计语言，更容易地描述计算问题并利用计算机解决计算问题。第一个广泛应用的高级语言是**诞生于 1972 年的 C 语言**。

8. 计算机执行源程序的两种方式：**编译和解释**

■ 源代码：采用某种编程语言编写的计算机程序，人类可读。例如：`result = 2 + 3`

■ 目标代码：计算机可直接执行，人类不可读 (专家除外)。例如：`11010010 00111011`

(1) 编译：将源代码一次性转换成目标代码的过程。执行编译过程的程序叫作编译器。

(2) 解释：将源代码逐条转换成目标代码同时逐条运行的过程。执行解释过程的程序叫作解释器。



9. 计算机分类

(1) 通用型计算机：常用的台式计算机、笔记本电脑、平板电脑等，或服务器、超级电脑等。

(2) 专用型计算机：为特定应用量身打造，内部的程序一般不能被改动。常被称为“**嵌入式系统**”。

10. 电子计算机从诞生起，经过了**电子管、晶体管、集成电路（IC）和超大规模集成电路（VLSI）**四个阶段的发展。（摩尔定律）集成电路上可以容纳的晶体管数目在大约每经过 18 个月到 24 个月便会增加一倍。

(1) 第一代计算机：**体积较大；运算速度较低；存储容量不大**；而且价格昂贵；使用也不方便。

(2) 第二代计算机：全部采用**晶体管**作为电子器件，其运算速度比第一代计算机的提高了近百倍，体积为原来的几十分之一。在软件方面，开始使用计算机算法语言。

(3) 第三代计算机：主要特征是**以中、小规模集成电路为电子器件**。重大突破是出现了操作系统。

(4) 第四代计算机：采用**大规模集成电路（LSI）和超大规模集成电路（VLSI）**为主要电子器件。另一个重要分支是以大规模、超大规模集成电路为基础发展起来的微处理器和微型计算机。

11. **数据**是指所有能输入到计算机并被计算机程序处理的，具有一定意义的数字、字母、符号和模拟量等的通称。计算机利用**模数转换器（ADC, Analog to Digital Converter）**，将模拟信号（即真实世界的连续信号）转换为数字信号（即用数值表示的离散信号），即 0 和 1 进行传输。

第二章 数字系统和信息编码

1. 十进制 (Decimal)、二进制 (Binary)、八进制 (Octonary) 与十六进制 (Hexadecimal)

2. 基数 (Base) 与位权 (Weight)

3. 不同进制间的转换

■ R 进制数转换为十进制数时, 将各位数与它的位权乘积相累加, 即一个二进制数 $a_n a_{n-1} \dots a_1 a_0$ 在十进制中的值 $A = a_n \times R^n + a_{n-1} \times R^{n-1} + \dots + a_1 \times R^1 + a_0 \times R^0$ 。

■ 十进制整数转换成 R 进制整数: 可用十进制整数连续地除以 R, 每次除法获得的余数即为相应 R 进制数一位, 最后按逆序输出结果。此方法称为“除 R 取余法”。

■ 十进制小数转换成 R 进制小数: 可用十进制的小数连续地乘以 R, 用得到的整数部分组成 R 进制的小数, 最后按顺序输出结果。此法称为“乘 R 取整法”。

■ 对于二进制数和八进制数、十六进制数之间的转换, 有简便快速的“三位一并法”和“四位一并法”。

4. 中央处理器(Central Processing Unit, CPU)是进行各种运算的硬件, 是一个非常小的集成电路芯片, 通过引脚(pin)与外部连接并交换数据。

CPU 一次只能够处理有限数位的二进制数。现在的计算机一般能一次能处理 32 个或 64 个比特的数据, 计算机能直接处理的最大的二进制整数是 232 或 264。计算机通常把整数分为两类:

■ 无符号整数(unsigned integer), 表示的是非负整数, n 位计算机能表示 $[0, 2^n - 1]$ 范围内的所有整数;

■ 带符号整数(signed integer), 可以表示正整数、负整数和 0, 因此需要占用一个比特位来表示整数的正负符号, 所能表示的正整数范围就会变小。

• 对于无符号整数, n 个比特所能表示的最大数是 $2^n - 1$ 。

➢ 用 8 个比特位表示的最大整数是 $2^8 - 1$ 。

➢ 8 个比特能够表示 $[0, 255]$ 之间的所有二进制整数。00000000₂ 表示 0, 00000001₂ 表示 1, 11111111₂ 表示 255。

• 在二进制加法中遵循“逢二进位”的法则, 即两数对应的位相加与前一位的进位的和, 大于 1 则产生进位, 把小于等于 1 的部分记为两数相加后该位的值。

$$\begin{array}{r} 00001000_2 (8_{10}) \\ + 00001000_2 (8_{10}) \\ \hline = 00010000_2 (16_{10}) \end{array}$$

• 溢出 (overflow): 无符号整数加法的一种异常情况, 即两个整数相加的位的位数大于这两个数的位数。

➢ 例如对于只能处理 8 位整数的计算机而言, 这个二进制加法的和造成了溢出:

$$\begin{array}{r} 10001001_2 (137_{10}) \\ + 10001000_2 (136_{10}) \\ \hline = 100010001_2 (171_{10}) \end{array}$$

↑
舍弃

实际结果 (错误)

➢ 从算术上看, “ $137 + 136 = 17$ ”的结果显然是错误的。而在计算机中产生这类错误的原因是两数相加的和超过了 CPU 所能处理的最大无符号整数 $2^8 - 1$, 即 255。溢出发生时, CPU 会报错。

• 二进制的乘法和除法的运算规则和十进制的运算规则是一样的。

$$\begin{array}{r} \text{被乘数} \quad 1001_2 (9_{10}) \\ \times \quad \text{乘数} \quad 1001_2 (9_{10}) \\ \hline 1001_2 \quad \leftarrow \text{移 1 位} \\ 0000_2 \quad \leftarrow \text{移 2 位} \\ 0000_2 \quad \leftarrow \text{移 3 位} \\ + 1001_2 \quad \leftarrow \text{移 3 位} \\ \hline \text{积} \quad 1010001_2 (81_{10}) \end{array}$$

• 二进制乘法是由基本的二进制加法和移位操作所完成的

➢ 当两个二进制数的位数之和大于计算机所能处理的位数 n 时, 乘法的结果很可能超过 n 位, 也就是出现溢出。

➢ 绝大部分计算机系统都有处理溢出的机制, 这里不再作深入讨论。

• 二进制除法是由基本的二进制减法和移位操作完成的

$$\begin{array}{r} 1001_2 (9_{10}) \quad \text{商} \\ \text{除数 } 1001 \quad \text{被除数 } 1010001_2 (81_{10}) \\ \hline - 1001_2 \\ \hline 1_2 \\ \text{补 1 位} \rightarrow 10_2 \\ \text{补 2 位} \rightarrow 100_2 \\ \text{补 3 位} \rightarrow 1001_2 \\ \hline - 1001_2 \\ \hline 0000_2 (0_{10}) \quad \text{余数} \end{array}$$

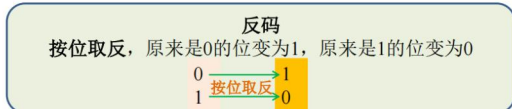
➢ 从最高位开始, 在被除数中取和除数同样多的位数, 所得数值减去除数, 直至所得的余数小于除数, 这个余数和被除数中的剩余位数拼接成新的数, 取其中和除数同样多的位数, 并减去除数……重复这个过程直到被除数的最后一位。

5. 【重要】计算机用第(2)种对应方式表示负数，也就是所谓的**补码**。

- 减法其实可以看作对负数的加法 → 如何在计算机的世界里表示负数？
- 在带符号数的运算中，计算机用1个比特表示该整数的符号，把一半的数定义为负数。

十进制数	无符号整数	带符号整数对应方式(1)	带符号整数对应方式(2)
255	11111111	-128	-1
254	11111110	-2	-2
...
128	10000000	-1	-128
127	01111111	127	127
126	01111110	126	126
...
0	00000000	00000000	00000000

- 负数-x的补码通过正整数x的反码加1得到。



- 计算机中的带符号数有三种表示方法：原码、反码、补码

➢ 对于带符号数中的正数，其原码、反码、补码的表示值是相同的。

原码 反码 补码

+2 00000010 00000010 00000010

➢ 对于带符号数中的负数，其原码、反码、补码的表示值是不同的。

原码 反码 补码

-2 10000010 11111101 11111110

反码是原码取反；补码是反码+1，或原码取反+1

8 位二进制无符号数的表示范围为：0~255

8 位二进制带符号数的补码表示范围为：-128~127

在用补码方式表示 n 位带符号整数时，最大数是 $2^{n-1}-1$ ，最小数是 -2^{n-1}

- 带符号数的减法：负数的加法；负数用补码表示

- 例：16 - 32：

- 16的补码：00010000

- -32的补码为32的反码加1：11100000

0	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0

- 11110000的高位是1说明是负数-x，x的反码加1
- 11110000-1以后的反码为00010000，说明x是16
- 则结果为-16

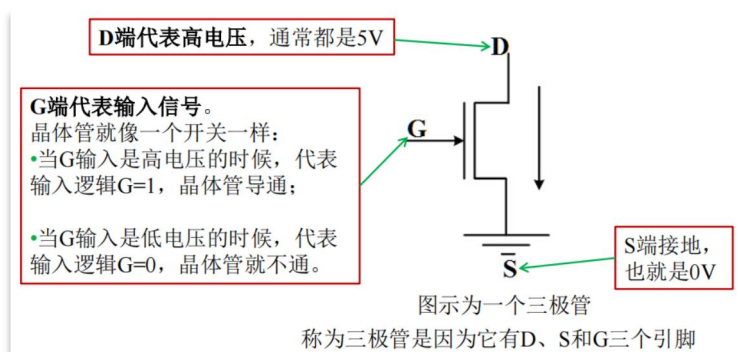
6. 逻辑运算是对逻辑变量和逻辑运算符号的组合序列所作的逻辑推理。

逻辑运算的变量只有两个，它们代表两种对立的逻辑状态。（真—假、是—否、有—无 → 1—0）

逻辑运算的基本运算：与（AND）“∧”，或（OR）“∨”，非（NOT）“¬”

7. 计算机电路中用晶体管实现逻辑。晶体管是以半导体材料为基础元件，例如各种半导体材料制成的二极管、三极管、场效应管、可控硅等。现在计算机芯片制造者已经可以用大规模生产的方式在几个平方毫米的半导体晶片上实现数百万个晶体管。

每个晶体管可以在不改变自身内部结构的情况下，根据外部电源的变化而展现不同的状态。我们可以通过控制晶体管的电源来控制它们开或关的状态。



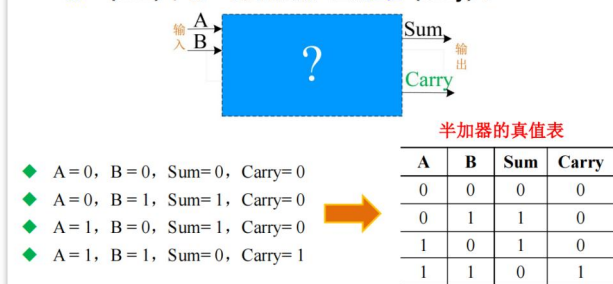
8. 二进制数的加法在计算机里是由每一个比特位的加法组成的

而每一位的加法都需要 3 个输入，并产生 2 个输出

3 个输入分别是两个相加位和一个由相邻低位产生的进位

2 个输出分别是一个相加得到的二进制数位和一个进位

- 半加器的输入是两个1位的二进制数A和B，它们的值是0或1；经过加法器的运算之后，给出两个1位的输出，一个是加法所产生的低位，称为“和”（sum）；另一个是加法所产生的进位（carry）。

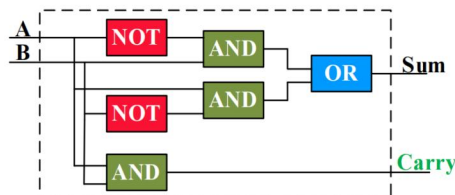


- 为了方便起见，我们常在写逻辑算式时省略逻辑与的符号，并且用“+”和“bar”代替逻辑或和相应变量的非运算。例如：

$$\text{Carry} = A \wedge B, \text{Sum} = (A \wedge \neg B) \vee (\neg A \wedge B)$$

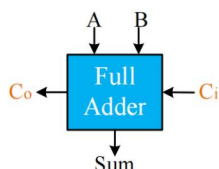
$$\text{改写为 } \text{Carry} = AB, \text{Sum} = A\bar{B} + \bar{A}B$$

- 我们可以根据这种逻辑运算的符号画出图示的电路设计图：

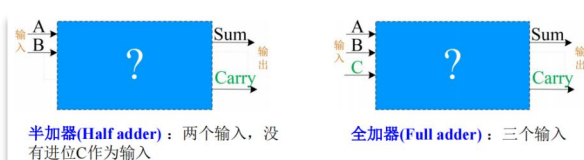


- 实现多位加法需要全加器，只要在半加器的基础上做一个小小改进，就可以得到全加器。

- 半加器的没有进位输入，而全加器需要输入低位的进位



- 三个输入：A和B是两个加数，Ci是从下一位获得的进位。
- 两个输出：给上一位的进位Co，以及两数相加的和在该位的值Sum。

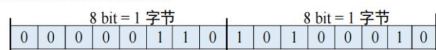


9. 数据的存储形式

计算机用二进制数的组合来表达所有需要保持的信息，这些二进制数的组合按照一定的规则存放就形成了计算机里的数据。

二进制数的数值 0 或 1 的存储方式跟随着物理介质的特性不同而不同，基本是利用物理材料的电信号、磁信号之类的状态来存储 0 或 1。这些载体就称为存储介质。存储介质和辅助数据存储和数据读写的电路、设备等组合在一起，构成了存储设备，例如我们常用的内存、磁盘、U 盘等。

- 在计算机内部，各种信息都是以二进制编码的形式存储的。
- 在二进制编码中，指定不同数量的0或1形成的不同的组合表示不同的意义。编码往往用到一大串0或1，必须要按照一定规则对这些信息进行分割和识别才能获得有用的信息。



计算机把单位信息分成以下三种：

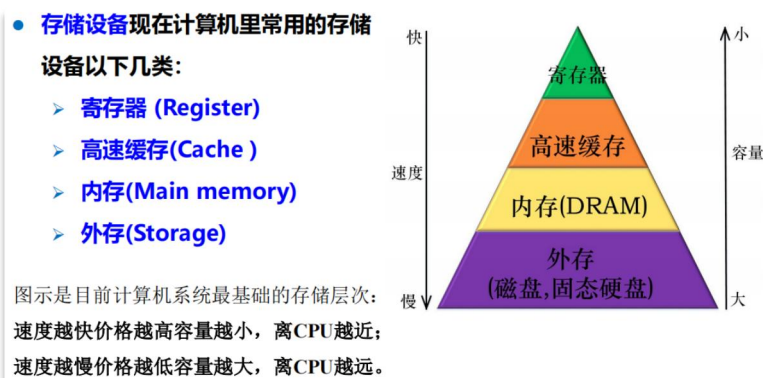
- 位(或称为“比特”，bit)：一个1或一个0即一个位。
- 字节(Byte)：一个字节由8位二进制数组成（1 Byte=8bit）。
- 字(Word)：字是字节的组合，CPU可以用“字”为单位来读写数据。大部分CPU的字长是32位（4个字节）或64位（8个字节）。

- 字节是信息存储中常用的基本单位。计算机的存储器（包括内存与外存）通常也是以多少字节来表示它的容量。常用的单位有：

- KB (Kilobyte)：K代表 2^{10} ，1KB= 2^{10} B=1024B；
- MB (Megabyte)：M代表 2^{20} ，1MB=1024KB；
- GB (Gigabyte)：G代表 2^{30} ，1GB=1024MB；
- TB (Terabyte)：T代表 2^{40} ，1TB=1024GB；
- PB (Petabyte)：P代表 2^{50} ，1PB=1024TB；
- EB (Exabyte)：E代表 2^{60} ，1EB=1024PB。

- 这些符号在不同场合有不统一的定义。通常在谈容量时用的是二进制的一套定义方法；在谈论计算机性能时常用的是单位制；而谈速度的时用的是十进制的一套定义。例如网络传输就是用的十进制。

10. 存储设备 – 存储层次

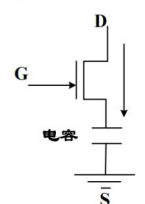


普通运算时需要将数据先放到 CPU 中的寄存器里，然后 ALU 对寄存器的值做计算，再存回寄存器里。

程序普通执行时的信息，包括程序指令和很多用到的数据都存放在内存中。

外存的一个显著特征是数据断电后不丢失，而当前以上三种存储在断电后都会丢失数据。

11. 存储设备 – 用晶体管制成 DRAM 和 SRAM

- DRAM和SRAM都是用许多**晶体管**组成的存储结构，一个DRAM的存储单元仅仅是由一个晶体管加上一个电容组合而成，表示一个bit。
 - 电容里存储的电荷数量用来表示这个bit是0或1。
 - 由于电容会一直慢慢漏电，漏电到一定程度就无法保存这个存储单元的信息。所以**规定每隔一定时间就刷新一次DRAM**，也就是给电容充电，所以它叫**动态存储单元**。
- 
- 该图是一个电路示意图，显示了一个晶体管（Gates G and D）与一个电容（C）的连接。电容下方标注为“电容”，整个结构下方标注为“1bit的DRAM存储单元”。

- 在高速缓存里，使用的是另一种更为复杂快速的存储单元，也就是**静态存储单元—SRAM**。
- SRAM不需要充电，它用多达**六个晶体管**组成了一个循环的结构。
- 同样是存储一个bit，**SRAM用6个晶体管**实现了更快更靠谱的性能，**不需要定时刷新也能保证数据不丢失**。
- SRAM相比DRAM的缺点也很明显，那就是同样存储量的SRAM需要6倍于DRAM的晶体管，而且硬件的面积也增大了很多，价格要贵很多。
- 晶体管需要通电才能表达0或1，一旦掉电就会丢失保存的信息，所以**断电后DRAM和SRAM都会丢失信息**。

12. 存储设备 – 掉电也能用的存储介质

磁盘、闪存（比如 **U 盘**、**固态硬盘**）。

闪存使用的是一种改进的晶体管。

第三章 程序是如何执行的

1. 计算机中有两个核心部件，分别是 CPU 和主存（内存）。CPU 是做运算的，主存存储程序和相关的变量，每一条程序语句和每一个变量在内存中都有相应的内存地址。

2. CPU 中的核心部件

- (1) 语句地址的存储——程序计数器 PC(Program Counter)
- (2) CPU 中存储程序语句——指令寄存器 IR(Instruction Register)
- (3) 执行运算——算术逻辑单元 ALU(Arithmetic Logic Unit)

3. 【重要】汇编指令

■ “读取 a 到 R”操作——load 指令

格式：load R1, (address) 例如：load R1, (1000) (address)表示 address 这个地址内存存储的值。

■ “R 赋值”操作——mov 指令

格式 1: mov R1, constant 例如：mov R1, 0Ah 前一个是寄存器，后一个是十六进制常数。

格式 2: mov R2, R1 (后赋前)

■ 加法指令 add

格式 1: add R2, R1, constant 即为 $R2 = R1 + \text{constant}$. 例如：add R2, R1, 01h

格式 2: add R1, R1, R2 即为 $R1 = R1 + R2$.

■ 减法指令 sub

“sub R2, R1, constant”，代表 $R2 = R1 - \text{constant}$.

■ 左移位指令 shiftl

“shiftl R3, R1, 05h”代表寄存器 R1 的二进制数左移 5 位，移出的那 5 位填 0。将最终值存入 R3。

‘05h’也可以用一个寄存器表示，例如“shiftl R3, R1, R2”代表 R1 的二进制值向左移 R2 位，存入 R3。

左移指令就相当于将 R1 做乘法。R1 左移一位，R1 值相当于乘 2，R1 左移 2 位，R1 值相当于乘 4。

■ 右移位指令 shiftr

向右移位，移完后的那些最高位填 0。

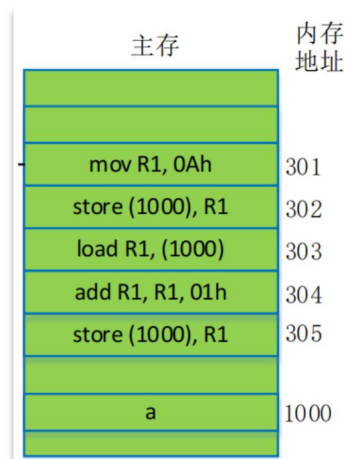
■ “将 R 存回 a”操作——store 指令

格式：store (address), R1

注：address 是内存地址，(address)表示该地址要存回的值，R1 是寄存器。也就是(address)=R1。

【演示】a=a+1 的完整执行过程

程序开始执行时，变量 a 存储在主存地址 1000 处。a=10, a=a+1 程序语句有五条汇编指令，从地址 301 处开始顺序存储每条指令。程序开始执行时，PC 指向汇编程序的首地址 301 处。



■ “比较 x 是否小于 y”——slt 指令

格式：slt R4, R1, R2

即比较寄存器 R1 中的数值是否小于 R2 中的数值，如果小于，则将寄存器 R4 置 1，否则置 0。

■ 判断小于或等于指令——sle 指令（同上）

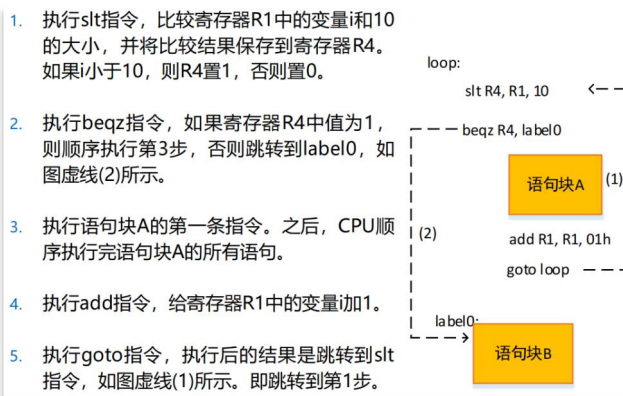
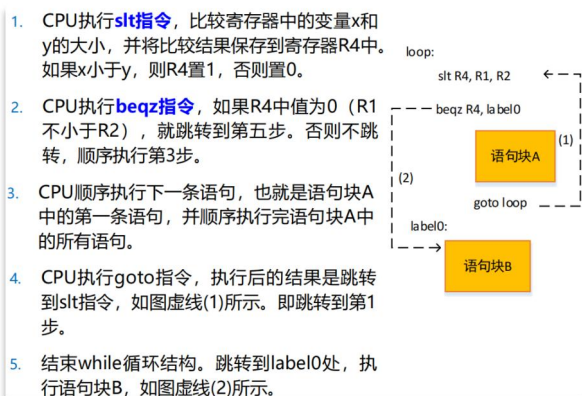
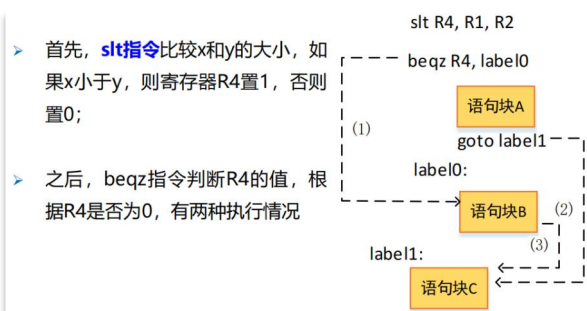
■ “选择跳转语句块”操作——beqz 指令

beqz R4, label2 表示：如果寄存器 R4 中的数值为零，则跳转到标签 label2 标记的指令块处。

■ “直接跳转到语句块”操作——goto 指令

只有一个操作数，即“标签”label。goto label3 表示：跳转到标签 label3 标记的指令处执行。

【应用】if-else 分支语句；while 循环语句；for 循环语句



4. 关于 Python 的函数调用

5. 函数调用过程的分析

栈：按照先进后出的原则存储数据

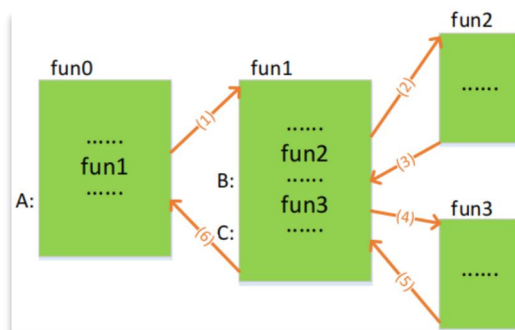
主调函数是指调用其他函数的函数；被调函数是指被其他函数调用的函数。一个函数很可能既调用别的函数，又被另外的函数调用。函数调用后返回时，程序会返回到主调函数中调用函数的语句的后一条语句继续执行。

函数调用的特点是：越早被调用的函数，越晚返回。

所以：采用“栈”来保存返回地址。

局部变量如同返回地址般也是存在栈里。

当函数执行时，这个函数的每一个局部变量就会在栈里有一个空间。在栈中存放此函数的局部变量和返回地址的这一块区域叫做此函数的栈帧(frame)。当此函数结束时，这一块栈帧就会被弹出。



第五章 计算思维的核心——算法

1. 递归

动态规划、分治法、贪心算法都是基于递归概念的方法。

- 例：平面划分问题、汉诺塔问题。

递归是一个过程或函数在它的定义或说明中又直接或间接调用它自己的一种方法。

递归本质是把一个复杂的大问题层层转化为一个与原问题相似的小问题，利用小问题的解来构筑大问题的解。学习用递归解决问题的关键就是找到问题的递归式。

在使用递归解决问题时要特别注意，一定要有一个明确的递归结束条件，否则就会陷入无限循环中。

写递归程序的诀窍就是：（1）怎么分，怎么合；（2）怎么终止。

2. 分治法

- 例：求 n 个数中的最小值

循环比较法： $(n-1)$ 次；递归比较法： $(n-1)$ 次；

分治比较法： $(n-1)$ 次。但是这种方法可能在多核的情况下要比前两种方法的效率高。

- 问题描述：假设有 n 个数，分别为 $a_1, a_2, a_3, \dots, a_n$ ，求 n 个数中的最小值。
- 求最小值例子：分治比较法
 - 要求 $M(1, n)$ ，可以先求得 $M(1, n/2)$ 和 $M(n/2+1, n)$ ， $M(1, n/2)$ 和 $M(n/2+1, n)$ 中较小的就是 $M(1, n)$ ；
 - 而要求 $M(1, n/2)$ ，可以先求得 $M(1, n/4)$ 和 $M(n/4+1, n/2)$ ，其中较小的就是 $M(1, n/2)$ 。
 - 按照上述基本思想，可以求得 n 个数中的最小值 $M(1, n)$ ，用公式可以表示为： $M(1, n) = \min(M(1, n/2), M(n/2+1, n))$
 - 用这种方法，同样需要比较 $n-1$ 次。

```
#<程序：最小值_分治>
def M(a):
    #print(a) 可以列出程序执行的顺序]
    if len(a) == 1: return a[0]
    return ( min(M(a[0:len(a)//2]),M(a[len(a)//2:len(L)])))
L=[4,1,3,5]
print(M(L))
```

```
#<程序：最小值和最大值_分治>
A=[3,8,9,4,10,5,1,17]
def Smin_max(a):
    if len(a)==1:
        return(a[0],a[0])
    elif len(a)==2:
        return(min(a),max(a))
    m=len(a)//2
    lmin,lmax=Smin_max(a[:m])
    rmin,rmax=Smin_max(a[m:])
    return min(lmin,rmin),max(lmax,rmax)
print("Minimum and Maximum:%d,%d"%(Smin_max(A)))
```

- 例：同时求 \min 和 \max ： $(1.5n-2)$ 次

- 问题描述：求 n 个数 $a_1, a_2, a_3, \dots, a_n$ 中的最小值和最大值。
- 如果用前面的方法，找最小值要比较 $n-1$ 次，找最大值也要比较 $n-1$ 次，要找到最小值和最大值共需 $2n-2$ 次比较
- 在找最小值和最大值时存在很多重复的比较。例如，求最小值时要比较 a_1 和 a_2 ，求最大值时还要比较一次 a_1 和 a_2 。其实可以用分治法同时找到最小值和最大值，它最多只需比较 $1.5n-2$ 次

3. 贪心算法（贪婪算法）：用来求解最优化问题的一种方法

一般来说，求解最优化问题的过程就是做一系列决定从而实现最优值的过程。最优解就是实现最优值的这些决定。贪心算法考虑局部最优，每次都做当前看起来最优的决定，得到的解不一定是全局最优解，但是有些问题能够用贪心算法求得最优解。

- 例：求两个正整数 x 和 y 的最大公约数

重要性质：如果 a 是 x 和 y 的最大公约数并且 $x > y$ ，那么 a 也是 $x-y$ 和 y

的最大公约数。（15 和 10 的最大公约数是 5， $15-10=5$ ，5 和 10 的最大公约数也是 5。）

两个整数的最大公约数等于其中较小的数和两数相除余数的最大公约数。

- 对比：

贪心算法考虑局部最优，每次都做当前看起来最优的决定，得到的解不一定是全局最优解。

动态规划考虑全局最优，得到的解一定是最优解。

```
def GCD(x,y):
    if x>y: a=x;b=y
    else: a=y;b=x
    if a%b ==0: return(b)
    return(GCD(a%b,b))
```

4. 动态规划

动态规划与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。

与分治法不同的是，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的，即子问题之间具有重叠的部分。在这种情况下，如果用分治法求解就会重复的求解这些重叠的部分。

动态规划只会对这些重叠的部分求解一次并用表格保存这些解，如此一来就可以避免大量的重复计算。

● 例：最长递增子序列问题

● **问题描述**：已知有n个数的序列L，求它的最长递增子序列的长度。假设序列L的一个递增子序列为 $[a_1, a_2, \dots, a_k]$ ，这些数必须满足 $a_1 < \dots < a_i < \dots < a_j < \dots < a_k (1 \leq i < j \leq k)$ ，而最长递增子序列就是所有递增子序列中长度最大的那个。

➤ 例：序列 $L=[5, 2, 4, 7, 6, 3, 8, 9]$ 的最长递增子序列是 $[2, 4, 7, 8, 9]$ ，其长度是5。

● **基本思路**：首先将原问题分解成小问题，再用小问题的解构筑原问题的解。因此，我们需要考虑的是“怎么分，怎么合”的问题。

● “怎么分”就是考虑怎么将n个数的最长递增子序列问题划分成n-1个数的最长递增子序列问题；

● “怎么合”就是考虑怎么用n-1个数的最长递增子序列问题的解构筑n个数的最长递增子序列问题的解。

5. 算法效率的度量：用依据该算法编制的程序在计算机上执行所消耗的时间来度量。

(1) **事后统计**：利用计算机内的计时功能，不同算法的程序可以用一组或多组相同的统计数据区分

(2) **事前分析估计**：依据的算法选用何种策略、问题的规模、程序语言、编译程序产生机器代码质量、机器执行指令速度。

6. 时间复杂度的渐进表示法

只需要考虑当问题规模充分大时，算法中基本语句的执行次数在渐近意义下的阶。

算法的时间复杂度：算法中基本语句重复执行的次数是问题规模n的某个函数f(n)，算法的时间量度记作： $T(n)=O(f(n))$ 。表示随着n的增大，算法执行的时间的增长率和f(n)的增长率相同，称渐近时间复杂度。

【重要】分析算法时间复杂度的基本方法

→找出语句频度最大的那条语句作为基本语句

→计算基本语句的频度得到问题规模n的某个函数f(n)

→取其数量级用符号“O”表示

时间复杂度是由嵌套最深层语句的频度决定的。

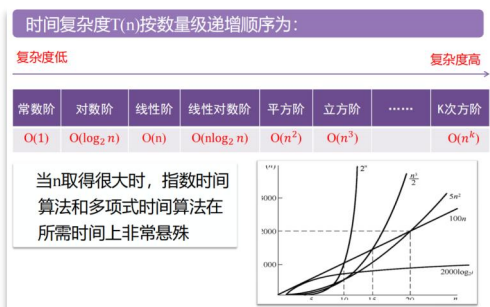
$$f(n) = 2n^3 + 3n^2 + 2n + 1$$

当n趋向无穷大时：

$$\lim_{n \rightarrow \infty} f(n) / n^3 = \lim_{n \rightarrow \infty} (2n^3 + 3n^2 + 2n + 1) / n^3 = 2$$

• 即当n充分大时，f(n)和 n^3 之比是一个不等于0的常数，即f(n)和 n^3 是同阶（同数量级）的。

• 用“O”来表示数量级，记作 $T(n)=O(f(n))=O(n^3)$



● 最好、最坏和平均时间复杂度

➤ **最好时间复杂度**：最好情况（最小值）

➤ **最坏时间复杂度**：最坏情况（最大值）

➤ **平均时间复杂度**：在所有可能情况下，按照输入实例以等概率出现时，计算加权平均值

➤ 很多情况下，平均时间复杂度难以确定，通常只讨论最坏时间复杂度（即上界）

➤ 本书内容均指最坏时间复杂度（除特别指明）

7. 空间复杂度

● 空间复杂度

➤ 采用渐进空间复杂度，记作：

$$S(n) = O(f(n))$$

● 算法占据的空间

① **算法本身要占据的空间**：输入/输出，指令，常数，变量等

② **对数据进行操作的辅助存储空间**

➤ 若算法执行时所需要的辅助空间相对于输入数据量而言是个常数，则称这个算法在原地工作，辅助空间为 $O(1)$ ；

➤ 有的算法需要占用临时的工作单元数与问题规模n有关。

● **示例**：数组逆序，将一维数组a中的n个数逆序存放到原数组组中。

$$S(n) = O(1)$$

原地工作

$$S(n) = O(n)$$

【算法1】
for(i=0; i<n/2; i++)
{
 t=a[i];
 a[i]=a[n-i-1];
 a[n-i-1]=t;
}

【算法2】
for(i=0; i<n; i++)
 b[i]=a[n-i-1];
for(i=0; i<n; i++)
 a[i]=b[i];

第六章 操作系统简介

1. BIOS

所有设备在开机启动过程中都包括三个阶段：**启动自检阶段**、**初始化启动阶段**、**启动加载阶段**。

这三个阶段主要由 **BIOS (Basic Input Output System)** 来完成的。

BIOS 是一组程序，包括基本输入输出程序、系统设置信息、开机后自检程序和系统自启动程序。

(1) 启动自检阶段

电脑刚接通电源，将读取 BIOS 程序，并对硬件进行检测。

这个检测过程也叫做**加电自检 (Power On Self Test, POST)**。功能是检查电脑整体状态是否良好。

(2) 初始化启动阶段

根据 BIOS 设定的启动顺序，找到优先启动的设备。

(3) 启动加载阶段

BIOS 会指定启动的设备来读取硬盘中的操作系统核心文件。

由于不同的操作系统具有不同的文件系统格式（如 FAT32，NTFS，EXT4 等等），因此需要一个启动管理程序来处理核心文件的加载，这个启动管理程序就被称为 **Boot Loader**。

(4) 内核装载阶段

操作系统利用内核程序，开始测试并驱动各个外围设备，包括存储装置、CPU、网卡、声卡等。

(5) 登录阶段

计算机主要完成以下两项任务：启动所有需要自动启动的 Windows 服务；显示登录界面。

2. 操作系统（计算机管理控制程序）

操作系统（OS）是管理计算机硬件与软件资源的计算机程序，是**软件与硬件的中间接口**。

操作系统很喜欢 sleep！所以要叫醒它！叫醒操作系统的方式叫做“**中断**”。中断的来源有三种。

【重要】中断的来源

- **硬件中断 (Hardware Interrupt)**：指由硬件发出的中断，包括 I/O 设备发出的数据交换请求、时钟中断等等。
- **软件中断 (Software Interrupt)**：指由应用程序触发的中断，就是正在执行的软件需要操作系统提供服务。软件中断主要包括各种系统调用（system calls），为应用程序提供不同的服务。
- **异常 (Exception)**：指当系统运行过程中出现了一些非正常事务，需要操作系统进行处理。例如用户程序读写一个地址，而这地址被保护起来，是不能被用户程序读写的，这也会发生异常中断。但是，异常并不全是错误。

3. 操作系统要管理的硬件资源最主要包括：**各种各样的 I/O 设备、计算资源、存储资源**。

(1) **I/O 设备**：**可以与计算机进行数据传输的硬件**。指键盘，显示器，U 盘等这些常用的设备，操作系统需要统一对这些硬件进行管理。

(2) **计算资源**：主要指 **CPU**。CPU 通常使用**轮询**和**硬件中断**两种方式检测设备的工作状态。

轮询方式的三个弊端：1) 检测中断速度慢；2) 可能存在设备处于“饥饿”状态，某一个设备有请求却一直得不到 CPU 的响应；3) 系统处理中断事务不灵活。

(3) **存储资源**：通常包括内存和外存，内存是 CPU 直接通过**系统总线**来访问的，而外存是通过**标准的 I/O**来管理的。

4. 驱动程序

任何新硬件如果要连接到计算机，必须提供**驱动程序（Device Driver）**。

（1）内核态（Kernel Mode）与用户态（User Mode）

绝对不允许用户程序直接访问硬件设备。必须要 CPU 提供硬件的支持。

禁止用户程序直接访问硬件设备的基本思想是 CPU 将指令集分为**需要特权的指令（Privileged）**和**一般的指令（Non-Privileged）**。而**所有的 I/O 指令都是属于需要特权的指令**。一般用户不能执行这类 privileged 指令，必须是**系统内核才能执行这类 privileged 指令**。

在 CPU 有个特殊的寄存器叫做**状态寄存器（Status Register）**，显示现在的 CPU 是在**内核态还是用户态**。假如 CPU 是在用户态，那么任何的 privileged 指令**都不可以执行**，一旦执行了，CPU 就发生异常错误。

CPU 如何从用户态转成内核态，这是现代操作系统的一个重要的技术：必须要使用**“中断”方式**，只有中断，CPU 才会进入内核态。

（2）系统调用（System Call）—— 软件中断

操作系统中设置了一组用于实现系统功能的子程序，称为**系统调用函数**。

系统调用函数的操作一定运行于内核态，而普通的函数调用由函数库或用户自己提供，运行于用户态。

（3）常用系统调用

① 进程控制，如：fork(), exit()等。

② **文件系统操作控制，如：read(), write()等。**

③ 系统控制，如：ioctl(), time()等。

④ 内存管理，如：mmap(), mprotect()等。

⑤ 网络管理控制，如：sethostname(), socket(), bind()等。

⑥ 用户管理，如：getuid(), getgid()等。

⑦ 进程间通信，如：signal(), kill()等。

在今后的学习工作中，**文件操作是最为常用**，包括文件的**打开、创建、读、写等系统调用**。

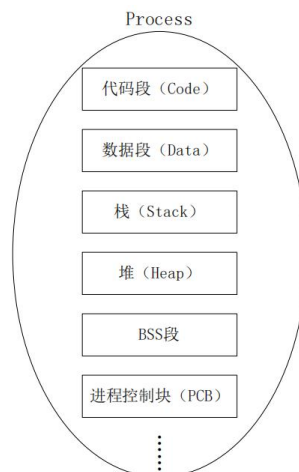
5. 进程

进程调度：在系统进行任务的换入，换出操作时，需要确定哪些任务需要换入 CPU 中执行，而哪些任务需要继续等待。

进程是一个程序的一次执行，包含了其执行时**所有的环境信息**。程序运行首先需要将源代码转化为**可执行程序**，其次还需要操作系统为其提供一个**运行环境**，而**这个运行环境就是进程**。

一个进程包含了**代码段、数据段、栈、堆、BSS 段以及进程控制块等部分**。

- **代码段**通常是指用来存放程序执行代码的一块内存区域。
- **数据段**通常是指用来存放程序中已经初始化的全局变量的一块内存区域。
- **栈（Stack）**是用户存放程序临时创建的局部变量区域。
- **堆（Heap）**是用于存放进程运行中动态分配的内存段，大小并不固定，可根据需要动态扩张或缩减。
- **BSS 段（Block Started by Symbol）**通常是指用来存放程序中未初始化的全局变量的一块内存区域。
- 为了统一管理进程，**进程控制块（Process Control Block, PCB）**，用来记录进程的特征信息，描述进程运动变化的过程。PCB 是操作系统感知进程存在的唯一标识，进程与 PCB 是一一对应的。



6. 进程状态——“三状态模型”

