

## 3.1 主存储器

数据存储的基本形式、数据地址的类型及转换

## 3.2 数值数据在计算机内的表示形式

## 3.3 字符数据在机内的表示形式

## 3.4 数据类型转换

## 3.5 浮点数据在机内的表示形式

## 学习重点

数据存储的基本形式

有符号整数和无符号整数的表示与存储

字符串的表示方法与存储

浮点数据的表示方法与存储

地址类型转换 与 数据类型转换



## 3.1 主存储器

### 3.1.1 数据存储的基本形式

内存条：用来存放程序 and 数据的装置

C 语言： char、short、int、double

对应长度： 1个、 2个、 4个、 8个字节

	字节	字	双字	四字
汇编语言：	byte	word	dword	qword



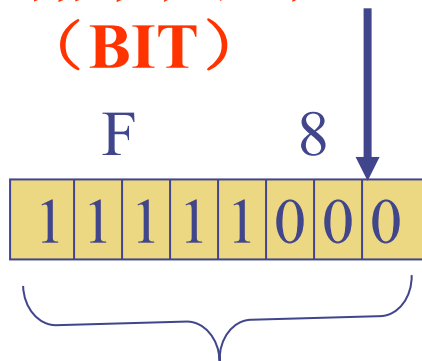
字节是最小的寻址单位

每一个字节都有一个地址

物理地址 (Physical Address, PA) 是唯一的

00012340H	
00012341H	
00012342H	
00012343H	
00012344H	
00012345H	
00012346H	F 8 H
00012347H	
00012348H	
00012349H	
0001234AH	
0001234BH	
0001234CH	
0001234DH	
0001234EH	
0001234FH	
00012350H	

主存的基本存储单位是位 (BIT)



8个位组成一个字节 BYTE

Q: 1M字节内存, 地址编码需要多少二进制位?

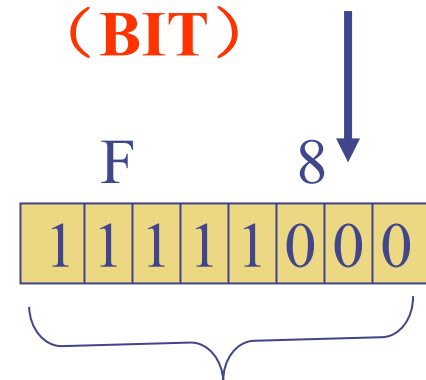
Q: 32位地址对应的内存大小可达到多大?

字节是最小的寻址单位

00012340H	
00012341H	
00012342H	
00012343H	
00012344H	
00012345H	
00012346H	F8H
00012347H	04H
00012348H	56H
00012349H	12H
0001234AH	
0001234BH	
0001234CH	
0001234DH	
0001234EH	
0001234FH	
00012350H	

字地址是这2个字节中低字节的地址

主存的基本存储单位是位 (BIT)



8个位组成一个字节 **BYTE**

} 两个相邻的字节组成一个字 **WORD**

**Q1:**两个黄色的字节组成的字的地址是多少？  
字中的内容又是多少？

**PA:00012346H**  
**DATA: 04F8H**

**Q2:**红色的呢？

**PA:00012347H**  
**DATA: 5604H**

00012340H	
00012341H	
00012342H	
00012343H	
00012344H	
00012345H	
00012346H	F8H
00012347H	04H
00012348H	56H
00012349H	12H
0001234AH	
0001234BH	
0001234CH	
0001234DH	78H
0001234EH	56H
0001234FH	
00012350H	

字数据的存放形式:

低8位在低字节；  
高8位在相邻的高字节中。

**Q3:**将5678H存放到地址为1234D的字单元中。

地址为12346H  
的**双字**是：  
44434241H  
即(00012346H) =  
44434241H

地址为12346H  
的**字**是：  
4241H  
即(00012346H) =  
4241H

地址为12346H  
的**字节**是：  
41H  
即(00012346H) =  
41H

00012340H	
00012341H	
00012342H	
00012343H	
00012344H	
00012345H	
00012346H	41H
00012347H	42H
00012348H	43H
00012349H	44H
0001234AH	
0001234BH	
0001234CH	
0001234DH	
0001234EH	
0001234FH	
00012350H	

**双字**  
四个连续的字节组成

其地址为四个字节中的最低字节的地址。



## 3.1.1 数据存储的基本形式

数据存储方法有两种

- 小端存储 (Little Endian)

- 大端存储 (Big Endian)

- Intel x86 系列采用小端存储方式

- 在小端存储方式中，最低地址字节中存放数据的最低字节，最高地址字节中存放数据的最高字节。按照数据由低字节到高字节的顺序依次存放在从低地址到高地址的单元中。

- 大端存储方式与小端存储方式相反。





## 3.1.1 数据存储的基本形式

```
char  t1[5] = { 48, 49, 127, 129, 0 };  
short t2[5] = { 48, 49, 32767, 32769, 0 };  
int   t3[5] = { 48, 49, 0x12345678, 0x11223344, 0 };
```

内存 1

地址: 0x010FF7B8 列: 自动

0x010FF7B8	30 00 00 00 31 00 00 00	0...1...
0x010FF7C0	78 56 34 12 44 33 22 11	xV4.D3".
0x010FF7C8	00 00 00 00 30 00 31 00	....0.1.
0x010FF7D0	ff 7f 01 80 00 00 9b 00	... €...?.
0x010FF7D8	30 31 7f 81 00 00 00 00	01.?....

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
t3	0x010ff7b8 {48, 49, 305419896, 287454020, 0}	int[5]
t2	0x010ff7cc {48, 49, 32767, -32767, 0}	short[5]
t1	0x010ff7d8 <字符串中的字符无效。>	char[5]

添加要监视



## 3.1.1 数据存储的基本形式

```
char  t1[5] = { 48, 49, 127, 129, 0 };
short t2[5] = { 48, 49, 32767, 32769, 0 };
int   t3[5] = { 48, 49, 0x12345678, 0x11223344, 0 };
int   *pt1 = t1; // 警告：从“char*”到“int*”的类型不兼容
int   q = *pt1;                                     int * pt1 = (int *)t1;
```

内存 1

地址: 0x010FF7D8 列: 自动

0x010FF7D8	30 31 7f 81 00 00 00 00	01. ?....
0x010FF7E0	ab 23 05 2d 04 f8 0f 01	?#. -. ?..

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
pt1	0x010ff7d8 {-2122370768}	int *
q	-2122370768	int
t1	0x010ff7d8 <字符串中的字符无效。>	char[5]
q.x	0x817f3130	int
&pt1	0x010ff7b4 {0x010ff7d8 {-2122370768}}	int **

添加监视





## 3.1.2 数据地址的类型及转换

给定一个地址后，可以根据该地址取一个字节、一个字、一个双字……

取多少字节的数据，取决于地址类型。

以 ... 为字节地址

以 ... 为字地址

以 ... 为双字地址





华中科技大学

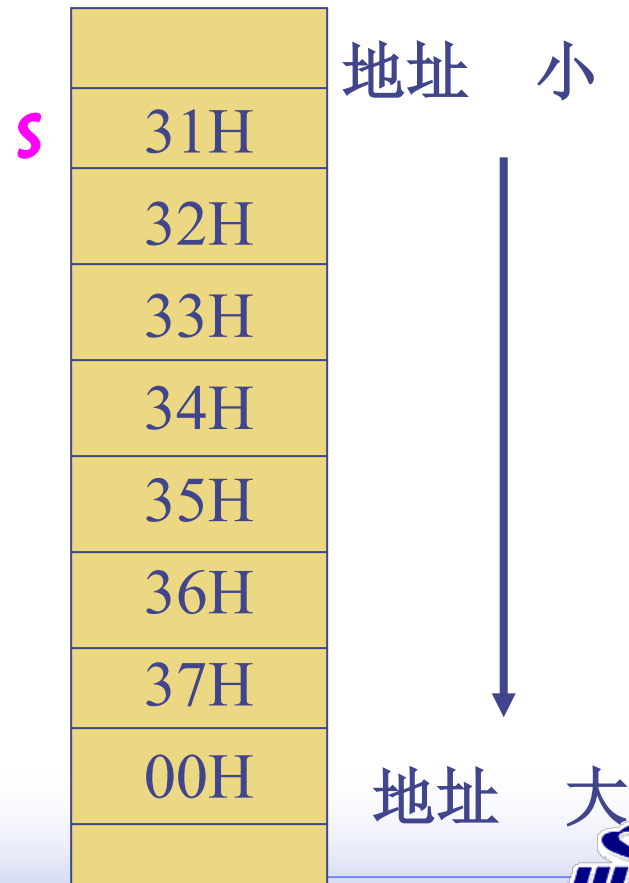
## 3.1.2 数据地址的类型及转换

C:\新书示例\C  
36353433  
3433  
51  
1BA

程序运行结果是什么？为什么？

```
char s[10];  
strcpy(s, "1234567");  
printf("%x \n", *(long *)(s+2));  
printf("%x \n", *(short *)(s+2));  
printf("%d \n", *(char *)(s+2));  
*(int *)(s+1)=16706;  
printf("%s \n",s);
```

关键词：地址类型转换





## 3.1.2 数据地址的类型及转换

```
char s[10];  
strcpy(s, "1234567");
```

内存 1

地址: 0x00AFFA18 列: 自动

0x00AFFA18	31	32	33	34	35	36	37	00	1234567.
0x00AFFA20	06	90	00	01	96	9d	d1	3a	.?...???:

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
▶ s	0x00affa18 "1234567"	char[10]
s[2]	51 '3'	char
*(short *)(s+2)	13363	short
*(short *)(s+2),x	0x3433	short
*(int *)(s+2),x	0x36353433	int

在调试窗口，观察地址类型转换后取数的结果





## 3.1.2 数据地址的类型及转换

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
union test {
    char s[10];
    char c;
    short x;
    int y;
} temp;
int main()
{   strcpy(temp.s, "1234567");
    printf(" %x \n", temp.x);
    printf(" %x \n", temp.y);
    printf(" %d \n", temp.c);
    return 0;
}
```

工程: union\_type

S

31H
32H
33H
34H
35H
36H
37H
00H

小地址



大地址

C:\新书示例\C03 内存

3231  
34333231  
49





## 3.1.2 数据地址的类型及转换

```
union test {  
    char s[10];  
    char c;  
    short x;  
    int y;  
} temp;  
strcpy(temp.s, "1234567");
```

工程: union\_type

监视 1

搜索(Ctrl+E)



搜索深度:

3



A

名称	值	类型
▸ &temp.x	0x0092a138 {union_type.exe!test temp} {12849}	short *
▸ &temp.y	0x0092a138 {union_type.exe!test temp} {875770417}	int *
▸ &temp.c	0x0092a138 "1234567"	char *
▸ temp.s	0x0092a138 "1234567"	char[10]





## 3.1.2 数据地址的类型及转换

### 练习：体会地址类型转换的应用

定义了结构 student，以及结构数组变量 s[3];

```
struct student {  
    char name[8];  
    short age;  
    float score;  
    char remark[200];  
};  
student s[3];  
student new_s[3];
```

编写程序，将 s[3] 中的信息紧凑存放到一个字符数组 message 中，然后从 message 转换到结构数组 new\_s[3] 中。







## 3.1.2 数据地址的类型及转换

### 练习

Microsoft Visual Studio 调试控制台

结构 student的大小 =214

结构数组 s 的大小 =642

张三 20 91.5 2021年获得三好学生称号, 2020年获得优秀干部奖

xuxiang 21 94.7 very good student

wang 22 87.6 none

packed :103

张三 20 91.5 2021年获得三好学生称号, 2020年获得优秀干部奖

xuxiang 21 94.7 very good student

wang 22 87.6 none

要求 s[0].name 为自己的姓名;

可以在给的程序中 补充

**void pack\_student(student\* s)**

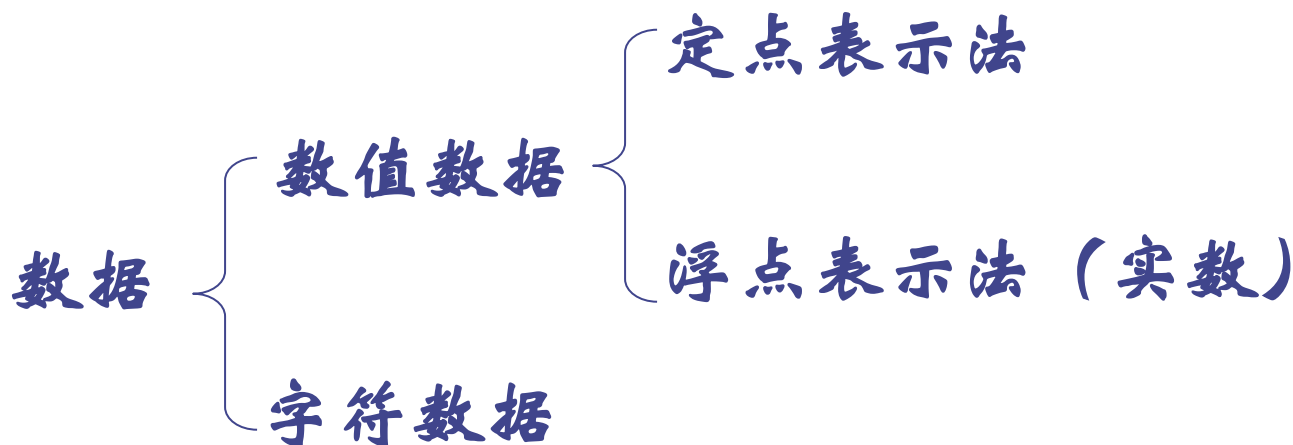
**void restore\_student(student\* s)**



## 3.2 数值数据在计算机内的表示形式



华中科技大学



常用的数值数据为二进制、十进制、十六进制、BCD码。

字符数据：ASCII（美国信息标准交换代码）

GB2312

GBK



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

整数的表示法（小数点固定在第0位的后面）

### 1. 十进制数转换成16进制数

18

30

347

12 H

1E H

15B H

### 2. 将以上数据转换成二进制数

10010 B

11110 B

101011011 B



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

### 3. 有符号数的n位二进制的补码表示

正数的补码是其本身；

负数的补码：先求其相反数的补码，然后对该补码的二进制逐位求反，最后加1。

设  $n=16$ ,  $-69DAH$  的补码表示是多少？

$-69DAH$  的相反数是  $69DAH$ ,  
对应的二进制是  $0110\ 1001\ 1101\ 1010\ B$ ,  
逐位求反  $1001\ 0110\ 0010\ 0101\ B$   
加1后:  $1001\ 0110\ 0010\ 0110\ B$   
 $[-69DAH]_{补} = 9\ 6\ 2\ 6\ H$



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

设  $n=16$ , — 69DA H 的补码表示是多少?

$[-69DAH]_{补} = 9626H$

观察: 二进制数	0110	1001	1101	1010	B
逐位求反	1001	0110	0010	0101	B
它们的和:	1111	1111	1111	1111	B

对16进制数的直接求反方法:

	6	9	D	A	H
+	?	?	?	?	
<hr/>					
	F	F	F	F	H

(9625 H)

设  $n=32$ , - 69DAH 的补码表示是多少?

FFFF9626H

一个二进制数的补码表示中, 其最高位 (即符号位) 向左扩展若干位后, 得到的仍然是该数的补码。



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

设  $n=16$ ,  $-69DAH$  的补码表示是多少?

$$[-69DAH]_{\text{补}} = 9626H$$

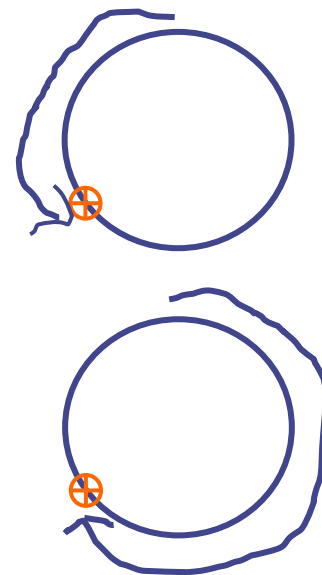
$$\begin{array}{r} 69DAH \\ + \quad ? ? ? ? \quad (9625H) \\ \hline FFFFH \end{array}$$

$$69DAH + 9625H = FFFFH$$

$$69DAH + 9625H + 1 = 10000H = 2^{16}$$

$$69DAH + [-69DAH]_{\text{补}} = 2^n \quad (n=16)$$

$$[-69DAH]_{\text{补}} = 2^n - 69DAH$$



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

8001H 当无符号short数，是 32769

当有符号short数，是 -32767

为什么一个数，能当有符号看，也可以当成无符号数看？

这两个数之间有什么关系？



摄影号电文三图

8 点，也可视为 到 0 点 差 4 个小时

12 点 即 0 点

从 0 出发，顺时针走 8 格

从 0 出发，逆时针走 4 格

对时钟：  $[-4]_{\text{补}} = 12 - 4 = 8$



## 3.2.1 有符号和无符号整数的表示形式

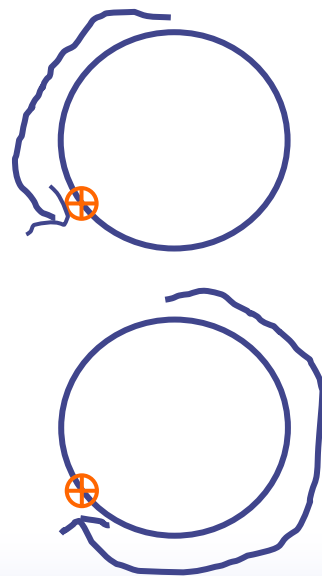


华中科技大学

Q: 设 `char sa[5] = {-100, 156, -255, 1, 'a', 97, 0};`  
`unsigned char usa[5] = {-100, 156, -255, 1, 'a', 97, 0};`  
数组 `sa`, `usa` 中存放的结果是什么? 为什么?

监视 1	
搜索(Ctrl+E)  搜索深度: 3	
名称	值
sa	0x003cf6f4 "潜\x1\x1aa"
[0]	-100 '?'
[1]	-100 '?'
[2]	1 '\x1'
[3]	1 '\x1'
[4]	97 'a'
[5]	97 'a'
[6]	0 '\0'
usa	0x003cf6ec "潜\x1\x1aa"

内存 1	
地址: 0x003CF6EC	列: 自动
0x003CF6EC	9c 9c 01 01 61 61 00 00 ??..aa..
0x003CF6F4	9c 9c 01 01 61 61 00 00 ??..aa..
0x003CF6FC	38 58 46 b6 20 f7 3c 00 8XF? ?<.
0x003CF704	b3 1c 84 00 01 00 00 00 ? ?

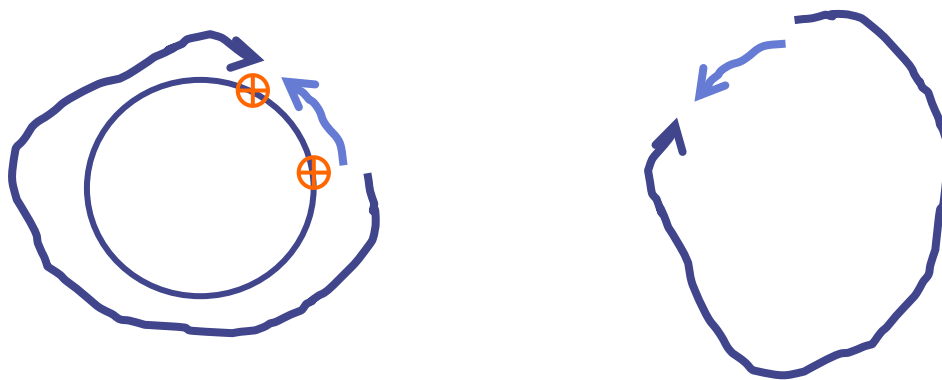




## 3.2.1 有符号和无符号整数的表示形式



华中科技大学



$m - n$  为什么等于  $m + [-n]$  补 ?

$$5 - 2 = 0x05 + 0xFE = \boxed{1} 3$$



## 3.2.1 有符号和无符号整数的表示形式



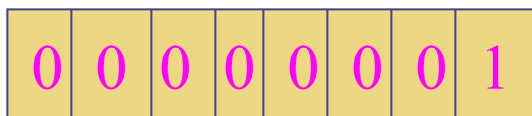
华中科技大学

### ■ 有符号数的表示范围

8位补码: 80H --- 7FH (-128, 127)

16位补码: 8000H --- 7FFFH (-32768, 32767)

正数



1



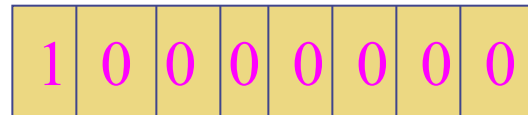
7F

N=8

负数



-1



-128



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

### ■ 有符号数（补码表示）的大小比较

设  $n = 16$ , 试比较:

5678H          vs          7345H ?          <

5678H          vs          8345H ?          >

3271H          vs          0A521H ?          >

0A521H        vs          0B521H ?          <



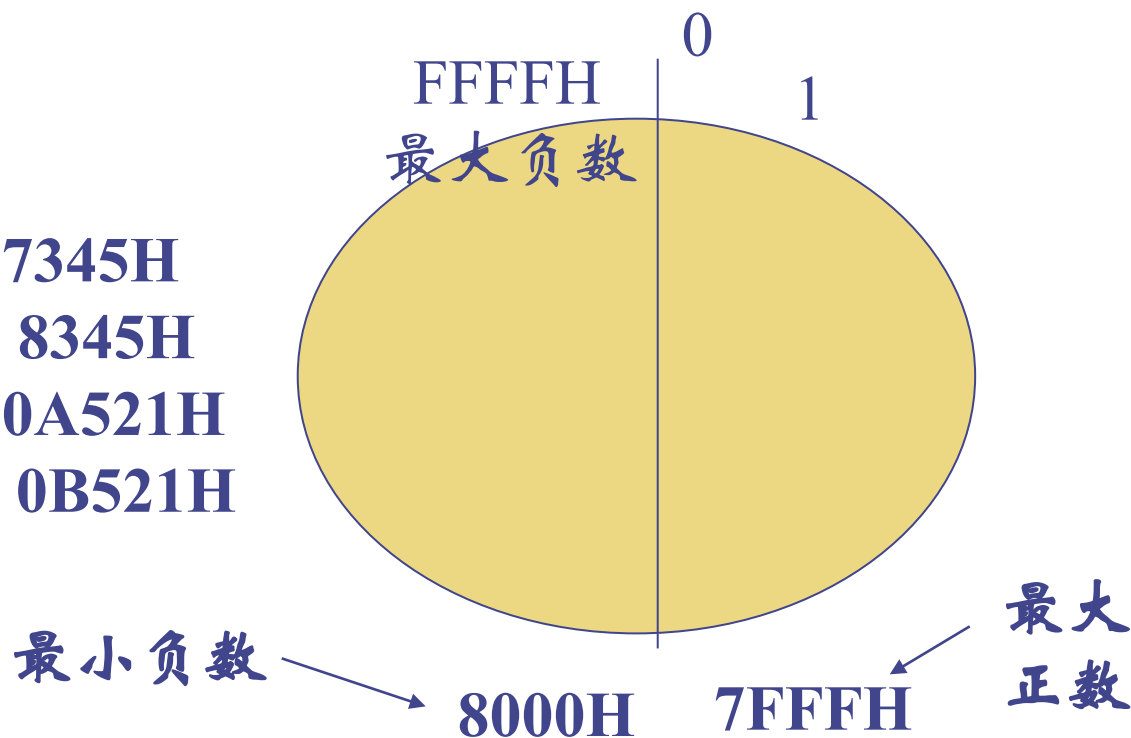
## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

### ■ 有符号数（补码表示）的大小比较

5678H	<	7345H
5678H	>	8345H
3271H	>	0A521H
0A521H	<	0B521H



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

### ■ 无符号数及其表示范围

一个段中的偏移地址，就是一个无符号数。  
最小的无符号数是0。

8位无符号数:      0H --- FFH      (0 ,255)

16位无符号数:    0H --- FFFFH    (0 ,65535)

无符号数的大小比较?



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

设  $n=8$ , 有一个数  $M$ ,

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

**Question:** 若将其看成一个数的补码, 它表示的什么数?  
-1

**Question:** 如果  $M$  作为无符号数, 它表示的是多少?  
255

**Question:** 哪到底将它看成什么数呢?

**Answer:** 这取决于访问该单元的指令。看一个C程序



## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

```
#include <stdio.h>

void main( )
{
    short x;
    unsigned short y;
    x= -1;        // x=65535
    y= -1;        // y=65535
    printf("%d  %d\n", x, y);
}
```

结果是：  
-1 65535

在汇编语言中也有类似于unsigned的约定。

工程： c\_有符号和符号的比较



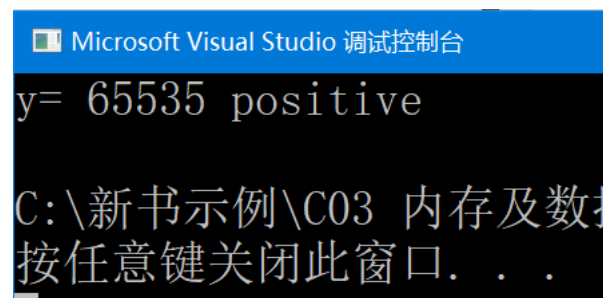
## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

```
#include <stdio.h>
void main( )
{
    short x;
    unsigned short y;
    x = -1; y = -1;
    if (x > 0)
        printf("x= %d positive\n",x);
    if (y > 0)
        printf("y= %d positive\n",y);
}
```

结果？



谁来把一个数当有符号数看，还是无符号数？

工程： c\_有符号和符号的比较



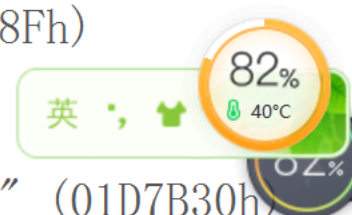


## 3.2.1 有符号和无符号整数的表示形式



华中科技大学

```
    if (x > 0) printf("x= %d positive\n", x);
001D1875 movsx     eax, word ptr [x]
001D1879 test     eax, eax
001D187B jle      __$EncStackInitStart+43h (01D188Fh)
001D187D movsx     eax, word ptr [x]
001D1881 push     eax
001D1882 push     offset string "x= %d positive\n" (01D7B30h)
001D1887 call    _printf (01D10CDh)
001D188C add     esp, 8
    if (y > 0) printf("y= %d positive\n", y);
001D188F movzx     eax, word ptr [y]
001D1893 test     eax, eax
001D1895 jle      __$EncStackInitStart+5Dh (01D18A9h)
001D1897 movzx     eax, word ptr [y]
001D189B push     eax
001D189C push     offset string "y= %d positive\n" (01D7B44h)
001D18A1 call    _printf (01D10CDh)
001D18A6 add     esp, 8
```





## 3.2.2 BCD码

BCD (Binary Coded Decimal): 二进制编码的十进制  
用4位二进制数表示一位十进制数。

1 = 0001 BCD

8 = 1000 BCD

2 = 0010 BCD

9 = 1001 BCD

.....

9781的  
压缩  
BCD码

1000	0001
<hr/>	
1001	0111

0000	0001
0000	1000
<hr/>	
0000	0111
<hr/>	
0000	1001

9781的  
非压缩  
BCD码



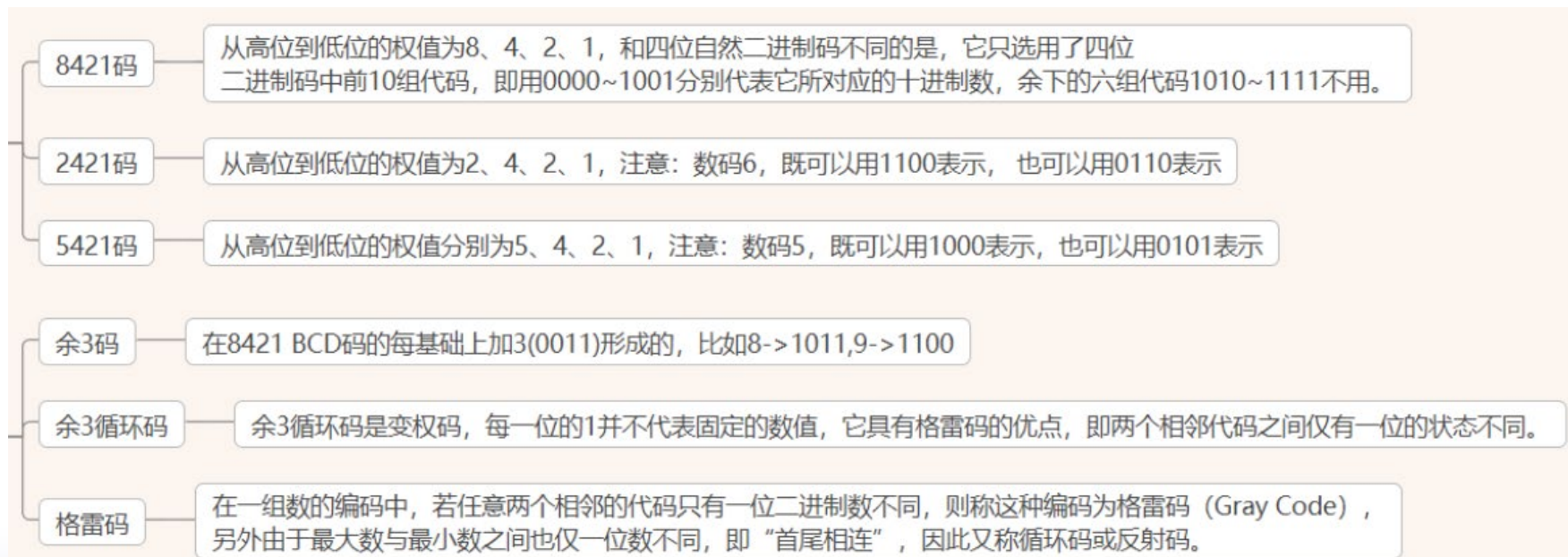


# BCD码 扩展知识

BCD码可分为两类：有权BCD码和无权BCD码

有权BCD码：8421码、2421码、5421码

无权BCD码：余3码、余3循环码、格雷码





# 格雷码 Gray Code

十进制数	自然二进制数	格雷码
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

自然二进制码→格雷码:

- 自然码的最左(最高)位不变;
- 从自然码的次高位开始, 每一位与其左边一位异或 (XOR), 作为对应格雷码在该位的值。
- 相邻的数字编码, 只有一个二进制位发生变化
- 常用于模拟—数字转换中



# 余3码



华中科技大学



十进制数	8421码	余3码
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

以8421码（即BCD）为基础，  
每位十进制数BCD码，加上0011得到。





# 3.3 字符数据在机内的表示形式

## ASCII码字符表

		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
↓		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0				0		P		p								
1	1				1	A	Q	a	q								
2	2				2	B	R	b	r								
3	3				3	C	S	c	s								
4	4			\$	4	D	T	d	t								
5	5				5	E	U	e	u								
6	6				6	F	V	f	v								
7	7				7	G	W	g	w								
8	8				8	H	X	h	x								
9	9				9	I	Y	i	y								
10	A	换行				J	Z	j	z								
11	B					K		k									
12	C					L		l									
13	D	回车				M		m									
14	E					N		n									
15	F					O		o									

American Standard Code for Information Interchange



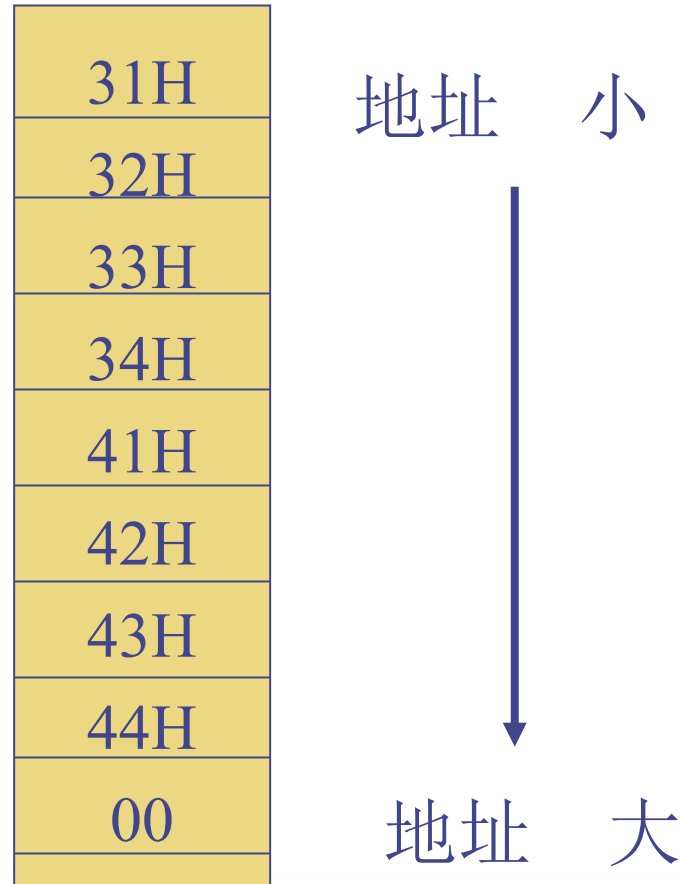


## 3.3 字符数据在机内的表示形式

字符串  
“1234ABCD”  
的表示结果：

以0 为串的结束

也可以写成'\0'





## 3.3 字符数据在机内的表示形式

### 汉字编码 - GB2312

GB2312是简体中文字符集，包含了6763个汉字和682个非汉字字符（图形符号），双字节编码。所有字符组成一个94x94的矩阵。矩阵的每一行称为一个“区”，每一列称为一个“位”（行列号称为区位号）。

- 01-09区为682个非汉字字符（图形符号）
- 16-55区为一级汉字（3755个最常用的汉字，按拼音字母顺序排列）
- 56-87区为二级汉字（3008个汉字，按部首次序排列）
- 10-15区、88-94是空白区（保留区）。







## 3.3 字符数据在机内的表示形式

### 汉字编码 - GB2312

区位码：由行号（区号）为高字节、列号（位号）位低字节组成的字

国际码（交换码）：区位码 + 0x2020（似乎没什么用）

机内码：区位码 + 0xA0A0

‘华’：区位码 = 27 10，即 0x1B0A

国际码 = 0x3B2A

机内码 = 0xBBAA（计算机存贮）






# 3.3 字符数据在机内的表示形式

```
char s[] = "123华中科技大学";
```

内存 1									
地址: 0x009FF998		列: 自动							
0x009FF998	31	32	33	bb	aa	d6	d0	bf	123?????
0x009FF9A0	c6	bc	bc	b4	f3	d1	a7	00	???????.

```
char16_t hz = '华';
```

监视 1		
搜索(Ctrl+E)  < > 搜索深度: 3		
名称	值	类型
 hz	48042 u'华'	char16_t
 hz,x	0xbbaa u'华'	char16_t



## 3.3 字符数据在机内的表示形式

### Unicode:

- 将全世界所有的字符包含在一个集合里;
- 只要支持该字符集, 就能显示所有字符, 不会有乱码了。
- 从0开始, 为每个符号指定一个编号(码点, code point)  
码点U+0041: 英语的大写字母A; 码点U+534E: 汉字“华”
- 在Unicode 7.0版中, 收入了109449个符号, 其中的中日韩文字为74500个。
- 最前面的65536个字符位, 码点从 U+0000 到 U+FFFF。  
称为基本平面 (BMP), 含最常见的字符。





## 3.3 字符数据在机内的表示形式

```
char16_t hz = '华'; //GB2312
char16_t hz1 = u'华'; //Unicode, UTF-16
```

hz	48042 u'𠂇'	char16_t
hz,x	0xbbaa u'𠂇'	char16_t
hz1,x	0x534e u'华'	char16_t
hz1	21326 u'华'	char16_t

猴:29492 吼:21564 厚:21402 候:20505 后:21518 呼:21628 乎:20046 忽:24573 瑚:29786 壶:22774  
葫:33899 胡:32993 蝴:34676 狐:29392 糊:31946 湖:28246 弧:24359 虎:34382 唬:21804 护:25252  
互:20114 沪:27818 户:25143 花:33457 哗:21719 华:21326 猾:29502 滑:28369

汉字的 unicode 编码表

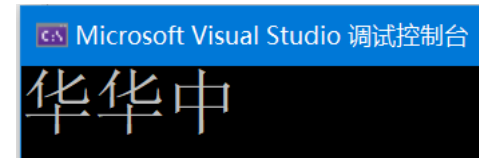
<https://blog.csdn.net/Aimless520/article/details/126692649>





## 3.3 字符数据在机内的表示形式

```
char p[] = "华\u534e中";  
cout << p << endl;
```



内存 1					
地址:	0x006FF6D4				列: 自动
0x006FF6D4	bb	aa	bb	aa	????
0x006FF6D8	d6	d0	00	cc	??.



## 3.3 字符数据在机内的表示形式

Unicode存在的问题:

- 如果所有字符都用2个字节、3个字节甚至更多字节来表示, 会造成存储空间的增加, 太费存储空间
- ASCII只需要一个字节来进行编码
- 如何区别该编码是Unicode还是ASCII ?
- 如何节约空间?
- 如何避免对编码误判?

**Unicode 的实现方式:** UTF-8、UTF-16、UTF-32





## 3.3 字符数据在机内的表示形式

### UTF-8

- 是一种变长的编码方法，长度从1个到4个字节不等；
- 越是常用的字符，字节越短；

- 对于单字节的符号

字节的第一位设为0，后面7位为这个符号的Unicode码。

对于英文字母，UTF-8 编码和 ASCII 编码是相同的。

- 对于n字节的符号 ( $n > 1$ )

第一个字节的前n位都设为1，第n+1位设为0，

后面字节的前两位一律设为10，

剩下没有提及的二进制位，全为这个符号的 Unicode 码。





## 3.3 字符数据在机内的表示形式

### ➤ 对于n字节的符号 ( $n > 1$ )

第一个字节的前n位都设为1，第n+1位设为0，

后面字节的前两位一律设为10，

剩下没有提及的二进制位，全为这个符号的 Unicode 码。

码点U+534E： 汉字“华” → UTF-8

0101 0011 0100 1110

15个有效二进制位，用3个字节来表示

1110 0101 10 00 11 01 10 00 1110

E5 8D 8E

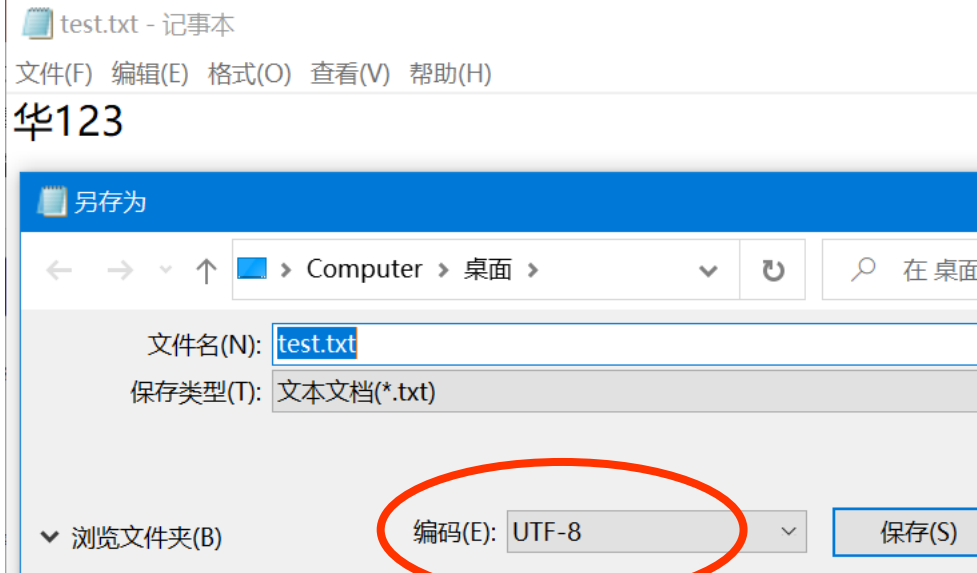






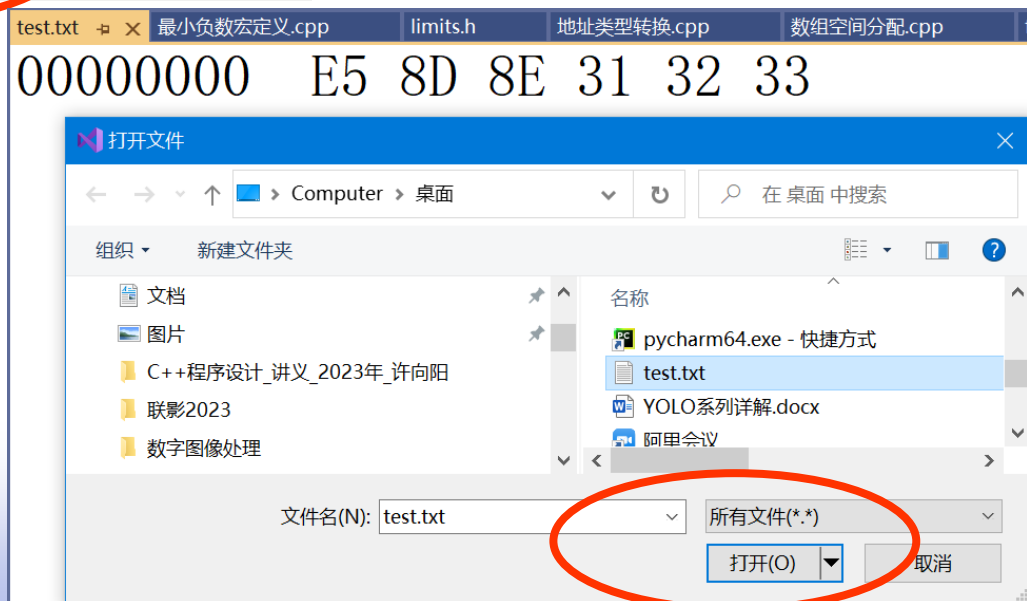
华中科技大学

## 3.3 字符数据在机内的表示形式



保存文件时，可以选择编码方式

打开方式中，以2进制形式打开





## 3.3 字符数据在机内的表示形式

### UTF-16

- 介于UTF-8和UTF-32之间，结合了定长和变长两种编码方式的特点；
- 基本平面的字符用2个字节，辅助平面的字符用4个字节。
- 2个字节 (U+0000 到 U+FFFF)

UTF-16编码就是码点 对应的16位无符号整数

- 4个字节 (U+010000 到 U+10FFFF)

先计算  $U' = U - 0x10000$

将 $U'$  写成二进制形式:  $yyyy\ yyyy\ yyxx\ xxxx\ xxxx$ ,

U的UTF-16编码:  $110110yyyyyyyyyy\ 110111xxxxxxxxxx$ 。





## 3.3 字符数据在机内的表示形式

### UTF-32

- UTF-32是最直观的编码方法
- 每个码点使用四个字节表示
- 字节内容一一对应码点
- 转换规则简单直观，查找效率高。缺点在于浪费空间
- 同样内容的英语文本，它会比ASCII编码大四倍。
- 实际上没有人使用这种编码方法

HTML 5标准就明文规定，网页不得编码成UTF-32。





## 3.3 字符数据在机内的表示形式

编码(E): GB18030

### ANSI 编码

- 不同的国家和地区制定了不同的标准
- 在简体中文Windows操作系统中，ANSI编码代表GB编码；
- 在日文Windows系统中，ANSI编码代表 Shift\_JIS 编码。
- 不同 ANSI 编码之间互不兼容；
- 对于ANSI编码而言，0x00 ~ 0x7F之间的字符，依旧是1个字节代表1个字符。





## 3.3 字符数据在机内的表示形式



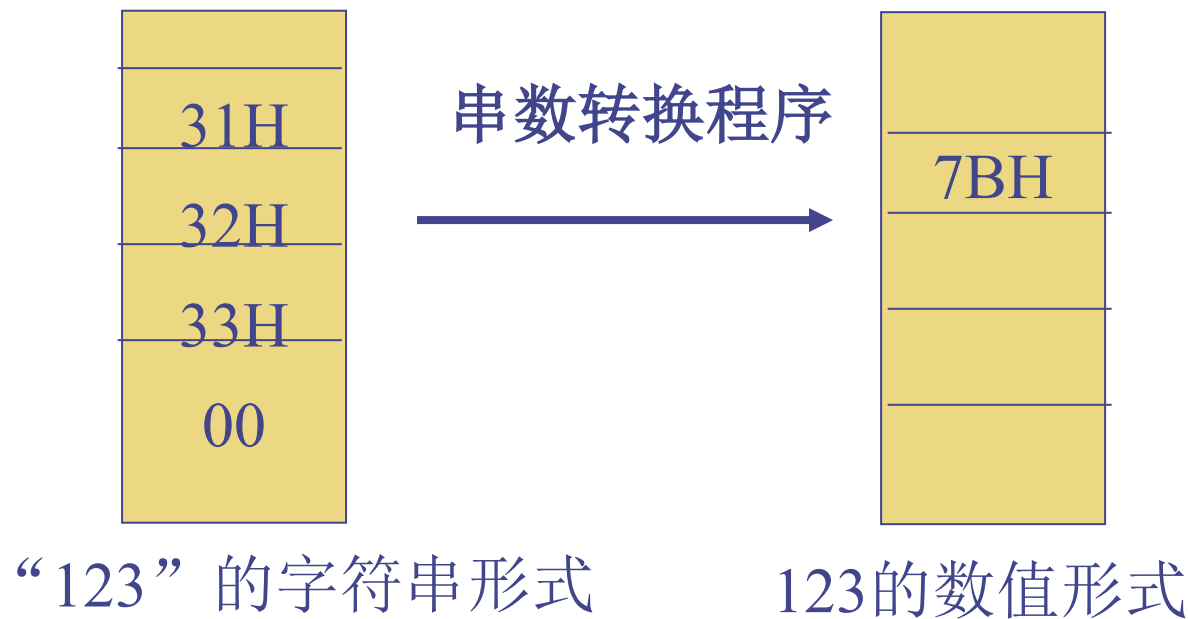
在 GB2312 中，“华”的区位码是 2710，第27区的第10个字（共 94 区，每区 94 个字）(1B 0A)。

区位码的每个字节加 A0，得到 ANSI 编码，即 BB AA。



## 3.3 字符数据在机内的表示形式

**Q:** 在键盘上输出123。计算机中得到是什么呢？  
若要用其作数值运算，怎么办呢？

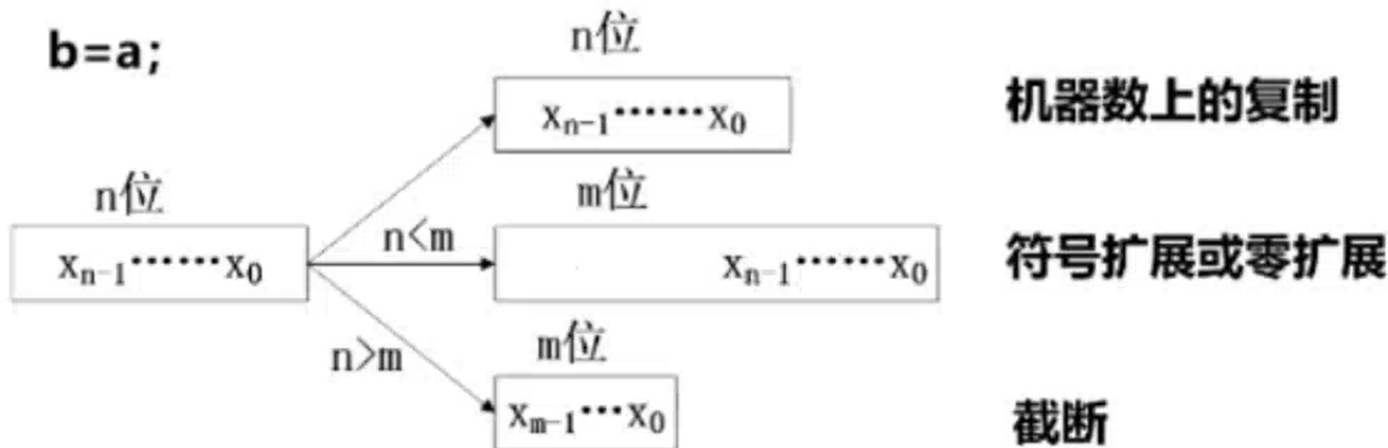


## 3.4 数据类型转换

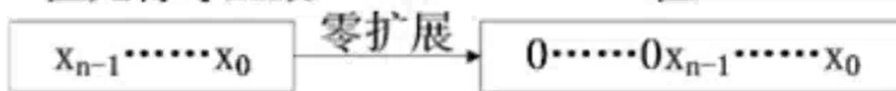
➤ 不同整数类型的变量赋值，如何进行转换？

**a和b都是整数**

**b=a;**



**n位无符号整数**



**n位有符号整数**



a

b

$n < m$



## 3.4 数据类型转换

➤ 不同**整数类型**的变量赋值，如何进行转换？

规则1：两边的长度一样，直接复制，与类型无关；

规则2：右边长，左边短，截断，保留低位；

规则3：右边短，左边长，右边的数需要扩展。

**Q：** 在需要扩展时，何时用符号扩展？

何时用 0 扩展？

```
int i1;   short s1;   char c1;   bool b1;  
unsigned int i2; unsigned short s2; unsigned char c2;
```







## 3.4 数据类型转换

```
short s1 = 0xffff;  
unsigned short s2 = 0xffff;  
  
int i1;  
i1 = s1;    // i1=0xfffffffff;  
  
i1 = s2;    // i1=0x0000ffff;
```

short  $\rightarrow$  int  $\Rightarrow$  赋值给 int

unsigned short  $\rightarrow$  unsigned int  $\Rightarrow$  赋值给 int

结论：以右边的类型进行扩展





## 3.4 数据类型转换

```
short s1 = 0xffff;           s1=-1;  
unsigned short s2 = 0xffff;  s2=-1;  
int i1;      unsigned int i2;
```

```
i1 = s1;    // i1=0xffffffff;  
i1 = s2;    // i1=0x0000ffff;
```

```
i2 = s1;    // i2=0xffffffff;  
i2 = s2;    // i2=0x0000ffff;
```

结论：以右边的类型进行扩展；  
与赋值号左边变量的类型无关



# 3.5 浮点数据在机内的表示形式

```
float f1 = 1.25;
00101865 F3 0F 10 05 40 7
0010186D F3 0F 11 45 F8
double d1 = 1.25;
00101872 F2 0F 10 05 48 7
0010187A F2 0F 11 45 E8
long double ld1 = 1.25;
0010187F F2 0F 10 05 48 7
00101887 F2 0F 11 45 D8
```

内存 1

地址: 0x0059FDD8

0x0059FDD8	00 00 a0 3f cc cc cc cc 00 fe 59 00	..??
0x0059FDE4	83 20 10 00 01 00 00 00 b0 92 ac 00	? ..
0x0059FDF0	18 7a ac 00 01 00 00 00 b0 92 ac 00	.z?.

监视 1

搜索(Ctrl+E)

搜索深度: 3

名称	值	类型
f1	1.25000000	float
&f1	0x0059fdd8 {1.25000000}	float *

添加要监视的项

float f1=1.25;

对应的4个字节内容为: 3f a0 00 00

## 3.5 浮点数据在机内的表示形式

```
double d1 = 1.25;
00101872 F2 0F 10 05 48 7B 10 00 movsd      xmm0,mmword ptr [__real@3ff4000000
0010187A F2 0F 11 45 E8
    long double ld1 = 1.25;
0010187F F2 0F 10 05 48 7B 10 00 movsd      xmm0,mmword ptr [__real@3ff4000000
00101887 F2 0F 11 45 D8
    printf("%f %f %f \n",
0010188C 83 EC 08
0010188F F2 0F 10 45 D8
00101894 F2 0F 11 04 24
00101899 83 EC 08
```

内存 1

地址: 0x0059FDC8 列: 自动

0x0059FDC8	00 00 00 00 00 00 f4 3f cc cc cc cc	....
0x0059FDD4	cc cc cc cc 00 00 a0 3f cc cc cc cc	????
0x0059FDE0	00 fe 59 00 83 20 10 00 01 00 00 00	.?Y.

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
f1	1.25000000	float
&f1	0x0059fdd8 {1.25000000}	float *
&d1	0x0059fdc8 {1.2500000000000000}	double *

double d1=1.25;

对应的8个字节内容为: 3f f4 00 00 00 00 00 00



## 3.5 浮点数据在机内的表示形式

$$\pm(a_n 2^n + \dots + a_1 2^1 + a_0 2^0 + a_{-1} 2^{-1} + \dots + a_{-m} 2^{-m} + \dots)$$

$$1.25 = 1.01\text{B};$$

**规格化数据：**  $\pm 1.XXXX * 2^n \text{ B}$

科学表示法：小数点左边恒为1

要素：正负号、尾数 XXXX、指数 n

**问题：**尾数 XXXX、指数各用多少位来表示？

**用什么形式来表达？**





## 3.5 浮点数据在机内的表示形式

31	30	23	22	0
符号	8 位指数			23 位尾数

单精度浮点数  
float

符号位：0表示正数，1表示负数

指 数：采用“移码”来表示。单精度移 7F

-126 D  $\rightarrow$  00000001 B

127 D  $\rightarrow$  11111110 B

尾 数：用原码表示。对于规格化浮点数，尾数的小数点前面的1，不需存贮。

$1.25 = 1.01\text{B} = 1.01 * 2^0$ ;

0 0111 1111 010 000000000000000000000000 B

对应的4个字节内容为： 3f a0 00 00





## 3.5 浮点数据在机内的表示形式

63	62	52	51	0
符号	11 位指数		52 位尾数	

双精度浮点数  
**double**

符号位：0表示正数，1表示负数

指 数：采用“移码”来表示。双精度移 3FF

-1022D  $\rightarrow$  000 0000 0001B

1023D  $\rightarrow$  111 1111 1110B

double d1=1.25 =  $1.01 * 2^0$  ;

0 011 1111 1111 0100 0000 ..... 0000

对应的8个字节内容为：3f f4 00 00 00 00 00 00





## 3.5 浮点数据在机内的表示形式

31	30	23	22	0
符号	8 位指数			23 位尾数

单精度浮点数  
**float**

63	62	52	51	0
符号	11 位指数			52 位尾数

双精度浮点数  
**double**

79	78	64	63	0
符号	15 位指数			63 位尾数

高精度浮点数  
**long double**

符号位：0表示正数，1表示负数

指 数：采用“移码”来表示。

单精度移 7F      双精度移 3FF

高精度移 3FFF







# 3.5 浮点数据在机内的表示形式

31	30	23	22	0
符号	8 位指数			23 位尾数

- (1) 规格化浮点数
- (2) 非规格化浮点数
- (3) 特殊数值 ( $\pm\infty$ , NaN等)

单精度浮点数: float

表 2.2 IEEE 754 浮点数的解释

值的类型	单精度 (32 位)				双精度 (64 位)			
	符号	阶码	尾数	值	符号	阶码	尾数	值
正零	0	0	0	0	0	0	0	0
负零	1	0	0	-0	1	0	0	-0
正无穷大	0	255(全1)	0	$\infty$	0	2047(全1)	0	$\infty$
负无穷大	1	255(全1)	0	$-\infty$	1	2047(全1)	0	$-\infty$
无定义数 (非数)	0 或 1	255(全1)	$\neq 0$	NaN	0 或 1	2047(全1)	$\neq 0$	NaN
规格化非零正数	0	$0 < e < 255$	$f$	$2^{e-127} (1.f)$	0	$0 < e < 2047$	$f$	$2^{e-1023} (1.f)$
规格化非零负数	1	$0 < e < 255$	$f$	$-2^{e-127} (1.f)$	1	$0 < e < 2047$	$f$	$-2^{e-1023} (1.f)$
非规格化正数	0	0	$f \neq 0$	$2^{-126} (0.f)$	0	0	$f \neq 0$	$2^{-1022} (0.f)$
非规格化负数	1	0	$f \neq 0$	$-2^{-126} (0.f)$	1	0	$f \neq 0$	$-2^{-1022} (0.f)$



## 3.5 浮点数据在机内的表示形式

规格化浮点数:  $\pm 1.XXXX * 2^n$  B (不存储整数位1)  
n (移码) 范围: [00000001, 11111110]

因此, 规格化单精度浮点指数范围: [-126, 127]

尾数范围: [000...000 (23个0), 111...111]

规格化单精度浮点数范围:

$\pm [1.0 * 2^{-126}, 1.111...111 * 2^{127}]$  B  $\Leftrightarrow$

$\pm [1.175494351 * 10^{-38}, 3.402823466 * 10^{38}]$  D

Q: 规格化浮点数能表示0和很小的数(如  $2^{-130}$ ) 吗?





## 3.5 浮点数据在机内的表示形式

非规格化浮点数:  $\pm 0.XXXX * 2^{-126}$  B

规定: 阶码(移码) = 00000000 (尾数不为0)

非规格化单精度浮点数范围:

$\pm [0.000...001 * 2^{-126}, 0.111...111 * 2^{-126}]$  B  $\Leftrightarrow$

$\pm [1.4013 * 10^{-45}, 1.17548 * 10^{-38}]$  D

对于区间  $(-1.4013 * 10^{-45}, 0)$ 、 $(0, 1.4013 * 10^{-45})$  的数,  
float 是无法表示的 !!!





## 3.5 浮点数据在机内的表示形式

### 特殊数值

**$+\infty$  符号位0、阶码全1、尾数为0**

0-1111111 1-0000000 00000000 00000000

**$-\infty$  符号位1、阶码全1、尾数为0**

1-1111111 1-0000000 00000000 00000000

**NaN (Not a Number) 符号位1或0、阶码全1、尾数不为0**

例如: 0-1111111 1-0000000 00000000 00000001

一些因素会引起 NaN (非数值), 如  $0 / 0$ 、 $\text{sqrt}(-1)$  等。

**$+0$  符号位0、阶码全0、尾数为0**

0-0000000 0-0000000 00000000 00000000

**$-0$  符号位1、阶码全0、尾数为0**

1-0000000 0-0000000 00000000 00000000





## 3.5 浮点数据在机内的表示形式

### Problem1:

```
float a = 10000;
```

```
float b = a + 0.0001f;
```

```
bool c1 = (a == b);
```

```
bool c2 = (10000 < 10000.0001f);
```

**c1, c2 = true or false ?**

### Problem2:

```
float a = 0x65536; //0x10000
```

```
float b;
```

**求满足  $b > a$  的条件下, IEEE754能表示的最小 b。**



# “Father” of the IEEE 754 standard

---

直到80年代初，各个机器内部的浮点数表示格式还没有统一  
因而相互不兼容，机器之间传送数据时，带来麻烦

1970年代后期，IEEE成立委员会着手制定浮点数标准

1985年完成浮点数标准IEEE 754的制定

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.



[www.cs.berkeley.edu/~wkahan/ieee754status/754story.html](http://www.cs.berkeley.edu/~wkahan/ieee754status/754story.html)



**Prof. William Kahan**



## 3.5 浮点数据在机内的表示形式

Q: 为什么指数使用移码表示，而不是补码表示？

在加法运算时，要对阶，用无符号数比较、移位更简单。

Q: 为什么规格化时，用  $\pm 1.XXXX * 2^n B$ ，  
而不是  $\pm 0.1XXXX * 2^n B$ ？

能够保存更多的有效位数，指数的表示范围也更大。





## 3.6 数据类型转换

同样一个数值，分别用 short, int, float, double 表示，结果相同吗？

```
int i; float f; double d;
```

Q: `i == (int)(float)i`; ?

```
#include <stdio.h>

int main()
{
    int i;
    i = 1234567890;
    printf("%d \n", i);
    i = (int)(float)i;
    printf("%d \n", i);
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
1234567890
1234567936
```







## 3.6 数据类型转换

同样一个数值，分别用 short, int, float, double 表示，结果相同吗？

```
int i; float f; double d;
```

Q:  $i == (\text{int})(\text{double})i$  ?

```
int main()
{
    int i;
    i = 1234567890;
    printf("%d \n", i);
    i = (int)(float)i;
    printf("%d \n", i);
    i = 1234567890;
    printf("%d \n", i);
    i = (int)(double)i;
    printf("%d \n", i);
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
1234567890
1234567936
1234567890
1234567890
```

float 23 位尾数

double 52位尾数





## 3.6 数据类型转换

同样一个数值，分别用 short, int, float, double 表示，结果相同吗？

```
int i; float f; double d;
```

Q: `f == (float)(int)f; ?`





华中科技大学

# 结构数组的数据存放

通过实例展示结构数组的存放结果  
不同编译开关下（对齐方式）结构数组数据的存放





# 总结

数据存储的基本形式

有符号整数和无符号整数的表示与存储

字符串的表示方法与存储

浮点数据的表示方法与存储

地址类型转换 与 数据类型转换

