数据表示
- 非数值数据：C语言char类型
- 数值数据：
  - 整数 Integers：C语言 short, int, long型
  - 非整数(Real/Non-integers)：C语言 float, double型

※汇编语言无数据类型 ⇒ 汇编语言直接面向硬件电路，电路运算对象为二进制数据；其本身没有类型属性，与数据类型相关是对该数据执行的操作

一、数值数据表示法

1、数据编码：①支持数据类型 ②数值范围，数值精度 ③数值存储与处理所需的硬件代价 ④可移植性

2、不同进制编码特点：①十进制：状态过多，运算组合状态过多 ② 二进制：符号数最少，"0" "1" 物理上容易实现；可以表示任何对象(字符、数值、逻辑值)；运算规则简单

少量简单的基本符号表示大量复杂的信息；单位信息所含状态简单，电路简单，运算处理方便

$N=\sum_{i=-k}^{m} D_i \times r^i$

$2^5=32$ $2^6=64$ $2^7=128$ $2^8=256$ $2^9=512$
$2^{10}=1024$(1Kilo) $2^{11}=2048$ $2^{12}=4096$
$2^{13}=8192$ $2^{14}=16384$ $2^{15}=32768$
$2^{16}=65536$ $2^{20}=1048576$ (1Mega)
$2^{30}=1073741824$ (1Giga) $2^{40}=1$Tera
更大的单位是多少？
$2^{50}=1$ Peta $2^{60}=1$ Exa $2^{70}=1$ Zetta $2^{80}=1$ Yotta

- 130=128+2=10000010
- 65539= 65536+3 =1 0000 0000 0000 0011
- 2010 = 2047-37=11111111111-32-4-1
- 11111110111 = 2^{12}-1-8
- 17/128=10001/10000000=0.0010001

3.定点数与机器表示

① 无符号数：所有位数均为数值位
② 有符号数：增加符号位

真值：用+/-表示正负的某进制数称为符号数的真值
机器码(机器内部使用)：将符号和数值一起编码表示的二进制数称为机器码

a. 原码表示法：高位符号为符号位 0为正，1为负 ①直观易懂 ②±0两个零 ③ +/-运算方式不统一 — { 设计加减运算器；判断参与运算数符号位 相同与否；a-b<0时需调整结果

用途 浮点数尾数用原码表示

定点小数 $[X]_原 = \begin{cases} X, & 0 \le X < 1 \\ 1-X, & -1 < X \le 0 \end{cases}$ 定点整数 $[X]_原 = \begin{cases} X, & 0 \le X < 2^n \\ 2^n - X, & -2^n < X \le 0 \end{cases}$

表示区间：$[-(1-2^{-n}), 1-2^{-n}] \xRightarrow{n\to\infty} (-1,1)$ 表示区间 $[-(2^n-1), 2^n-1] \Rightarrow (-2^n, 2^n)$ 区间对称

b. 反码表示法：负数数值位按位取反(1的补码) ①用逻辑门易实现 ②±0两个零 ③加减统一，但仍需调整 ⇒ 进位时需将符号位进位 加到LSB上 ④有无符号数 加减不统一

不再使用

定点小数 $[X]_反 = \begin{cases} X, & 0 \le X < 1 \\ (2-2^{-n})+X, & -1 < X \le 0 \end{cases}$ 定点整数 $[X]_反 = \begin{cases} X, & 0 \le X < 2^n \\ (2^{n+1}-1)+X, & -2^n < X \le 0 \end{cases}$

表示区间 $[-(1-2^{-n}), 1-2^{-n}] \Rightarrow (-1,1)$ 表示区间 $[-(2^n-1), 2^n-1] \Rightarrow (-2^n, 2^n)$ 区间对称

c. 补码表示法：将负数模取符号位进位位权值(模2的补码) ①零表示方法唯一 ②符号位可参与运算 ③减法可化为加法运算电路统一 ④负数多一个

表示负数可利用模性质转换成正数，可将原码运算中减法变为加法运算

- 正值直接取其原来的二进制码，符号位为0
- 负值则逐位取反，末位加1，符号位为1
- 求负：连符号位一起取反加1
- $[-1010\,1010]_补$
- = 1 01010101+1=1 01010110
- $[-0.010101]_补$=1.101011
- 扫描法
  - 从最右侧开始扫描找到第一个1，该数位左侧所有数据位取反，其他数据位不变

定点小数 $[X]_补 = \begin{cases} X, & 0 \le X < 1 \\ 2+X, & -1 \le X < 0 \end{cases}$ 模2 $=[X]_反+2^{-n}$

定点整数 $[X]_补 = \begin{cases} X, & 0 \le X < 2^n \\ 2^{n+1}+X, & -2^n \le X < 0 \end{cases}$ 模2^{n+1} $=[X]_反+1$

$[-1.0]_补=[-1.000]_补=1.000$ (0.1111+0.0001)，$[-1000000]_补=0111111+1=1000000$ -128 = 256+(-128) = 80H

补码符号位讨论
右移：$[X]_补 \Rightarrow [\frac{X}{2^n}]_补$ 算术右移
原符号位不变，符号位与数值位向右移n位 符号向右扩展
$[X]_补=1010 \Rightarrow [\frac{X}{2}]_补=11001$

左移：$[X]_补 \Rightarrow [2^n \cdot X]_补$
左移，末位补0，若符号位变化则溢出

※位数不同的补码相加减，在位数较小的整数左补符号位数码至对齐 符号位扩展不改变补码真值

※有符号/无符号数加法运算统一 ※加法和减法运算统一

$[X+Y]_补=[X]_补+[Y]_补$，$[X-Y]_补=[X]_补+[-Y]_补$，$[Y]_补=[-[-Y]_补]_补$，$[-Y]_补=![Y]_补+1$ 运算简单

减数转为相反数补码
取反+1 用于求补码真值

※双符号位补码(变形补码)

定点小数 $[X]_补 = \begin{cases} X, & 0 \le X < 1 \\ 4+X, & -1 \le X < 0 \end{cases}$ 定点整数 $[X]_补 = \begin{cases} X, & 0 \le X < 2^n \\ 2^{n+2}+X, & -2^n \le X < 0 \end{cases}$ ①增加符号位不会改变补码真值 ②正确的变形补码两个符号位相同

表示范围 $[-1, 1-2^{-n}] \Rightarrow [-1,1)$ 表示范围 $[-2^n, 2^n-1] \Rightarrow [-2^n, 2^n)$ ※少一个0，多一个 $-2^n/-1$ 非对称区间

※当计算结果发生溢出时，符号位会产生异常

d、移码表示法

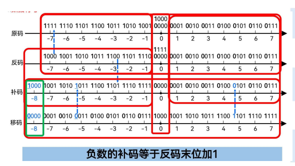
用途：仅用于表示整数，用于表示浮点数的阶码

$[X]_移=2^n+X$，$-2^n \le X < 2^n$ ⇒ 保持数据原有大小顺序

※与补码符号位相异，数据位相同

+1010 ⇒ 0,1010（补） ⇒ 11010（移）
-1010 ⇒ 10110 ⇒ 00110

对一字节八位移码 $[-128]_移=0$，$[127]_移=0FFH$



	$-2^n < x \le 0$	$-1 < x \le 0$	$x \ge 0$
原码	$2^n - x$	$1-x$	x
反码	$2^{n+1}-1+x$	$2-2^{-n}+x$	x
补码	$2^{n+1}+x$	$2+x$	x
移码	2^n+x	无	2^n+x

	定点整数		定点小数	
原码反码	$[1-2^n, 2^n-1]$	$(-2^n, 2^n)$	$[-2^{-n}-1, 1-2^{-n}]$	(-1, 1)
补码	$[-2^n, 2^n-1]$	$[-2^n, 2^n)$	$[-1, 1-2^{-n}]$	[-1, 1)
移码	$[-2^n, 2^n-1]$	$[-2^n, 2^n)$	小数无移码	

※内存中补码 不同数据类型对内存数据进行不同解释 INT 4字节 32位 CHAR 1字节 8位

- 无符号整数
  - unsigned char，unsigned short，unsigned int
  - 一般用于地址运算，编号表示
  - 指针是什么数据类型？
- 有符号整数
  - char short int long
  - 采用补码表示
- 无符号整数的最大值大于位数相同的带符号整数的最大值
  - 8位无符号整数最大是255（1111 1111）
  - 8位带符号整数最大为127（0111 1111）

```
#define INT8_MIN  (-128)
#define INT16_MIN (-32768)
#define INT32_MIN (-2147483647 - 1)
#define INT64_MIN (-9223372036854775807LL - 1)

#define INT8_MAX  127
#define INT16_MAX 32767
#define INT32_MAX 2147483647
#define INT64_MAX 9223372036854775807LL

#define UINT8_MAX  0xff        /* 255U */
#define UINT16_MAX 0xffff      /* 65535U */
#define UINT32_MAX 0xffffffff  /* 4294967295U */
#define UINT64_MAX 0xffffffffffffffff
```

why not

与C语言对常数解释有关 PPT P52

```
main()
{
  char a=127,b=128,c=129,d=-1;    //8bits
  printf("\nchar 127 = %d = %X",a,a);
  printf("\nchar 128 = %d = %X",b,b);
  printf("\nchar 129 = %d = %X",c,c);
  printf("\nchar  -1 = %d = %X",d,d);
  Output32BitHex(&a);
}
```

机器码赋值 真值输出 机器码16进制输出 内存输出

```
char 127 =  127 = 7F
char 128 = -128 = FFFFFF80
char 129 = -127 = FFFFFF81
char  -1 =   -1 = FFFFFFFF
8081FF7F
```

为什么不是FF81807F?

随机存储而不按定义顺序在地址中找存储

```
main()
{
  int x=-1;
  unsigned u = 2147483648;
  printf ("x = %u = %X = %d\n",x,x,x);
  printf ("u = %u = %X = %d\n",u,u,u);
  return;
}
```

真值赋值 真值输出 机器码输出

```
x = 4294967295 = FFFFFFFF = -1
u = 2147483648 = 80000000 = -2147483648
```

4. 浮点数 $N=R^e \times m = 2^E \times M = 2^{\pm e} \times (\pm 0.m)$

定点整数 ↓ 定点小数

E_0	$E_1E_2 \cdots E_m$	M_0	$M_1M_2 \cdots M_n$
阶符	阶值	尾符	尾数

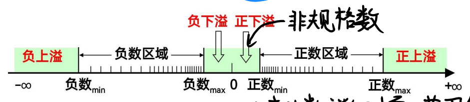
→ 位宽决定浮点数个数
→ 位宽一定，阶码位宽个 ⇒ 尾码位宽↓ ⇒ 浮点数稀疏个，范围个

{ 阶码长，表示范围大；尾码长，表示精度高

※规范化浮点数 ⇒ 尾数最高有效位为1的浮点数称为规格化数 两种规格化数 { 1.XXXX；0.1XXXX 隐藏位 ⇒ 不存储

①保障数据表示的唯一性，方便交换数据
②简化浮点运算算法
③提升了表示精度，尾数去掉了左侧無的0

可表示数字个数：2^{1+E+M} 移码 尾码



※表达数字数目不变，范围个，值越大，越稀疏

- 浮点数在数轴上的分布不均匀，越大越稀疏
- E→+∞ 产生正上溢或者负上溢
- E→-∞ 产生正下溢或者负下溢
- 机器字长一定时，阶码越长，表示范围越大
- 阶码一定，尾码越长，精度越高

32浮点数 float：S(1bit) | E(23~30共8bit) | M(0~22共23bit)
64浮点数 double：S(1bit) | E(52~62共11bit) | M(0~51共52bit)

float 规格化：max $E_{max}=254$ $1.111\cdots1 \times 2^{254-127} = 2^{127} \times (2-2^{-23})$ 约3.4×10^{38}；min $E_{min}=1$ $1 \times 2^{-126} = 2^{-126}$

double 规格化：max $E_{max}=2046$ $1.111\cdots1 \times 2^{2046-1023} = 2^{1023} \times (2-2^{-52})$；min $E_{min}=1$ $1 \times 2^{-1022} = 2^{-1022}$

float 非规格：max $E=0$ $2^{-126} \times (1-2^{-23})$ $M=0.\underbrace{111\cdots1}_{23}$；min $E=0$ $2^{-23} \times 2^{-126} = 2^{-149} \approx 1.4 \times 10^{-45}$ $M=2^{-23}$

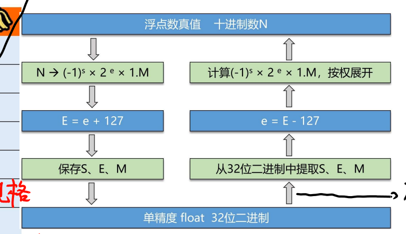
double 非规格：max $E=0$ $2^{-1022} \times (1-2^{-52})$ $M=0.\underbrace{111\cdots111}_{52}$；min $E=0$ $2^{-52} \times 2^{-1022} = 2^{-1074} \approx 4.5 \times 10^{-325}$ $M=2^{-52}$

- 构成：阶码E，尾数M，符号位S
- 阶码和尾码的位数的分配实际是表示范围和精度的折衷
- 阶码移码表示，偏移量127/1023，尾数原码表示，符号位放在第一位 尾码符号位为S，放在第一位
  - 方便定点数逻辑比较浮点数大小

- 规格化数(Normal) 仅保存M
  - 1≤E≤254/2046 $(-1)^s \times 1.M \times 2^{E-127}$ $(-1)^s \times 1.M \times 2^{E-1023}$
- 规格化数尾数最高数字位恒为1，该位为隐藏位，可节约尾数数据位
- 非规格化数(Subnormal)
  - E=0 $(-1)^s \times 0.M \times 2^{-126}$ $(-1)^s \times 0.M \times 2^{-1022}$
- 尾数原码表示，表示区间对称

符号位	阶码	尾数	表示
0/1	255	1xxxx	QNaN 运算错误
0/1	255	非零 0xxxx	sNaN 未初始化的数值
0	255	0	+∞
1	255	0	-∞
0/1	1~254	M	$(-1)^S \times (1.M) \times 2^{(E-127)}$
0/1	0	M (非零)	$(-1)^S \times (0.M) \times 2^{(-126)}$ 非规格
0/1	0	0	+0/-0

标初始化float数值相等



$1.\underbrace{XX\cdots XX}_{23}$ { 00 舍去；11 进位 } 存在舍入

※浮点数不满足结合律 损失精度 $\begin{cases} 2^{-126}+(2^{10}-2^{10}) = 2^{-126} \\ (2^{-126}+2^{10})-2^{10} = 0 \end{cases}$ 吸收 → 2^{10}

$\frac{int}{0}$ 报错；$\frac{float}{0}$ = INF

浮点数%d输出，直接截断小数部分

```
int i; float f; double d;
i==(int)(float)i
f==(float)(int)f
i==(int)double(i)
f==(float)(double)f
d==(float)d
f==-(-f)
(d+f)-d==f
```

- 不成立，int与float部分重叠 → 浮点数不能精确表示一些整数
- 不成立
- 成立，int与double完全重叠
- 成立，float为double子集
- 不成立，同上
- 成立，原码表示，取负不改数值
- 不成立，浮点数在数轴分布不均匀（d 被吸收）

double：52bit尾数
float：23bit尾数
int：32bit
char：8bit

1.1_{10} → IEEE 754 float

十进制转换二进制 1.1_{10} = 1.00011001100110011001100110011…$_2$
规格化 = 1.00011001100110011001100 11 × 2^0
取23位尾数，舍入后会**变大**
8位阶码 = 0 + 127 = 01111111_2

符号	阶码								尾码																						
0	0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1

3F8CCCCD

3.3_{10} → IEEE 754 float

十进制转换二进制 3.3_{10} = 11.0100110011001100110011001 1…$_2$
规格化 = 1.10100110011001100110011 0011 × 2^1
取23位尾数，舍入后会**变小**
8位阶码 = 1 + 127 = 10000000_2

符号	阶码								尾码																						
0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

40533333