计算机导论

第二章 计算机的基础知识

厦门大学计算机系

严严

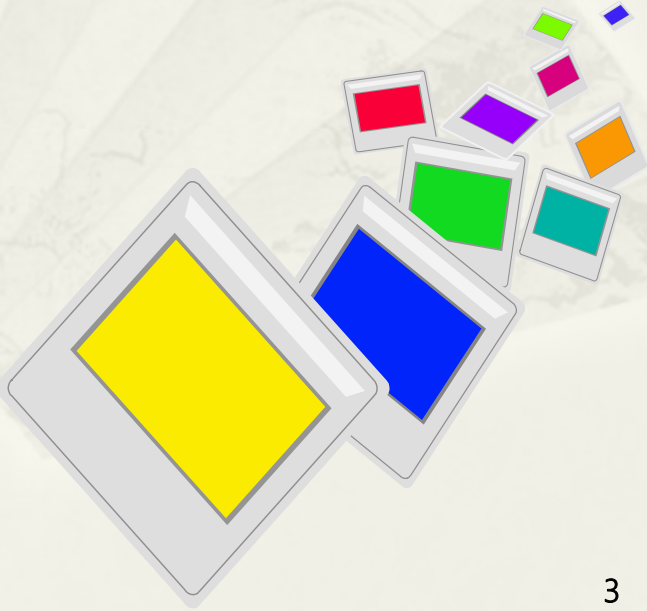
第2章 计算机的基础知识

内容提要

- 计算机的运算基础
- 命题逻辑与逻辑代数基础
- 计算机的基本结构与工作原理
- 程序设计基础
- 算法基础
- 数据结构基础

基本要求：

- 掌握数制间的转换方法以及数据在计算机内部的表示形式
- 理解逻辑代数、计算机的工作原理、程序设计以及算法与数据结构的基本知识，为学习本书的以下各章和后续课程打好基础



进制的概念

- 进制（数制）
  - 即进位计数值。就是用进位的方法进行计数。
  - 三要素：

数码:一组用来表示某种数制的符号。

基数：数制所使用的数码个数称为“基数”或“基”，常用“R”表示，称R进制。

位权:指数码在不同位置上的权值。在进位计数制中，处于不同数位的数码代表的数值不同。

- 不同进制数的表示方法
- 数制之间的转换

1. 常用的进位计数制

1）十进制（Decimal System）

由0、1、2、…、8、9十个数码组成，即基数为10。

2 ）二进制（Binary System）

由0、1两个数码组成，即基数为2。

3）八进制（Octal System）

由0 …、7八个数码组成，即基数为8。

4）十六进制（Hexadecimal System）

由0、…、9、A、 …、 F十六个数码组成，即基数为16。

十进制、二进制、八进制、十六进制之间的对应关系

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0	0	0	9	1001	11	9
1	1	1	1	10	1010	12	A
2	10	2	2	11	1011	13	B
3	11	3	3	12	1100	14	C
4	100	4	4	13	1101	15	D
5	101	5	5	14	1110	16	E
6	110	6	6	15	1111	17	F
7	111	7	7	16	10000	20	10
8	1000	10	8	17	10001	21	11

二进制

- 二进制：使用数字0和1等符号来表示数值且采用“逢二进一”的进位计数制。

- 二进制数制的特点：
  - 仅使用0和1两个数字。
  - 最大的数字为1，最小的数字为0。
  - 每个数字都要乘以基数2的幂次，该幂次由每个数字所在的位置决定。

- 二进制加法和乘法运算规则：

0＋0＝0	0 × 0＝0
0＋1＝1	0 × 1＝0
1＋0＝1	1 × 0＝0
1＋1＝1	1 × 1＝1

- 例题，书31页例【2-1】。

二进制

❖ **例题：**

【例 2-1】 计算二进制数 1011×101 的值。

解：

```
    1 0 1 1
  ×   1 0 1
  ---------
    1 0 1 1
  0 0 0 0
+ 1 0 1 1
  ---------
  1 1 0 1 1 1
```

即 1011×101＝110111，相当于十进制 11×5＝55。

八进制和十六进制

- 八进制：使用数字0、1、2、3、4、5、6、7等符号来表示数值的，且采用“逢八进一”的进位计数制。

例如，八进制数$(7654.345)_8$可表示为：

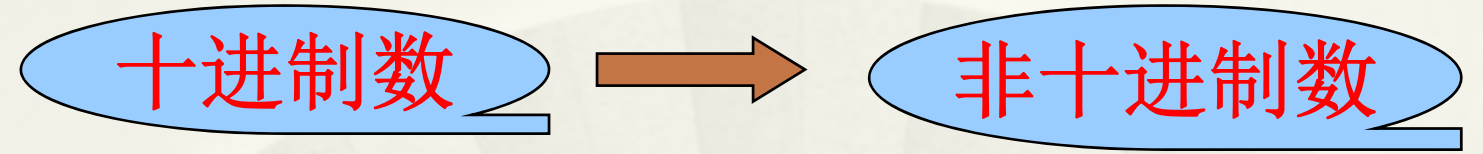
$$(7654.345)_8 = 7\times8^3+6\times8^2+5\times8^1+4\times8^0+3\times8^{-1}+4\times8^{-2}+5\times8^{-3}$$

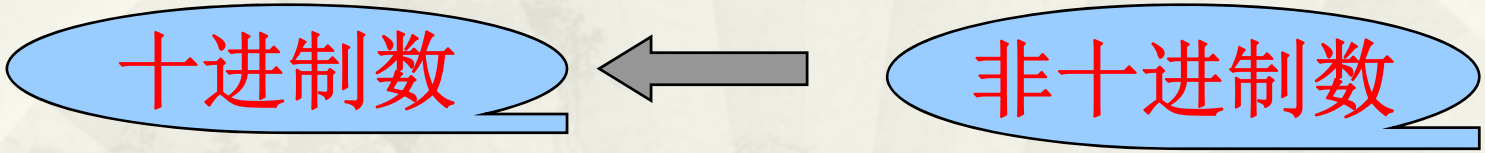
- 十六进制：使用数字0、1、2、3、4、5、6、7、8、9和A、B、C、D、E、F等符号来表示数值，其中A、B、C、D、E、F分别表示数字10、11、12、13、14、15。十六进制的计数方法为“逢十六进一”。

表 2-1　常用数制的基数和数字符号

数制	十进制	二进制	八进制	十六进制
基数	10	2	8	16
数字符号	0～9	0,1	0～7	0～9,A,B,C,D,E,F

进制之间的转换





二、八、十六进制之间的转换

非十进制数 ⟶ 十进制数

位权法： 把各非十进制数按权展开求和

转换公式： $(F)_{10} = a_1 \times x^{n-1} + a_2 \times x^{n-2} + \ldots +$

$a_{m-1} \times x^{1} + a_m \times x^{0} + a_{m+1} \times x^{-1} + \ldots$

示例：

$(1011.1)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}$

$= 8 + 0 + 2 + 1 + 0.5$

$= (11.5)_{10}$

十进制整数 ⟶ 非十进制整数

- 整数部分和小数部分采用不同的方法
- 整数部分采用除基数逆向取余法
- 小数部分采用乘基数正向取整法

十进制整数转换为非十进制整数

❖ **除基取余法：“除基取余，先余为低（位），后余为高（位）”。**

【例 2-2】 将十进制整数 55 转换为二进制整数。

解：

		余数
2	55	1
2	27	1
2	13	1
2	6	0
2	3	1
2	1	1
	0	

则得：$(55)_{10}=(110111)_2$

【例 2-3】 将十进制整数 55 转换为八进制整数。

解：

		余数
8	55	7
8	6	6
	0	

则得：$(55)_{10}=(67)_8$

十进制整数转换为非十进制小数

- **乘基取整法:“乘基取整，先整为高(位),后整为低(位)”。**

【例 2-5】 将十进制小数 0.625 转换为二进制小数。

解:

		整数
	0.625	
×	2	
	1.25	1
	0.25	
×	2	
	0.5	0
×	2	
	1.0	1

则得:$(0.625)_{10}=(0.101)_2$

❖ 乘基取整法:“乘基取整，先整为高(位),后整为低(位)”。

【例 2-6】 将十进制小数 0.32 转换为二进制小数。

解：

		整数
	0.32	
×	2	
	0.64	0
×	2	
	1.28	1
	0.28	
×	2	
	0.56	0
×	2	
	1.12	1
	⋮	

则得：$(0.32)_{10}=(0.0101\cdots)_2$

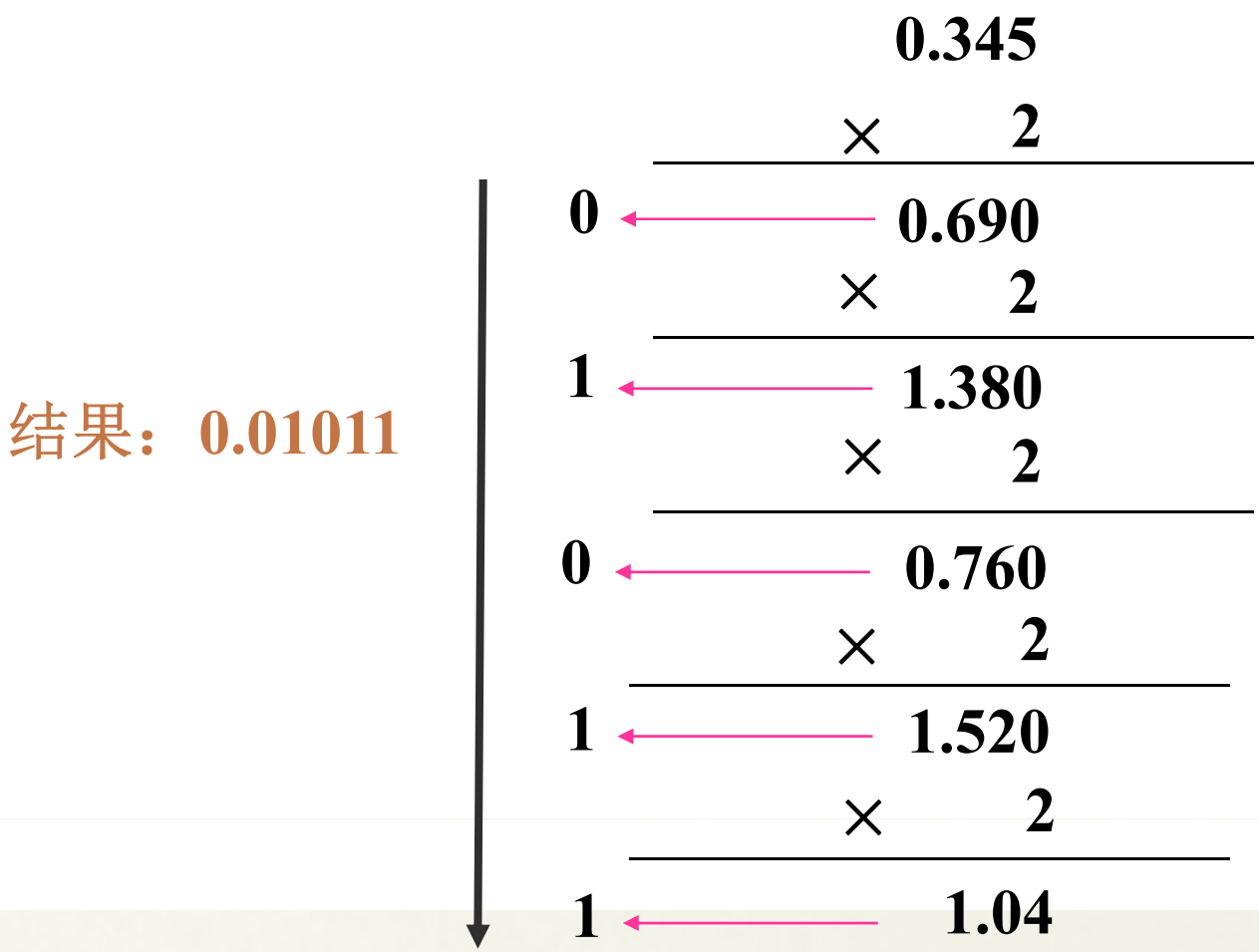
十进制数转换为非十进制数

- 十进制小数并不是都能够用有限位的其他进制数精确地表示，这时应根据精度要求转换到一定的位数为止，作为其近似值。

- 如果一个十进制数既有整数部分，又有小数部分，则应将整数部分和小数部分分别进行转换。

例：100.345 D=(**1100100. 01011**)B

小数部分：乘基数正向取整法



结果：**0.01011**

说明

- 十进制小数并不是都能够用有限位的其他进制数精确地表示
  - 应根据精度要求转换到一定的位数为止
  - 可以采用0舍1入的方法进行处理（类似于十进制中的四舍五入的方法）作为其近似值。
- 如果一个十进制数既有整数部分，又有小数部分，则应将整数部分和小数部分分别进行转换

非十进制之间的转换

整数从右向左三位并一位

小数从左向右三位并一位

二进制 ⇄ 八进制

一位拆三位

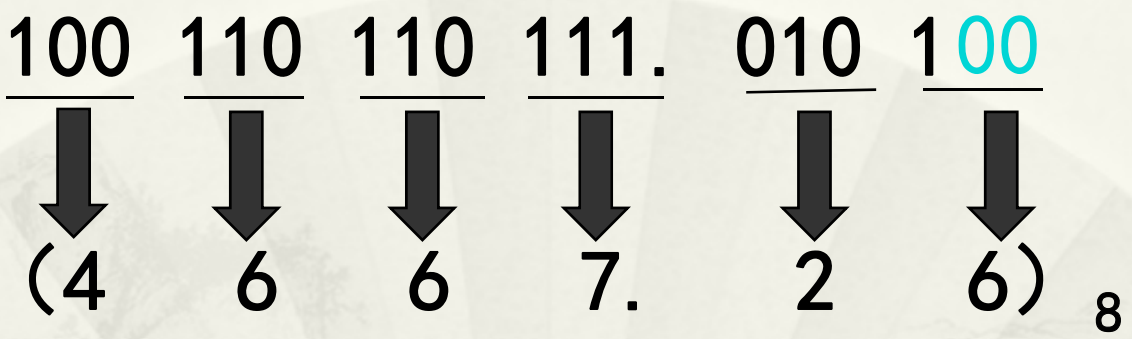
整数从右向左四位并一位

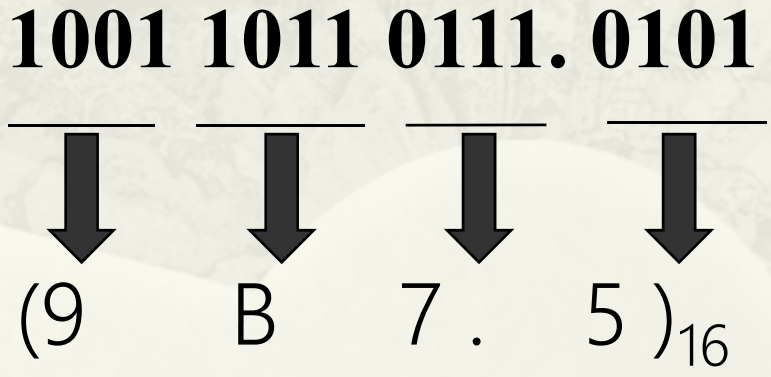
小数从左向右四位并一位

二进制 ⇄ 十六进制

一位拆四位

例：100110110111.0101B=(　)O=(　)H





思考

- 计算机内部为何采用二进制？
  - 易于用物理元件表示:计算机是由**逻辑电路**组成，而逻辑电路通常只有**两个状态**
  - **运算规则简单**
  - **可靠性高**
    - 两个状态表示的二进制两个数码，数字传输和处理不容易出错
  - **逻辑性强**
    - 计算机工作原理是建立在**逻辑运算**基础上的，**逻辑代数**是逻辑运算的理论依据

计算机中数据的单位

- 位（b）:位是计算机存储信息的**最小单位**
- 字节(B):字节是信息处理的基本单位，一个字节由八位二进制数组成，即1Byte=8bit。
  - 1KB=1024B=2^{10}B
  - 另外还有MB、GB、TB
- **字长**:字是CPU通过**数据总线**一次**存取、加工和传送数据**的长度。
  - 一个字通常由一个或若干个字节组成，字长越长，性能越强。
  - 常用的字长有8位、16位、32位、64位。

计算机中的信息表示

数值信息在计算机中的表示

非数值型数据在计算机中的表示

整数在计算机中的表示

- 在计算机中，按照既定的二进制位数（称为码长），
  - 最左边的那一位（称为符号位）用来表示一个整数的正负
    - 0 表示正数，1 表示负数
  - 符号位之后的那些位（称为数值位），用来表示这个整数的绝对值
- 在计算机中，数可以有三种不同的二进制表示方法（差别在于负数之数值位的表示不同）：
  - 原码表示
  - 反码表示
  - 补码表示

原码表示

- 在给定码长后，根据一个整数的正负填写符号位，再将这个整数之绝对值的二进制表示，按照数值位的长度在前面补足必要的 0 后，就得到原码表示。

若码长为 8，则 $123_{(10)}$ 的原码表示是：

01111011

$-123_{(10)}$ 的原码表示是：

11111011

若码长为 16，则 $123_{(10)}$ 的原码表示是：

0000000001111011

$-123_{(10)}$ 的原码表示是：

1000000001111011

n位二进制**原码**的表数范围： $-(2^{n-1}-1) \le N \le (2^{n-1}-1)$

	真值	8位原码		真值	16位原码
$2^{n-1}-1$	+127	0111 1111	7FH	+32767	7FFFH
	+126	0111 1110	7EH	+32766	7FFEH
	…	…	…	…	…
	+2	0000 0010	02H	+2	0002H
	+1	0000 0001	01H	+1	0001H
	0	0000 0000	00H	0	0000H
→	-0	1000 0000	80H	-0	8000H
	-1	1000 0001	81H	-1	FFFFH
	-2	1000 0010	82H	-2	FFFEH
	…	…	…	…	…
	-126	1111 1110	FEH	-32766	8002H
$-(2^{n-1}-1)$	-127	1111 1111	FFH	-32767	8001H
	-128	----------	----	-32768	-------

原码中 0 有两种表达方式 (+0、−0)

原码表示方法简单直观，但机器中原码不便于运算！

反码表示

- 规定：
- 一个正整数的反码表示与其原码表示相同；
- 一个负整数的反码表示：对其原码表示的数值位进行按位变反（按位将 1 换成 0、将 0 换成 1）的结果。
- 例如（若码长为 8）：

$(26)_{(反)} = (26)_{(原)} = 0\ 0011010$

$(-26)_{(反)} = 11100101$ *(10011010 → 11100101)*

- 0 也有两种反码表示：

00000000

11111111

补码表示

- 我们先来看一个例子(汽车上的里程表)

补码表示

- 我们先来看一个例子(汽车上的里程表)

补码表示

- 在这个例子中，当里程表上的数字是 999999.9 时，再行进 0.1 公里，里程表显示的是 000000.0。
- 如果我们只看整数部分：由于 999999 + 1 = 000000，（从仪表盘上看到的结果），所以从算术运算的角度看， 这里999999 的作用相当于 -1。
- 说明，当限制了数据的表示长度时，要得到一个与正整数k对应的负数表示，可以认为：要得到的那个数加上这个正整数之后等于 0。我们称之为求补。
  - 在上面的例子中，要得到 1 的负数表示 -1，就是看哪个数加上 1 后等于 0。
  - 这个数便是 999999。 Why?

补码表示

- 给定码长的二进制表示上来：

  例如，当码长为 8（即数值位数为 7），则

  $26_{(10)} = 0011010$

  那么，要得到 $-26_{(10)}$，就是求一个二进制数 c：使得：

  $c + 0011010 = 0000000$

  这样的 c 就是 $|-26_{(10)}|$ 的二进制表示：

  1100110

  因为：

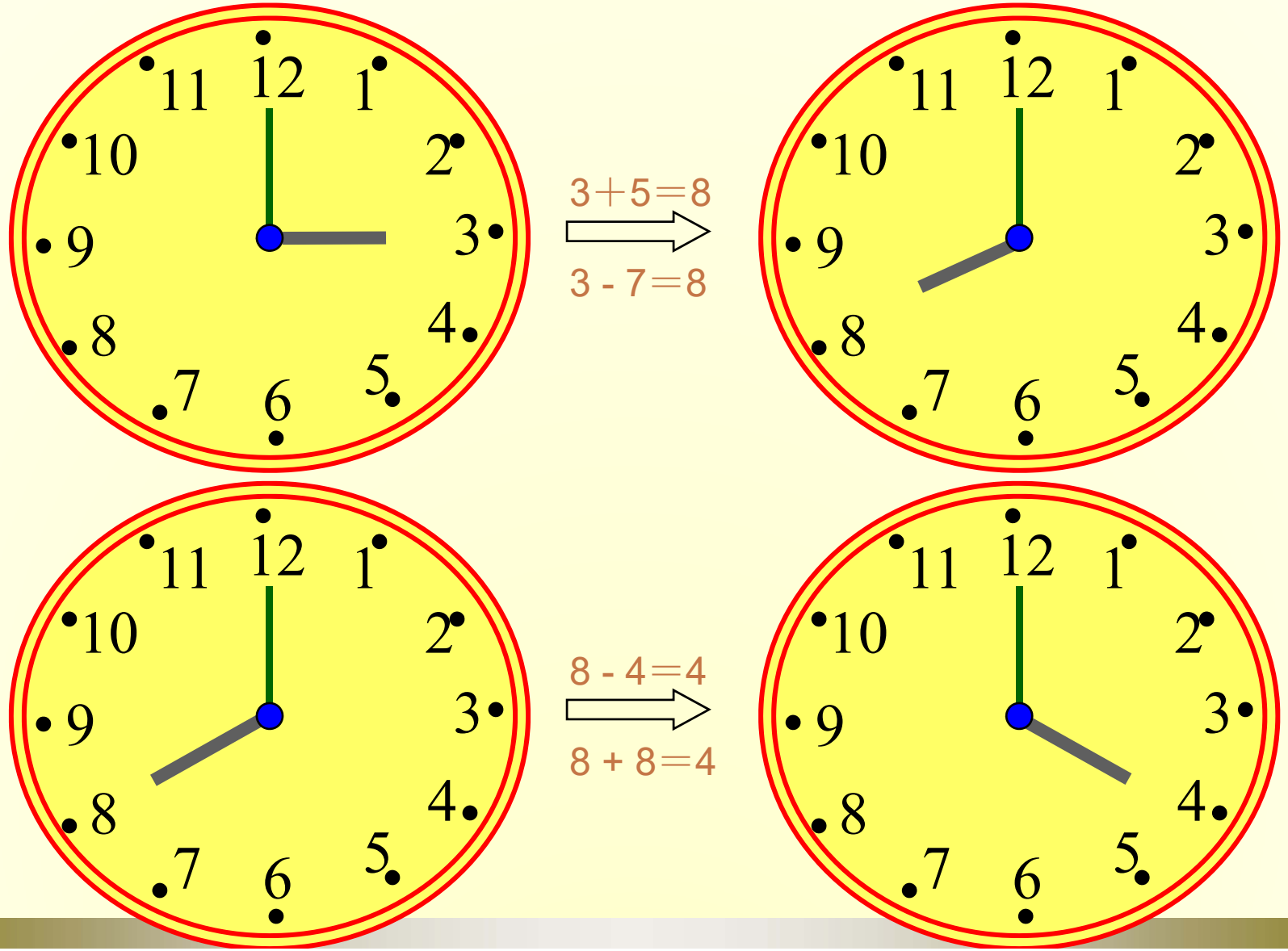
```
   1100110
+) 0011010
-------------
   0000000
```

因码长有限，
进位被丢弃

补码表示

- 规定：
  - 一个正整数的补码表示与它的原码表示相同；
  - 一个负整数的补码表示
    - 符号位为 1，数值位是其绝对值的求补结果
- 对于一个负整数，怎样求它的补码表示？
  - 对其原码表示的数值位按位变反后加 1。
  - 例：当码长为 8，求 $-26_{(10)}$ 的补码表示（11100110）：
    - 原码表示是：10011010
    - 按位变反后：11100101
    - 加 1 后得到：11100110，即得到其补码表示。

另一个例子



n位二进制补码的表数范围：$(-2^{n-1}) \le N \le (2^{n-1}-1)$

	真值	8位补码		真值	16位补码
$2^{n-1}-1$	+127	0111 1111	7FH	+32767	7FFFH
	+126	0111 1110	7EH	+32766	7FFEH
	…	…	…	…	…
	+2	0000 0010	02H	+2	0002H
	+1	0000 0001	01H	+1	0001H
	0	0000 0000	00H	0	0000H
	-0			-0	
	-1	1111 1111	FFH	-1	FFFFH
	-2	1111 1110	FEH	-2	FFFEH
	…	…	…	…	…
	-126	1000 0010	82H	-32766	8002H
	-127	1000 0001	81H	-32767	8001H
-2^{n-1}	-128	1000 0000	80H	-32768	8000H

(-2^{n-1}) 的补码 100……000 为按等效原则定义的！

（无法从原码转换而来）

计算机中为什么使用补码表示数

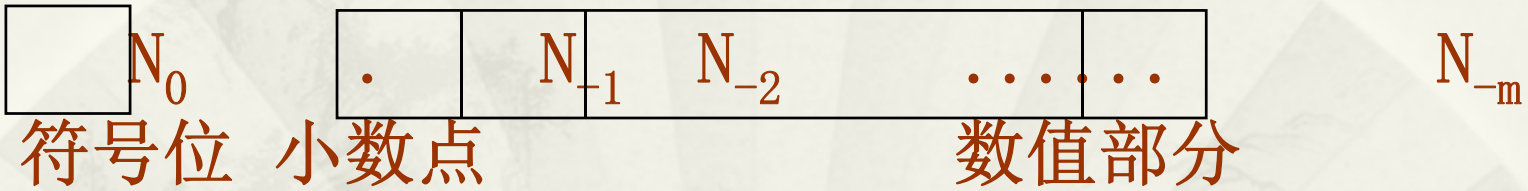
- 使符号位能与有效值部分一起参加运算,从而简化运算规则
- 使减法运算转换为加法运算,进一步简化计算机中运算器的线路设计

定点数和浮点数

* 定点表示法

在机器中，小数点位置固定的数称为**定点数**

1、**定点小数表示法**，即小数点固定在符号位与最高位之间



2、**定点整数表示法**，此时将小数点固定在数的最低位的后面

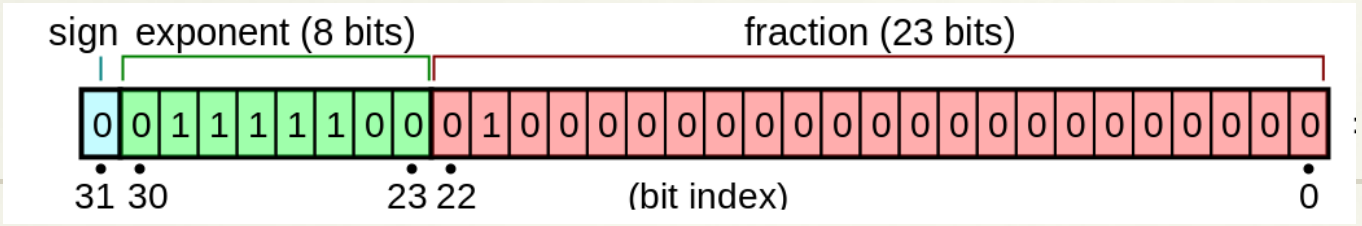
[注意]定点数的运算规则比较简单，但不适宜对数值范围变化比较大的数据进行运算

浮点表示法

- 浮点数可以扩大数的表示范围
- 浮点数由两部分组成
  - 一部分用以表示数据的有效位，称为尾数；
  - 一部分用于表示该数的小数点位置，称为阶码。
  - 一般阶码用整数表示，尾数大多用小数表示。
    - 一个数N用浮点数表示可以写成：

$$N = M \cdot R^e$$

- M表示尾数，e表示指数，R表示基数。
- 基数一般取2，8，16。一旦机器定义好了基数值，就不能再改变了
- 在浮点数表示中基数不出现，是隐含的。



$$(-1)^{b_{31}} \times (1.b_{22}b_{21}\ldots b_0)_2 \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127}$$

$$\texttt{7f7f ffff} = (1 - 2^{-24}) \times 2^{128} \approx 3.402823466 \times 10^{38}$$

$$\texttt{0080 0000} = 2^{-126} \approx 1.175494351 \times 10^{-38}$$

```
0000 0000  = 0
8000 0000  = -0
```

```
7f80 0000  = infinity
ff80 0000  = -infinity
```

作业

1. 将下面的十进制数转换为二进制数：
   6，286，1024，0.25，7.125，2.625
2. 将下面的二进制数转换为十进制数：
   110111，10011101，0.101，10.01，1010.001
3. 将下面的二进制转换为八进制和十六进制：
   10011011.0011011，1010101010.0011001
4. 将下面的八进制或十六进制转换为二进制：
   $(75.612)_8$，$(64A.C3F)_{16}$
5. 什么是原码，补码和反码，并写出下列各数的反码，补码和原码（用八位表示）：
   127，-127，135，-120，0.75，-0.75
6. 一台浮点计算机，数码为8位，阶码为3位，则他能表示的数的范围是多少？
7. 在计算机系统中，位，字节，字，和字长所表示的含义各是什么？

信息的编码

- 数值的编码
- 文字的编码
  - 字符编码
  - 汉字编码
    - 汉字交换码
    - 汉字机内码
    - 汉字字形码
    - 汉字输入码

信息的编码

- 基本符号包括字母、运算符、标点符、控制符、大量的汉字等。
- 而计算机只能识别0和1两个数字符号。因此必须对信息编码，用二进制表示各种符号。
- 为了帮助检查和纠错，可在编码子中曾加一些校验位，或使用检错码和纠错码。

BCD码

- BCD码：是一种二-十进制的编码，使用四位二进制数表示一位十进制数。

- 【例 2-20】 将十进制数 5678 转换为 BCD 码。

  解：

十进制数：	5	6	7	8
	↓	↓	↓	↓
BCD 码：	0101	0110	0111	1000

- 【例 2-21】 将 BCD 码 1001 0110 1000 0101 转换为十进制数。

  解：

BCD 码：	1001	0110	1000	0101
	↓	↓	↓	↓
十进制数：	9	6	8	5

  即 BCD 码 1001 0110 1000 0101 的十进制数为 9685。

字符编码

1）字符编码：

- 目前采用的字符编码主要是ASCII码
  - 美国标准信息交换代码-American Standard Code for Information Interchange的缩写
  - 已被国际标准化组织ISO采纳，作为国际通用的信息交换标准代码
- ASCII码是一种西文机内码，有7位ASCII码和8位ASCII码
  - 7位ASCII码称为标准ASCII码，8位ASCII码称为扩展ASCII码
  - 7位标准ASCII码用一个字节（8位）表示一个字符，并规定其最高位为0，实际只用到7位，因此可表示128个不同字符
  - 同一个字母的ASCII码值小写字母比大写字母大32

L \ H	0000	0001	0010	0011	0100	0101	0110	0111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	,	7	G	W	g	w
1000	BS	CAN	)	8	H	X	h	x
1001	HT	EM	(	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	’	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

字符编码

2）汉字编码

* （1）汉字交换码：由于汉字数量极多，一般用连续的两个字节（16个二进制位）来表示一个汉字。

* 1980年，我国颁布了第一个汉字编码字符集标准，即GB2312-80《信息交换用汉字编码字符集基本集》，该标准编码简称国标码

  * 是我国大陆地区及新加坡等海外华语区通用汉字交换码

  * GB2312-80收录了6763个汉字，以及682符号，共7445个字符，奠定了中文信息处理的基础

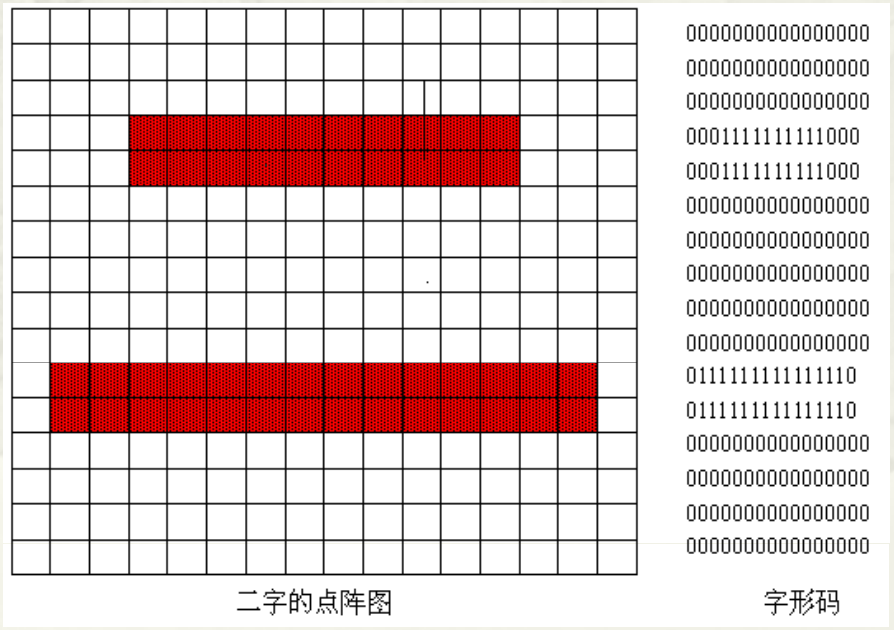
字符编码

* （2）汉字机内码
* 但是国标码GB2312不能直接在计算机中使用
  * 它没有考虑与基本的信息交换代码ASCII码的冲突。
  * 比如：“大”的国标码是3473H，与字符组合“4s”的ASCII相同，“嘉,”的汉字编码为3C4EH,与码值为3CH和4EH的两个ASCII字符“<”和“N”混淆。

- 为了能区分汉字与ASCII码，在计算机内部表示汉字时把交换码（国标码）两个字节最高位改为1，称为“机内码”。
  - 当某字节的最高位是1时，必须和下一个最高位同样为1的字节合起来，代表一个汉字。

* （3）汉字字形码
  * 用来将汉字显示到屏幕上或打印到纸上所需要的图形数据。
  * 汉字字形码记录汉字的外形，是汉字的输出形式。
* 记录汉字字形通常有两种方法：点阵法和矢量法
  * 分别对应两种字形编码：点阵码和矢量码
  * 所有的不同字体、字号的汉字字形构成汉字库

- 点阵码是一种用点阵表示汉字字形的编码
- 一个16×16点阵的汉字要占用32个字节
- 一个32×32点阵的汉字则要占用128字节



- （4）汉字输入码
- 将汉字通过键盘输入到计算机采用的代码
- 也称为汉字外部码（外码）
- 汉字输入码的编码原则应该易于接受、学习、记忆和掌握
- 根据编码规则，这些汉字输入码可分为流水码、音码、形码和音形结合码四种

数据校验码

- 奇偶校验码：在表示数据的N位代码中增加一位奇偶校验位，使N＋1位中“1”的个数为奇数（奇校验）或偶数（偶校验）。它只能检测一位错误，且不能指出哪一位错。
- 海明校验码：在有效信息代码中增加校验位，用来校验代码中“1”的个数是奇数（奇校验）还是偶数（偶校验），通过奇偶校验可以发现代码传输过程中的错误并自动校正。
- 应用：用于计算机各部件之间信息传输以及计算机网络的信息传输。

逻辑代数基础

- 命题：有具体意义且能够判断真假的陈述句。
- 命题的真值：
  - 命题所具有的值“真”（true，简记为T）或“假”（false，该简记为F）称为其真值。
- 命题标识符：表示命题的符号，标识符称为命题常量。
- 原子命题：不能分解为更为简单的陈述句的命题；
- 复合命题：将原子命题用连接词和标点符号复合而成的命题。

连接词“与”（ ∧）

“与”（ ∧）:两个命题A和B的“与”(又称为A和B的“合取”)是一个复合命题，记为A∧B。

当且仅当A和B同时为真时A∧B为真，在其他的情况下A∧B的真值均为假。

A∧B的真值表:

A	B	A∧B
T	T	T
T	F	F
F	T	F
F	F	F

连接词 “或”（∨）

- “或”（∨）：两个命题A和B的“或”（又称为A和B的“析取”）是一个复合命题，记为A∨B
- 当且仅当A和B同时为假时A∨B为假，在其他的情况下A∨B的真值均为真
- A∨B的真值表：

A	B	A∨B
T	T	T
T	F	T
F	T	T
F	F	F

连接词“非”（¬）

- “非”（¬）：命题A的“非”（又称为A的“否定”）是一个复合命题，记为 ¬A。若A为真，则¬A为假；若A为假，则¬A为真。
- ¬A的真值表：

*

A	¬A
T	F
F	T

连接词 “异或”（⊕）

- “异或” （⊕）：两个命题的A和B的“异或”（又称为A和B的“不可兼或”）是一个复合命题，记为A⊕B
- 当且仅当A和B同时为真或者同时为假时A⊕B为假，在其他的情况下A⊕B的真值为真
- A⊕B的真值表：

A	B	A⊕B
T	T	F
T	F	T
F	T	T
F	F	F

连接词“条件”（ →）

❖“条件”（ →）：两个命题的A和B的“条件”是一个复合命题，记为A→B, 读作“如果A，则B”。

❖ 当且仅当A的真值为真，B的真值为假时，A→B为假，在其他的情况下A→B的真值均为真。

❖ A→B的真值表：

A	B	A →B
T	T	T
T	F	F
F	T	T
F	F	T

连接词 “双条件”（ ↔ ）

- “双条件”（↔）:两个命题的A和B的“双条件”（又称为A当且仅当B）是一个复合命题，记为A ↔ B，读作“A当且仅当B”。 当且仅当A的真值与B的真值相同时， A↔ B为真，否则A ↔B的真值均为假。
- A ↔ B的真值表：

A	B	A↔B
T	T	T
T	F	F
F	T	F
F	F	T

命题公式

- **命题公式：** 由命题变元、连接词和括号组成的合式的式子称为命题公式。
- 命题公式等价：如果两个不同的命题公式P和Q，无论其命题变元取什么值它们的真值都相同，则称该两个命题公式等价，记为P＝Q。

〖例2-25〗证明 ¬（A→B）与A∧¬ B是等价的。

A	B	¬（A→B）	A∧¬ B
T	T	F	F
T	F	T	T
F	T	F	F
F	F	F	F

命题公式的等价律

* 其中A、B、C等为命题变元，T表示“真”，F表示“假”

❖零律：　$A \vee F = A$

　　$A \wedge F = F$

❖幺律：　$A \vee T = T$

　　$A \wedge T = A$

❖幂等律：$A \vee A = A$

　　$A \wedge A = A$

❖求补律：$A \vee \neg A = T$

　　$A \wedge \neg A = F$

❖交换律：$A \vee B = B \vee A$

　　$A \wedge B = B \wedge A$

命题公式的等价律（续）

- **结合律：** $A\vee(B\vee C)=(A\vee B)\vee C$
  $A\wedge(B\wedge C)=(A\wedge B)\wedge C$
- **分配律：** $A\wedge(B\vee C)=A\wedge B\vee A\wedge C$
  $A\vee B\wedge C=(A\vee B)\wedge(A\vee C)$
- **吸收律：** $A\wedge B\vee A\wedge\neg B=A$
  $(A\vee B)\wedge(A\vee\neg B)=A$
- **狄－摩根定律：** $\neg(A\vee B)=\neg A\wedge\neg B$
  $\neg(A\wedge B)=\neg A\vee\neg B$
- **双重否定律：** $\neg\neg A=A$

证明狄一摩根定律

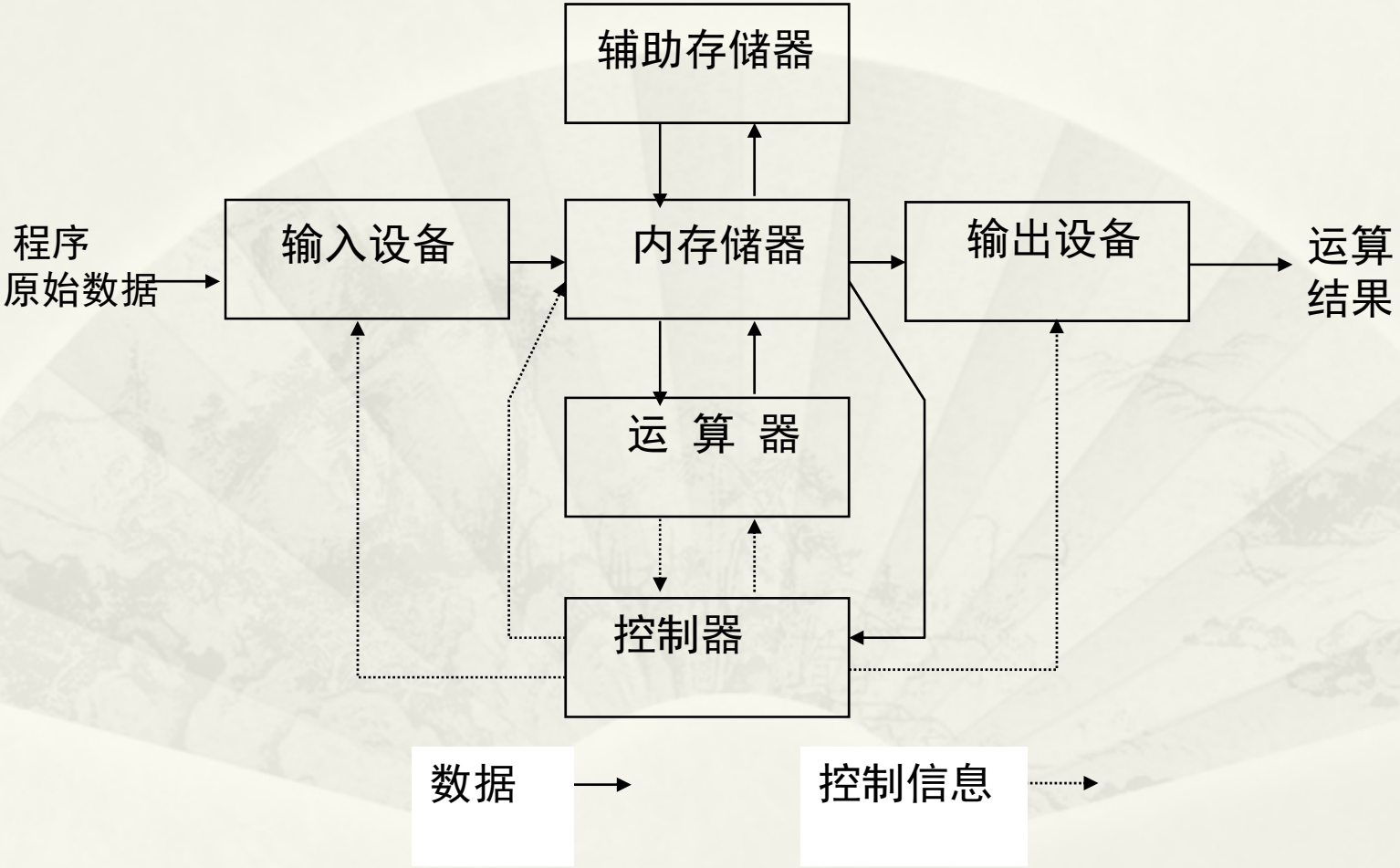
* 〖例2-26〗证明狄一摩根定律之一：¬（A∧B）＝¬A∨¬B。

A	B	A∧B	¬（A∧B）	¬A	¬B	¬A∨¬B
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

逻辑函数的化简

* 〖例2-27〗试将逻辑函数F＝A＋$\bar{A}$ B化简。
* 解：F＝A＋$\bar{A}$ B
* ＝（A＋$\bar{A}$ ）(A＋B)　　（分配律）
* ＝1 (A＋B)　　（求补律）
* ＝A＋B　　（幺律）
* 〖例2-28〗试将逻辑函数F＝AB＋A $\bar{B}$ ＋$\bar{A}$ B＋ $\overline{(A\ B)}$ 化简。
* 解：F ＝ AB＋A $\bar{B}$ ＋$\bar{A}$ B＋ $\overline{(A\ B)}$
* ＝ A（B＋ $\bar{B}$ ）＋$\bar{A}$ （B＋ $\bar{B}$ ）　　（分配律）
* ＝ A＋$\bar{A}$　　（求补律）
* ＝ 1　　（求补律）

计算机硬件的基本结构



程序设计语言

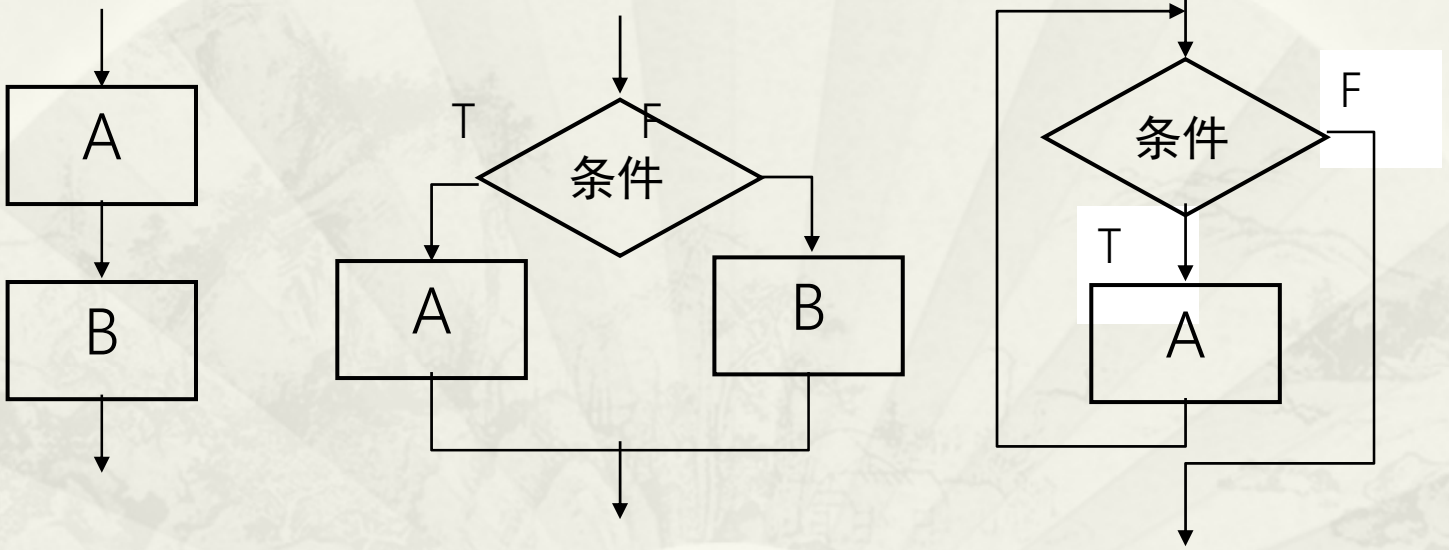
- 机器语言
  - 由计算机的指令系统组成，使用机器语言编写的程序计算机能够直接理解并执行
  - 但编程和理解都十分的困难
- 汇编语言
  - 使用“助忆符”来表示指令的操作码
  - 并使用存储单元或寄存器的名字表示地址码，以便于记忆和书写
- 高级程序设计语言
  - 是一种与机器的指令系统无关、表达形式更接近于被描述的问题的程序设计语言，便于程序的编写。
  - 使用高级程序设计语言编写的程序称为源程序，它必须经过程序设计语言翻译系统的处理后才能执行。
  - 面向过程程序设计语言
  - 面向对象程序设计语言

程序设计

- 一个使用程序设计语言产生一系列的指令以告诉计算机该做什么的过程。
- 广义的程序设计：
  - 需求分析
  - 总体设计
  - 详细设计
  - 编码
  - 测试
  - 运行与维护

结构化程序设计

*结构化程序设计：采用自顶向下逐步求精的设计方法和单入口单出口的控制成分（顺序、分支和循环）。



（a）顺序结构　　（b）选择型分支结构　　（c）循环结构

算法

- 由一系列规则组成的过程，这些规则确定了一个操作的顺序，以便能在有限步骤内得到特定问题的解
- 算法的性质
  - 确定性
  - 通用性
  - 有限性
- 算法的描述工具
  - 自然语言
  - 流程图
  - 决策表
  - 算法描述语言

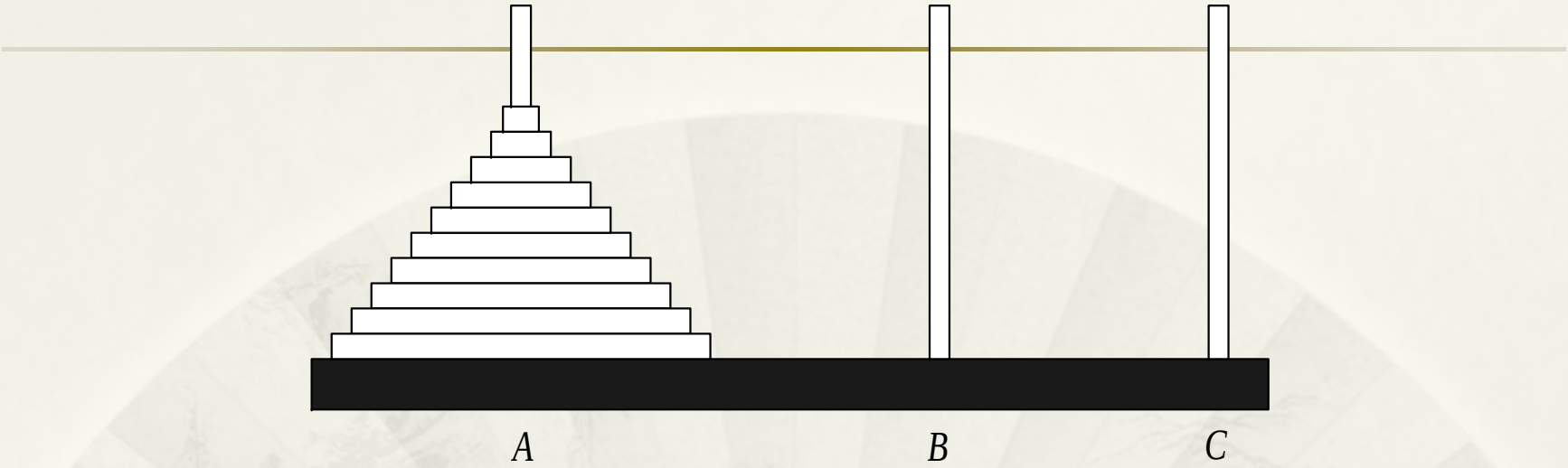
Hanoi问题

下面我们来用这种方法解答一个关于世界末日的传说里的问题。

这个传说出自古代的印度，历史学家鲍尔是这样描述的：

- 在世界中心贝拿勒斯（位于印度北部的一个佛教圣地）的圣庙里，安放着一块黄铜板，黄铜板上插着三根宝石针。
- 印度教的主神梵天在创造世界的时候，在其中的一根针上从下到上穿好了由大到小的64片金片，这就是所谓的梵塔，又称汉诺塔。
- 不论白天黑夜，都有一个僧侣按照梵天不渝的法则 在三根针上移动这些金片：
  - 一次只能移动一片，并且要求不管在哪一根针上，小片永远在大片上面。
- 当所有的64片金片都从梵天穿好的那根针上移到另外一根针上时，世界就将在一声霹雳中灰飞烟灭，而梵塔、庙宇和众生也都将同归于尽。

Hanoi问题

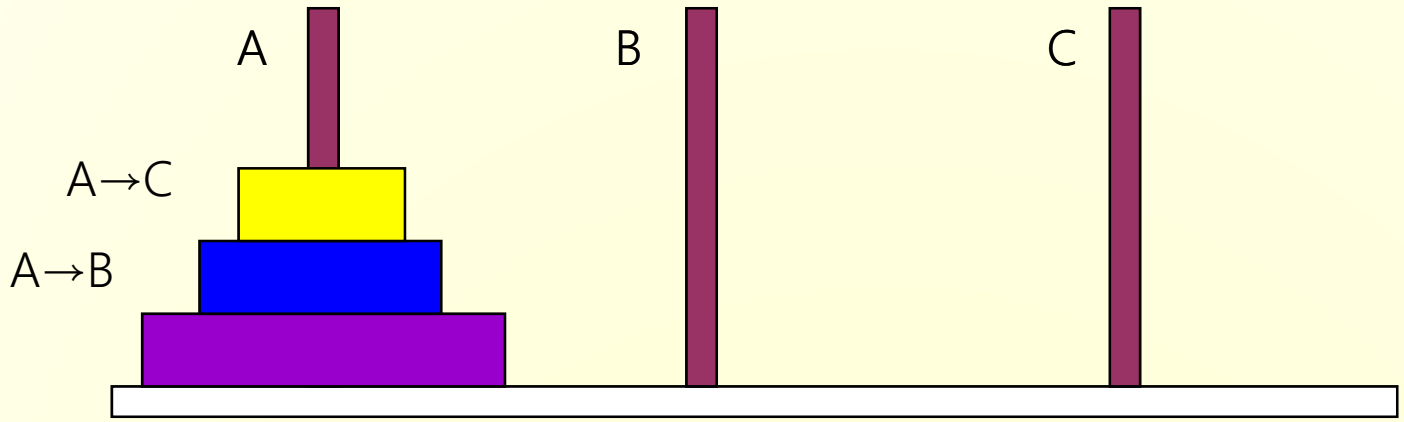


一个黄铜板上，插着三根宝石针，其中一根针上从下到上放了由大到小的64块金片，这就是Hanoi塔。

Hanoi塔问题：就是如何将64块金片按照梵天不渝法则，由一根宝石针全部移动到另一根宝石针上去。

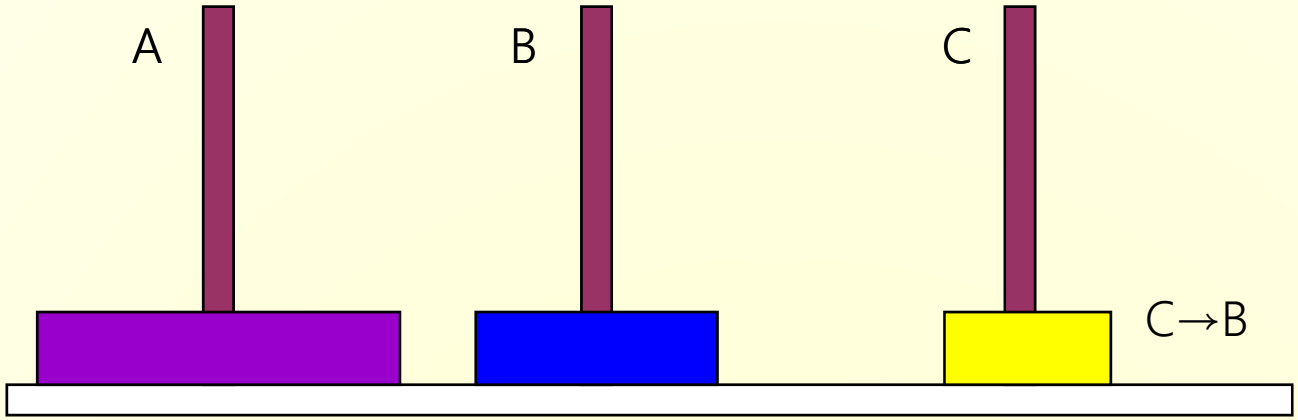
Hanoi问题

- 3个圆盘的移动过程演示



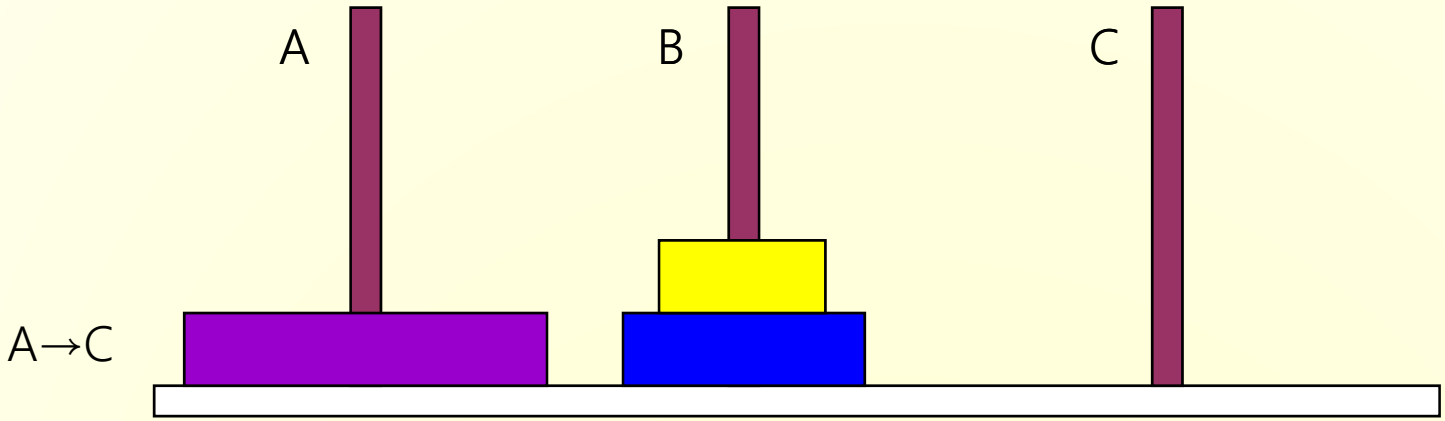
Hanoi问题

- 3个圆盘的移动过程演示



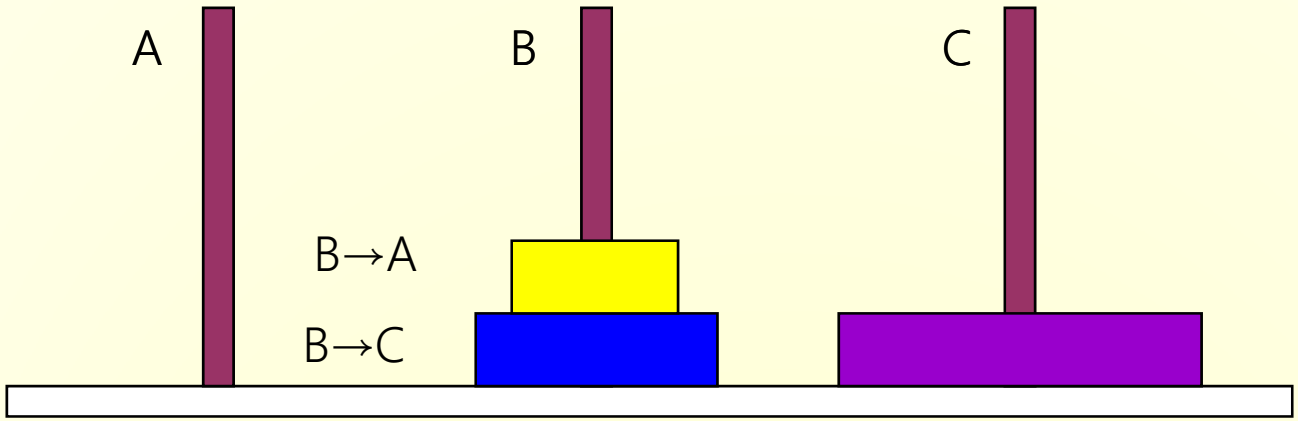
Hanoi问题

- 3个圆盘的移动过程演示



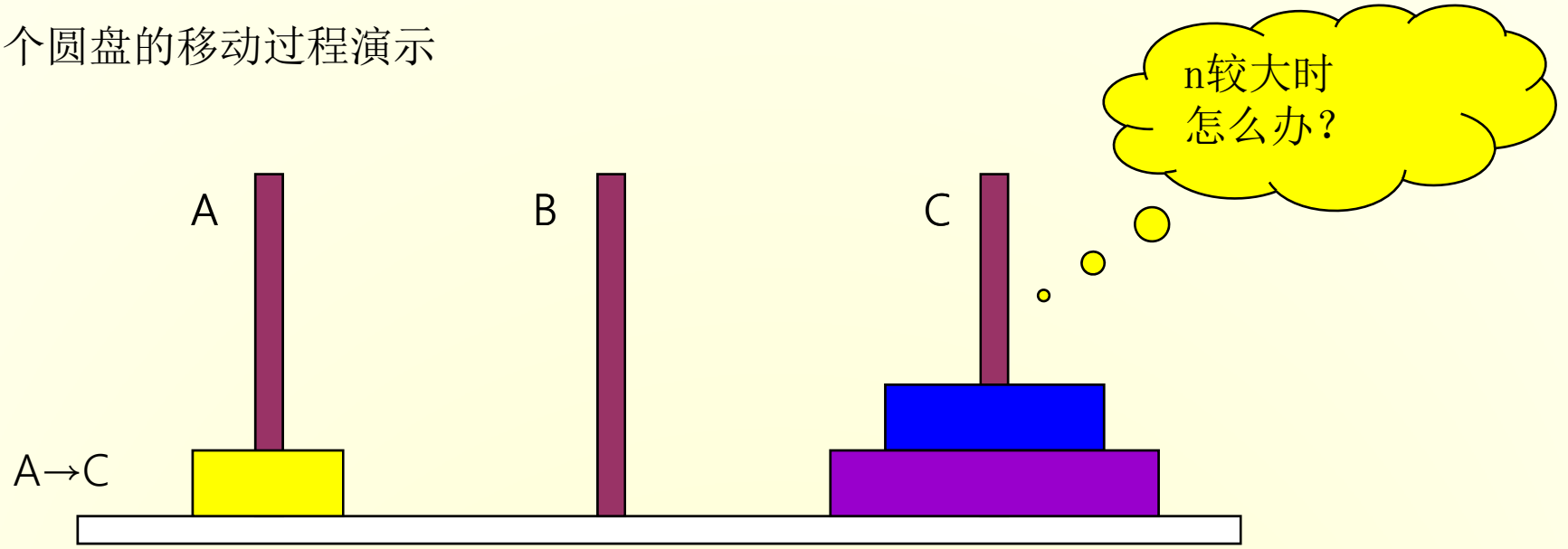
Hanoi 问题

- 3个圆盘的移动过程演示



Hanoi 问题

➢3个圆盘的移动过程演示



3个圆盘一共需要7次移动：

A→C，A→B，C→B，
A→C，B→A，B→C，
A→C

n个圆盘需要2^n-1次移动

n=64时，需要$2^{64}-1$次移动，即1844亿亿次移动。

若每次移动需用1微秒，则总共需要60万年时间！

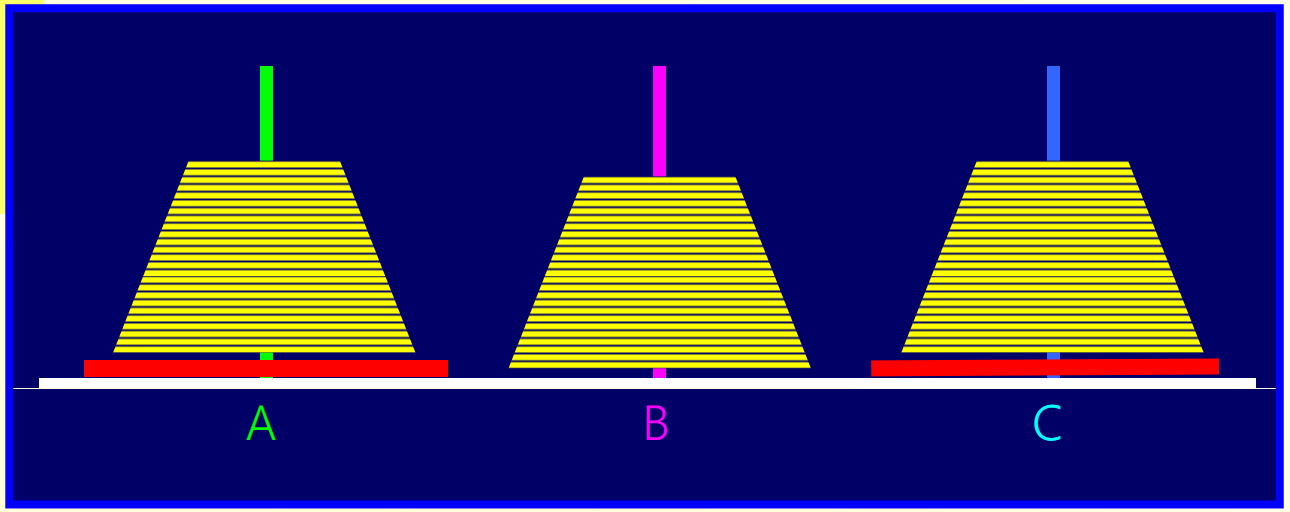
Hanoi问题

➢n个圆盘的移动方法：

n个圆盘分为2部分

{ 上面(n-1)个圆盘
{ 最下面的第n号圆盘

移动步骤：
① (n-1)个圆盘A→B
② 第n个圆盘A→C
③ (n-1)个圆盘B→C



(n-1)个圆盘怎么移动？

递归何时结束？

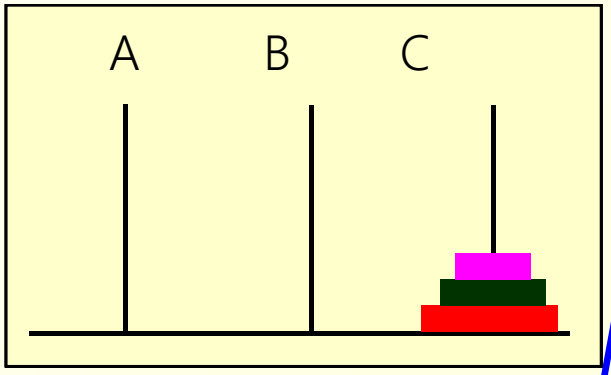
Hanoi 问题

- 递归法求解Hanoi问题

```
void  Han(int n, int A, int B, int C)
{   if (n==1)   Move(n, A, C);
    else
     {  Han(n-1, A, C, B);
        Move(n, A, C);
        Han(n-1, B, A, C);
      }
}
```

// 将n个盘子从A移到C,借助于B
// 将1个盘子从A移到C
// 将n-1个盘子从A移到B,借助于C
// 将第n个盘子从A移到C
// 将n-1个盘子从B移到C,借助于A

请特别注意这3个参数的顺序




```
Han( 3, A, B, C)
{   if (n==1)
        Move(n, A, C);
    else   {  Han(2, A, C, B);
              Move(3, A, C);
              Han(2, B, A, C);
           }
}
```

```
Han( 2, A, C, B)
{   if (n==1) Move(n, A, B);
    else   {  Han(1, A, B, C);
              Move(2, A, B);
              Han(1, C, A, B);
           }
}
```

```
Han( 1, A, B, C)
{   if (n==1)
        Move(1, A, C);
}
```

```
Han( 1, C, A, B)
{   if (n==1)
        Move(1, C, B);
}
```

```
Han( 2, B, A, C)
{   if (n==1) Move(n, B, C);
    else   {  Han(1, B, C, A);
              Move(2, B, C);
              Han(1, A, B, C);
           }
}
```

```
Han( 1, B, C, A)
{   if (n==1)
        Move(1, B, A);
}
```

```
Han( 1, A, B, C)
{   if (n==1)
        Move(1, A, C);
}
```

数据结构

- 数据：描述客观事物的数、字符以及所有能输入到计算机并被计算机程序处理的符号的集合，如数值、字符、图形、图像、声音等。
- 数据结构：带有结构的数据元素的集合
  - 结构反映了数据元素相互之间存在的某种联系
  - 数据结构是计算机科学技术的一个分支
    - 主要研究数据的逻辑结构和物理结构以及它们之间的关系
    - 对这种结构定义相应的运算，设计出实现这些运算的算法。

线性表

- 是*n*个数据元素的有限序列
- 线性表的运算：
- 设L为一个线性表
  - 置空表SETNULL（L）
  - 求表的长度LENGTH（L）
  - 取表元素GET（L，i）
  - 在表中查找特定元素LOCATE（L，x）
  - 插入新元素INSERT（L，i，b）
  - 删除表元素DELETE（L，i）
- 线性表的存储结构：
  - 顺序存储结构
  - 链式存储结构

堆 栈（stack）

- 是一种受限的线性表，即只能在表的一端（表尾）进行插入和删除操作
- 进栈和退栈操作按“后进先出”（Last In First Out，LIFO）的原则进行。
- 堆栈的运算：

  设S为一个堆栈
  - 置空栈SETNULL（S）
  - 进栈PUSH（S，x）
  - 退栈POP（S）
  - 取栈顶元素TOP（S）
  - 判断堆栈是否为空EMPTY（S）
- 堆栈的存储结构：顺序存储结构

堆 栈

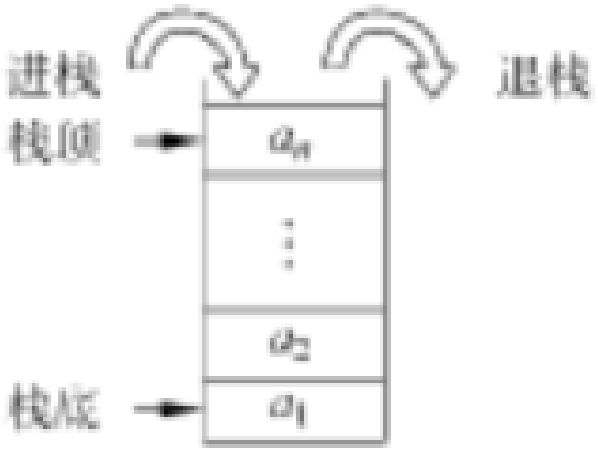


图 2-10　进栈和退栈操作的示意图

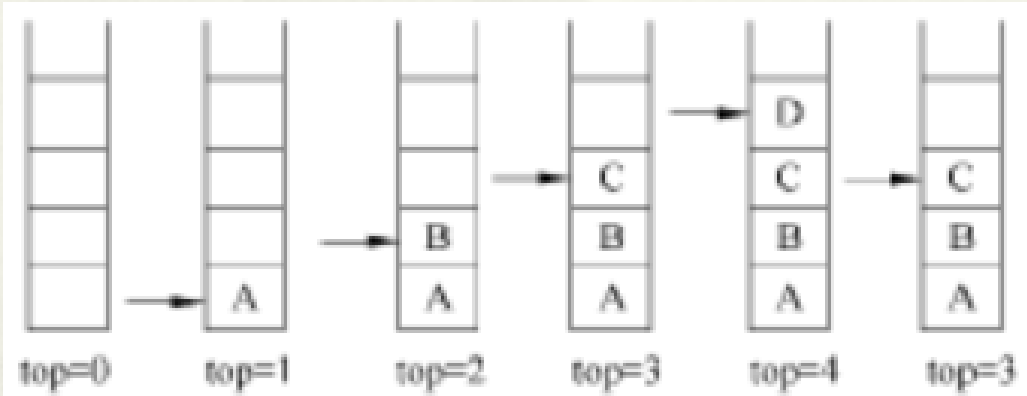


图 2-11　堆栈的存储结构示例

队 列（queue）

- 是一种受限的线性表，只能在表的一端（队尾）进行插入，在表的另一端（队首）进行删除操作。
- 进、出队列操作按“先进先出”（First In First Out，FIFO）的原则进行。
- 队列的运算

  设Q为一个队列
  - 置空队列SETNULL（Q）
  - 进入队列ADDQUEUE（Q，x）
  - 退出队列DELQUEUE（Q）
  - 取队首元素FRONTQUE（Q）
  - 判断队列是否为空EMPTY（Q）
- 队列的存储结构：链式存储结构，一个链队列需要设置队首指针和队尾指针。

队 列

图 2-12　队列示意图

图 2-13　链队列示意图

本章小结

- 本章介绍了计算机的一些基本知识，包括：数制与码制、数的定点与浮点表示、信息的编码；逻辑代数与逻辑电路基础；计算机的基本结构与工作原理；程序设计语言、结构化程序设计、程序设计风格；以及算法与数据结构的基础知识。
- 通过本章的学习，应掌握数据在计算机内部的表示形式及数制间的转换方法，理解逻辑代数、逻辑电路、计算机的结构、程序设计语言、结构化程序设计方法以及算法与数据结构的基本知识，为进一步学习本书的以下各章和后继课程打好基础。

作业

1. 堆栈的存取规则是？队列的存取规则是？
2. * 什么是命题公式，怎么判断两个命题公式是等价的？
3. 一个堆栈的入栈顺序是a,b,c,d,e,则不可能的出栈顺序是？
   A.edcba B.decba C. dceab D. abcde
4. 写出下列函数的真值表：
   $F = (\bar{A}BC + A\bar{B}C + AB\bar{C})$
5. 请用真值表证明下面的等式：
   a) $AB + \bar{A}B = (A + \bar{B})(\bar{A} + B)$
   b) $\bar{A}B + A\bar{B} = A \oplus B$
6. 请用逻辑代数的基本等价式证明下列等式：
   a) $A + \bar{A}B = A + B$
   b) $A(\bar{A} + B) = AB$
   c) $(A + B)(B + C)(C + D) = AC + BC + BD$