

大计基笔记

PKstu

彭康书院学业辅导与发展中心出品



彭.大计基笔记（背诵版）

计算机发展的四个阶段

电子管计算机 (第一台:1946 年) -> 晶体管计算机 -> 集成电路 -> 大规模或超大 规模集成电路

微机系统组成

微机系统组成	硬件系统（外部设备与主机系统的关系是信息交换	主机系统：CPU（核心）， 存储器， 输入输出接口（IO接口）， 总线
		外部设备：键盘鼠标扫描仪/显示器打印机/软驱（被淘汰掉了）、硬盘（盘片 + 驱动器一体）、光驱（光盘驱动器、（输入/输出/存储设备）
	软件系统	系统软件：（1）操作系统（核心） （2）系统应用程序
		应用软件：

****注意****：我们发现19年仲英出版的大计基小助手将系统软件与应用软件的下属混淆

硬件系统

主机：CPU 存储器 接口 内存条 主机板

判断的充要条件：

能否与处理器（CPU)进行直接信息交流（不通过接口）

注意：主机箱≠主机（引入主机板）

外设=输入 + 输出

输入设备：向计算机输入信息的设备

输出设备：接受计算机输出信息的设备。（注意：一个设备可以分兼两职，磁盘写入数据时为输出设备，读取设备为输入设备）

软件系统：系统 + 应用

功能：管理、监控、维护计算机软硬件的软件。

分级：

操作系统（OS）：

存储器管理、文件管理、进程管理、设备管理等。运行的一切系统软件与应用程序依赖于OS的支持。

系统应用程序：

语言及处理程序、监控管理程序、支撑软件

主机

CPU

定义：

中央处理器(微处理器)， 提供运算和控制功能

(复习：相当于图灵机中的读写头)

结构

寄存器， 运算器， 控制器， 与片内总线

- ALU运算器：算术逻辑单元：完成算术计算与逻辑计算（例如：与或非逻辑）
- CU控制器：指挥控制**中心**
- REGISTERS内部寄存器组：暂时存储数据的单元。避免 CPU 频繁访问存储器, 缩短运行时间。一个专用的寄存器：PC寄存器， 程序计数器。

存储器

定义：存放数据，完成**读&写**两个基本操作

读：取出数据，原有数据不挂失

写：保存数据，原有数据被覆盖

（复习，微机系统组成中的外设，包括输入与输出两个功能）

内存存储器

特点:

1. 可与 CPU 直接进行信息交换.
2. 存储速度快, 容量小, 单位字节容量价格高.
3. 断电后, 存储信息丢失.

区分 ROM 与 RAM: ROM (只读储存器) 在断电时数据不丢失; RAM (随机储存器) 在断电时数据会丢失.

外存储器

外存：属于外部设备

1. 记录面=磁头数
2. 磁道：记录面上的同心圆
3. 扇区：每个磁道划分为扇区
4. 读写方式：磁头径向进退找磁道，盘片旋转读扇区。

存储容量计算

硬盘的存储容量=**磁头数×柱面数×扇区数×扇区容量**

一般每扇区容量是：512B（字节）（复习，单位是bit）

存储单元

定义：内存按照单元组织；

1. 每个单元大小为一个字节
2. 每个单元都对应一个地址，以实现单元内容的寻址

内存容量

来源：地址码（用二进制表示）地址码长度体现内存容量。

位和字节：

位：bit,是存储的最小数据单位。

字节：B,基本的数据单位，一个字母占据1B,一个汉字占据2B。

换算：

1B=8bit

1kB= 2^{10} B（1024，全都用1024！）

1MB= 2^{10} kB= 2^{20} B

1GB= 2^{10} mb= 2^{30} B

I/O 接口

作用示意: CPU \leftrightarrow I/O 接口 \leftrightarrow 外设

功能: CPU 与外设的速度匹配, 信息的输入输出, 信息的转换, 总线隔离.

主机板

芯片

典型芯片组：南桥&北桥芯片

北桥芯片：

北桥芯片是芯片组的核心，主要负责处理CPU,内存存储器和显卡三者之间的“交通”。发热量较大，故加装散热片。

南桥芯片：

主要负责硬盘等存储设备与PCI(外围器件互联) 之间的数据流通。

总线

1. **按层次结构**，可分为：CPU总线，系统总线，外设总线。
2. **按照传送信息类型**，分为：地址总线、数据总线、控制总线。
 - **前端总线**：指从CPU引脚上引出的连接线。用于实现CPU与主存储器、控制芯片组，I/O接口芯片以及其他CPU之间的连接。
 - **系统总线**（I/O通道总线）：主机系统与外围设备的通信通道。
 - **外设总线**：计算机主机与外部设备接口总线

计算性

计算的必要条件：能够被图灵机完成的工作。

“不可解决性” 反映在两方面：无限步骤，无限时间。

图灵机

图灵机不是一种具体的机器，而是一种思想模型

组成：

1. 一条无限长的纸带Type（对应计算机中的存储器）
2. 一个读写头Head（对应计算机中的处理器cpu）
3. 一套控制规则Table（对应程序，指令）
4. 一个状态寄存器

习题

1. 计算机系统的主要资源有（A）
 - A.处理器、存储器、I/O 设备、程序和数据
 - B.处理器、存储器、接口、程序和数据
 - C.处理器、存储器、接口、外部设备
 - D.处理器、存储器、I/O 设备、用户接口
2. 以下哪种说法是错误的（A）
 - A.计算机系统是指包括外部设备在内的所有硬件
 - B.系统软件是管理、监控和维护计算机软硬件资源的软件
 - C.主板上的 BIOS 芯片中保存了控制计算机硬件的基本软件
 - D.现代微型计算机均采用了多总线系统结构
3. 微型计算机系统的概念结构由（A）组成
 - A.微处理器，总线，存储器，输入输出设备，I/O 接口，软件系统
 - B.微处理器，总线，存储器，输入输出设备，I/O 接口，主机
 - C.微处理器，主机，存储器，输入输出设备，I/O 接口
 - D.微处理器，存储器，输入输出设备，I/O 接口，软件系统
4. 计算机的字长是指（C）
 - A.计算机所能处理的数据的位数
 - B.数据总线的位数
 - C.一次能处理的二进制数据的位数
 - D.存储器地址的位数
5. 微处理器主要包含了（B）等几个功能部件
 - A.总线、计算器、控制器、存储器
 - B.寄存器、运算器、控制器、片内总线（内部总线）
 - C.程序计数器、指令寄存器、指令译码器、操作控制电路
 - D.运算器、移位器、数据暂存器、控制器
6. 存储容量为 1 KB 的存储器具有（C）
 - A.1000 个存储单元
 - B.1000 个二进制元
 - C.1024 个存储单元
 - D.1024 个二进制元
7. 计算机主板上的北桥芯片（A）
 - A.负责 CPU、存储器、显示接口之间的数据交换

- B.负责系统总线和 I/O 接口之间的数据交换
 - C.负责 CPU 和南桥之间的数据交换
 - D.负责 CPU 和 I/O 接口之间的数据交换
8. I/O 接口是指 (D)
- A.微处理器与存储器之间的接口
 - B.外部设备与存储器之间的接口
 - C.外部设备中用于与主机进行数据传输的接口
 - D.主机与外部设备之间的接口
9. 外部设备 (D)
- A.只能通过总线连接到主机
 - B.可以通过芯片连接到主机
 - C.可以直接与主机的系统总线连接
 - D.必须通过输入输出接口连接到主机
10. 在 PMA 方式的 I/O 数据传输过程中(A)
- A.控制数据传输过程的是专用的硬件
 - B.控制数据传输过程的是 CPU
 - C.控制数据传输过程的是存储器
 - D.控制数据传输过程的是总线

二进制

算术运算

加法规则： $0+0=0$ $0+1=1$ $1+0=1$ $1+1=10$

乘法规则： $0\times 0=0$ $0\times 1=0$ $1\times 0=0$ $1\times 1=1$

减法规则： $0-0=0$ $1-0=1$ $1-1=0$ $0-1=1$ (有借位)

运算规律

加法

计算10110110B和01101100B的和。

进 位	1 11111000
被 加数	10110110
加 数 +)	01101100
	1 00100010

减法运算规则

计算11000100B和00100101B的差。

借 位	01111110
被减数	11000100
减 数 -)	00100101
	10011111

二进制乘法原理：

输入			输出	
Ci-1	Ai	Bi	Si	Ci
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

http://blog.csdn.net/qq_36148847

二进制除法

110.1

110

100111

110

111

110

110

110

0

数制及其转换

不同进制的表示方法：

- (1) 括号右下角加进制数
例如(1011)₂、(56)₈、(987)₁₀、(1A3)₁₆
- (2) 数字后加字母
H 十六进制 D 十进制 B 二进制 O 八进制

十进制转化为N进制

整数部分:除N取余数
小数部分: 乘N取整数

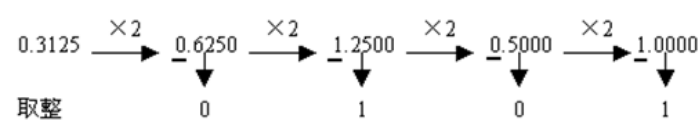
例1:将十进制整数25转换为二进制数

2	25	余数
2	12	1 —— 最低位
2	6	0
2	3	0
2	1	1
	0	1 —— 最高位

例2: 将十进制小数0.3125转换为二进制数。

用2乘十进制小数，可以得到积，将积的整数部分取出，再用2乘余下的小数部分，又得到一个积，再将积的整数部分取出，如此进行，直到积中的整数部分为零，或者整数部分为1，此时0或1为二进制的最后一位。

所以：0.3125D=0.0101B



具体到题目中所说的(0.787)₁₀ (即：0.787D)

0.787* 2=1.574取出整数部分1

0.574* 2=1.148取出整数部分1

0.148* 2=0.296取出整数部分0

0.296* 2=0.592取出整数部分0

0.592* 2=1.184取出整数部分1

0.184* 2=0.368取出整数部分0

0.368* 2=0.736取出整数部分0

0.736* 2=1.472取出整数部分1

故 $(0.787)_{10} = (0.11001001\dots)_B$

例3: 将十进制整数193转换为十六进制数。

除以“十六取余”

因此, $193D=C1H$

N进制转化为十进制数

过程:

(1) N进制数按权展开

(2) 展开的算式按十进制运算

运算结果为转换后的十进制数

例 4:将二进制数100110.101转换成十进制数。

$$\begin{aligned} & 100110.101_B \\ &= 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 32 + 0 + 0 + 4 + 2 + 0 + 0.5 + 0 + 0.125 = 38.625_D \end{aligned}$$

例5: 将十六进制数2BA转换成十进制数。

$$\begin{aligned} 2BA &= 2 \times 16^2 + B \times 16^1 + A \times 16^0 \\ &= 512 + 176 + 10 \\ &= 69 \end{aligned}$$

机器数

定义: 计算机中存储和处理二进制进制数。

分类

有符号数: 最高位表示数据的正或负

例如一个字节的范围-128~+127

无符号数: 最高位也是绝对值的一部分

例如一个字节的范围0~255

有符号数入门

通过符号位与数值位表示。后面的数值位的意思是**真值**, 也即**对应着**绝对值。

+52=0 0110100

-52=1 0110100

在计算机一开始发明的时候, 同时处理8位

原码

定义:

最高位符号位, 后面是真值。其中, 0表示正值, 1表示负值。

用 $[X]_{\text{原}}$ 表示原码。

缺点:

这个方法对计算机很麻烦。

1. 在运行程序: 判断符号, 判断真值, 进行运算的过程中, 计算机要做的事情太多了。

2. 0的表示存在混淆, ± 0 的表示不同。

原因: 00000000/10000000都是0。

反码:

正: 反码等于原码

负: 符号位不变, 然后按位取反。

取-52

反码表示=1 1001011

取0

+0: 00000000

-0: 11111111

但是发现：依旧没有解决0的表示唯一性问题

补码：

正：补码=原码=反码

负：补码=反码+1（复习：反码：符号位不变，数值位按位取反）

取-52

反码：11001011

补码：11001100（复习：二进制的四则运算）

补码的还原：

如果是正的，第一位符号位没的说，后面就是真值，补码=原码=反码

如果是负的，此补码再取补码

0的补码

+0: 00000000

-0: 原码：10000000——反码：11111111——补码100000000

变成了九位，此时1看不到了，0的表示唯一了

补码的作用：

把减法运算转化为加法运算

$$(X - Y)_{\text{补}} = X_{\text{补}} + (-Y)_{\text{补}}$$

例如：66-51

66: 01000010; -51=11001101

相加=100001111，此时头上的1看不到，而00001111=15

总结：取补码运算的程序

首先明确是相减运算，然分别转化为补码，进行相加，如果出现9位溢出，那么首位放弃。

编码

ASCII 码 (美国标准信息交换代码)

标准 ASCII 码: 7 位二进制码表示一个符号, 可表示 128 个字符, 最高位默认为 0.

扩展 ASCII 码: 八位表示一个字符.

重点: 大写字母与其对应的小写字母 ASCII 码之差为 32.

计算机网络

按覆盖区域分类

1. 广域网

特点: 覆盖地理范围大 (几十千米到几千千米), 传输速率低, 传播延迟大, 适应大 容量与突发性要求...

2. 局域网

特点: 覆盖地域小 (几米到几十千米), 传输速率高, 传播延迟小

3. 城域网

特点: 覆盖范围在广域网和局域网之间, 运行方式与局域网类似

按拓扑结构分类

1. **星型结构**: 优点: 易于构建, 扩充; 控制相对简单. 缺点: 对中心节点的可靠性要求比 较高
2. **树形结构** (层次结构): 规模较大的网络一般都采用树形结构
3. **总线型结构**: 优点: 结构简单, 成本低. 缺点: 实时性较差
4. **环形结构**
5. **点到点部分连接的不规则形结构 (网状结构)**: 多用于广域网
6. **点对点的全互联结构**

TCP/IP 协议及其体系结构

TCP/IP 协议有四个层次, 从上到下分别是应用层, 传输层, 网际层, 网络接口层.

1. **应用层**: 包含了很多面向应用的协议, 如简单邮件传输协议 (SMTP), 超文本传输 协议 (HTTP), 文件传输协议 (FTP).
2. **传输层**: 两个主要传输协议: 无连接的用户数据报协议 (UDP, 不可靠传输协议), 面 向连接的传输控制协议 (TCP, 可靠传输协议). 两者的主要区别在于, 前者只负责 发送数据, 无需提前建立连接, 发后不管; 后者会建立连接, 与接受的主机进行多次 反馈.
3. **网际层**: 又称互联网层或网络层. 为网络上不同主机提供通信服务, 及解决主机到 主机的通信问题. 核心协议是 IP 协议.
4. **网络接口层**: 网络接口层没有定义什么具体内容, 一般用 OSI 模型中的数据链路层 和物理层代替.

网络应用模式

这一部分熟悉 C/S 模式和 B/S 模式即可,P2P 模式了解即可.

- C/S 模式: 客户/服务器模式, 有客户端
- B/S 模式: 浏览器/服务器模式, 无专门的客户端, 以网络浏览器为客户端

IP地址与端口号

IPv4 地址 IPv4 地址用三十二位二进制编码表示. 为了便于记忆, 人们将 32 位 IP 地址分为四 个字节, 用等效十进制代替.IP 地址有 A,B,C,D,E,F 五类, 常用的 IP 地址是 A,B,C 三 类.A 类给大型网络用,B 类分配给中等规模的网络,C 类给规模较小的局域网.

- **A类**: 网络号有七位, 可使用的网络号有 $126(2^7 - 2)$ 个, 网络号全为 0 的 IP 地址 和网络号为 127(01111111) 不允许使用. 每个 A 类地址网络 允许最大主机数为 $16\,777\,214(2^{24} - 2)$
- **B类**: 网络号有 14 位. 规定 128.0.0.0 不能使用, 可用网络号有 $16\,383(2^{14} - 1)$ 个, 最大主机数 $65\,534(2^{16} - 2)$.
- **C类**: 网络号有 21 位, 规定 192.0.0.0 不能使用, 可用网络号有 $2\,097\,151(2^{21} - 1)$ 个. 最大主机数 $254(2^8 - 2)$ 此外还有一部分私有地址 不可使用, 私有地址用于没有合法的 IP 地址的单位或家 庭用户自己组建局域网.

10.0.0.0 10.255.255.255(1 个A类地址块)
172.16.0.0 172.31.255.255(15个B 类地址块)
192.168.0.0 192.168.255.255(1个C 类地址块)

端口号

端口号是用来标识同一主机中不同进程的机制. 如果把 IP 地址看作港口的地址, 端 口号就时港口中泊位的编号. 端口号是传输层和应用层之间数 据交换的一种机制. 一些常见的 TCP 熟知的端口号所对应的应用层协议 (了解)

1. 21 FTP(文本传输协议)
2. 23 Telnet(远程登录)
3. 25 SMTP(邮件传输协议)
4. 80 HTTP(超文本传输协议)
5. 110 POP3(邮局协议)

子网和子网掩码

为了更有效的利用 IPv4 的地址空间, 将主机号的一定位数划分出来作为网络号的一部分, 划分出来的就是子网号. 这样,IP 地址就变成了网络-子 网-主机三个部分组成

下面用一个例子演示如何划分, 有一个 C 类网络 202.117.58.0 划分为四个子网, 那 么需要从主机号中取出两位作为子网号.

子网 1:202.117.8.00000000
子网 2:202.117.8.01000000
子网 3:202.117.8.10000000
子网 4:202.117.8.11000000

子网掩码是用来识别各个子网的, 也是一个 32 位二进制数,IP 地址的网络号和子网 号部分, 子网掩码对应是 1, 对应于主机号部分, 子网掩码中相 应应为 0. 上一个例子中子 网掩码就是 255.255.255.192(11000000)

要得到子网地址, 只需要把 IP 地址和子网掩码进行 ‘与’ 运算即可

习题

- 1. 二进制数 1110111.11 转换成十进制数是(B)
A.119.125 B.119.75 C.119.375 D.119.3
- 2. 下列叙述中,正确的是(D)
A.汉字的计算机内码就是国标码
B.存储器具有记忆能力,其中的信息在任何时候都不会丢失
C.所有十进制小数都能准确地转换为有限位二进制小数
D.所有二进制小数都能准确地转换为十进制小数
- 3. 十六进制数 FF.1 转换成十进制数是(A)
A.255.0625 B.225.125 C.127.0625 D.127.125
- 4. 当二进制数 X=-10000 时,则有
A.[X]原=100000 B.[X]反=100001 C.[X]补=101111 D.[X]补=110000
- 5. 下列二进制运算中,正确的是(D)
A.1 *0=1 B.0* 1=1 C.1+0=0 D.1+1=10
- 6. 为了避免混淆,八进制数在书写时常在后面加字母(B)
A. H B. O C. H D. B
- 7. 有一个数值 152,它与十六进制 6A 数相等,那么该数值是(B)进制数
A. 二 B. 八 C. 十 D. 十六
- 8. 数据 111.H 的最左边的 1 相当于 2 的(A)次方
A.8 B.9 C.11 D.2
- 9. 用汉语拼音输入“西安”两个汉字,输入“xi'an”5 个字符,那么“西安”两字的机内 码所占用的字节数(B)
A. 2 B. 4 C. 8 D. 5

逻辑运算与逻辑门

“与”门和“或”门

“与”(用 \wedge 或者乘号 \cdot 表示) 和 “或”(用 \vee 或者加号 $+$ 表示) 的运算规则:

$$a \wedge b = \begin{cases} 1, a = b = 1 \\ 0, ect \end{cases}$$

$$a \vee b = \begin{cases} 0, a = b = 0 \\ 1, etc \end{cases}$$

$0 \wedge 0 = 0$	$1 \wedge 0 = 0$	$0 \wedge 1 = 0$	$1 \wedge 1 = 1$
$0 \vee 0 = 0$	$1 \vee 0 = 1$	$0 \vee 1 = 1$	$1 \vee 1 = 1$

对于多位的情况, 右对齐按位做与运算即可. 空位补 0.

“非”运算

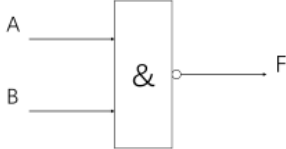
符号: 数值上面加一条横线, 如 $B = \overline{A}$.
运算法则: 按位取反 (0 转 1, 1 转 0). 一位的情况:

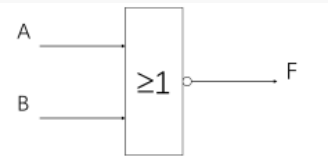
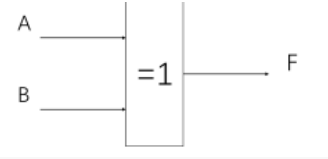

$\overline{0} = 1$

$\overline{1} = 0$

多位的情况逐位操作即可.

其他逻辑运算

名称	符号	运算规则	真值表	门电路逻辑符号
与非	$\overline{A \wedge B}$,	先与后非	$\overline{0 \wedge 0} = 1, \overline{0 \wedge 1} = 1, \overline{1 \wedge 0} = 1, \overline{1 \wedge 1} = 0$	

名称	符号	运算规则	真值表	门电路逻辑符号
或非	$\overline{A \vee B}, \overline{A+B}$	先或后非	$\overline{0 \vee 0} = 1, \overline{0 \vee 1} = 0, \overline{1 \vee 0} = 0, \overline{1 \vee 1} = 0$	
异或	$A \oplus B = \overline{A} \wedge B + A \wedge \overline{B}$	同0异1	$0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0$	
同或	$\overline{A \oplus B}$	同1异0	$\overline{0 \oplus 0} = 1, \overline{0 \oplus 1} = 0, \overline{1 \oplus 0} = 0, \overline{1 \oplus 1} = 1$	

时序逻辑电路

触发器

触发器 (trigger, flip-flop) 也称双稳态多谐振荡器, 在外部触发信号的作用下可以改变电压, 而当外部触发信号不出现, 便可以使输出端状态稳定的保持不变

如图3.5所示的电路中, 两个与非门 G_1, G_2 门输出端与输入端交叉连接到了一起, 构成一种新的功能期间, 称为RS触发器

在 RS 触发器基础上增加两个与非门. 下图位 D 触发器的示意图, 其中 CP=0 时输 出状态保持不变; CP=1 时输出取决于 D 端状态.

作用

1. 触发器是具有记忆功能的逻辑部件 (任何时候输出端都能保持一个确定的稳定状 态 (0 或 1).
2. 一个触发器能够存储 1 位二进制数.
3. 触发器是构成计算机存储装置的基本单元

半加器

实现两个1位二进制数相加。有加数和被加数以及“和”与“进位”两个输出，不考虑来自低位进位的加法器

全加器

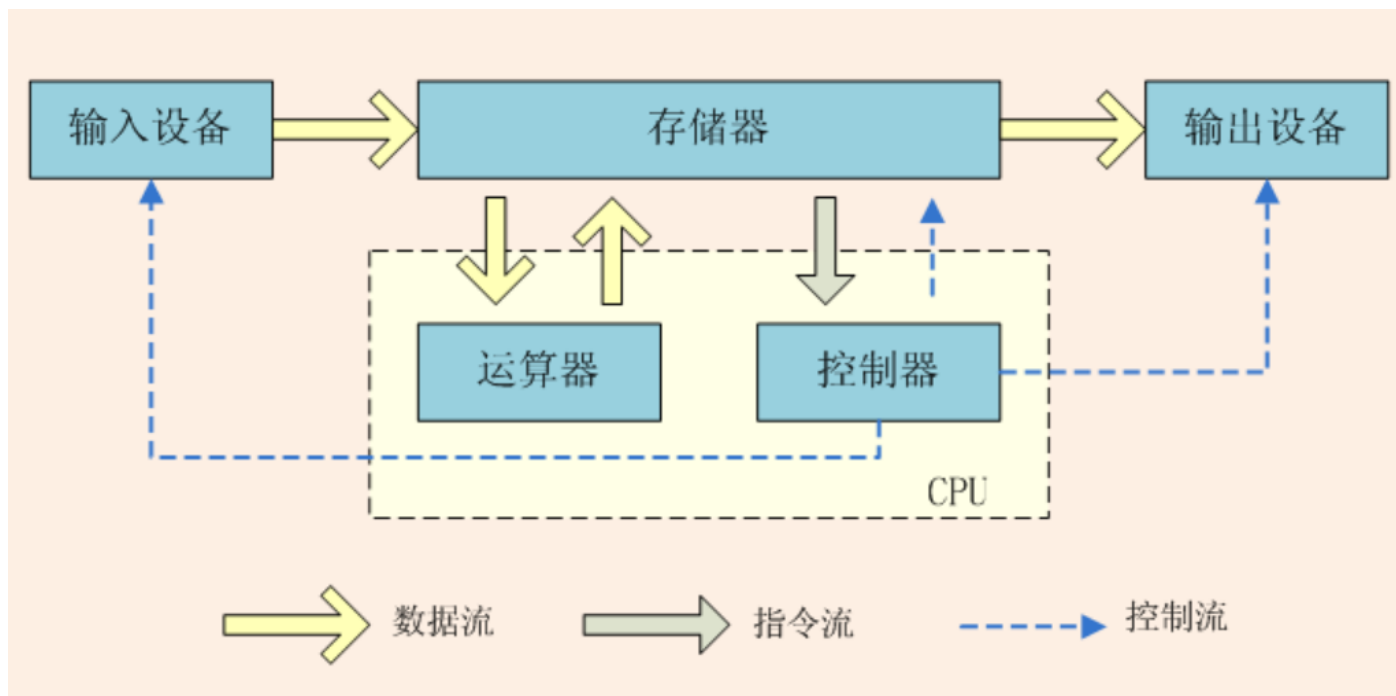
如果在半加器中添加一个或门来接受低位的进位输出信号，则两个半加器构成一个全加器

如图，CARRY IN是来自低位的进位信号，CARRY OUT是向高位输出的进位信号，SUM是相加的和。



冯诺依曼结构与原理

1. 采用二进制: 计算机中所有信息 (数据和指令) 统一用二进制表示.
2. 设计计算机硬件由五个部分构成: 运算器, 逻辑控制装置, 存储器, 输入设备, 输出 设备
3. 存储程序原理: 如图 2.8. 特点: 以运算器为核心, 所有信息的输入和输出都需要通 过运算器.



指令与程序

概念

1. **指令**: 控制计算机完成某项操作的, 能够被计算机识别的“命令”(二进制形式描述的机器指令).
2. **指令系统**: 计算机能够识别的所有指令的集合.
3. **程序**: 按一定顺序组织在一起的指令序列.

指令格式

指令的格式为操作码 + 操作数. 其中操作码说明指令的功能, 操作数说明指令操作的对象

程序计数器 (PC)

步骤:

1. PC 用来产生和存放下一条将要读取的指令的地址.
2. PC 每输出一次地址, 就指向内存的一个单元, CPU 将该单元的指令自动取出.
3. PC 中内容自动加 1, 准备读取下一条指令.

PC 的作用: 程序执行的“指挥棒”. PC 指向哪里, CPU 就到哪里取指令

两种执行方式比较

1. **顺序执行**: 一条指令执行完了再执行下一条指令.
2. **并行执行**: 同时执行两条或多条指令.

时间比较: 并行拥有更高的效率, 同时用于更高的复杂度. 相比顺序执行, 并行执行的优势用速率比

冯诺依曼计算机基本原理

存储程序控制原理, 以运算器为核心, 采用二进制.

进一步可表示:

1. 将计算过程描述为由多条指令按一定顺序组成的程序, 并放入存储器保存;
2. 指令按其在存储器中存放的顺序执行;
3. 由控制器控制整个程序和数据的存取以及程序的执行.

冯诺依曼系统的局限性

1. CPU 与存储器之间会有大量的数据交互, 造成总线瓶颈.

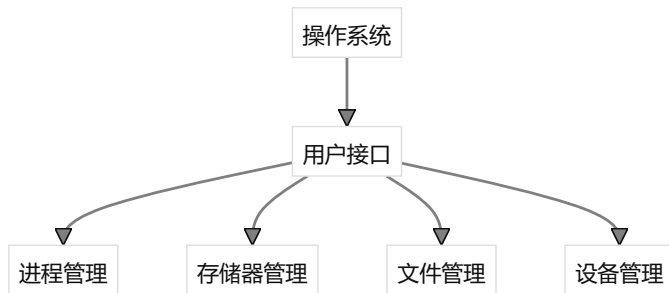
2. 指令的执行顺序由程序计数器控制, 使得即使有关数据已经准备好, 也必须逐条执行指令序列.
3. 指令的执行顺序由程序决定, 对一些大型的, 复杂的任务是比较困难;
4. 以运算器为中心, I/O 设备与存储器间的3数据传送都要经过运算器, 使处理效率, 特别是对非数值数据的处理效率比较低.

操作系统 (OS)

概念

操作系统是一组控制和管理计算机软、硬件资源, 为用户提供便捷使用计算机的程序集合; 是用户和计算机之间进行“交流”的界面.

基本功能



进程

概念:

1. 进程是程序的一次执行过程. 是系统进行资源分配和调度的一个独立单位. (一个程序可以对应多个进程, 一个进程也可以对应多个程序)
2. 在多道程序环境中, 对系统内部资源的分配和管理是以进程为基本单位.
3. 任何程序要运行, 都必须为它创建进程.

进程与程序的关系 :

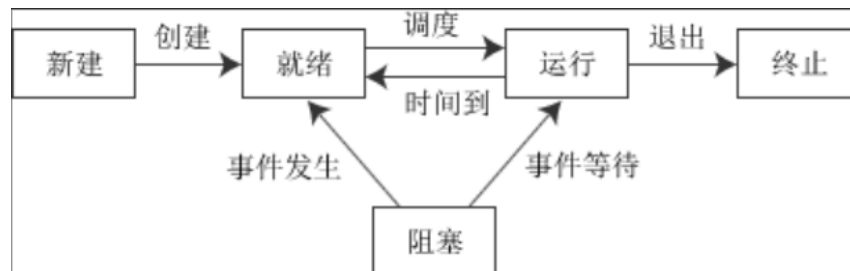
1. 进程是动态的, 程序是静态的: 进程是程序的一次执行, 程序是有序代码的集合
2. 进程是暂时的, 程序是永久的: 进程有生命周期, 会消亡, 程序可长期保存在外存储器上
3. 进程与程序密切相关: 同一程序的多次运行对应到多个进程; 一个进程可以通过调用激活多个程序.

进程的基本状态与状态转换

受资源的制约, 进程在其生命周期中的执行过程是间断性的. 有三种基本状态:

1. **就绪状态**: 已经获得了除 CPU 之外所必需的一切资源, 一旦分配到 CPU, 就可立即执行.
2. **运行状态**: 已经获得 CPU 及其它一切所需资源, 正在运行.
3. **等待状态**: 由于某种资源得不到满足, 进程运行受阻, 处于暂停状态, 等待分配到所需资源后, 再投入运行.

进程之间的转换如下图



存储器管理的主要功能

1. 负责将程序从联机外存储器 (硬盘) 调入内存 地址变换: 将程序中的地址对应到内存中的地址 存储分配: 为程序分配相应的内存空间
2. 将硬盘和内存实行统一的管理 (存储器系统)
3. 存储扩充: 解决在小的存储空间中运行大程序的问题
4. 存储保护: 保护各类程序及数据区免遭破坏

存储器扩充

- 1. 主导思想: 如何在有限的内存空间中, 处理大于内存的程序.
- 2. 虚拟存储技术
 - 将内存与部分硬磁盘统一在一起管理, 使其构成一个整体, 从而将部分外存空 间作为内存使用.
 - 从用户的角度, 相当于有一个容量足够大的内存空间. 用户可以在这个地址空 间内编程 (存储扩充), 而完全不考虑内存的大小.

变量名的命名规则:

- 1. 只能由英文字母, 数字和下划线开头
- 2. 标志符必须以字母或下划线开头

数据类型

数据类型分类				
基本类型	数据类型分类		关键字	变量声明实例
	整型	基本整型	int	int a;
		长整型	long	long a;
		短整型	short	short a;
	实型	单精度实型	float	float a;
		双精度实型	double	double a;
		字符型	char	char a;

常见的转义字符

字符	含义
'\n'	换行
'\r'	回车 (不换行)
'\"'	双引号
'\"'	单引号
'\\'	反斜线
'\?'	问号
'ddd'	ddd表示1~3个八进制数字
'xdd'	dd表示2个十六进制数字

函数printf () 格式转换说明符

说明符	用法
%d	输出带符号的十进制整数, 整数的符号省略
%u	无符号的十进制整数输出
%c	输出一个字符
%s	输出字符串
%f	十进制小数
%e	指数形式
%o	八进制输出
%x	十六进制输出
% %	输出一个百分号

在用scanf () 输入时, 遇到以下几种情况都认为输入结束

- 1. 遇到空格, 回车符, 制表符 (tab)
- 2. 达到输出宽域

字符串处理库函数

功能	一般形式	功能描述
字符串复制	strcpy(str1,str2)	将str2复制到字符数组str1中
字符串连接	strcat(str1,str2);	将str2添加到str1的末尾
字符串比较	strcmp(str1,str2);	当str1大于str2时，函数返回值大于0 当str1小于str2时，函数返回值小于0 当str1等于str2时，函数返回值等于0
求字符串长度	strlen(str);	返回str实际长度，即不包括'\0'

因为这些库函数的说明存放在头文件string.h中，所以如果要在程序中调用这些函数，应在源程序最前面加上一个文件包含的编译预处理命令：

```
#include<string.h>
```

算法

基本概念

算法是计算机处理信息的本质，因为计算机本质上是一个算法来告诉计算机确切的步骤来执行一个指定的任务。

五大特性

- 1. **输入**：具有0个或多个输入
- 2. **输出**：至少有1个输出
- 3. **有穷性**：在有限的步骤之后会自动结束而不会无限循环，并且每一个步骤可以在**可接受的时间内**完成
- 4. **确定性**：算法中的每一步都有确定的含义，**不会出现二义性**
- 5. **可行性**：算法的每一步都是可行的，也就是说每一步都能够执行**有限**的次数完成

描述方法

- 1. **自然语言**，不严格
- 2. **伪代码描述**，用自然语言和计算机语言描述算法
- 3. **流程图描述**

时间复杂度

时间复杂度又称**计算复杂度**，是**算法有效性的**量度之一。时间复杂度是一个算法运行时间的相对量度。

时间频度与时间复杂度

- **时间频度**
一个算法花费的时间与算法中语句的执行次数成正比例，哪个算法中语句执行次数多，它花费时间就多。
一个算法中的语句执行次数称为语句频度或时间频度。记为 $T(n)$ 。
- **时间复杂度**
在 $T(n)$ 中， n 为问题的规模。
一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数，用 $T(n)$ 表示。
若有某个辅助函数 $f(n)$,使得当 n 趋近于无穷大时， $T(n)/f(n)$ 的极限值为不等于零的常数，则称 $f(n)$ 是 $T(n)$ 的同数量级函数。记作 $T(n) = O(f(n))$,称 $O(f(n))$ 为算法的渐进时间复杂度，简称时间复杂度。

常见的时间复杂度量级

- 常数阶 $O(1)$
- 对数阶 $O(\log N)$
- 线性阶 $O(n)$
- 线性对数阶 $O(n\log N)$
- 平方阶 $O(n^2)$
- 立方阶 $O(n^3)$
- k 次方阶 $O(n^k)$

- 指数阶 $O(2^n)$

上面从上至下依次的时间复杂度越来越大，执行的效率越来越低。

为了更好的理解时间复杂度的概念以及了解时间复杂度量级，以下几个例子。

常数阶 $O(1)$

```
int i = 1;    //1
int j = 2;    //1
int m = i + j; //1
//T(n)=3
//f(n)=1
```

像以上例子，语句执行总次数只与语句条数有关，而与变量大小无关。
这样的代码无论有多长，其时间复杂度都可以被表示为 $O(1)$ 。

线性阶 $O(n)$

```
for(i=1; i<=n; i++) //1
{
    j = i;           //n
    j++;             //n
}
//T(n)=2n+1
//f(n)=n
```

像以上例子，在循环中，循环里的代码会被执行 n 遍。因此，执行代码消耗时间与 n 有关。这类代码其时间复杂度都可以被表示为 $O(n)$ 。

对数阶 $O(\log N)$

```
int i = 1;    //1
while(i<n)    //1
{
    i = i * 2; //log2(n)
}
//T(n)=2+log2(n)
//f(n)=log2(n)
```

像以上例子，该循环一共循环了 $\log_2 n$ 次，其中的语句执行次数为 $\log_2 n$ 。
因此，该代码的时间复杂度为 $O(\log_2 n)$ 。

线性对数阶 $O(n \log N)$

线性对数阶可以理解为将时间复杂度为 $O(\log N)$ 的代码循环 n 次，如以下例子：

```

for(m=1; m<n; m++) //1
{
    i = 1;          //m
    while(i<n)      //m
    {
        i = i * 2; //m*log2(n)
    }
}
//T(n)=m*log2(n)+2m+1
//f(n)=nlog2(n)

```

平方阶 $O(n^2)$

如果把 $O(n)$ 的代码再嵌套循环一遍，它的时间复杂度就是 $O(n^2)$ 了。

```

for(x=1; i<=n; x++) //1
{
    for(i=1; i<=n; i++) //n
    {
        j = i;          //n*n
        j++;             //n*n
    }
}
//T(n)=2*n*n+n+1
//f(n)=n*n

```

空间复杂度

- **空间复杂度**

算法在计算机内执行时所需存储空间的度量，也是问题规模 n 的函数。
空间复杂度不是程序占用了多少bytes的空间，算的是变量的个数。

- **常见的空间复杂度**

$O(1)$, $O(n)$, $O(\log n)$, $O(n^2)$

排序算法

排序分为：

- **内部排序**：排序过程仅在内存中进行（数据量少）
- **外部排序**：排序过程仅在内外存中进行（数据量少）

排序算法有很多：

- 冒泡排序
- 选择排序
- 快速排序
- 插入排序
- 希尔排序
- 堆排序
-

本节**必须掌握冒泡排序**，**建议掌握选择排序**，**快速排序**仅作了解。

查找算法

若一组待查找数据中存在给定值，则称查找是**成功**的；否则称查找**不成功**。
待查数据元素的集合称为**查找表**。

常见的查找算法有以下四种：

- 顺序查找
- 折半查找/二分查找
- 插入查找
- 斐波那契查找

关键代码填空

- 顺序表：
创建表，销毁表，插入，查找
- 栈：
创建栈，入栈，出栈，判断是否为空栈
- 队列：
创建队列，入队，出队

注：

- 在以下所有提到的操作中，涉及指针操作均可以用数组进行替代。
- 使用数组时不需进行malloc操作

数据结构的定义

1. **数据**：是描述客观事物的数和字符的集合
2. **数据项**：是具有独立含义的数据最小单位，也称为字段或域
3. **数据对象**：是指性质相同的数据元素的集合，它是数据的一个子集。
4. **数据结构**：是指所有数据元素以及数据元素之间的关系，可以看作是相互之间存在着某种特定关系的数据元素的集合。

数据结构的组成

1. 数据的**逻辑结构**：由数据元素之间的逻辑关系构成
2. 数据的**存储结构**：数据元素及其关系在计算机存储器中的存储表示，也称为数据的物理结构。
3. 数据的运算：施加在该数据上的操作

逻辑结构的类型

1. **集合**：指数据元素之间除了“同属于一个集合”的关系以外别无其他关系
2. **线性结构**：指该数据结构中的数据元素之间存在**一对一**的关系
3. **树形结构**：指该结构中的数据元素之间存在**一对多**的关系
4. **图形结构**：指该结构中的数据元素之间存在**多对多**的关系

存储结构的类型

1. **顺序存储结构**
采用一组连续的存储单元存放所有的数据元素。即所有数据元素在存储器中占有一整块存储空间，而且两个逻辑上相邻的元素在存储器中的存储位置也相邻。
优点：存储效率高，通过下标来直接存储
缺点：不便于数据修改，对元素的插入或删除操作可能需要移动一系列的元素
2. **链式存储结构**
每个逻辑元素用一个内存结点存储，每个结点是单独分配的，所有的结点的地址不一定是连续的。通过指针域将所有结点链接起来。
优点：便于数据修改，对元素进行插入或删除操作只需修改相应结点的指针域，不必移动结点。
缺点：存储空间的利用率低，不能对元素进行随机存取。
3. **索引存储结构**
指在存储数据元素信息的同时还建立附加的索引表。存储所有数据元素信息的表称为主数据表，其中每个数据元素有一个关键字和对应的存储地址。
优点：查找效率高。
缺点：需要建立索引表，增加了空间开销。
4. **哈希（或散列）存储结构**
根据元素的关键字通过哈希（或散列）函数直接计算出一个值，并将这个值作为该元素的存储地址
优点：查找速度快
一般适合要求对数据能够进行快速查找和插入的场合

线性表

线性表：

1. 线性表 (linear list) 是n个具有相同特性的数据元素的有限序列。
2. 线性表在逻辑上是线性结构，但是在物理结构上并不一定是连续的。（这里的物理结构一般指物理地址空间）。

线性表的逻辑特征：

- 在非空的线性表，有且仅有一个开始结点 a_1 ，它没有直接前趋，而仅有一个直接后继 a_2 ；
- 有且仅有一个终端结点 a_n ，它没有直接后继，而仅有一个直接前趋 a_{n-1} ；
- 其余的内部结点都有且仅有一个直接前趋 a_{i-1} 和一个直接后继 a_{i+1} 。
- 元素个数有限。
- 线性表是一种逻辑结构，表示元素之间一对一相邻的关系。
- 线性表顺序存储结构占用一片连续的存储空间。知道某个元素的存储位置就可以计算其他元素的存储位置。

基本操作注意事项：

- 创建线性表时是否创建成功
- 添加数据时表是否已满
- 取出/删除数据时表是否为空

顺序表：线性表的顺序存储结构

顺序表：

1. 是线性表
2. 物理结构上是连续的
3. 顺序表中任意一个数据元素都可以随机存取

顺序表的特点

- 利用数据元素的存储位置表示线性表中相邻数据元素之间的前后关系，即线性表的逻辑结构与存储结构一致
- 在访问线性表时，可以快速地计算出任何一个数据元素的存储地址。因此可以粗略地认为，访问每个元素所花时间相等。这种存取元素的方法被称为随机存取法。

顺序表的优缺点

优点：

- 存储密度大(结点本身所占存储量/结点结构所占存储量)
- 可以随机存取表中任一元素

缺点：

- 在插入、删除某一元素时，需要移动大量元素
- 浪费存储空间
- 属于静态存储形式，数据元素的个数不能自由扩充

顺序表的分类

1. 静态顺序表：使用定长数组存储
2. 动态顺序表：使用动态开辟的数组存储

顺序表操作

```
SqList * CreatList(int n);           //创建线性表
int InsertToList(SqList * sqlist, int pos, int data); //插入元素
int DeleteFromList(SqList * sqlist, int pos);        //删除元素
int GetPriorElement(SqList * sqlist, int n, int * data); //获取元素前驱
int FindElement(SqList * sqlist, int pos, int data);  //查找元素
void PrintList(SqList * sqlist); //输出线性表
void DestroyList(SqList * sqlist); //销毁线性表
```

创建线性表

注意:

返回的是结构体指针!

```
// 创建最大表长度为n的线性顺序表
SqList *CreatList(int n)
{
    SqList *sqlist = (SqList *)malloc(sizeof(SqList));
    // 为线性表分配内存
    if (sqlist != NULL)
    {
        sqlist->list = (int *)malloc(sizeof(int) * n);
        if (sqlist->list == NULL)
        {
            return NULL;
            // 如果分配失败返回NULL
        }
        sqlist->length = 0;
        // 初始化当前顺序表长度为0, 即为空表
        sqlist->MaxLength = n;
        // 设置最大表长度
    }
    return sqlist;
    // 返回顺序表结构体指针
}
```

插入元素

在某一位置插入元素后, 注意**原该位置及之后**的数据全部要后移一位。

C

```
// 在线性表的pos位置插入数据data
int InsertToList(SqlList *sqlist, int pos, int data)
{
    if (pos < 0 || pos > sqlist->length || sqlist->length == sqlist->MaxLength)
        // 假如插入位置不正确或者线性表已满，插入失败
    {
        return -1;
    }
    for (int n = sqlist->length; n > pos; n--)
    {
        sqlist->list[n] = sqlist->list[n - 1];
        // 自pos起，所有元素后移一位
    }
    sqlist->list[pos] = data;
    // 插入新的元素
    return ++sqlist->length;
    // 表长+1，返回表长
}
```

查找元素

就是普通的顺序查找，只不过换成了结构体、指针和函数来表示。

C

```
//在线性表中查找下标>=pos中数据data的下标
int FindElement(SqlList *sqlist, int pos, int data)
{
    //顺序查找
    for (int n = pos; n < sqlist->length-1; n++)
    {
        if(data==sqlist->list[n])
        {
            return n;
            //找到数据返回下标
        }
    }
    return -1;
    //查找失败返回-1
}
```

销毁线性表

释放空间。

C

```
//销毁线性表
void DestroyList(SqlList *sqlist)
{
    free(sqlist->list);
    sqlist->length=0;
    //释放空间
}
```

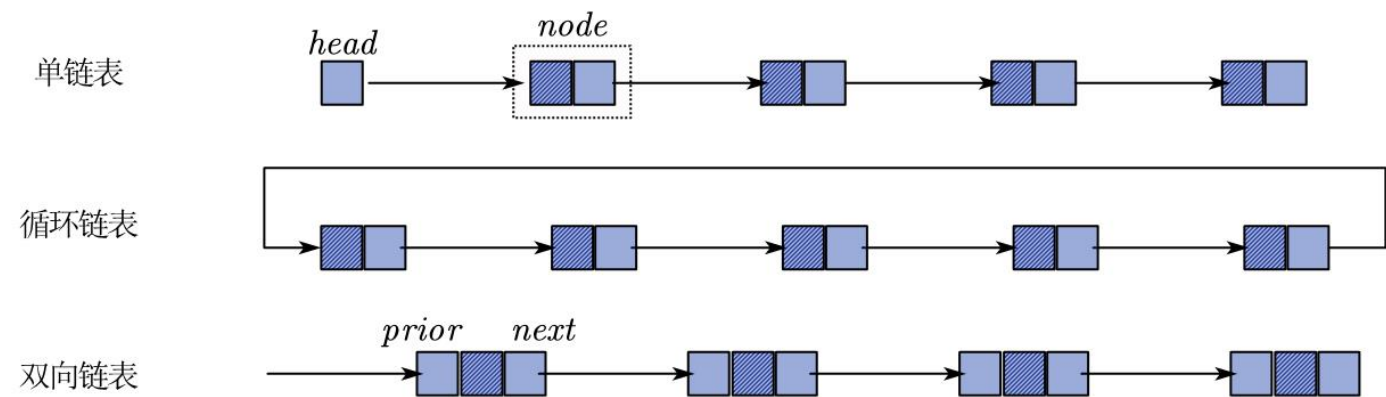
链表：线性表的链式存储结构

链表特点

- 结点在存储器中的位置是任意的，即逻辑上相邻的数据元素，在物理上不一定相邻；
- 线性表的链式表示又称为非顺序映像或链式映像。
- 用一组物理位置任意的存储单元来存放线性表的数据元素。
- 这组存储单元既可以是连续的，也可以是不连续的，甚至是零散分布在内存中的任意位置上的。
- 链表中元素的逻辑次序和物理次序不一定相同。

链表词汇

- 结点: 数据元素的存储映像。由数据域和指针域两部分组成。
- 链表: n个结点由指针链组成一个链表。
它是线性表的链式存储映像，称为线性表的链式存储结构。
- 头指针: 是指向链表中第一个结点的指针
- 首元结点: 是指链表中存储第一个数据元素a₁的结点
- 头结点:是在链表的首元结点之前附设的一个结点



单链表

单链表是线性表的链式存储。**只能从头节点依次向后遍历。**

单链表是由表头唯一确定，因此单链表可以用头指针的名字来命名。若头指针名是L，则把链表称为表L。

在单链表中插入数据需将插入位置前一个数的下一个结点设置为插入数据，插入数据的下一个结点设为原数据。

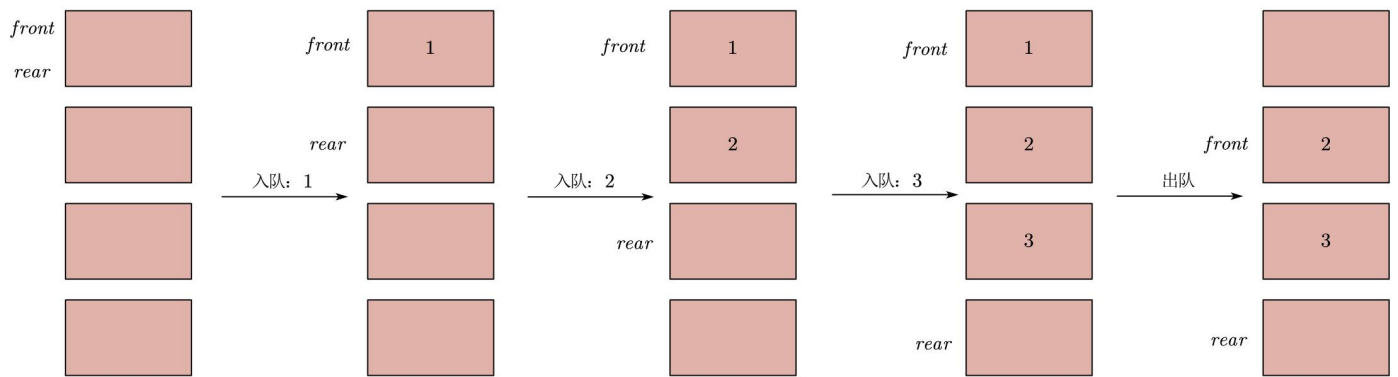
队列：操作受限的线性表

限制：仅允许在表的一端进行插入操作，而在表的另一端进行删除操作。

1. 把进行**插入**的一端成为队的**队尾 (rear)**，把进行**删除**的一端称为**队头或队首 (front)**。
2. 向队列中插入新元素称为**进队或入队 (enqueue)**，新元素进队后就成为新的队尾元素
3. 从队列中删除元素称为**出队或离队 (dequeue)**，元素出队后，其**直接后继元素**就成为队首元素。（不像顺序表需要将后面所有元素向前移一位）

基本操作

```
void InitQueue(Queue *Q);           //初始化
void EnQueue(Queue *Q, int x);      //入队
void DeQueue(Queue *Q);             //出队
```

初始化

```
//初始化
void InitQueue(Queue *Q)
{
    Q->base=(int *)malloc(sizeof(int)*Maxsize);//开辟存储空间
    if(Q->base==NULL)
    {
        //开辟失败
        return NULL;
    }
    //初始化时，队头和队尾都在0位置
    Q->front=0;
    Q->rear=0;
}
```

C

入队

```
//入队
void EnQueue(Queue *Q, int x)
{
    //判断是否还有存储空间
    if(Q->rear>=Maxsize)
    {
        return;
    }
    //假如还有存储空间，数据入队
    //队尾下标+1
    Q->base[Q->rear++]=x;
}
```

C

出队

```
//出队
void DeQueue(Queue *Q)
{
    //判断是否为空
    if(Q->front==Q->rear)
    {
        return;
    }
    //不为空，出队
    Q->front++;
}
```

栈：操作受限的线性表

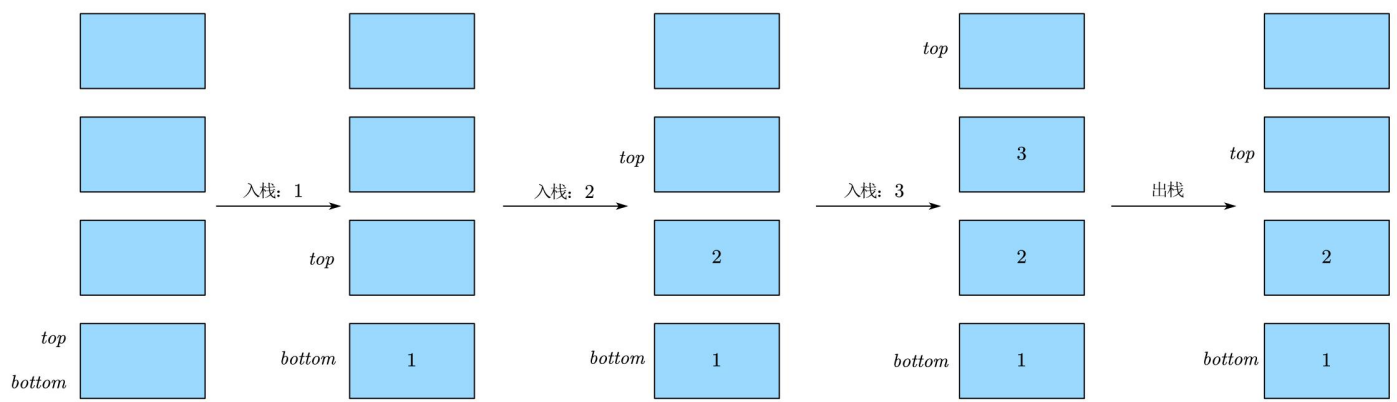
限制：仅允许在表的一端进行插入删除操作。

- 进行数据插入和删除操作的一端为**栈顶**，另一端为**栈底**。
- 栈中的数据元素遵守**后进先出**的原则。
- 压栈：栈的插入操作叫做进栈/压栈/入栈，即从栈顶插入数据。
- 出栈：栈的删除操作叫做出栈，即从栈顶删除数据。

栈的实现可以使用**数组**和**链表**，相对而言数组的结构实现更优，因为数组在尾上插入数据的代价较小，数组的特征也更符合栈的特性。

栈的基本操作

```
Stack * CreatStack(int length); //创建栈
int IsStackEm(Stack * stack); //判断是否为空栈
void Push(Stack * stack, int data); //入栈
int Pop(Stack * stack); //出栈
```



创建栈

C

```
//创建栈
Stack * CreatStack(int length)
{
    Stack * stack=(Stack *)malloc(sizeof(Stack)); //开辟空间
    if(stack)
    {
        //非空表示申请成功
        stack->array=(int *)malloc(sizeof(int)*length);
        //假如使用数组无需进行这一步
    }
    //开辟空间失败
    if(stack->array==NULL)
    {
        return NULL;
    }
    //初始化
    stack->bottom=0;
    stack->top=0;
    stack->max=length;
}
```

判断是否为空栈

C

```
//判定是否为空栈
int IsStackEm(Stack * stack)
{
    if(stack->bottom==stack->top)
    {
        //空栈返回0
        return 0;
    }
    //非空栈返回1
    return 1;
}
```

入栈

C

```
//入栈
void Push(Stack * stack, int data)
{
    if(stack->top==stack->max)
    {
        //判断栈是否已满
        return 0;
    }
    //数据入栈
    stack->array[stack->top]=data;
    //栈顶上移
    stack->top++;
}
```

出栈

C

```
//出栈
int Pop(Stack * stack)
{
    if(stack->top>stack->bottom)
    {
        //栈不为空
        //栈顶下移
        stack->top--;
        //返回被取出的栈顶数据
        return stack->array[stack->top];
    }else{
        printf("栈空, 出栈失败");
        return -1;
    }
}
```