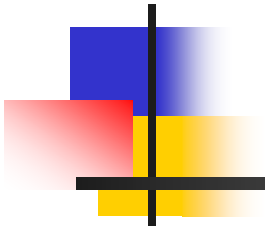


# 第一章 基本概念

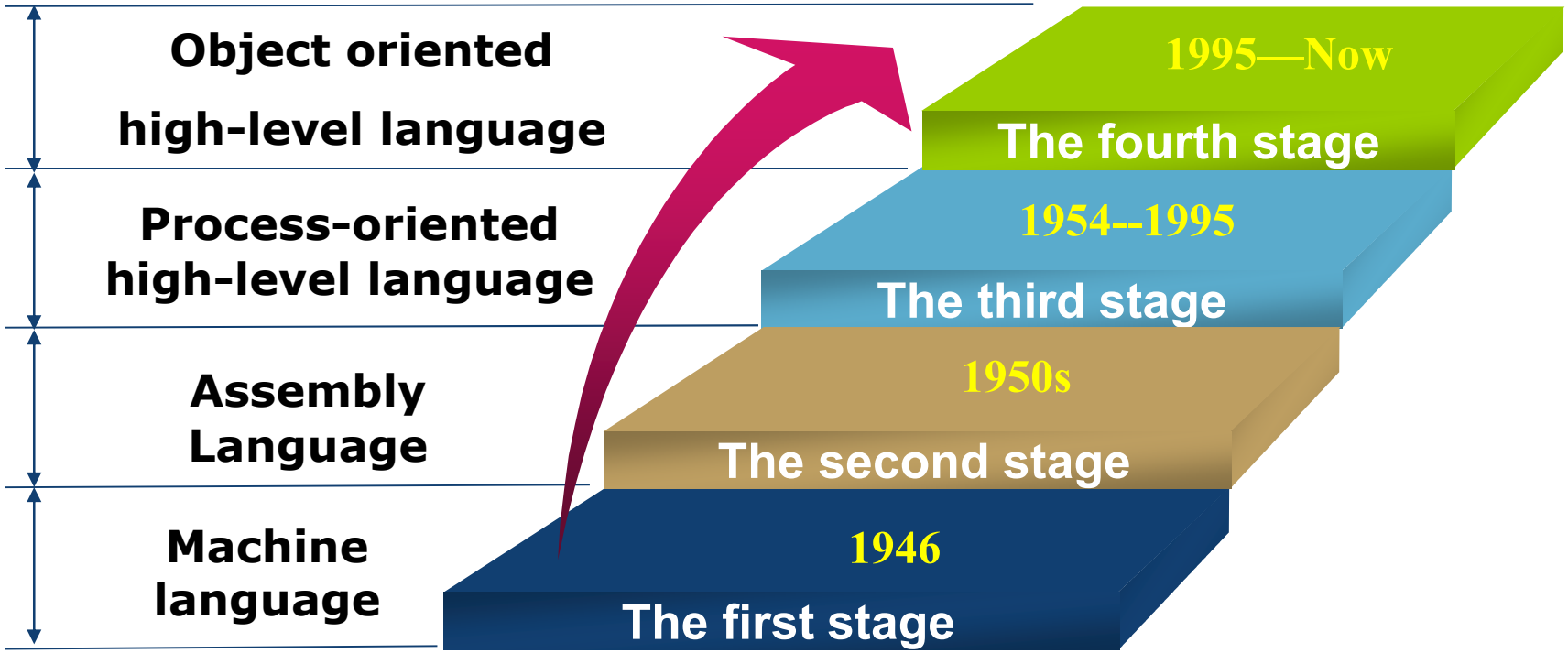
郑贵滨

# 第1章 基本概念

- 1.1 汇编语言简介
- 1.2 虚拟机的概念
- 1.3 数据的表示方法
- 1.4 布尔运算

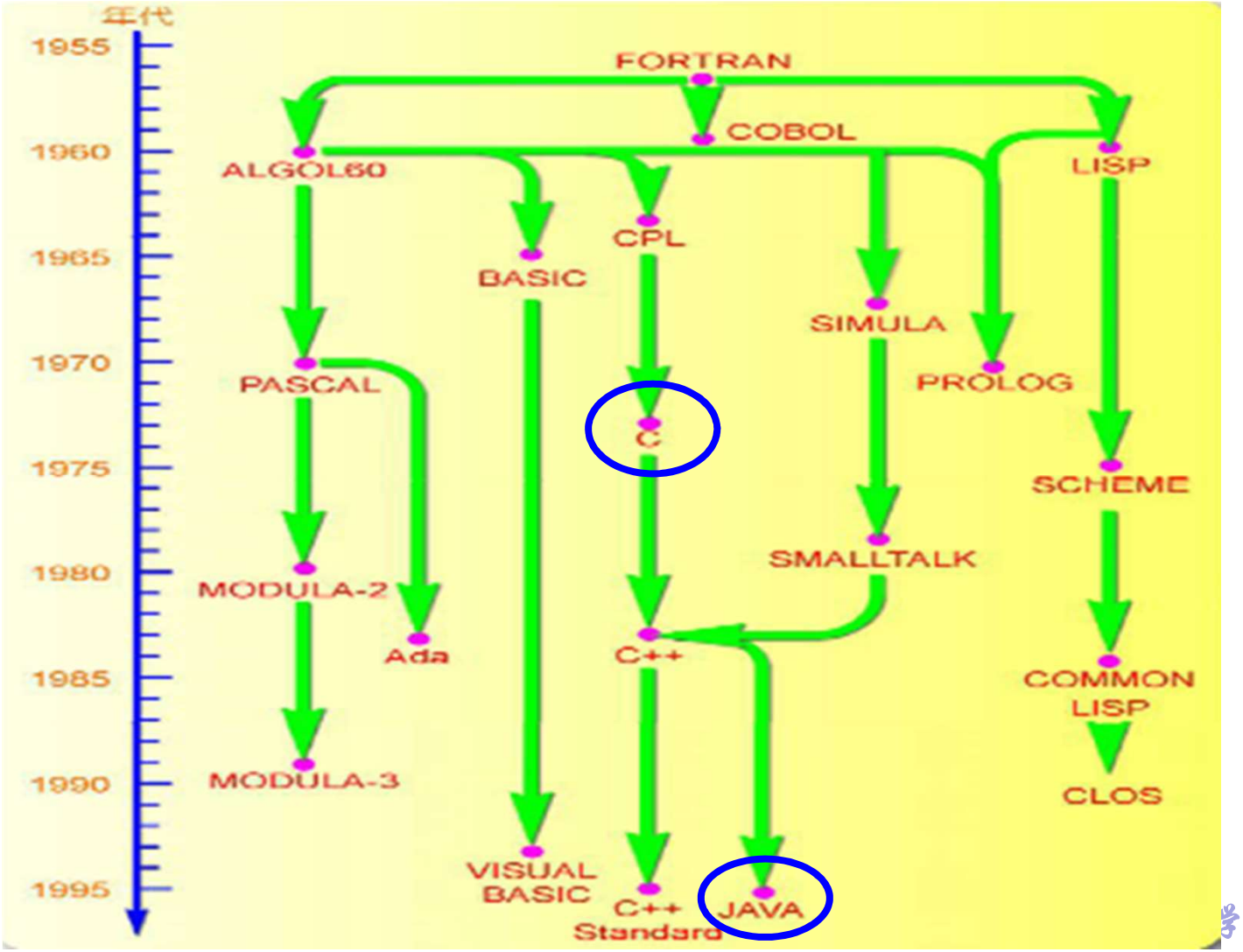
# 1.1 汇编语言简介

- 从计算机诞生至今，编程语言总数超过2500种
- 编程语言的发展简史——四个阶段



# 1.1 汇编语言简介

- 编程语言的发展简史——编年史



# （1） 机器语言

- 是一种二进制语言，由二进制数0、1组成的指令代码的集合，机器能直接识别和执行。
- 每一条语句都是二进制形式的代码。

  例如：1000 0000（加法）

  其他例子。
- 每条指令都简单到能够用相对较少的电子电路单元即可执行。
- 各种机器的指令系统互不相同。

# (1) 机器语言

- 采用穿孔纸带保存程序(1打孔，0不打孔)

| 优点： | 缺点： |
| --- | --- |
| 1. 速度快 | 1. 可移植性差 |
| 2. 占存储空间小 | 2. 编译难度大 |
| 3. 翻译质量高 | 3. 直观性差 |
| | 4. 调试困难 |

# （1） 机器语言

- 示例

应用8086CPU完成运算：

S = 768 + 12288 - 1280

机器指令码：

10110000000000000000011

00000101000000000110000

00101101000000000000101

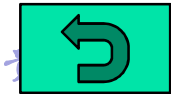
假如将程序错写成以下这样，请找处错误：

10110000000000000000011

00000101000000000110000

00010110100000000000101





# （2）汇编语言

## ❖ 汇编语言的产生

### ➢ 汇编语言指令——汇编语言的主体

- ☺ 汇编指令是机器指令便于记忆和阅读的书写格式——助记符，与人类语言接近，ADD、MOV、SUB和CALL等。
- ☺ 用助记符代替机器指令的操作码，用地址符号或标号代替指令或操数的地址。

机器指令：1000100111011000

操　　作：寄存器BX的内容送到AX中

汇编指令：MOV AX,BX

- ☺ 汇编指令同机器指令是一一对应的关系。

# 寄存器

- 寄存器：是CPU内部用于存放数据的有特定名称的存储单元。
- 一个CPU中有多个寄存器。
  - AX是其中一个寄存器的名称，
  - BX是另一个寄存器的名称。

- 更详细的内容我们在以后的课程中将会讲到。

# （2）汇编语言

- 示例

应用8086CPU完成运算：

S = 768 + 12288 - 1280

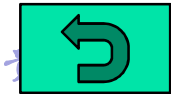
机器指令：

```
10110000000000000000011
00000101000000000110000
00101101000000000000101
```

汇编指令：

```
mov  S, 768      ;S是长度16位的字变量
add  S, 12288
sub  S, 1280
```





# （2）汇编语言

除汇编指令，汇编语言还包括：

- 伪指令　（由编译器执行）
- 其它符号（由编译器识别）

**汇编指令是汇编语言的核心，它决定了汇编语言的特性。**

**汇编语言的程序如何运行？**

计算机能读懂的只有机器指令

# （2）汇编语言

优点：

1. 执行速度快；

2. 占存储空间小；

3. 可读性有所提高。

缺点：

1. 类似机器语言；

2. 可移植性差；

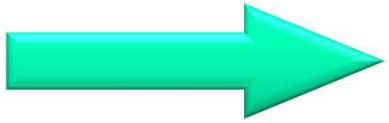
3. 与人类语言还相差很悬殊。

# （3）高级语言

- C++和Java等高级语言与汇编语言的关系

C++和Java等**高级语言与汇编语言及机器语言之间是一对多的关系**。一条简单的C++语句会被扩展成多条汇编语言或者机器语言指令。

```
X = （Y + 4）* 3;
```



```
mov eax, Y
add eax, 4
mov ebx, 3
imul  ebx
mov X, eax
```

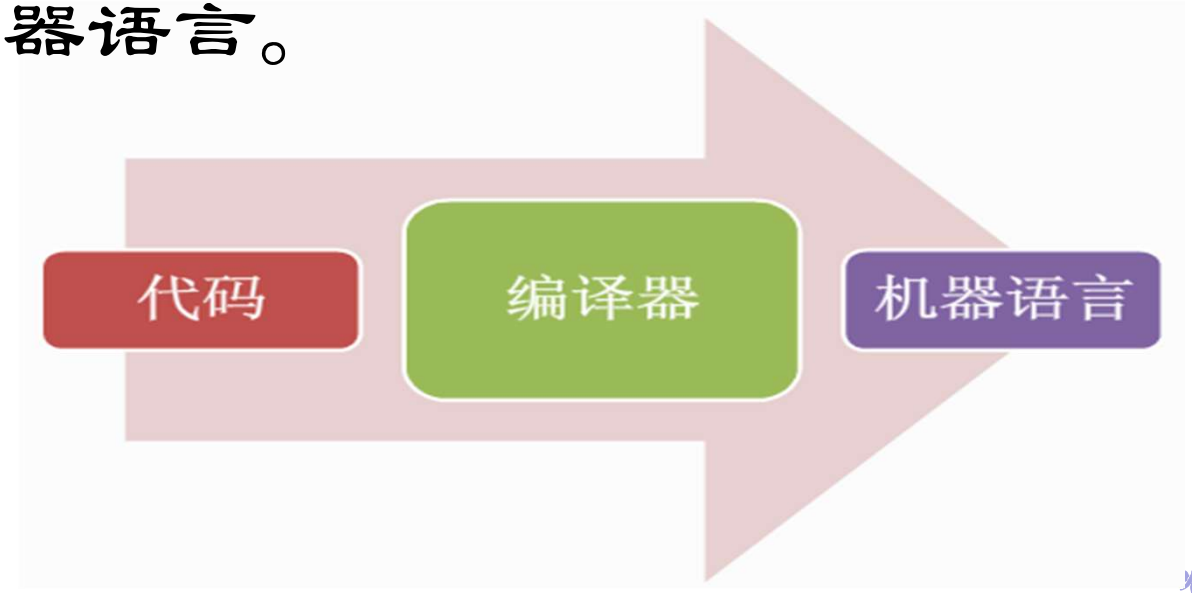
# （4）高级语言到机器语言的转换方法

- 解释方式

通过解释程序，逐行转换成机器语言，转换一行运行一行。

- 编译方式（翻译方式）

通过编译程序（编译、链接）将整个程序转换成机器语言。



# （5）汇编语言和高级语言的比较

- **可移植性**：如果一种语言的源程序代码可以在多种计算机系统上编译运行，那么这种语言就是可移植的。
  - 汇编语言总是和特定系列的处理器捆绑在一起。
  - 当今有多种不同的汇编语言，每种都是基于特定系列的处理器或特定计算机的。
  - 汇编语言没有可移植性。
  - 高级语言的可移植性好。

# （5）汇编语言和高级语言的比较

| 应用程序类型 | 高级语言 | 汇编语言 |
| --- | --- | --- |
| 用于单一平台的中到大型商业应用软件 | 正式的结构化支持使组织和维护大量代码很方便 | 最小的结构支持使程序员需要人工组织大量代码，使各种不同水平的程序员维护现存代码的难度极高 |
| 硬件驱动程序 | 语言本身未必提供直接访问硬件的能力，即使提供了也因为要经常使用大量的技巧而导致维护困难 | 硬件访问简单直接。当程序很短并且文档齐全时很容易维护 |
| 多种平台下的商业应用软件 | 可移植性好，在不同平台可以重新编译，需要改动的源代码很少 | 必须为每种平台重新编写程序，通常要使用不同的汇编语言，难于维护 |
| 需要直接访问硬件的嵌入式系统和计算机游戏 | 由于生成的执行代码过大，执行效率低 | 很理想，执行代码很小并且运行很快 |

# （6）为什么学汇编？

- 深入了解计算机体系结构和操作系统
- 在机器层次思考并处理程序设计中遇到的问题
- 在许多专业领域，汇编语言起主导作用：
  - 嵌入式系统
  - 游戏程序
  - 设备驱动程序
- 软件优化，通过汇编语言使用最新最快的CPU指令，获得最高的处理速度。 ***...实例：多媒体信息处理运算***
- **后继课程的学习**

# 关联课程...

--------完成的课程---------

- ✓ 计算机专业导论 1秋
- ✓ 高级语言程序设计Ⅰ 1秋
- ✓ 高级语言程序设计Ⅱ 1春
- ✓ C++ 1夏
- ✓ Java 1夏

------并行的课程 2秋-------

- ➢ 软件设计与开发实践Ⅰ
- ➢ 数据结构与算法
- ➢ 汇编语言程序设计
- ➢ 数字逻辑设计

---------后继课程----------

- ? 计算机组成原理 2春
- ? 计算机设计与实践 2夏
- ? 计算机体系结构 3春
- ? 接口技术 3春
- ? VLSI设计 3夏
- ? *操作系统* *3秋*
- ? *编译原理* *3春*

# （7）汇编学什么？

- 了解计算机体系结构的基本原理；
- 了解布尔逻辑如何被用于程序设计和计算机硬件；
- 了解IA-32处理器如何管理内存，如何使用实模式、保护模式和虚拟模式；
- 高级语言如何在机器语言层次实现算术运算、循环和逻辑结构；
- 高级语言编译器如何将源代码翻译成汇编语言和机器语言

# （7）汇编学什么？

- 了解数据的表示方法，包括有符号和无符号整数、实数以及字符数据
- 了解应用程序如何通过中断服务程序和系统调用与操作系统进行交互，以及操作系统如何加载和执行应用程序
- 学习汇编语言代码与C++程序之间的接口

# （8）编程环境

- 汇编编译器，也称汇编程序（Assembler）把汇编语言源程序转换为机器语言的目标文件(.obj)；
- 链接器把一个或多个目标文件和库文件合成一个可执行文件（.exe)；教材提供的两个链接器：Microsoft的16位链接器LINK.EXE和32位的链接器LINK32.EXE。
- 调试器/调试程序为程序员提供了跟踪程序执行过程以及查看寄存器和内存内容的工具。(debug、windbg)
- Intel处理器可使用的两种最流行的汇编编译器

  **Microsoft MASM** (MacroAssembler)

  **TASM** (Borland Turbo Assembler)

# （8）编程环境

- 调试程序（Debugger）
  - **DEBUG**：DOS下的调试程序，也可编辑程序。

    若在Windows下使用需注意：开始-〉运行command开启16位模拟DOS环境,然后debug（windows自带）。（不用cmd——32位控制台环境）
  - MASM采用**CodeView**
  - 其他还有**Turbo Debugger**等
  - **WinDbg**是微软提供的Windows调试程序

    WinDbg支持源程序级调试，但需要在汇编、连接过程中分别加入参数"/Zi"（大写Z、小写i）、"/ debug ":
    - ML /c /coff /Fl /Zi First.asm
    - LINK32 /subsystem:console /debug First.obj

# （8）编程环境：集成开发环境(IDE)

- 为方便同学实验，建议采用：
  - **轻松汇编QASMII**(调用TASM 5.0)
    - Windows平台下的汇编语言集成开发环境，集成了文本编辑器（QASM自带）、汇编程序（TASM）、连接程序（TLINK）、调试程序（TDEBUG）等。
    - 它只是在内部调用TASM、TLINK、TDEBUG，所以它遵守这些软件的工作特点。
    - 上述这些软件是Borland公司DOS下的汇编语言工具，遵守DOS的规定。如文件名（包括路径名）不能是汉字，不能超过8个字符。扩展名缺省是.asm，这在做实验时必须注意。否则编译出错
  - **Visual Studio 2010/2013/2015**

    微软出品，windows下最强大的IDE，不能用于编译实模式16位dos程序。

# 1.2 虚拟机的概念

- 机器语言——L0语言：编程非常困难
- 创建一种语言——L1：易于编程
- L1源程序的执行有两种方法：
  - **解释方式**：当执行L1程序时，使用L0编写的解释程序对L1程序的每条指令解码并执行，每条指令在执行之前都必须进行解码处理。
  - **编译方式**：用特别设计的L0程序将整个L1源程序翻译成L0程序，生成的L0程序就可以直接在计算机硬件上执行了。

# 1.2 虚拟机的概念

- 以此类推，可建立起一个由虚拟机组成的多层结构：
  - 虚拟机VM1执行用L1语言书写的指令
  - 虚拟机VM0执行用L0语言书写的指令。


- 如虚拟机VM1是实实在在的计算机，为虚拟机VM1书写的程序就可以直接在硬件上执行。
- 否则，为VM1编写的程序需要通过解释或编译的方法转到虚拟机VM0上执行。

# 1.2 虚拟机的概念

❖ 计算机语言的虚拟机层次结构：

高级语言程序通常由编译器翻译成汇编语言的程序，再由汇编编译器将其翻译成机器语言

使用助记符，很容易翻译到指令体系结构层，其他一些汇编语句，则由操作系统直接执行

计算机操作系统管理系统的软、硬资源。操作系统软件被翻译成机器码在第2层运行

一套固化在处理器内部的指令集——机器语言。1条机器语言指令分解为几条微指令执行

处理器内部的硬件解释器

计算机的数字逻辑硬件


图1.1 虚拟机的第0层到第5层

# 1.3数据的表示

主要内容：

- 进制、进制的转换
- 原码反码补码
- 字符的存储

# 1.3.1 进制

- 进制：逢“k”进1
  - “满十进一”就是十进制(Decimal)，
  - “满二进一”就是二进制(Binary)
  - “满八进一”就是八进制(Octal System)
  - “满十六进一”就是十六进制(Hexadecimal System)
  - “满k进一”就是k进制 （k叫做基数,单个数字位最多能表示的符号数）
- 例子：
  - 一小时有六十分——六十进制
  - 一个星期有七天——七进制
  - 一年有十二个月——十二进制
  - 电子计算机——二进制
  - 半斤八两？

# 1.3.1 进制

- 数的通用表示

$N= \pm a_n a_{n-1} \ldots a_1 a_0 . b_1 b_2 \ldots b_m$

若基值为k, 则N= ?

10进制：

$3721 = 3 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 1 \times 10^0$

k进制：

$$N= \pm a_n \times k^n + a_{n-1} \times k^{n-1} + \ldots + a_1 \times k^1 + a_0 \times k^0 + b_1 \times k^{-1} + b_2 \times k^{-2} + \ldots + b_m \times k^{-m}$$

其中$a_i$，$b_j$是$0 \sim k-1$中的一个数码

# 二进制数

- 特点：逢二进一，由0和1两个数码组成，基数为2，各个位权以$2^i$表示
- 二进制数：

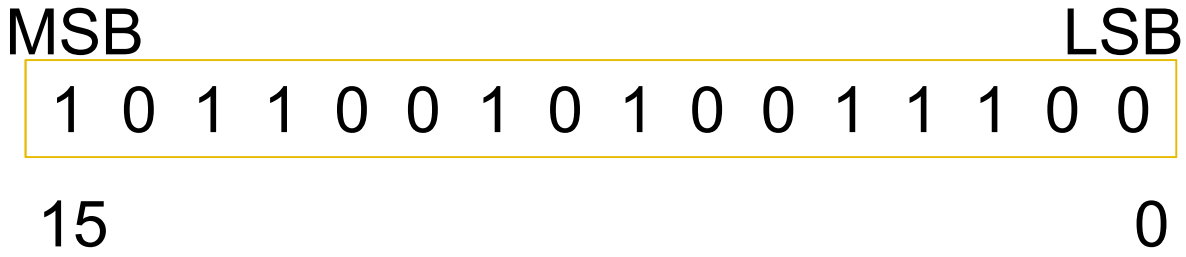
$$a_n a_{n-1} \ldots a_1 a_0 . b_1 b_2 \ldots b_m =$$
$$a_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2^1 + a_0 \times 2^0$$
$$+ b_1 \times 2^{-1} + b_2 \times 2^{-2} + \ldots + b_m \times 2^{-m}$$

其中$a_i$，$b_j$非0即1

**便于计算机存储、算术运算简单、支持逻辑运算**

# 二进制数

- MSB：最高有效位（Most Significant Bit）
- LSB：最低有效位（Least Significant Bit）




**数字串长、书写和阅读不便**

# 十六进制数

- 基数16，逢16进位，位权为$16^i$，16个数码：

  0, 1，2，3，4，5，6，7，8，9，A，B，C，D，E，F

- 十六进制数：

  $a_n a_{n-1} \ldots a_1 a_0 . b_1 b_2 \ldots b_m =$

  $$a_n \times 16^n + a_{n-1} \times 16^{n-1} + \ldots + a_1 \times 16^1 + a_0 \times 16^0$$

  $$+ b_1 \times 16^{-1} + b_2 \times 16^{-2} + \ldots + b_m \times 16^{-m}$$

  其中$a_i$，$b_j$是0～F中的一个数码

# 十六进制数的加减运算

- 十六进制数的加减运算类似十进制
  - 逢16进位1，借1当16

$$23D9H + 94BEH = B897H$$

$$A59FH - 62B8H = 42E7H$$

- 二进制和十六进制数之间具有对应关系：

  每4个二进制位对应1个十六进制位

  00111010B = 3AH，F2H = 11110010B

**与二进制数相互转换简单、阅读书写方便**

# 进制转换

- 十进制整数转换为k(2、8或16)进制数

  整数转换：用除法——除基取余法

  - 十进制数整数部分不断除以基数k(2、8或16)，并记下余数，直到商为0为止
  - 由最后一个余数起，逆向取各个余数，则为转换成的二进制和十六进制数

  126＝01111110B　二进制数用后缀字母B

  126＝7EH　十六进制数用后缀字母H

# 进制转换

- 十进制小数转换为k(2、8或16)进制数...

  小数转换：用乘法—乘基取整法

  乘以基数k，记录整数部分，直到小数部分为0为止

  $0.8125 = 0.1101B$

  $0.8125 = 0.DH$

  - 小数转换会发生总是无法乘到为0的情况
  - 可选取一定位数（精度）
  - 将产生无法避免的转换误差

# 进制转换

- k进制数转换为十进制数

  方法：按权展开

  - 二进制数转换为十进制数

    0011.1010B

    $= 1\times 2^{1} + 1\times 2^{0} + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} = 3.625$

  - 十六进制数转换为十进制数

    $1.2\text{H} = 1\times 16^{0} + 2\times 16^{-1} = 1.125$

- 2、8、16进制间的转换

  4个2进制位对应1个16进制位

  3个2进制位对应1个8进制位

# 1.3.2 原码、反码、补码

- **真值**：实际的数值（包括正负号）。

  如：+35，-1001110B，-100101B

- **机器数**：数在计算机中的表示；
  - 在计算机中，数只能用二进制表示，符号也用二进制数位表示；
  - 存放在寄存器或储存单元中。

    例如：10000111，11001001，01100011

字节（BYTE） 8位（bit）

字的实际长度和CPU型号有关(CPU字长)：16、32、64bits

汇编中的字长？

# 1.3.2 原码、反码、补码

- 机器数...
  - 定点数：固定小数点的位置表达数值的机器数
    - 定点整数：将小数点固定在机器数的最右侧表达的整数
    - 定点小数：将小数点固定在机器数的最左侧表达的小数
  - 浮点数：小数点浮动表达的实数
  - 无符号数：只表达0和正整数的定点整数
  - 有符号数：表达负整数、0和正整数的定点整数
- 机器数的表示：原码、反码、补码

# 原码、反码、补码

- 原码： 数的最高位是符号位，其余不变。
  - 符号位（最高位）用“0”和“1”分别表示数的符号“+”和“-”
  - 数值部分不变
  - 数x的原码一般表示为 $[x]_{原}$
- 反码：
  - 正数的反码：与原码相同；
  - 负数的反码：符号位为“1”(同原码)，数值位按位取反。
  - 数x的反码一般表示为 $[x]_{反}$

如：

$x = +1001001 \text{ ----> } [x]_{反} = 01001001$

$x = -1100100 \text{ ----> } [x]_{反} = 10011011$

# 原码、反码、补码

- 补码：最高位表示符号：正数用0，负数用1
  - 正数补码：与原码相同；
  - 负数补码：将对应正数补码取反加1

[105]补码＝01101001B

[-105]补码＝[01101001B]取反＋1

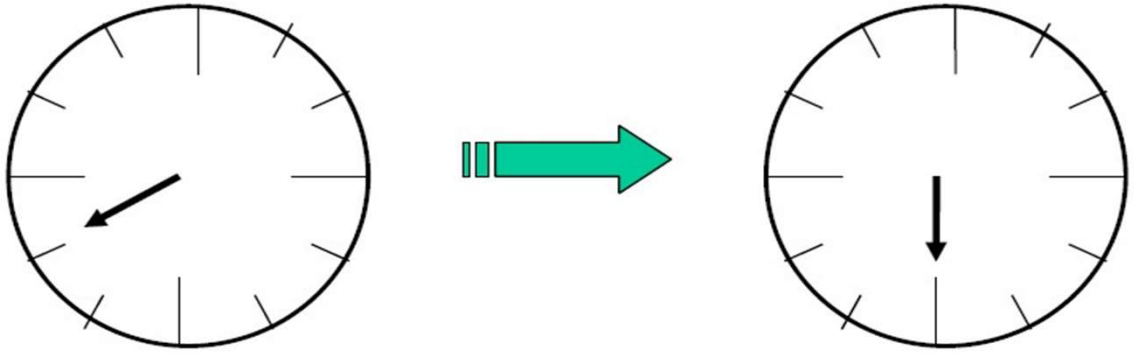
＝10010110B＋1＝10010111B

- 补码的表示范围：
  - 字节— 8位二进制补码：-128～+127
  - 字 — 16位二进制补码：$-2^{15}$～$+2^{15}-1$, 32768～32767
  - 双字— 32位二进制补码：$-2^{31}$～$+2^{31}-1$，
    -2147483648～2147483647

**有符号整数在计算机中默认采用补码**

# 补码·引入的思路（1）

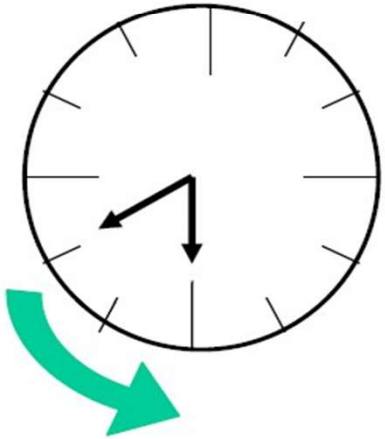
- 由钟表拨表针的方法得到启示；
- 例如：把表上的8点钟改为6点钟



方法一：

反时针

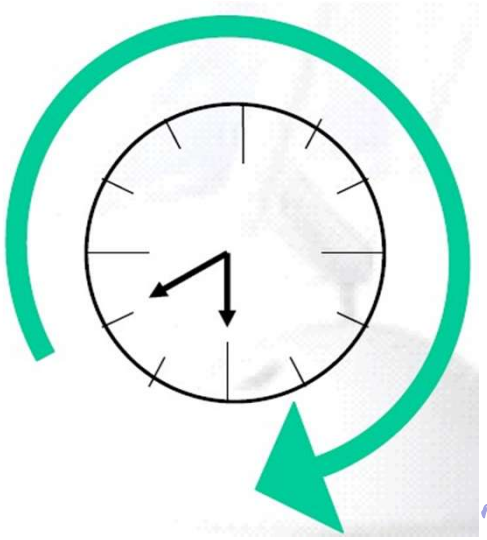
拨2格



方法二：

顺时针

拨10格



# 补码•引入的思路（2）

- 拨针方法小结：

$$8 - \mathbf{2} = 6$$

$$8 + \mathbf{10} = 6$$

- 思考：为什么会出现这种现象？计算机中是否也有这种现象？

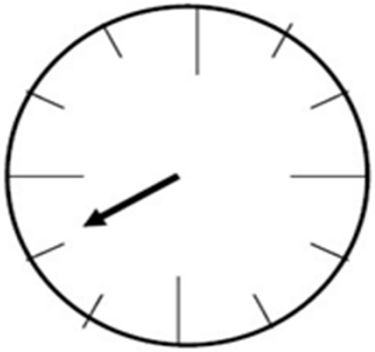
(表盘是圆的，可循环计时。)

  - 计算机储存一个数也有与钟表相同的特点：循环计数
  - 因此对于计算机，要计算像8-2这样的减法式子，也可以化为加法形式来进行。

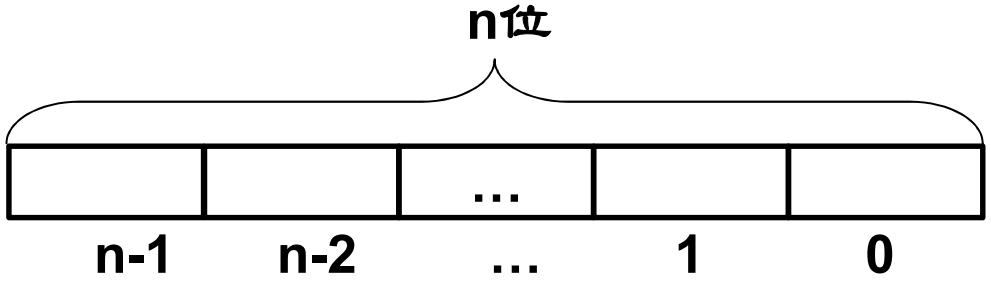
- 思考：在计算机中，8-2是否也可以化为8+10？如果不行，那么应化为什么样的式子？

# 补码•引入的思路（3）

- 不同之处：
  - 表计时的最大数是12
  - 计算机计数的最大数不是12（是多少？）
  - 计数的最大数称为 **模**，求模运算：%、mod
  - 计算机的模与字长有关。**n位机的模是$2^n$**



**模12**




**模$2^n$**

# 补码•引入的思路（4）

- 观察钟表拨针的两种方法：

  $8 - 2 = 6$

  $8 + 10 = 6$

减去一个数 a 相当于加上（模-a）一样，而在计算机中也有相同情况。

- 在n位字长的计算机中：

  减去一个数a相当于加上（$2^n$-a）一样。

- （$2^n$-a）称为a的**补**数，其二进制表示形式称为**补码**。

# 补码•补码的求法（1）

- 正数：与原码相同
- 负数：“求反加1”

例：x=+1001100B，则[x]补=01001100B=[x]原

x=-1001100B，则[x]补=10110100B

x=-1001100B时

$[x]_{补}=2^8-1001100B$

$=256-1001100B$

$=255-1001100B+1$

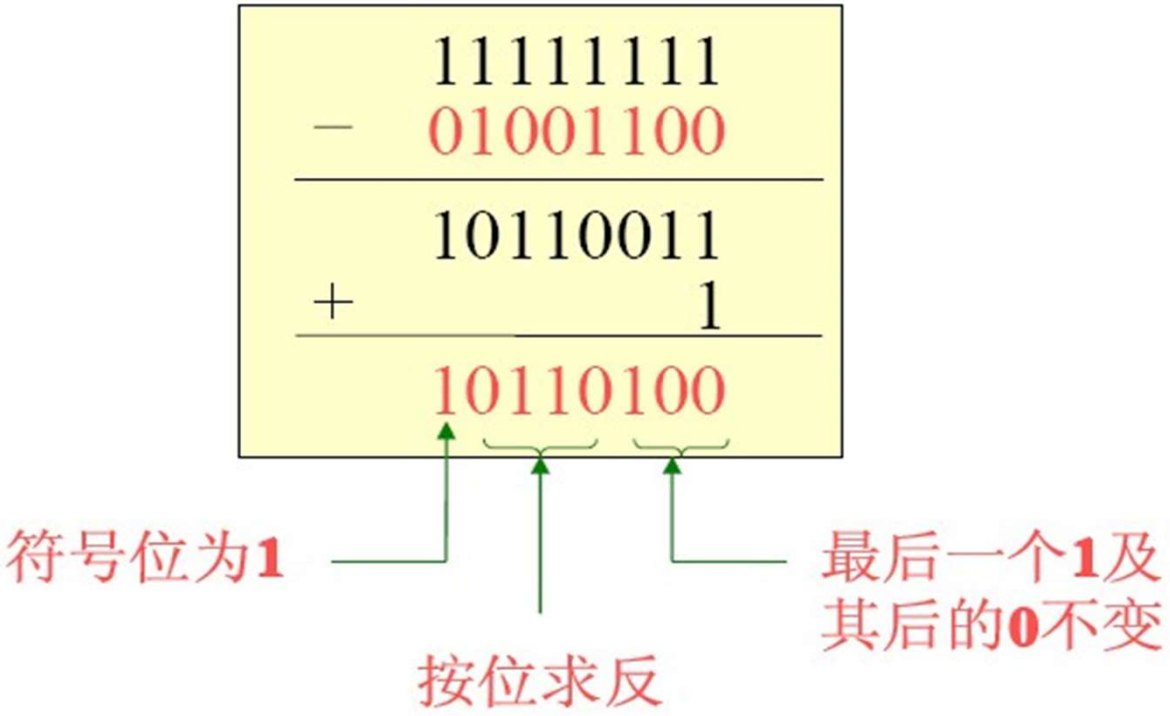
$=11111111B-1001100B+1$

$=10110100B$

```
  11111111
- 01001100
----------
  10110011   ← 求反
+        1   ← 加 1
----------
  10110100
```

# 补码•补码的求法（2）

- 负数补码的快速求法：
  - 符号位为1
  - 真值中最后一个1以前的各位按位求反，而最后一个1及其后的0保持不变。



# 补码•补码的求法（3）

❖ 练习

1、求出下列各数的补码（8位）

(1) – 67　　10111101

(2) +1011001B　　01011001

(3) – 45　　11010011

# 补码•补码的求法（4）

- 用补码表示计算机中的数后，加减运算均可统一为加法。

例： 设 x=+0000111 , y=+0000100, 计算式子： x—y

(先算$[x]_{补}$=00000111, $[-y]_{补}$=11111100 , **x—y=x+(-y)** )

| 补码运算： | |
|---|---|
| 00000111 | $[x]_{补}$ |
| + 11111100 | $[-y]_{补}$ |
| 1 00000011 | $[x-y]_{补}$ |
| ↑ 自然丢失 | |

| 原码运算： | |
|---|---|
| 0000111 | x |
| – 0000100 | y |
| 0000011 | x-y |

# 补码•补码的作用与效果

- 计算机内部，带符号数均用补码表示
  - 采用补码进行运算后，结果也是补码
  - 欲得真值，需作转换。

转换方法：
- 0开头：将0换成“+”号，其余数位不变。
- 1开头：1换成“–”号，其余
  - 方法1：减1求反
  - 方法2：求反加1

- 溢出：根据最高位的进、借位情况进行判断。
  - 溢出：“有进无出”或“无进有出”
  - 正常：“有进有出”或“无进无出”

# BCD码（Binary Coded Decimal）

- BCD码：二进制编码的十进制数

  一个十进制数位0~9用4位二进制编码来表示

  - 常用8421 BCD码：低10个4位二进制编码表示
  - 压缩BCD码：一个字节表达两位BCD码
  - 非压缩BCD码：一个字节表达一位BCD码（低4位表达数值，高4位常设置为0）
  - BCD码很直观

  BCD码：0100 1001 0111 1000.0001 0100 1001

  十进制真值：　4978.149

**BCD码便于输入输出，表达数值准确**

# 1.3.3 字符的存储

- ASCII码

美国标准信息交换码American Standard Code for Information Interchange

  - 标准ASCII码用7位二进制编码，有128个字符。
  - 不可显示的控制字符

前32个和最后一个编码

回车CR：0DH　换行LF：0AH　响铃BEL：07H

  - 可显示和打印的字符：20H后的94个编码
    - 数码0~9：30H~39H
    - 大写字母A~Z：41H~5AH
    - 小写字母a~z：61H~7AH
    - 空格(space)：20H
  - 扩展ASCII码：最高D7位为1，表达制表符

| 高3位 / 低4位 | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 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 |

# 1.3.3 字符的存储

- ANSI字符集

8位，表示256个字符：前128个字符对应于标准美国键盘上的字母和符号；后128个字符表示特殊字符。

Unicode标准：

- Unicode标准——处理包含多于256个字符的字符集
  - 定义字符和符号的统一方法，包括世界各种常用语言，是个很大的集合，现在的规模100多万。
  - 每个符号的编码都不一样，A：U+0041、汉字“严”：U+4E25。具体的符号对应表，可以查询unicode.org

# 1.3.3 字符的存储

- **Unicode的三种编码形式：**

  （1） **UTF-8**：以8位为单元对UCS(Unicode Character Set)进行编码，是一种变长的字节编码系统。ASCII码占用一个字节，字节值和ASC II码值一样。

| Unicode符号范围(16进制) | UTF-8编码方式(2进制) |
|---|---|
| 0000 0000-0000 007F | 0xxxxxxx |
| 0000 0080-0000 07FF | 110xxxxx 10xxxxxx |
| 0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx |
| 0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx |

# Unicode的三种编码形式：

**(2) UTF-16:**每个字符编码长16位，用于访问效率和存储空间并重的环境中。

VC的wchar_t *类型的字符串(包括硬编码在.h/.cpp里的字符串字面值)，自动采用UTF-16的编码

**(3) UTF-32:**用于不太关心存储空间的环境，每个字符宽度固定为32位。

# 汉字编码——国标码GB2312

- GB2312又称为GB2312-80字符集，全称为《信息交换用汉字编码字符集•基本集》，由原中国国家标准总局发布，1981年5月1日实施。
- 汉字信息在计算机内部也是以二进制方式存放。由于汉字数量多，用一个字节的128种状态不能全部表示出来，因此在GB2312-80中规定用两个字节的十六位二进制表示一个汉字，每个字节都只使用低7位（与ASCII码相同），即有128×128=16384种状态。

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | x | x | x | x | x | x | x | 0 | x | x | x | x | x | x | x |

# 汉字编码——国标码GB2312

- 由于ASCII码的34个控制代码在汉字系统中也要使用，为不致发生冲突，不能作为汉字编码，128除去34只剩94种，所以汉字编码表的大小是94×94=8836，用以表示国标码规定的汉字6763个（一级汉字，是最常用的汉字，按汉语拼音字母顺序排列，共3755个；二级汉字，属于次常用汉字，按偏旁部首的笔划顺序排列，共3008个），数字、字母、符号等682个，共7445个。

# 汉字编码——区位码

- 将整个代码表分为94个区，每个区94个位，区号对应高字节，位号对应低字节，合起来就是区位码。

  如“保”字在二维代码表中处于17区第03位，区位码即为1703D=1103H 。

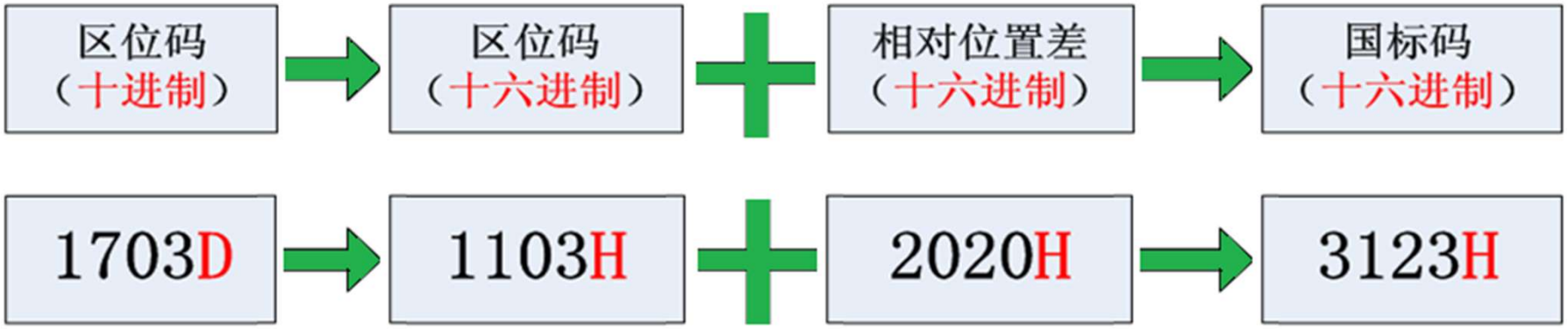
- 注：区位码的高字节和低字节的取值范围均是01H~5EH。

# 汉字编码——区位码

- 区位码和国标码的关系如下：

  区码 = 高字节国标码-20H

  位码 = 低字节国标码-20H



# GB码区位示例（续）

| 56区 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | 于 | 丌 | 兀 | 丐 | 廿 | 卅 | 丕 | 亘 | 丞 |
| 1 | 鬲 | 孬 | 噩 | 丨 | 禺 | 丿 | 匕 | 毛 | 夭 | 爻 |
| 2 | 卮 | 氐 | 囟 | 胤 | 馗 | 毓 | 睾 | 鼗 | 丶 | 亟 |
| 3 | 鼐 | 乜 | 乩 | 亓 | 半 | 孛 | 嗇 | 碞 | 仄 | 厍 |
| 4 | 厝 | 厣 | 厥 | 厮 | 厩 | 厶 | 匚 | 匝 | 砸 | 匮 |
| 5 | 匾 | 赜 | 卦 | 卣 | 刂 | 刈 | 刎 | 刭 | 刨 | 刿 |
| 6 | 剴 | 剌 | 剞 | 剡 | 剜 | 蒯 | 剽 | 劂 | 劁 | 劐 |
| 7 | 劓 | 冂 | 冏 | 冫 | 冃 | 仉 | 仂 | 仨 | 仡 | 仫 |
| 8 | 仞 | 伛 | 仳 | 伢 | 佤 | 仵 | 伥 | 伧 | 伉 | 伫 |
| 9 | 佞 | 佧 | 攸 | 佚 | 佝 | | | | | |

| 57区 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | 佟 | 佗 | 伲 | 伽 | 佶 | 佴 | 侑 | 侉 | 侃 |
| 1 | 侏 | 佾 | 佻 | 侪 | 佼 | 侬 | 侔 | 俦 | 俨 | 俪 |
| 2 | 俅 | 俚 | 俣 | 俜 | 俑 | 俟 | 俸 | 倩 | 偌 | 俳 |
| 3 | 倬 | 倏 | 倮 | 倭 | 俾 | 倜 | 倌 | 倥 | 倨 | 偾 |
| 4 | 偃 | 偕 | 偈 | 偎 | 偬 | 偻 | 傥 | 傧 | 傩 | 傺 |
| 5 | 僖 | 僦 | 僭 | 僬 | 僵 | 僮 | 僖 | 僭 | 仝 | 氽 |
| 6 | 佘 | 金 | 俎 | 龠 | 汆 | 籴 | 兮 | 巽 | 黉 | 馘 |
| 7 | 糇 | 糭 | 勹 | 匍 | 訇 | 匐 | 凫 | 夙 | 兕 | 亠 |
| 8 | 兖 | 亳 | 衮 | 袤 | 亵 | 裔 | 裒 | 禀 | 嬴 | 蠃 |
| 9 | 羸 | 冫 | 冱 | 冽 | 冼 | | | | | |

# 汉字编码——区位码

- 01-09区为特殊符号；16-55区为一级汉字，按拼音排序；56-87区为二级汉字，按部首/笔画排序；10-15区及88-94区则未有编码。

| 16区 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | 啊 | 阿 | 埃 | 挨 | 哎 | 唉 | 哀 | 皑 | 癌 |
| 1 | 蔼 | 矮 | 艾 | 碍 | 爱 | 隘 | 鞍 | 氨 | 安 | 俺 |
| 2 | 按 | 暗 | 岸 | 胺 | 案 | 肮 | 昂 | 盎 | 凹 | 敖 |
| 3 | 熬 | 翱 | 袄 | 傲 | 奥 | 懊 | 澳 | 芭 | 捌 | 扒 |
| 4 | 叭 | 吧 | 笆 | 八 | 疤 | 巴 | 拔 | 跋 | 靶 | 把 |
| 5 | 耙 | 坝 | 霸 | 罢 | 爸 | 白 | 柏 | 百 | 摆 | 佰 |
| 6 | 败 | 拜 | 稗 | 斑 | 班 | 搬 | 扳 | 般 | 颁 | 板 |
| 7 | 版 | 扮 | 拌 | 伴 | 瓣 | 半 | 办 | 绊 | 邦 | 帮 |
| 8 | 梆 | 榜 | 膀 | 绑 | 棒 | 磅 | 蚌 | 镑 | 傍 | 谤 |
| 9 | 苞 | 胞 | 包 | 褒 | 剥 | | | | | |

| 17区 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | 薄 | 雹 | 保 | 堡 | 饱 | 宝 | 抱 | 报 | 暴 |
| 1 | 豹 | 鲍 | 爆 | 杯 | 碑 | 悲 | 卑 | 北 | 辈 | 背 |
| 2 | 贝 | 钡 | 倍 | 狈 | 备 | 惫 | 焙 | 被 | 奔 | 苯 |
| 3 | 本 | 笨 | 崩 | 绷 | 甭 | 泵 | 蹦 | 迸 | 逼 | 鼻 |
| 4 | 比 | 鄙 | 笔 | 彼 | 碧 | 蓖 | 蔽 | 毕 | 毙 | 毖 |
| 5 | 币 | 庇 | 痹 | 闭 | 敝 | 弊 | 必 | 辟 | 壁 | 臂 |
| 6 | 避 | 陛 | 鞭 | 边 | 编 | 贬 | 扁 | 便 | 变 | 卞 |
| 7 | 辨 | 辩 | 辫 | 遍 | 标 | 彪 | 膘 | 表 | 鳖 | 憋 |
| 8 | 别 | 瘪 | 彬 | 斌 | 濒 | 滨 | 宾 | 摈 | 兵 | 冰 |
| 9 | 柄 | 丙 | 秉 | 饼 | 炳 | | | | | |

# GB码区位示例（续）

| 86区 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | 觥 | 觫 | 觯 | 訾 | 謦 | 靓 | 雩 | 雳 | 雯 |
| 1 | 霆 | 霁 | 霈 | 霏 | 霎 | 霪 | 霭 | 霰 | 霾 | 龀 |
| 2 | 龃 | 龅 | 龆 | 龇 | 龈 | 龋 | 龌 | 龙 | 龚 | 龛 |
| 3 | 龝 | 佳 | 隼 | 隽 | 雎 | 雒 | 雟 | 雠 | 銎 | 銮 |
| 4 | 鋈 | 錾 | 鍪 | 鏊 | 鎏 | 鐾 | 鑫 | 鱿 | 鲂 | 鲅 |
| 5 | 鲆 | 鲇 | 鲈 | 稣 | 鲋 | 鲎 | 鲍 | 鲑 | 鲒 | 鲔 |
| 6 | 鲕 | 鲚 | 鲛 | 鲞 | 鲟 | 鲠 | 鲡 | 鲢 | 鲣 | 鲋 |
| 7 | 鲦 | 鲧 | 鲨 | 鲩 | 鲫 | 鲭 | 鲮 | 鲰 | 鲱 | 鲲 |
| 8 | 鲳 | 鲴 | 鲵 | 鲶 | 鲷 | 鲺 | 鲻 | 鲼 | 鲽 | 鳄 |
| 9 | 鳅 | 鳆 | 鳇 | 鳊 | 鳋 | | | | | |

| 87区 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | 鳌 | 鳍 | 鳎 | 鳏 | 鳐 | 鳓 | 鳔 | 鳕 | 鳗 |
| 1 | 鳘 | 鳙 | 鳜 | 鳝 | 鳟 | 鳢 | 靼 | 鞅 | 鞑 | 鞒 |
| 2 | 鞔 | 鞯 | 鞫 | 鞣 | 鞲 | 鞴 | 骱 | 骰 | 骷 | 鹘 |
| 3 | 骺 | 骼 | 髁 | 髀 | 髅 | 髂 | 髋 | 髌 | 髑 | 魅 |
| 4 | 魃 | 魇 | 魉 | 鱼 | 鱽 | 鱾 | 鱿 | 猃 | 赝 | 飨 |
| 5 | 飨 | 餍 | 鬯 | 鬲 | 鬶 | 鬷 | 鬻 | 鬽 | 鬾 | 魇 |
| 6 | 鬻 | 鬃 | 鬓 | 鬟 | 鬣 | 麽 | 麾 | 縻 | 麂 | 麇 |
| 7 | 麈 | 麋 | 麒 | 麓 | 麝 | 麟 | 黛 | 黜 | 黝 | 黠 |
| 8 | 黟 | 黢 | 黩 | 黧 | 黥 | 黪 | 黯 | 鼢 | 鼬 | 鼯 |
| 9 | 鼹 | 鼷 | 鼽 | 鼾 | 齄 | | | | | |

# 汉字编码——机内码

- 国标码的前后字节的最高位为0，与ASCII码发生冲突
  - “保”字，国标码为31H和23H（区位码为1103H)
  - 西文字符“1”和“#”的ASCII码也分别为31H和23H。

- 现假如内存中有两个字节为31H和23H，这到底是一个汉字，还是两个西文字符“1”和“#”？于是就出现了二义性。

# 汉字编码——机内码

- 机内码（汉字内码）——计算机中中文字符的表示

由于国标码不能直接存储在计算机内，为方便计算机内部处理和存储汉字，又区别于ASCII码，将国标码中的每个字节在最高位改设为1，这样就形成了在计算机内部用来进行汉字的存储、运算的编码叫机内码（或汉字内码，或内码）。

内码既与国标码有简单的对应关系，易于转换，又与ASCII码有明显的区别，且有统一的标准（内码是惟一的）。

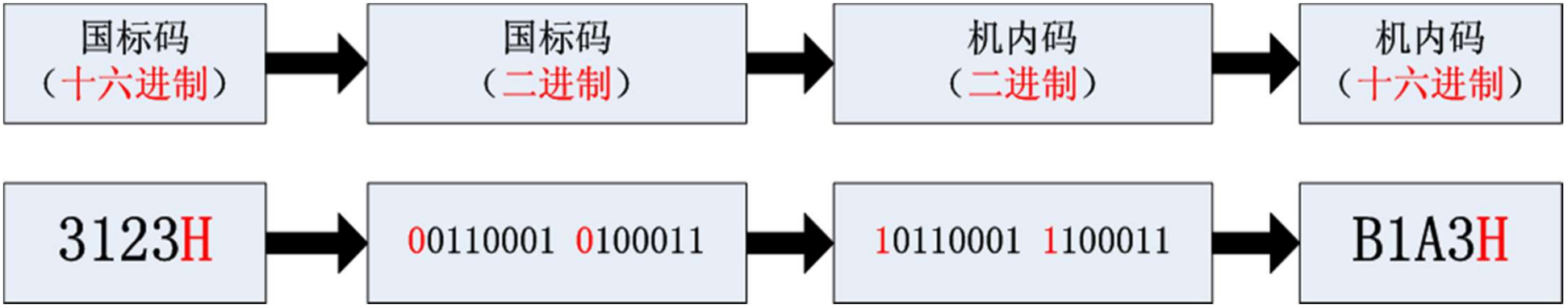
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | x | x | x | x | x | x | x | 1 | x | x | x | x | x | x | x |

# 汉字编码——机内码

- 机内码与国标码的变换：

  国标码的每个字节都加上128(80H)，即将每个字节的最高位由0改1，其余7位不变。或将区码和位码各加A0H(十进制160）。

  内码=区位码+A0A0H =国标码+8080H




如：“保”字的国标码为3123H，前字节为00110001B，后字节为00100011B，高位分别改1为10110001B和10100011B，即为B1A3H，因此，“保”字的机内码就是B1A3H。

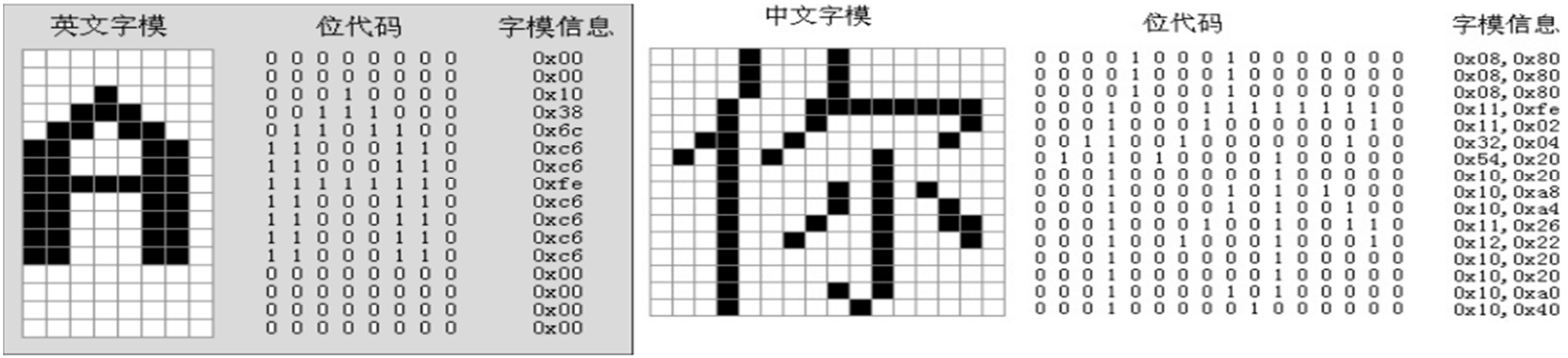
# 汉字编码——输入码

- 无论是区位码或国标码都不利于输入汉字，为方便汉字的输入而制定的汉字编码，称为汉字输入码，又称“外码”，即各种输入法编码方案。
- 常见的输入法有以下几类：
  - 按汉字的排列顺序形成的编码（流水码）：如区位码；
    按汉字的读音形成的编码（音码）：如全拼、简拼、双拼等；
  - 按汉字的字形形成的编码（形码）：如五笔字型、郑码等；
  - 按汉字的音、形结合形成的编码（音形码）：如自然码、智能ABC。

**输入码在计算机中必须转换成机内码，才能进行存储和处理。**

# 汉字编码——字形码

- 为了将汉字在显示器或打印机上输出，把汉字按图形符号设计成点阵图，就得到了相应的点阵代码——字形码（输出码、字模）。




- 全部汉字字形码的集合叫汉字字库
  - 软字库以文件的形式存放在硬盘上，现多用这种方式
  - 硬字库则将字库固化在一个单独的存储芯片中，再和其它必要的器件组成接口卡，插接在计算机上，通常称为汉卡

# 汉字编码——字形码

- 用于显示的字库叫显示字库。
  - 显示一个汉字一般采用16×16点阵或24×24点阵或48×48点阵。已知汉字点阵的大小，可以计算出存储一个汉字所需占用的字节空间。

例：用16×16点阵表示一个汉字，就是将每个汉字用16行，每行16个点表示，一个点需要1位二进制代码，16个点需用16位二进制代码（即2个字节），共16行，所以需要16行×2字节/行=32字节，即16×16点阵表示一个汉字，字形码需用32字节。

字节数=点阵行数×点阵列数/8

- 用于打印的字库叫打印字库，其中的汉字比显示字库多，而且工作时也不像显示字库需调入内存。

# 汉字编码——其他字符集

- BIG5 字符集

  又称大五码或五大码，1984年由台湾财团法人信息工业策进会和五间软件公司宏碁 (Acer)、神通 (MiTAC)、佳佳、零壹 (Zero One)、大众 (FIC)创立，故称大五码。

- GB 18030

  GB 18030的全称是GB18030-2000《信息交换用汉字编码字符集基本集的扩充》，是我国于2000年3月17日发布的新的汉字编码国家标准。2001年8月31日后在中国市场上发布的软件必须符合本标准。

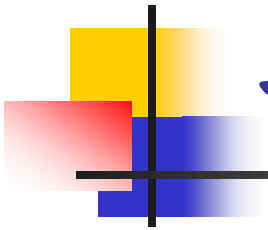
  超过150万个编码位，收录了27484个汉字，覆盖中文、日文、朝鲜语和中国少数民族文字。满足中国大陆、香港、台湾、日本和韩国等东亚地区信息交换多文种、大字量、多用途、统一编码格式的要求。与Unicode 3.0版本兼容，填补Unicode扩展字符字汇“统一汉字扩展A”的内容。并且与以前的国家字符编码标准（GB2312，GB13000.1）兼容。

# 1.4 布尔运算

- 布尔运算(逻辑运算):基于真(true)假(false)的运算:
  - AND 与运算，记为^ 或 ·
  - OR 或运算，记为 v 或+
  - NOT 非运算，记为¬ 或~ 或 `
  - XOR 异或运算，记为 ⊕
- 按位进行运算
- C中按位运算符： & | ^ ~，与、或、异或、非(取反)

思考： $P_{17}$ 28、29 $P_{20}$ 8、 9

- 将一个数中的某几位取反？
- 将一个数的每位置1？
- 取出一个数中的某几位？



# 作业

$P_{16}$ 1.3.7：11、13、17、18、19、23

$P_{20}$ 1.4.2