

南京大学 计算机科学与技术系

Department of Computer Science & Technology, NJU

计算机中的信息表示

补充阅读材料

How a computer represents information internally?



刘奇志

进制

一个数可以用不同的进制来表示。常用的进制有：

- 十进制 (0~9, 逢十进一)
- 二进制
 - 二进制 (0~1, 逢2进一)
 - 八进制 (0~7, 逢8进一)
 - 十六进制 (0~9、A~F, 逢16进一)

例如，对于十进制数：29

- 二进制表示为：11101
- 八进制表示为：35
- 十六进制表示为：1D

二进制数的运算

$$\begin{array}{r} (1 1 0)_2 \\ + (1_1 1_1 1 _1 1)_2 \\ \hline (1 1 0 0)_2 \end{array}$$

$$\begin{array}{r} (3)_8 \\ + (3_1)_8 \\ \hline (7)_8 \end{array}$$

$$\begin{array}{r} (1)_{16} \\ + (1_1)_{16} \\ \hline (3)_{16} \end{array}$$

二进制与十进制之间的转换（整数）

二进制转成十进制（29）

- $(11101)_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 29$
- $(35)_8 = 3 \times 8^1 + 5 \times 8^0 = 29$
- $(1D)_{16} = 1 \times 16^1 + 13 \times 16^0 = 29$

十进制（29）转成二进制

2		29	
2		14	1
2		7	0
2		3	1
2		1	1
		0	1

（二进制）

8		29	
8		3	5
		0	3

（八进制）

16		29	
16		1	13(D)
		0	1

（十六进制）

二进制与十进制之间的转换（小数）

二进制转成十进制

- $(0.1101)_2 = 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0.8125$

十进制转成二进制

- 乘以2取整数位，直到乘积为整

	0.8125	× 2
	1.625	× 2
	1.25	× 2
	0.5	× 2
↓	1.0	

思考

● $(0.1)_{10}$ 转成二进制是多少？能精确地用二进制表示？

不能。 $0.1_{10} = \underline{0.000110011\dots}_2$ 0.099609375_{10}

有些十进制小数不能精确表示成二进制小数。

二进制与八、十六进制之间的转换

- 整数：从低位向高位每三/四位一组，高位不足补0
- 小数：从高位向低位每三/四位一组，低位不足补0

$$\begin{aligned} & (\quad 11101.1101)_2 \\ = & (\quad 011 \ 101.110 \ 100)_2 = (35.64)_8 \\ = & (\ 0001 \ 1101.1101)_2 = (1D.D)_{16} \end{aligned}$$

机器数

❁ 真值：在数值前面用“+”号表示正数，“-”号表示负数的带符号二进制数。

- +1010111
- -1010111

❁ 机器数：用“0”表示符号“+”，用“1”表示符号“-”，即把真值的符号“数值化”后的二进制数（原码）。

- 8位（1字节）：01010111, 11010111
- 16位（2字节）：00000000 01010111, 10000000 01010111

溢出问题

假定用一个字节存储数据

➡ 不考虑符号

$$\begin{array}{r} (1111\ 1111) \\ + (0000\ 0001) \\ \hline 1\ (0000\ 0000) \end{array}$$

➡ 考虑符号

$$\begin{array}{r} (0111\ 1111) \\ + (0000\ 0001) \\ \hline (1000\ 0000) \end{array}$$

解决办法：加宽存储空间

原码存在的问题

❁ 问题1：0的机器数（原码）不唯一

- 以8位为例：00000000，10000000

❁ 问题2：减法运算借位不方便；改成加负数, 结果不正确

- 以8位为例：

	(0000 0011)	原码	3
-	(0000 1101)	原码	-13
<hr/>			
	(?10)	原码

	(0000 0011)	原码	3
+	(1000 1101)	原码	-13
<hr/>			
	(1001 0000)	原码	?

❁ 解决办法：补码

补码 (complement)

补码的定义

- 符号位: $+\rightarrow 0$, $-\rightarrow 1$
- 其他位:

$$[X]_{\text{补}} = \begin{cases} X & 0 \leq X < 2^{n-1} \\ 2^{n-1} - |X| & -2^{n-1} \leq X < 0 \end{cases}$$

$[X]_{\text{补}}$ 所能表示数的范围为:

$$[-2^{n-1}, (2^{n-1} - 1)]$$

$X = -10000000$ (-128)

8位:

$[X]_{\text{原}} = \text{溢出}$

$[X]_{\text{补}} = 1\ 0000000$

补码的简单求法

- 正数：同原码
- 负数：符号位同原码，其余各位取反，末位加1

真值X:	+1010111	-1010111
$[X]_{\text{原}}(8\text{位})$:	01010111	11010111
$[X]_{\text{原}}(16\text{位})$:	0000000001010111	1000000001010111
$[X]_{\text{补}}(8\text{位})$:	01010111	10101001
$[X]_{\text{补}}(16\text{位})$:	0000000001010111	1111111110101001

-
- 由于每位二进制数只有0和1两种，故 $2^{n-1} - |X|$ 就相当于是对 $|X|$ 的各位“取反加1”，“取反加1”只是对补码定义中 $2^{n-1} - |X|$ 的简单记忆法。

补码的优势

- 数据0的补码只有00000000(8位)
 - 10000000(8位)是-128的补码
- 可将减法变成加法，结果正确

	(0000 0011)	补码	3
+	(1111 0011)	补码	-13
	<hr/>		
	(1111 0110)	补码	-10

	(0000 0011)	原码	3
-	(0000 1101)	原码	-13
	<hr/>		
	(? 10)	原码	

	(0000 0011)	原码	3
+	(1000 1101)	原码	-13
	<hr/>		
	(1001 0000)	原码	?

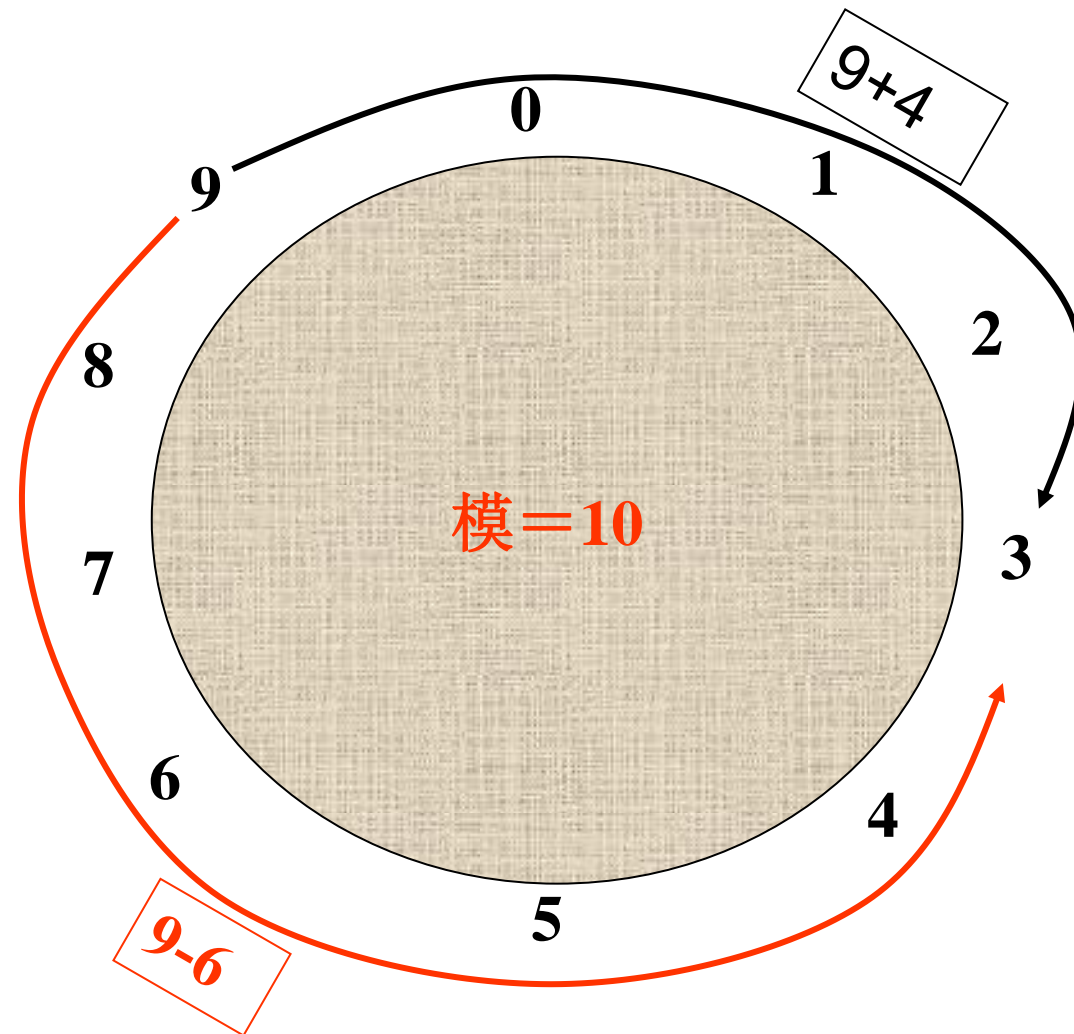
为什么补码可以将减法变成加法？

❁ 假设在圆盘上进行加减运算：

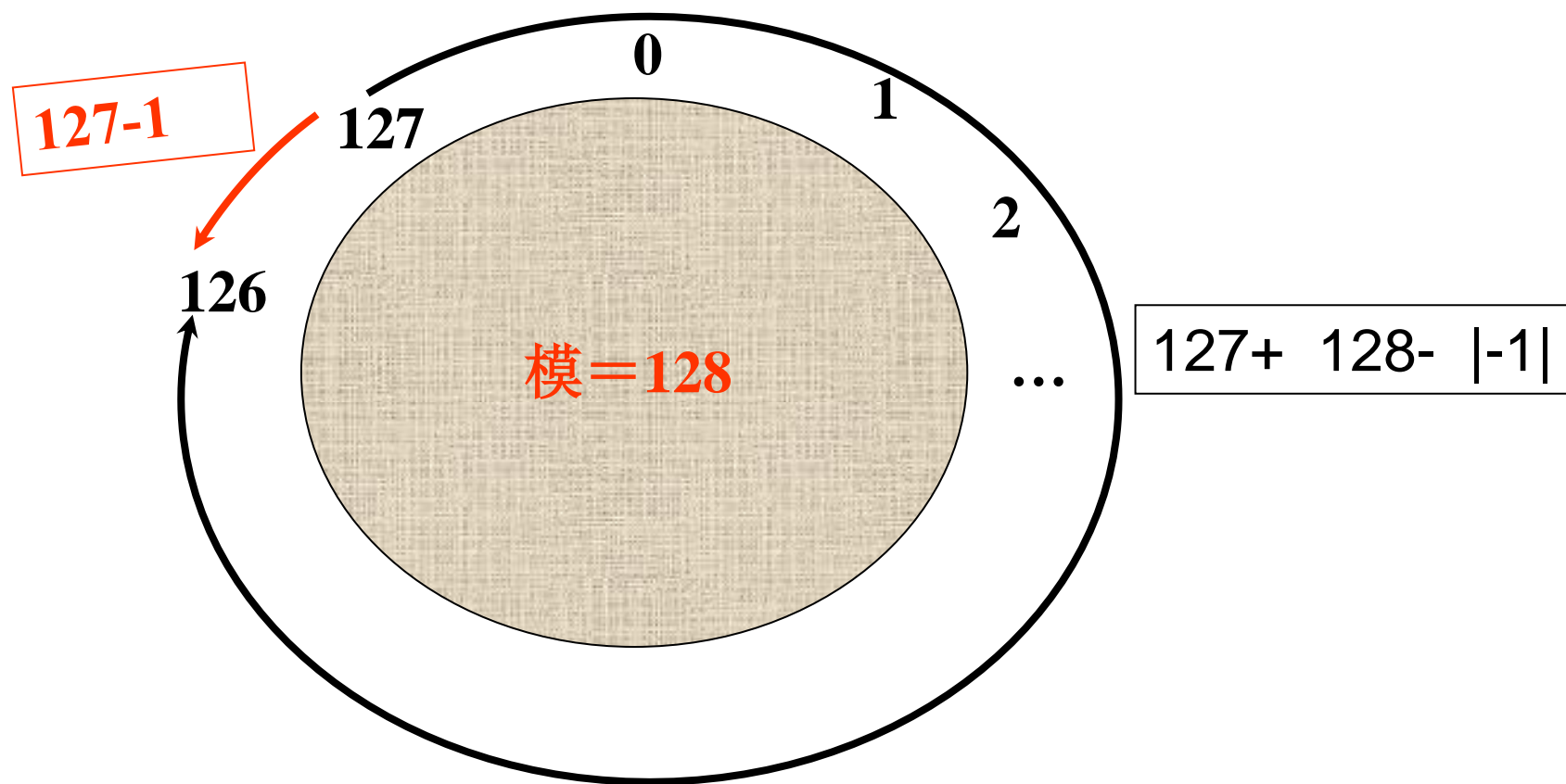
- 加法：顺时针运动找到结果；
- 减法：逆时针运动找到结果；
- 运动一周，终点回到起点。

□ 任一个数 $X(9)$ ，减去一个数 $A(6)$ ，总可以找到另一个数 $B(4)$ ，使得 $X+B=X-A$
 $A(9+4=9-6)$ ，于是称 $B(4)$ 是 $-A(-6)$ 的补数。

□ 因为逆时针运动能到达的点，顺时针运动一定也能到达，两种运动跨度之和叫做模
($= |-A| + B$)。



- 再假设这个圆盘的模改为128，则圆盘上的任一个数 $X-A$ ，相当于 $X+B$ ，其中， $B=128-|-A|$ ， B 是 $-A$ 的补数。



推广

$$\rightarrow [X]_{\text{补}} = \text{模} - |X|$$

□ 对于8位二进制数：模为1000 0000，则

$$[X]_{\text{补}} = 1000\ 0000 - |X|$$

□ 如果模为 2^{n-1} ，则

$$[X]_{\text{补}} = 2^{n-1} - |X|$$



思考：

对于内存中的二进制数：

11111111 11111111 11111111 11111111 (4字节)

显示出来的十进制数是多少？

机器中的二进制数，

可以当成一个数的原码，反推出该数为 **-2147483647**

也可以当成一个数的补码，反推出该数为 **-1**

通常约定当补码看待

计算机中的实数** (non-integral numbers)

- 定点数
- 浮点数
- BCD
- 对数数字系统 (Logarithmic number systems)
- 任意精度 (arbitrary-precision) 浮点数计算系统
- 比例计算 (rational arithmetic) 软件包 (分数)
- 其它计算软件 (Computer algebra systems) , 如Mathematica等...

定点数 (Fixed-point)

- 定点数就是约定小数点的位置固定不变。
- 定点整数（纯整数）：小数点在数的最右方
- 定点小数（纯小数）：小数点在符号位之后
- 小数点是隐含的，不另开辟空间存放
- 定点小数（纯小数）的小数点前的0也是隐含的，不另开辟空间存放

定点数可表示数值的范围

码制	定点小数		定点整数	
	最大数	最小数	最大数	最小数
原码	$1-2^{-n}$	$-(1-2^{-n})$	$2^n - 1$	$-(2^n - 1)$
补码	$1-2^{-n}$	-1	$2^n - 1$	-2^n

注意：表中的n
不包括小数点或
符号位

定点小数:

--	--

原码: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 $-(1-2^{-15})$
~0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 $2^{-1}+2^{-2}+...+2^{-15} = 1-2^{-15}$

补码: 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
~0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 $1-2^{-15}$

定点数的运算

- 参与运算的数要么都调整成整数

→ $3 + 0.5 = (3 \times 10 + 0.5 \times 10) / 10$

比例因子

- 参与运算的数要么都调整成小数

→ $3 + 0.5 = (3 / 100 + 0.5 / 100) \times 100$

比例因子

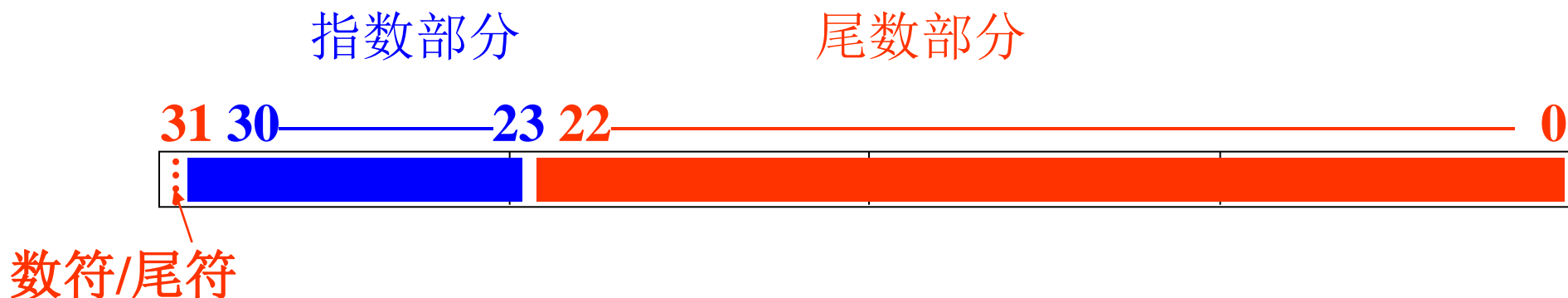
- 比例因子选择不当，会产生溢出或降低运算精度。

浮点数 (Floating point)

● $\pm M \cdot R^E$

- $\pm(S)$, sign, 数符.
- M, mantissa, 尾数, 一般为定点小数, 常用补码或原码表示, 指明有效数字的位数, 因而决定浮点数的精度.
- R, radix, 比例因子的基数, 一般为2, 8或16.
- E, exponent, 比例因子的指数, 又称为浮点数的阶码, 一般为定点整数, 常用补码或移码表示, 指明小数点在数据中的位置, 因而决定浮点数的表示范围.

单精度浮点数格式 (IEEE754)



尾数——原码，且规格化，且隐含存储1

指数——移码，是127的偏移（ $[X]_{\text{移}} = 127 + X$ ）

将-0.5按IEEE754单精度格式存储。

先将-0.5换成二进制并写成规格化： $-0.5 = -0.1_2 = -1.0 \times 2^{-1}_2$

这里 $s = 1_2$ ，M为全 0_2 ， $E = 127 + (-1) = 126 = 01111110_2$ ，

则存储形式为： **10111111 000000000000000000000000**

单精度浮点数值域

$$N_{\max} = 2^{127} \cdot (2 - 2^{-23})$$

$$N_{\min} = -2^{127} \cdot (2 - 2^{-23})$$

$$|N|_{\min} = 2^{-126}$$

指数范围为-126~127，对应移码为01~FE. (十六进制)

对于移码为00或为FF的情况，IEEE有特殊的规定：

移码为00：尾数是0，表示0

尾数不是0，表示绝对值非常小的数

移码为FF：尾数是0，表示无穷大

尾数不是0，不是一个数(NaN)

单精度浮点数的尾数用23位存储（除符号位外），

$$2^{(23)} = 8388608 ,$$

故单精度浮点数的有效位数约是7位。

双精度浮点数的尾数用52位存储（除符号位外）， $2^{(52)} = 4503599627370496,$

故双精度的有效位数约是16位。

浮点数的规格化

为什么要进行浮点数的规格化?

浮点数尾数的位数表示数的有效数位，有效数位越多，数据的精度越高。为了在浮点数运算过程中，尽可能多地保留有效数字的位数，使有效数字尽量占满尾数数位，必须经常对浮点数进行规格化操作。

规格化数的标志

尾数最高位具有非零数字。即：

若基为 R ，则 $|M| \geq 1/R$

若 $R=2$ ，则 $M=\pm 0.1bb\dots bb$ 或 $M=\pm 1.bb\dots bb$ (IEEE754格式)

若用补码表示尾数，则，规格化数的标志为：

“尾数的符号位和最高位具有不同的代码”

BCD (Binary-coded decimal)

- 用四个二进制位储存一个十进制的数码
 - 有利于二进制和十进制之间进行快捷转换(相对于一般的浮点式表示法, BCD可减少电脑作浮点运算时所耗费的时间)
 - 有利于用二进制来精确表示十进制数

- BCD码有多种形式，常用的是8421码，每一位十进数码用四位二进制数码表示，不允许出现1010~1111六种组合。

0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001
10	0001 0000		
123	0001 0010 0011		
123.4	0001 0010 0011 . 0100		

- 可以采用1010~1111六种组合中的某种组合表示正号、负号或小数点等信息。

-
- 这种编码技巧，最常用于会计系统的设计里，因为会计制度经常需要对很长的数字串作准确的计算。BCD也常用于其他需要高精度的计算场合。此外，基于BCD码设计电路，可以便于数码管显示数字，13(00001101):
0001 0011

- 采用BCD码运算时，逢16进1比逢10进1“迟钝”了6。怎么做才能得到正确的结果？

+6、60、66等修正

$$\begin{array}{r} (0111)_{\text{BCD}} \quad 7 \\ + (0110)_{\text{BCD}} \quad 6 \\ \hline (1101)_{\text{BCD}} \quad 13? \end{array}$$

$$\begin{array}{r} (1101) \\ + (0110)_{\text{BCD}} \quad 6 \\ \hline (1 \ 0011)_{\text{BCD}} \quad 13 \end{array}$$

❁ 但基于BCD码，不便于复杂计算（比如求 $\sin(x)$ ），所以很多情况下都不采用BCD码。

- ❁ 对数数字系统 (Logarithmic number systems)
- ❁ 任意精度 (arbitrary-precision) 浮点数计算系统
- ❁ 比例计算 (rational arithmetic) 软件包 (分数)
- ❁ 其他计算软件 (Computer algebra systems)，如Mathematica等...

ASCII码 (American Standard Code for Information Interchange)

美国标准信息交换码

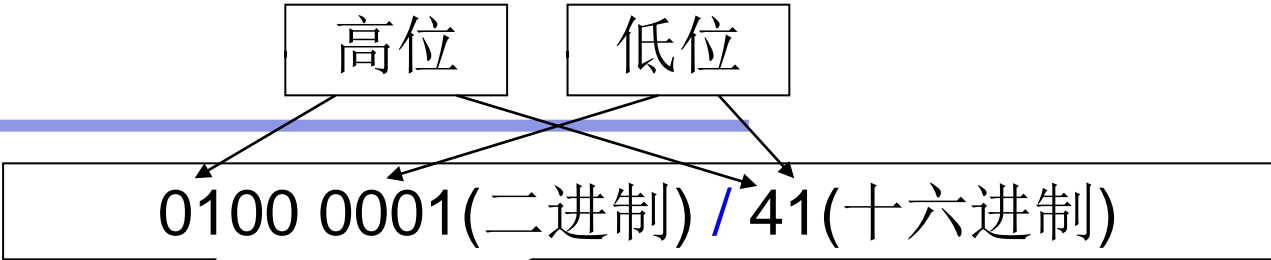
- ASCII码是常用的将字符转换成二进制数的标准代码。
- ASCII码由8位二进制数组成，其中最高位为校验位，用于传输过程检验数据正确性，其余7位二进制数表示一个字符，共有128种组合。若校验位也扩展为数据位，则有256种组合。
- 为了方便起见，常常用十六进制数或十进制数来描述二进制ASCII码，有些系统还定义了控制符来描述。
- 原来是美国的国家标准，1967年被定为国际标准。

ASCII码		字符	控制字符	意义	ASCII码		字符	控制字符	意义
十进制	十六进制				十进制	十六进制			
000	00		NULL		016	10	▶	DLE	
001	01	☺	SOH		017	11	◀	DC1	
002	02	☺	STX		018	12	↕	DC2	
003	03	♥	ETX		019	13	❗	DC3	
004	04	♦	EOT		020	14	⌋	DC4	
005	05	♣	ENQ		021	15	§	NAK	
006	06	♣	ACK		022	16	—	SYN	
007	07	•	BELL	振铃	023	17	↕	ETB	
008	08	◼	BS	退格键	024	18	↑	CAN	
009	09		HT	定位键	025	19	↓	EM	
010	0A		LF	line feed	026	1A	→	SUB	档案结束
011	0B	♂	VT	home	027	1B	←	ESC	escape
012	0C	♀	FF	form feed	028	1C	L	FS	向右键
013	0D		CR	carriage return	029	1D	↔	GS	向左键
014	0E	♪	SO		030	1E	▲	RS	向上键
015	0F	⚙	SI		031	1F	▼	US	向下键

ASCII 码		字符		ASCII 码		字符		ASCII 码		字符		ASCII 码		字符
十进制	十六进制			十进制	十六进制			十进制	十六进制			十进制	十六进制	
032	20			056	38	8		080	50	P		104	68	h
033	21	!		057	39	9		081	51	Q		105	69	i
034	22	"		058	3A	:		082	52	R		106	6A	j
035	23	#		059	3B	;		083	53	S		107	6B	k
036	24	\$		060	3C	<		084	54	T		108	6C	l
037	25	%		061	3D	=		085	55	U		109	6D	m
038	26	&		062	3E	>		086	56	V		110	6E	n
039	27	'		063	3F	?		087	57	W		111	6F	o
040	28	(064	40	@		088	58	X		112	70	p
041	29)		065	41	A		089	59	Y		113	71	q
042	2A	*		066	42	B		090	5A	Z		114	72	r
043	2B	+		067	43	C		091	5B	[115	73	s
044	2C	,		068	44	D		092	5C	\		116	74	t
045	2D	-		069	45	E		093	5D]		117	75	u
046	2E	.		070	46	F		094	5E	^		118	76	v
047	2F	/		071	47	G		095	5F	_		119	77	w
048	30	0		072	48	H		096	60	`		120	78	x
049	31	1		073	49	I		097	61	a		121	79	y
050	32	2		074	4A	J		098	62	b		122	7A	z
051	33	3		075	4B	K		099	63	c		123	7B	{
052	34	4		076	4C	L		100	64	d		124	7C	
053	35	5		077	4D	M		101	65	e		125	7D	}
054	36	6		078	4E	N		102	66	f		126	7E	~
055	37	7		079	4F	O		103	67	g		127	7F	

ASCII码			ASCII码			ASCII码			ASCII码		
十进制	十六进制	字符	十进制	十六进制	字符	十进制	十六进制	字符	十进制	十六进制	字符
128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
129	81	ü	161	A1	í	193	C1	ł	225	E1	β
130	82	é	162	A2	ó	194	C2	ṽ	226	E2	γ
131	83	â	163	A3	ú	195	C3	ṽ	227	E3	π
132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	ñ	197	C5	†	229	E5	σ
134	86	â	166	A6	ª	198	C6	‡	230	E6	μ
135	87	ç	167	A7	º	199	C7	‡	231	E7	τ
136	88	ê	168	A8	¿	200	C8	Ł	232	E8	φ
137	89	ë	169	A9	ƒ	201	C9	Œ	233	E9	θ
138	8A	è	170	AA	¬	202	CA	ℒ	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	⌈	235	EB	δ
140	8C	î	172	AC	¼	204	CC	‡	236	EC	ε
141	8D	ì	173	AD	ı	205	CD	=	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	‡	238	EE	€
143	8F	Å	175	AF	»	207	CF	⊥	239	EF	∩
144	90	É	176	B0	⋮	208	D0	ℒ	240	F0	≡
145	91	æ	177	B1	⋮	209	D1	⌈	241	F1	±
146	92	Æ	178	B2	⋮	210	D2	⌈	242	F2	≥
147	93	ô	179	B3		211	D3	ℒ	243	F3	≤
148	94	ö	180	B4	†	212	D4	ℒ	244	F4	┌
149	95	ò	181	B5	‡	213	D5	ƒ	245	F5	└
150	96	û	182	B6	‡	214	D6	ƒ	246	F6	÷
151	97	ù	183	B7	⌈	215	D7	‡	247	F7	≈
152	98	ÿ	184	B8	⌈	216	D8	‡	248	F8	°
153	99	Ö	185	B9	‡	217	D9	┘	249	F9	•
154	9A	Ü	186	BA		218	DA	ƒ	250	FA	·
155	9B	ƒ	187	BB	⌈	219	DB	■	251	FB	√
156	9C	f	188	BC	┘	220	DC	■	252	FC	n
157	9D	¥	189	BD	┘	221	DD	■	253	FD	²
158	9E	₤	190	BE	┘	222	DE	■	254	FE	■
159	9F	f	191	BF	┘	223	DF	■	255	FF	

常用字符的ASCII码值



ASCII码值 十进制	字符	ASCII码值十 进制	字符	ASCII码值十 进制	字符
0	(null)	65	A	97	a
10	换行	66	B	98	b
32	空格
48	0	90	Z	122	z
49	1	91	[123	{
...	...			127	DEL
57	9				

其它码制

- 汉字字符集的个数远远超过256，可以用2个字节表示的编码（65536种组合）
 - GB2312（简体）
 - Big5（繁体）
- 国际通用大字符集
 - Unicode
- ...
- 它们一般都兼容ASCII码

Unicode (*Universal Coded Character Set*)

- 使用16位的编码空间，也就是每个字符占用2个字节。这样理论上一共最多可以表示 2^{16} 即65536个字符。基本满足各种语言的使用。实际上当前版本的Unicode尚未填满，保留了大量空间作为特殊使用或扩展。
- UTF-8 (Unicode Transformation Format - 8 - bit) 是一种根据 Unicode 标准定义的可变长度字符编码。兼容ASCII码。由 Ken Thompson和 Rob Pike于1992年创建。现在已经标准化为RFC 3629。UTF-8用1到6个字节编码Unicode字符。用在网页上可以统一页面显示中文简体繁体及其它语言（如英文，日文，韩文）。

Thanks!

