南京理工大学
NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

第 2 章 数据信息的表示与运算单元

主讲 张功萱

非数值型数据的表示

第2.6节

1. 怎么表示非数值型数据?
2. 逻辑数又怎么表示?
3. 汉子表示有哪几个部分?

非数值型数据的表示

- 为了处理非数值领域的问题，需要在计算机中引入文字、字母及一些专用符号等，以便表示文字语言、逻辑语言等信息。但由于计算机硬件能够直接识别和处理的只是“0”、“1”二进制信息，因此在计算机中对这类数据必须用二进制代码来表示。
- 非数值型数据表示：逻辑数、字符、字符串、文字及某些专用符号等的二进制代码。
- 这些二进制代码并不表示数值，所以称为非数值型数据或符号数据。

2.6.1 逻辑数 —— 二进制串

- 在计算机中一个逻辑数是用一个二进制串来表示的。逻辑数具有下面几个特点：
- (1) 逻辑数没有符号的问题。逻辑数中各位之间是相互独立的，既没有位权问题，也没有进位问题。
- (2) 逻辑数中的“0”与“1”不代表值的大小，仅代表一个命题的真与假、是与非等逻辑关系。
- (3) 逻辑数只能参加逻辑运算，并且是按位进行的。
  - 例： 1011＋1100＝1111

2.6.2 字符与字符串

- 字符与字符串数据是计算机中用得最多的符号数据，它是人和计算机联系的桥梁。为使计算机硬件能够识别和处理字符，必须对字符按一定规则用二进制编码。

1. 字符编码

- 目前广泛使用的是 ASCII码(美国国家信息交换标准字符码)和EBCDIC码(扩展的二 — 十进制交换码)。
- ASCII码是用七位二进制表示一个字符，它包括10个数字(0～9)，52个英文大、小写字母(A～Z，a～z)，33个专用字符(如，、、％、#等)和33个控制字符(如NUL、LF、CR、DEL等)共128个字符。
- ASCII字符编码符号的排列次序为$b_6b_5b_4b_3b_2b_1b_0$，其中$b_6b_5b_4$为高位部分，$b_3b_2b_1b_0$为低位部分。

$b_6b_5b_4$ / $b_3b_2b_1b_0$	000	001	010	011	100	101	110	111
0000	NUL	DEL	SP	0	@	P	`	p
0001	SOH	DC_1	!	1	A	Q	a	q
0010	STX	DC_2	”	2	B	R	b	r
0011	ETX	DC_3	#	3	C	S	c	s
0100	EQT	DC_4	$	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

*RISC-V ASCII码

ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

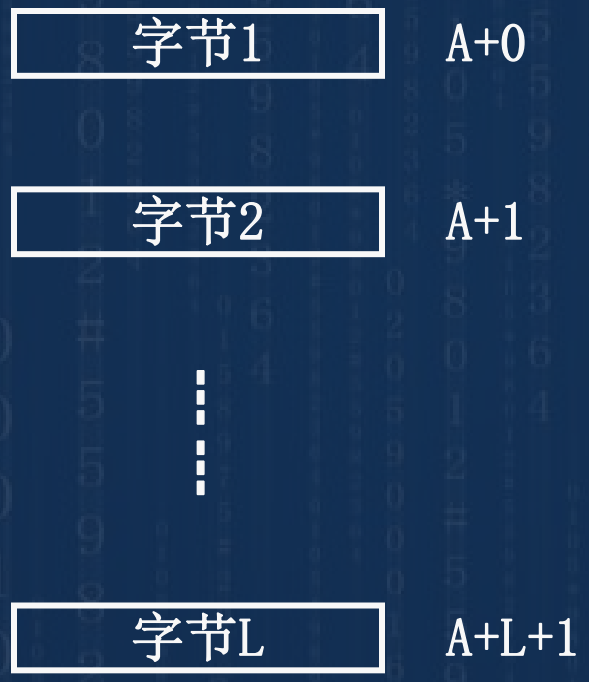
FIGURE 2.15 ASCII representation of characters. Note that upper- and lowercase letters differ by exactly 32; this observation can lead to shortcuts in checking or changing upper- and lowercase. Values not shown include formatting characters. For example, 8 represents a backspace, 9 represents a tab character, and 13 a carriage return. Another useful value is 0 for null, the value the programming language C uses to mark the end of a string.

- NUL 空
- SOH 标题开始
- STX 文本起始
- ETX 文本结束
- EOT 传输结束
- ENQ 询问
- ACK 应答（肯定）
- BEL 响铃
- BS 退一格
- HT 水平制表符
- LF 换行
- SYN 空转同步
- ETB 信息组传送结束
- CAN 作废
- EM 纸尽
- SUB 取代
- ESC 中断
- VT 垂直制表
- FF 换页
- CR 回车
- SO 移位输出
- SI 移位输入
- SP 空间(空格)
- DLE 数据连接断开
- DC1 设备控制1
- DC2 设备控制2
- DC3 设备控制3
- DC4 设备控制4
- NAK 反向应答（否定）
- FS 文件分隔符
- GS 组分隔符
- RS 记录分隔符
- US 单元分隔符
- DEL 作废

- 在计算机中，通常用一个字节表示一个字符。由于ASCII编码为七位二进制，字节的最高位的作用：
- ① 用作奇偶校验位，用来检测错误。
- ② 用于表示字符，形成扩展的ASCII码。如EBCDIC码。
- EBCDIC(EstendedBinary Coded Decimal Interchange Code)是IBM公司常用的一种字符编码。它采用八位二进制数表示一个字符。
- ③ 在我国用于区分汉字和字符。如规定字节的最高位为“0”表示ASCII码，为“1”表示汉字编码。

2. 字符串数据

- 字符串：连续的一串字符。
- 通常一个字符串占用主存中多个连续的字节进行存放。

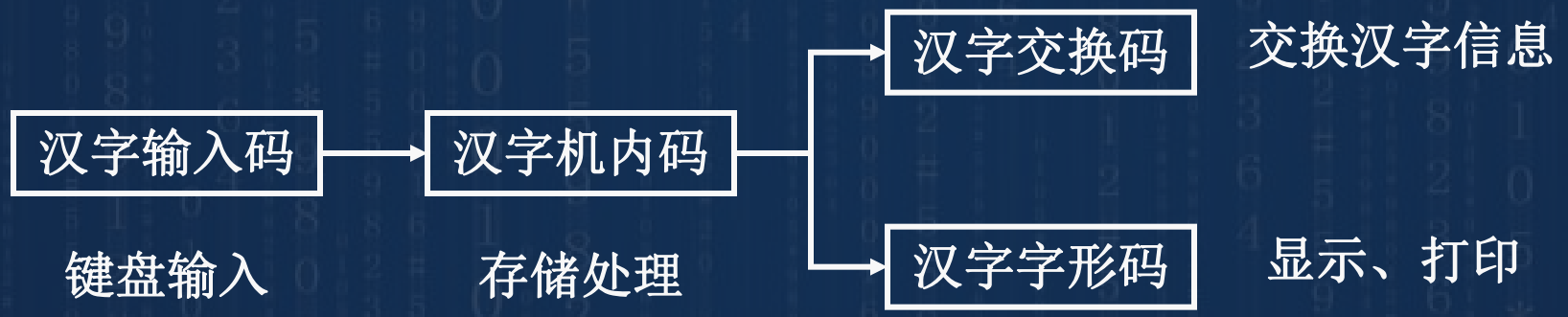


3. 十进制数串的表示

- 为了满足某些应用领域的需要，要求某些计算机内部能直接对十进制数进行运算和处理，为此要求对十进制数字进行二进制编码，且能够便于处理。
- 常见的有**BCD**码
- 相关知识已在计算机逻辑中叙述（回顾自习）

2.6.3 汉字信息的表示

- 为使计算机能够处理各种汉字信息，必须对汉字进行编码。
- 汉字在计算机中的表示比较特殊。因为在计算机中使用汉字，需要涉及到汉字的输入，存储与处理、汉字的输出等几方面的问题，因此汉字的编码也有多种类型。



汉字处理过程

- 在汉字信息处理系统中，人们使用键盘把汉字以汉字输入码的形式输入到计算机内，将其变换成计算机内部表示的汉字机内码，进行存储和处理。处理结果，如果送往终端设备或其它汉字系统，则把汉字机内码变换成标准汉字交换码，再传送出去。如果把处理结果显示或打印，则把汉字机内码变换成汉字地址码到字库取出汉字字形码送往显示器或打印机。

1. 汉字输入码

- 汉字输入码：汉字输入操作者使用的汉字编码。
- (1) 音码：利用汉字的字音属性对汉字的编码。
- 如：全拼、双拼、智能ABC、紫光拼音输入法等。
- 特点：易记。但击键次数多，重码多，不能盲打。
- (2)形码：以汉字的笔划和顺序为基础的编码。也称字形编码。
- 如：五笔字型、郑码等。
- 特点：便于快速输入和盲打，但要经过训练和记忆。

- (3) 音形结合码：将音码和形码结合起来的编码。
- 如：声韵笔形码、郑码、自然码等。
- (4) 数字码：用固定数目的数字来代表汉字。
- 如：电报码、区位码。
- 特点：无重码，输入码与机内码的转换比较方便，但难记忆。
- ① 电报码：用4位十进制数字表示一个汉字。
- ② 区位码：用数字串代表一个汉字输入。常用的是国标区位码，它是将国家标准局公布的6763个两级汉字分为94个区，每个区分94位，实际上把汉字表示成二维数组，每个汉字在数组中的下标就是区位码。区码和位码各两位十进制数字，因此输入一个汉字需按键四次。例如“中”字位于第54区48位，区位码为5448。

2. 汉字交换码

- 汉字交换码：用于不同汉字系统间交换汉字信息，具有统一的标准。
- 1981年国家标准总局公布了《信息交换用汉字编码字符集》，即GB2312—80，简称国标码。也称为国标交换码、交换码。
- 该标准共收集汉字6763个，其中一级汉字3755个，二级汉字3008个，再加上各种图形符号682个，共计7445个。
- 国标码规定每个汉字、图形符号都用两个字节表示，每个字节只使用最低七位。

- 国标码是在区位码的基础上规定的。
- 区位码无法用于汉字通信，因为它可能与通信使用的控制码（00H~1FH）（即0~31）发生冲突。
- ISO2022规定每个汉字的区号和位号必须分别加上32（即二进制数00100000），经过这样的处理而得的代码就是国标码。

- 例：“学”字的区号为49，位号为07，其区位码为4907，用2个字节的二进制数表示为：
  00110001 00000111
- “学”字的国标码计算为：
- 00110001 00000111
  +00100000 +00100000
  01010001 00100111
- 用十六进制数表示为5127H。

3. 汉字机内码

- 汉字机内码：用于汉字信息的存储、检索等操作的机内代码，一般采用两个字节表示。
- 英文字符的机内代码是七位的ASCII码，当用一个字节表示时，最高位为“0”。
- 为了与英文字符能相互区别，目前我国的计算机系统中汉字内码都是以国标码为基础，在国标码基础上把每个字节的最高位置“1”，作为汉字标识符。即机内码＝国标码＋8080H
- 有些系统中，字节的最高位作为奇偶校验位，在这种情况下就用三个字节表示汉字内码。

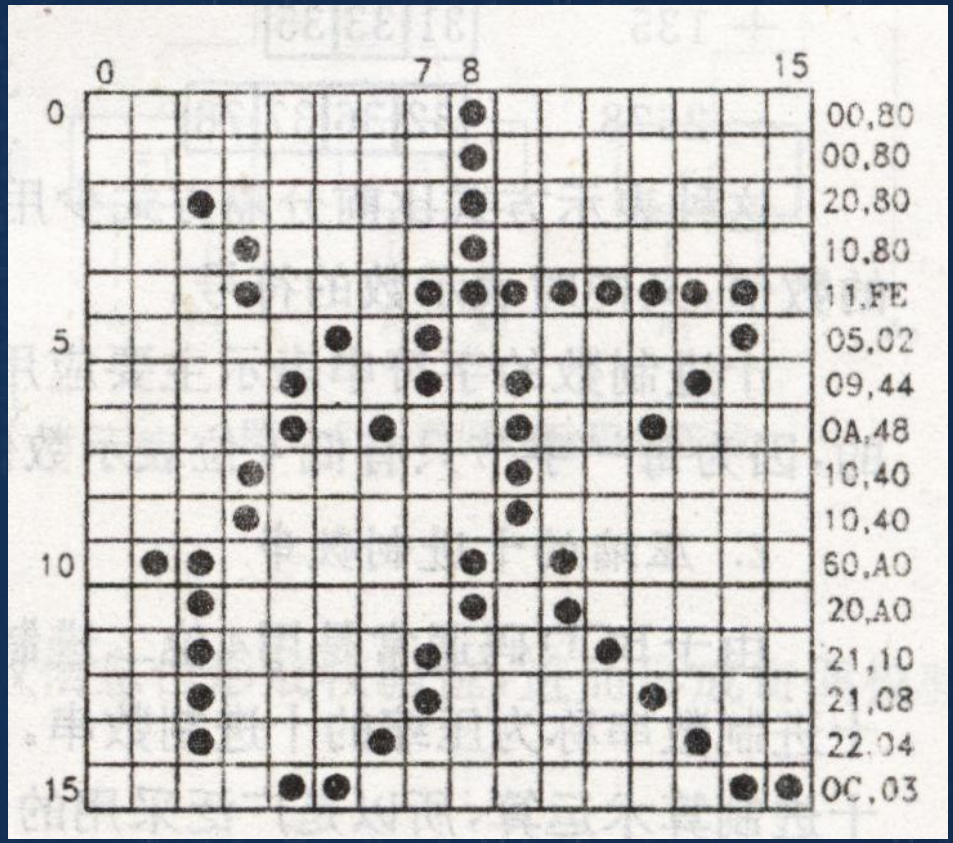
- 例：二进制编码 00111100和01000110两个字节分别表示ASCII码字符的“<”和“F”。
- 二进制编码 10111100和11000110两个字节一起表示一个汉字的内码。这两个字节是汉字“计”的汉字内码。

4. 汉字字形码

- 汉字字形码用于记录汉字的外形，是汉字的输出形式。又称字模。主要用于汉字的显示和打印。
- 汉字字形有两种记录方法
- (1) 点阵法
- 点阵法对应的字形编码称为点阵码。
- (2) 矢量法
- 矢量法对应的字形编码称为矢量码。

点阵法

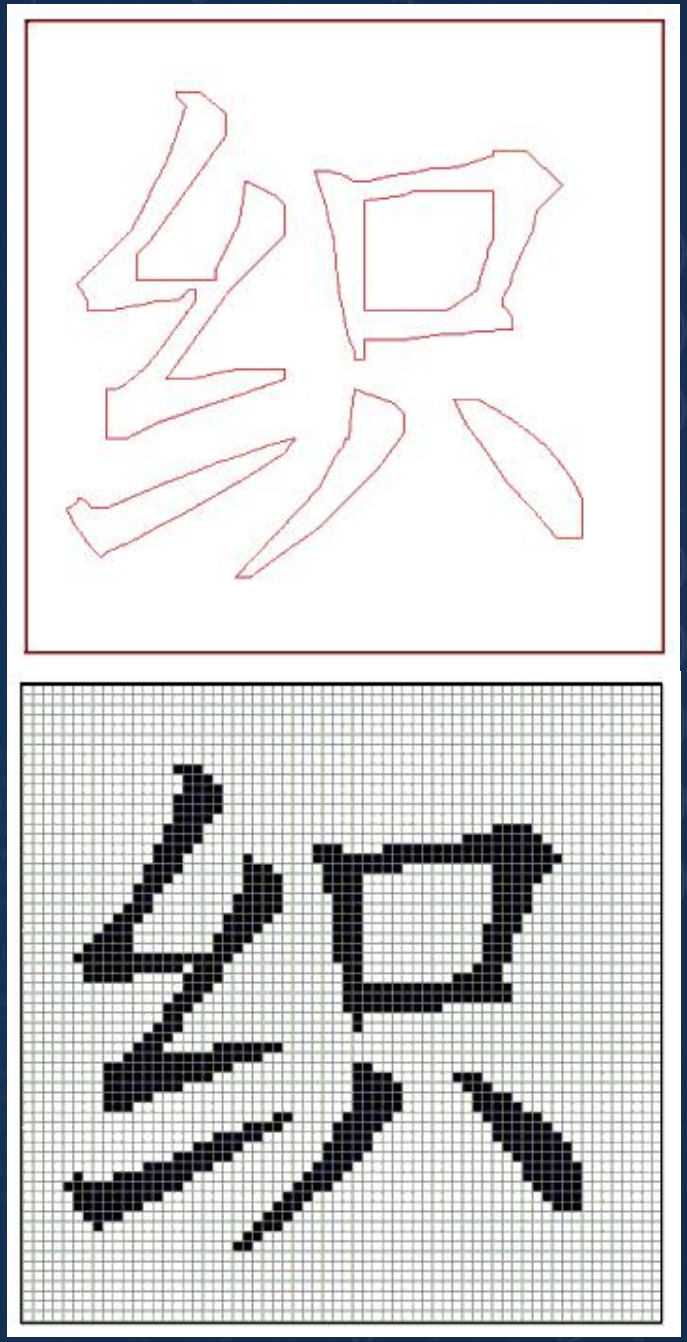
- 点阵法用点阵表示汉字字形。根据汉字输出的要求不同，点阵的多少也不同。
- 简易型汉字为16×16点阵，多用于显示。
- 提高型汉字为24×24点阵、32×32点阵，48×48点阵、64×64点阵、128×128点阵，甚至更高，多用于打印。



16×16的汉字字形点阵，每个汉字占32个字节。

矢量法

- 矢量码使用一组数学矢量来记录汉字的外形轮廓，矢量码记录的字体称为矢量字体或轮廓字体。
- 矢量字体能很容易地放大缩小而不会出现锯齿状边缘，屏幕上看到的字形和打印输出的效果完全一致。



曲线法

- 用若干个直线段和若干个曲线段描述汉字字形。其中曲线段采用二次曲线函数或三次曲线函数进行描述。
- 在目前使用的系统中，已普遍使用轮廓字体（即True Type字体，采用二次Bezier样条曲线描述字体）。
- 如中文Windows中提供了宋体、黑体、楷体、仿宋体等True Type字体的汉字库文件。

- 因为汉字字形码的信息量很大，所占存储空间也很大，因此汉字字形码不用于机内存储，而采用汉字字库存储。
- 所有的不同字体、字号的汉字字形码构成了汉字字库。只有需要输出汉字时，才将汉字机内码转换为相应的汉字字库地址，检索字库，输出字形码。
- 目前汉字字库通常是以多个字库文件的形式存储在硬盘上。

- 随着计算机技术和因特网技术的发展，汉字信息处理的应用范围不断扩大，各种领域对字符集提出了多文种、大字量、多用途的要求，GB2312—80中的6763个汉字明显不够用。为满足各方面应用的需要，国家开始对原来的基本汉字集进行扩充。

- 为了满足不同国家不同语系的字符编码要求，一些计算机公司结成了一个联盟，创立了一个称为Unicode的编码体系。
- 目前Unicode体系已成为了一种国际标准，即ISO 10646。在Unicode体系中，每个字符和符号被赋予一个永久、惟一的16位值，即码点。

Elaboration: Reflecting the international nature of the web, most web pages today use Unicode instead of ASCII. Hence, Unicode may be even more popular than ASCII today.

- **Unicode体系目前普遍采用的是UCS-2，即用两个字节来编码一个字符，共有65536个码点，可以表示65536个字符。由于每个字符长度固定为16位，使得软件的编制简单了许多。**
- **UCS-2将世界上几乎所有语言的常用字符都收录其中，包含20902个汉字、（汉字Unicode编码的区间为：0x4E00--0x9FA5），韩文、日文等不同国家的字符以及造字区、保留字符等，共计65534个，方便了信息交流。**
- **Unicode也有UCS-4规范，用4个字节来编码字符。**

- 注意：汉字的输入码、机内码、字形码是计算机中用于输入、内部处理、输出三种不同用途的编码，不能混为一谈。

汉字库的读取

- 由机内码读点阵字模
- 设**ch1ch2**表示某汉字机内码,则**16**点阵库中读取该汉字字模的位置是:
- **(1) 213**汉字系统的定位算法为
- **Location=((unsigned)ch1-160-1)*94+((unsigned)ch2-160-1)*32**

- **(**汉字库文件为: **HZK16)**

汉字库的读取（续）

- (2)UCDOS的计算为
- 运行RD16.COM后,加载汉字到内存,
- 然后
- _DH=ch1; _DL=ch2;
- getinterrupt(0x7F);
- Buf=MK_FP(_DX,0);
- 其中, unsigned char *Buf;

数据的长度与存储方式

第2.7节

1. 怎么存储不同长度数据?
2. 各类型数据转换注意什么?
3. 什么是数据的大小端存储?

2.7.1 数据的长度

- 不同类型数据的二进制长度各不相同，因此计算机存储数据时，必须按照规定的顺序组织存储。
- 1. 位、字节、字和字长
- 位→在计算机系统中，一位二进制数据“0”或“1”称为一个“位”或一个“比特”（bit，简写b）。位是存储、传输和处理信息的最小单位。
- 字节→计算机系统中西文字符通常采用8个二进制位表示，因此将8个bit称为一个字节（Byte，简写为B）。
- 字节是计算机系统中最常用的二进制计量单位。
- 字→“字”是指计算机系统中可以在同一时间内被同时处理的一组二进制数，“字”在计算机中通常作为一个整体被存取、传输和处理。

字长

- 字中包含的二进制位数称为字长。
- 字长反映了CPU内部数据通道的宽度，也反映了CPU中通用寄存器的宽度。不同的计算机系统中字的长度不是一定的，有的机器一个字的字长是1个或2个字节，如早期的Intel系列8位和16位处理器；有的机器一个字的字长是4个或8个字节，如现在常用的32位和64位微处理器。
- 字长反映了一台计算机的计算精度，同时为适应不同用户的要求、协调运算精度和硬件造价间的关系，大多数计算机均支持变字长运算，即机内可实现半字长、全字长（或单字长）和双倍字长运算。一般来说，在其他指标相同的情况下，字长越长，计算机处理数据的速度越快。

C语言中的整数类型

- C语言中的无符号整数的表示：
- Unsigned short、Unsigned int（Unsigned） Unsigned long等
- C语言中带符号整数采用补码表示：
- Short、int、long等

- 常在数据的后面加一个“u”或“U”来表示无符号数。
- 例：12345U、0x5B6Fu

2. C语言中基本数据类型的长度和格式

- C语言的基本数据类型包含数值型和字符型（char）两大类。
- 数值型数据分为整型数（int）和实型数（float，也称浮点数型）。
- 整型数又分为无符号整数（unsigned）和带符号整数（signed）。
- 各种数据根据长度不同分为短数据（short）、长数据（long）和正常数据。
- 如：整型数int、短整型数short int、长整型数long int、无符号整型数unsigned int、浮点数float、双精度浮点数double（double float）、长双精度浮点数long double（long double float）等。

- C语言中的字符型数据是按字符所对应的ASCII码值来存储的，因此char型数据的长度通常为1个字节。
- char型数据可以用于表示单个字符，也可以用于表示8位整数。

C语言中的整型数

- C语言中的整型数用于表示整数。
- ① 无符号整数规定数据对应的机器数字长的所有二进制位均表示数值。
- ②带符号整数采用补码表示，用机器数最高位的“0”、“1”表示数值的正负。
- 当C语言中整型数据的长度为n时：
- 无符号整数的数据范围是$0\sim 2^{n-1}$
- 带符号整数的数据范围是$-2^{n-1}\sim+(2^{n-1}-1)$。

- C语言中允许无符号整数与带符号整数之间的转换，转换后的数的真值是将原二进制机器数按转换后的数据类型重新解释得到。
- 例：在32位机中，有以下C代码：
- 1 int x=－1
  2 Unsigned u=2147483648 （即2^{31}）
  3
  4 printf(“x=%u=%d\n, x,x)
  5 printf(“u=%u=%d\n, u,u)
- 输出结果：
- x＝4294967295＝－1
- u＝2147483648＝－2147483648

- x的输出结果中
- －1的补码整数表示为“111•••1”
- 通过%d作为带符号数解释时，其值为“－1”
- 通过%u作为无符号数解释时，在32位机中其值为：$2^{32}-1=4294967295$
- u的输出结果中
- 2^{31}的无符号整数表示为“100•••0”
- 通过%u作为无符号数解释时，其值为2^{31} “2147483648”
- 通过%d作为带符号数解释时，其值为“－2147483648 ”

- C语言中，如果在执行一个运算中有带符号整数与无符号数同时参加，则C编译器会隐含地将带符号整数强制类型转换为无符号数，因而会带来意想不到的结果。
- 例：以下无符号运算的结果与直觉不符
- －2147483648＝＝ 2147483648U　结果为√
- ∵ 无符号运算中 100•••0B＝100•••0B

- －2147483648＜ 2147483647U　结果为×
- ∵ 无符号运算中
  100•••0B (2^{32})＞011•••1B ($2^{32}-1$)

C语言中的实型数

- C语言中的实型数主要用于表示浮点数。在机器内部，实型数均采用IEEE754 浮点数标准格式。
- C语言中浮点数的表示：
- float：对应IEEE754单精度浮点数格式
- double：对应IEEE754双精度浮点数格式
- long double：对应IEEE754扩展双精度浮点数格式，但其长度和格式随编译器和处理器的不同而有所不同。

注意

- 在实际工作中，我们常说系统工作在某平台上，所谓平台是指计算机系统所使用的CPU、操作系统和编译器。
- 作为高级语言，C语言中各种数据类型在不同的平台上分配的字节数不一定相同。

C语言中的数据长度的规定

- (1) char类型一般是8bit，但某些嵌入式编译器使用的char类型可能是16bit。
- (2) short类型和long类型的长度不相同。
- (3) int类型通常与具体机器的物理字长一致。
- (4) 虽然每种编译器可以根据硬件的不同自由确定各种数据类型的长度, 但short和int类型最少是16bit, long类型最少是32bit, short类型必须比int和long类型要短。

Elaboration: RISC-V software is required to keep the stack aligned to "quadword" (16 byte) addresses to get better performance. This convention means that a `char` variable allocated on the stack may occupy as much as 16 bytes, even though it needs less. However, a C string variable or an array of bytes *will* pack 16 bytes per quadword, and a Java string variable or array of shorts packs 8 halfwords per quadword.

C语言中常用数据类型的字节数

字节数 / 数据类型	32位编译器	64位编译器
char	1	1
char *（指针变量）	4	8
short int	2	2
int	4	4
long int	4	8
float	4	4
Double	8	8
Long	4	8
long long	8	8

注意！！！

- 在不同类型的数据之间进行运算和转换，有可能出现的意想不到的情况（在32位的编译环境中）：
- (1) 在整数运算中，如果同时有无符号数和带符号数参加运算，则C编译器会隐含地将带符号整数强制地转换为无符号整数，从而造成运算出错。
- (2) 从int转换为float时，不会发生溢出，但因为32位单精度浮点数的尾数有效长度短于int的数值部分的长度，所以可能出现数据被舍入的情况。
- (3) 从int或 float转换为double时，因为double的尾数有效长度更长，所以可以保留更多的精确值。

- (4) 从double转换float时，因为float的表示范围小于double，所以可能发生溢出；由于float的有效位数少，故可能有数据被舍入。
- (5) 从float或double转换为int时，因为int 没有小数，故可能丢失有效数据。因为int的表数范围小，故可能发生溢出。

- 例：设变量i、f、d的类型分别是int、float、double，它们的取值是除＋∞、－∞、NaN以外的任意值。请判断下列每个C语言关系表达式在32位机上运行时是否永真。
- ⑴ i＝＝(int)(float) i
- × ∵int精度比float高，∴当i转换为float再转换为int时，可能丢失有效位数。
- ⑵ f＝＝(float)(int) f
- × ∵float有小数部分，∴当f转换为int 再转换为float时，可能丢失小数部分。

- ⑶ i＝＝(int)(double) i
- √ ∵ double比int精度高范围大，∴当i转换为double再转换为int时，数值不变。
- ⑷ f＝＝(float)(double) f
- √ ∵ double比float精度高范围大，∴当f转换为double 再转换为float时，数值不变。
- ⑸ d＝＝(float) d
- × ∵ double比float精度高范围大，∴当d转换为float时，数值可能改变。
- ⑹ f＝＝－(－f)
- √ ∵ 浮点数取负就是简单地将数符取反。

- ⑺ (d＋f) －d＝＝f
- × ∵ double比float精度高范围大， ∴当d与f进行计算时，有可能因为对阶的原因使f的有效位丢失，而使计算出现误差。
- 例如当d＝1.79×10^{308}，f＝1.0时，左边的的计算由于(d＋f) 的对阶，而使f＝0，因此结果为0，但右边结果为1。

从数据表示的角度：C语言的数据类型是“多变长度”，是“不安全不可靠的数据”，也就是说：C语言是“不可靠的语言”

2.7.2 数据的存储方式

- 在计算机中存储数据时，二进制的0/1串从低位到高位可以有不同的存放顺序。需要规定数据的最高位和最低位，以便避免歧义。
- 最高有效位（MSB， Most Significant Bit）
  数据的最高位
- 最低有效位（LSB，Least Significant Bit）
- 数据的最低位
- 例：对于带符号数，符号位是MSB，数值部分的最低位为LSB。

字节排列顺序

- 计算机在访问多字节数据时，对每个数据只会给出一个地址，然后按规定顺序访问数据的各个字节。
- 例：一个字符串数据在按字节编址的主存中存放时，既可以从低位字节向高位字节的顺序存放，也可从高位字节向低位字节的顺序存放。
- 当访问某一占8个字节的字符串数据时，若给出该数据所在的内存地址为1000H，则必须规定1000H对应的是该数据8个字节中的哪个字节单元以及应该按什么顺序访问这8个字节。

大端排序和小端排序

- 大端排序方式（big endian）
- 数据的最高有效字节MSB存放在低地址单元中，最低有效字节LSB存放在高地址单元中。

- 小端排序方式（little endian）
- 数据的最高有效字节 MSB存放在高地址单元中，最低有效字节LSB存放在低地址单元中。

- 例：IF A>B THEN READ (K) 这一字符串包括空格在内共有20个字符。
- 在按字节编址的内存中存放，可以按照从低地址字节单元到高地址字节单元或从高地址字节单元到低地址字节单元的顺序存放。

从低字节向高字节顺序存放（大端方式）

A+0H	I
A+1H	F
A+2H	
A+3H	A
A+4H	>
A+5H	B
A+6H	
A+7H	T
A+8H	H
A+9H	E

A+AH	N
A+BH	
A+CH	R
A+DH	E
A+EH	A
A+FH	D
A+10H	
A+11H	(
A+12H	K
A+13H	)

按字节编址

从高字节向低字节顺序存放（小端方式）

A+0H	)
A+1H	K
A+2H	(
A+3H	
A+4H	D
A+5H	A
A+6H	E
A+7H	R
A+8H	
A+9H	N

A+AH	E
A+BH	H
A+CH	T
A+DH	
A+EH	B
A+FH	>
A+10H	A
A+11H	
A+12H	F
A+13H	I

按字节编址

不同的机器、不同操作系统之间数据传输也有此类问题！

- 例：设某32位数据12345678H，连续存放在以4000H开头的4个字节单元中，其按大端方式和小端方式，该数据在内存中的存放形式分别如下所示：

内存地址	大端方式存放内容	小端方式存放内容
4000H	12H	78H
4001H	34H	56H
4002H	56H	34H
4003H	78H	12H

- 例：设某机器指令ADD eax 0A049468H在内存中存放的形式如图所示，存放的起始地址是0483BDH，判断其存放采用的是什么方式。

- 答：根据数据的存放顺序可知采用的是小端排序方式。

确定系统数据的字节排列顺序

- 每个计算机系统在处理和保存数据时都需要确定其字节的排列顺序。
- 有的系统采用大端方式，有的系统采用小端方式，还有的系统既能工作于小端方式又能工作于大端方式，只需要在系统加电启动时选择确定采用小端还是大端方式即可。
- 注意：在字节排列顺序不同的系统之间进行数据通信时，必须按照规定进行顺序转换。
- 在调试底层机器级程序时，要清楚每个数据的字节排列顺序，以便将正确地实现机器数与真值之间的转换。