2. 数据的机器级表示与处理

2.1. 整数和字符的表示
2.2. 数据的转换和运算
2.3. 浮点数表示和运算

2. 数据的机器级表示与处理

- 学习目标
  - 掌握计算机内部各种数据的编码表示及其运算方法
  - 了解高级语言程序中的各种类型变量对应的表示形式
  - 在高级语言程序的变量、机器数和底层硬件（寄存器、加法器、ALU等）之间建立关联
  - 综合运用所学知识，分析高级语言和机器级语言程序中遇到的各种与数据表示和运算相关的问题，解释执行结果

2. 数据的机器级表示与处理

2.1. 整数和字符的表示
- 计算机中常用计数制
- 整数的表示（无符号整数、带符号整数）
- 字符的表示

2.2. 数据的转换和运算
- 按位运算/逻辑运算/移位运算
- 位扩展和位截断运算
- 无符号和带符号整数的加减运算
- 无符号和带符号整数的乘除运算
- 变量与常数之间的乘除运算

2.3. 浮点数表示和运算
- 浮点数的表示
- 浮点数的加减乘除运算

围绕C语言中的表示和运算，解释其在底层机器级的存储和实现。

计算机中的数制

- 十进制
  - 编程时使用

    42，$(42)_{10}$，42 D

    ```
    int x = 42;
    ```
- 二进制
  - 冯·诺伊曼结构：以二进制表示指令和数据
  - 计算机内部的信息存储、运算、输入/输出

    42 D=101010 B=$(101010)_2$

二进制

- 每位代码非0即1
- 高位权是低位权的2倍
- 加减运算法则：逢二进一，借一当二

	1	**0**	**1**	**0**	**1**	**0**	B
权	2^5	2^4	2^3	2^2	2^1	2^0	

$= 1\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0$

$= 32 + 0 + 8 + 0 + 2 + 0$

$= 42$

	1	0	1	0	1	0	B
+					1	0	B
=	1	0	1	1	0	0	B

计算机中的数制

- 十进制：符合人类直觉
- 二进制：方便计算机系统设计实现　书写阅读不便
- 十六进制
  - 方便二进制数的书写阅读
  - 数码有0，1，2，…，9，A，B，C，D，E，F
  - 基数为16，高位权是低位权的16倍
  - 加减运算法则：逢十六进一，借一当十六

42 D=2A H=$(2A)_{16}$=0x2A

十六进制

- 十六进制与二进制的关系
  - 每4位二进制用1位十六进制表示

```
    10 1010 B
= 0010 1010
=    2    A H
```

4 位二进制数	等值的1位十六进制数	4 位二进制数	等值的一位十六进制数
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

练习：十六进制

1. 写出二进制数 10 1001 1010 1111 对应的十六进制数。
2. 计算十六进制加法 39 H + 7A H = ?
3. 计算十六进制减法 45 H - 26 H = ?

```
 10 1001 1010 1111 B
= 2    9    A    F H
```

```
   39 H
+ 7A H
= B3 H
```

```
   45 H
- 26 H
= 1F H
```

数制转换

- 二/十六进制数→十进制数
  - 算法：每位的代码和该位的权值相乘，再求累加和

1101.11 B = ? D

$= 1\times2^{3}+1\times2^{2}+0\times2^{1}+1\times2^{0}+1\times2^{-1}+1\times2^{-2}$

= 8 +4 +0 +1 +0.5 +0.25

= 13.75 D

1F.8 H = ? D

$= 1\times16^{1}+15\times16^{0}+8\times16^{-1}$

= 16 +15 +0.5

= 31.5 D

数制转换

- 二进制→十六进制
  - 算法：四位二进制数为一组，每组用等值的十六进制代换

```
101011.11 B = ? H
            = 0010,1011.1100 B = 2B.C H
```

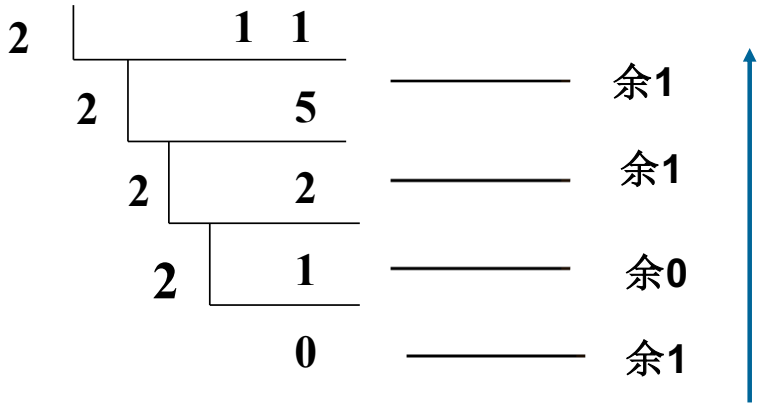
- 十六进制→二进制
  - 算法：一位十六进制数用等值的四位二进制数代换

```
17E.58 H = ? B
         = 0001,0111,1110.0101,1000 B
         =    1,0111,1110.0101,1 B
```

数制转换

- 十进制→二进制
  1. 十进制整数→二进制
     - 算法：除2取整，直到商为0为止，倒排余数

11 D = ? B



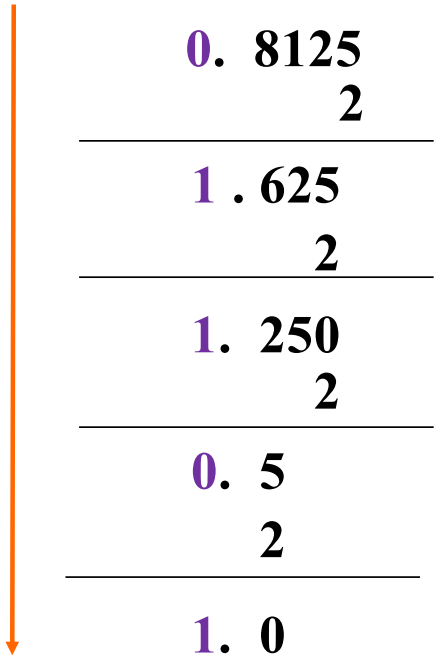
11 D = 1011 B

数制转换

- 十进制→二进制
  - 2. 十进制纯小数→二进制
    - 算法：小数部分乘2，取出整数部分，直到小数部分为0时止，顺排

0.8125 D = ? B

0.8125 D = 0.1101 B



数制转换

- 十进制→二进制
  3. 十进制带小数→二进制
     - 算法：整数、小数部分分别计算，再合并

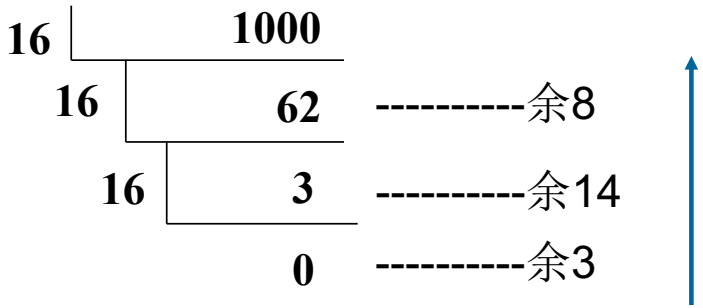
```
11.8125 D = ? B

    11 D = 1011 B
    0.8125 D = 0.1101 B
    11.8125 D = 1011.1101 B
```

数制转换

- 十进制→十六进制
  - 算法：除16取整，直到商为0为止，倒排余数

1000 D = ? H



1000 D = 3E8 H

数制转换

- 二/十六进制数→十进制数
  - 算法：每位的代码和该位的权值相乘，再求累加和
- 二进制→十六进制
  - 算法：四位二进制数为一组，每组用等值的十六进制代换
- 十六进制→二进制
  - 算法：一位十六进制数用等值的四位二进制数代换
- 十进制→二进制
  1. 十进制整数→二进制
     - 算法：除2取整，直到商为0为止，倒排余数
  2. 十进制纯小数→二进制
     - 算法：小数部分乘2，取出整数部分，直到小数部分为0时止，顺排
  3. 十进制带小数→二进制
     - 算法：整数、小数部分分别计算，再合并
- 十进制→十六进制
  - 算法：除16取整，直到商为0为止，倒排余数

2. 数据的机器级表示与处理

计算机中数据的编码

- 计算机如何表示
  - 有符号整数
  - 无符号整数
  - 字符

计算机中数据的编码

- 计算机如何表示
  - 有符号整数
  - 无符号整数

  - 基本C数据类型的典型大小（单位：字节）
  - 分配的字节数受程序是如何编译的影响而变化

指针变量的大小和指向的数据类型无关，只和系统架构和编译方式有关。

C声明		字节数	
有符号	无符号	32位	64位
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned int	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

计算机中数据的编码

- 计算机如何表示
  - 字符
    - 美国标准信息交换码（ASCII）编码
    - 七位二进制编码表示一个字符
    - 共128（2^7）个字符

ASCII(0)=011 0000=48

ASCII(e)=110 0101=101

ASCII 字符编码

低 位	高 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

计算机中数据的编码

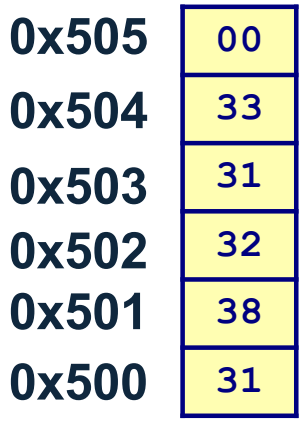
- 计算机如何表示
  - 字符
    - 美国标准信息交换码（ASCII）编码
    - 七位二进制编码表示一个字符
    - 共128（2^7）个字符
  - Unicode统一字符集
    - 32位
    - 汉字
      - 汉字输入码：用户通过键盘输入汉字时使用的编码，用于将汉字转换为计算机可识别的形式。
      - 汉字内码：计算机内部存储和处理汉字时使用的编码，确保汉字在系统中的唯一性。
      - 汉字字模码：描述汉字字形点阵信息的编码，用于显示或打印汉字。

计算机中数据的编码

- 计算机如何表示
  - 字符
    - 美国标准信息交换码（ASCII）编码
    - 七位二进制编码表示一个字符
    - 共128（2^7）个字符
  - 字符串
    - 每个字符的ASCII码依次存放
    - 字符NUL（000 0000）表示串终止

ASCII(0)=011 0000 B=48 D=30 H

```
char s[6] = “18213”;
```



无符号整数的编码

- 全部是正数运算且不出现负值结果
  - 地址运算，编号表示，…

```
unsigned int x = 1000;
```

32位，0000 0000 0000 0000 0000 0011 1110 1000 (0x000003E8)

```
unsigned short x = 1000;
```

16位，0000 0011 1110 1000 (0x03E8)

无符号整数的编码

- n位无符号整数可表示的值范围 $0 \sim 2^n-1$

C数据类型	位数	最小值	最大值
unsigned char	8	0	255
unsigned short	16	0	65535
unsigned int	32	0	4294967295
unsigned long	64	0	18,446,744,073,709,551,615

多字节数据的存放规则

- 多字节对象被存储为连续的字节序列
- 对象的地址为所使用字节中最小的地址

```
unsigned int x = 1000;                                    &x=0x500
32位，0000 0000 0000 0000 0000 0011 1110 1000 (0x000003E8)
```

大多数Intel兼容机

小端方式

0x503	00
0x502	00
0x501	03
0x500	E8

低位字节存放在低地址单元
高位字节存放在相邻的高地址单元

increasing byte address

0x503	E8
0x502	03
0x501	00
0x500	00

大端方式

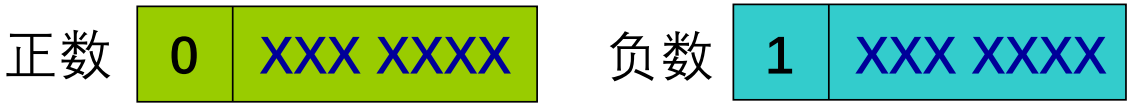
高位字节存放在低地址单元
低位字节存放在相邻的高地址单元

练习

- 设Intel机器中，有一个int型变量x的地址为0xFFFFC000，x=0x12345678，则内存单元0xFFFFC001中存放的内容的十六进制表示为____？

有符号整数的编码

- 计算机系统以二进制（0/1）表示数据，符号也需要用0/1表示
- 真值
  - 数据的实际数学值或逻辑值，是人类理解的自然形式。
  - 用“+”表示正数，用“–”表示负数
    - +42，-42
- 机器数
  - 真值在计算机内部的二进制表示形式，是计算机实际存储和处理的数据。
  - 符号数值化
    - 用0表示“+”，用1表示“-”，连同符号位在一起作为一个机器数
      - 8位机器数



有符号机器数的不同表示方法

- 原码
  - 原码表示的有符号数，最高位为符号位，数值位是该数的绝对值。
  - 8位机器数
    - 最高位是符号位，后7位是数值位
    - 42 D=2A H=101010 B

    +42的原码机器数为：00101010

    -42的原码机器数为：10101010

有符号机器数的不同表示方法

- 反码
  - 正数的反码同原码，符号位和数值位都不动
  - 负数的反码为 符号位不动，数值位按位取反
  - 8位机器数

    +42的反码机器数为：~~00101010（原码）~~ 00101010

    -42的反码机器数为：~~10101010（原码）~~ 11010101

有符号机器数的不同表示方法

- 补码
  - 正数的补码同原码和反码，符号位和数值位都不动
  - 负数的补码为 符号位不动，数值位按位取反后（反码）末位加1
  - 8位机器数

    +42的补码机器数为：~~0~~0101010（原码） ~~0~~0101010（反码） 00101010

    -42的补码机器数为：~~1~~0101010（原码） ~~1~~1010101（反码） 11010110

结论1： 一个负数的补码等于对应正数补码的“各位取反、末位加1”

有符号机器数的不同表示方法

- 负数补码的另一种计算方法
  - **结论2：一个负数的补码等于模（2^n）减该负数的绝对值**
    - 补码源于数学上的（相对于一个模下的）补数概念
  - 8位机器数

    -42的补码机器数为：~~10101010（原码）~~ ~~11010101（反码）~~ 11010110

```
 1 0000 0000 模
-  0010 1010 绝对值
=  1101 0110 补码
```

补码的设计

- 机器是有位长的，天然适合模运算
  - 类比：钟表是一个模为 12 的系统
    - 从 10 点拨到 6 点有两种方法

    1. 10 – 4 = 6
    2. 10 + 8 = 18 ≡ 6 (mod 12)
  - 推论：**数a减去（小于模的）数b，等价于数a加上数b的补码**
- 补码的优势
  - 统一加法和减法运算
    - 只需要加法器，运算电路更加简单
  - 数值转换更加便利
    - 符号位可以直接参与运算

练习：补码

- 设机器数有8位，求123和-123的补码表示。

解：

123 = 0111 1011 B

[123]补 = 0111 1011

[-123]反 = 1000 0100，[-123]补 = 1000 0101

[-123]补 = 1 0000 0000 (2^8) - 0111 1011 = 1000 0101

练习：补码

- 求-123的16位补码的十六进制表示。

解：

123 = 0000 0000 0111 1011 B

[-123]反 = 1111 1111 1000 0100

[-123]补 = 1111 1111 1000 0101 = FF85 H

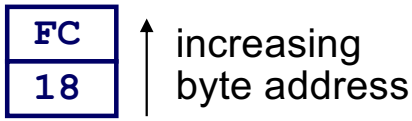
练习

- 给出以下两个C语言变量声明语句中变量的内存表示（Intel机器）。

```
1. short x = -1000;
2. int x = -1000;
```

解:

```
1. short x = -1000;
16位
1000=0000 0011 1110 1000
[-1000]反=1111 1100 0001 0111
[-1000]补=1111 1100 0001 1000=FC18
```

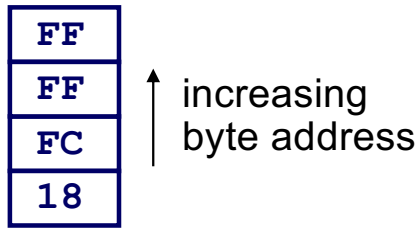


练习

- 给出以下两个C语言变量声明语句中变量的内存表示（Intel机器）。
  1. `short x = -1000;`
  2. `int x = -1000;`

解:

```
int x = -1000;
32位
1000=0000 0000 0000 0000 0000 0011 1110 1000
[-1000]反=1111 1111 1111 1111 1111 1100 0001 0111
[-1000]补=1111 1111 1111 1111 1111 1100 0001 1000=FFFFFC18
```



补码转换为真值

- 设机器数有8位，[X]补=36 H，则X=(？)$_{10}$

解：

[X]补=36 H=0011 0110 B

X正数

[X]原=[X]补=0011 0110 B=54 D

补码转换为真值

- 设机器数有8位，[X]补=96 H，则X=(？)$_{10}$

解：

[X]补=96 H=1001 0110 B

X负数

- 方法一：

[X]补-1=1001 0101 B，[X]原=1110 1010 B=-106 D

- 方法二：

|X|=1 0000 0000-1001 0110=0110 1010

[X]原=1110 1011 B=-106 D

结论1：一个负数的补码等于对应正数补码的“各位取反、末位加1”

结论2：一个负数的补码等于模（2^n）减该负数的绝对值

$[X]补=2^n-|X|$

推论：$|X|=2^n-[X]补$

练习

- 设字长为8位，已知[X]补=E5 H，则X=(?)$_{16}$

解:

[X]补=E5 H=1110 0101 B

X负数

- 方法一:

[X]补-1=1110 0100 B，[X]原=1001 1011 B=-1B H

- 方法二:

|X|=1 0000 0000-1110 0101=0001 1011

[X]原=1001 1011B=-1B H

无符号整数的编码

- n位无符号整数可表示的值范围 $0 \sim 2^n-1$

C数据类型	位数	最小值	最大值
unsigned char	8	0	255
unsigned short	16	0	65535
unsigned int	32	0	4294967295
unsigned long	64	0	18,446,744,073,709,551,615

有符号数的编码

- **n位补码所能表示的真值范围** $-2^{n-1} \sim 2^{n-1}-1$
  - n=8
  - 正数+0=非负数
    - 最大：0 111 1111=2^7-1=127
    - 最小：0 000 0000=0
  - 负数
    - [X]补=1 111 1111，|X|=1 0000 0000-1 111 1111=0000 0001=1，X=-1
    - [X]补=1 111 1110，|X|=1 0000 0000-1 111 1110=0000 0010=2，X=-2
    - …
    - [X]补=1 000 0001，|X|=1 0000 0000-1 000 0001=0111 1111=2^7-1=127，X=-127
    - [X]补=1 000 0000，|X|=1 0000 0000-1 000 0000=1000 0000=2^7=128，X=-128（没有-0）

有符号数的编码

- n位补码所能表示的真值范围

  $-2^{n-1} \sim 2^{n-1}-1$

  - n位原码：$-2^{n-1}-1, \cdots, -1, -0, +0, 1, \cdots, 2^{n-1}-1$
    - 1位符号位，n-1位数值位
  - 补码的优势+1
    - 表示范围更大
      - -2^{n-1}

有符号数的编码

- n位补码所能表示的真值范围：$-2^{n-1} \sim 2^{n-1}-1$

C数据类型	位数	最小值	最大值
char	8	-128	127
short	16	-32768	32767
int	32	-2 147483 648	2147483 647
long	64	-9223 372 036 854 775 808	9223 372 036 854 775 807

C语言中char类型的宽度为1个字节，

可表示一个字符（非数值数据，`char x = ‘8’; 0011 1000`），

也可表示一个8位整数（数值数据，`char x = 8; 0000 1000`）。

特殊数的补码

- 设机器数有n位
- $[-2^{n-1}]$补$=2^n-2^{n-1}=10\cdots0$（n-1个0）
  - 最小负数：[X]补=1 000 0000，|X|=1 0000 0000-1 000 0000=1000 0000=2^7=128，X=-128
- [-1]补$=2^n-0\cdots01=11\cdots1$（n个1）
  - 最大负数：[X]补=1 111 1111，|X|=1 0000 0000-1 111 1111=0000 0001=1，X=-1
- [0]补=[+0]补=[-0]补=0…0（n个0）
  - 最小非负数：0 000 0000=0

补码小结

- 没有负零的补码，或者表述为正零和负零的补码相同
- 正数的原码、反码、补码与真值数相同
- 由于补码表示的机器数更适合运算，所有现代计算机都采用补码表示整数

补码小结

数	字长w			
	8	16	32	64
$UMax_w$	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	255	65 535	4 294 967 295	18 446 744 073 709 551 615
$TMin_w$	0x80	0x8000	0x80000000	0x8000000000000000
	−128	−32 768	−2 147 483 648	−9 223 372 036 854 775 808
$TMax_w$	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	127	32 767	2 147 483 647	9 223 372 036 854 775 807
−1	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	0x00	0x0000	0x00000000	0x0000000000000000

图 2-14　重要的数字。图中给出了数值和十六进制表示

- 无符号整数的真值范围：0 ~ 2^n-1
- 有符号整数的真值范围：-2^{n-1} ~ 2^{n-1}-1
  - 不对称，负数比正数多一个
- -1和U_{Max}有相同的补码位表示：全1
- 真值0具有相同的补码表示：全0