第20章　ASCII码和字符映射

数字计算机存储器按位存储，所以，需要在计算机上处理的信息必须按位的形式存储。我们已经知道如何用位来表示数和机器码，下一个问题是如何用它来表示文本。毕境世界上大量堆积的信息是文本形式的，就像装满图书馆的书、杂志和报纸。尽管我们最终要用计算机来存放声音、图像和电影信息，但我们还是以较容易的文本存放开始。

为了以数字形式表示文本，必须开发一些系统使得系统里的每一个字母有唯一的编码。文本中也存在数字和标点符号，所以也必须有它们的编码。简单地说，所有的字母、数字和符号都要编码，这样的系统叫作字符编码集，每一个编码叫作字符编码。

第一个问题是：这些编码需要多少位？这并不是容易回答的问题。

当考虑用位表示文本的时候，需要切合实际。我们习惯于看到书中、报刊、杂志上精美的文本格式，段落按照相同的间隔整齐地分成一行一行，但这些并不是文本的本质。当我们在杂志上看到一个小故事，几年后在一本书中又看到同样故事的时候，我们不会因为书中文本间距的不同而认为是不同的故事。

换句话说，不要以这种印刷成行列的二维格式来看待文本，应该把文本看成是一维的字母、数字和标点符号流，此外，也许还有额外的编码用来表示一段的结束和另一段的开始。

再来看看，如果在杂志上看到一个故事，后来又在书中看到同样的故事但字样有些不同，这是一个大问题吗？如果杂志上的写法为

Call me Ishmael

而书中的写法为

```
Call me Ishmael
```

这些差别难道是我们真正关心的吗？恐怕不是。印刷样式是微妙地影响了文本的观感，但故事本身并没有因为样式的改变而不同。样式可以经常修改，但不会带来什么影响。

接下来另外一个简化问题的方法是：用平版的文本。没有斜体，没有粗体，没有下划线，没有颜色，没有空心体，没有上下标，没有音调标记，没有 Å、 é 、ñ、ö 等符号，只有99%英语文本里纯粹的拉丁字母。

在对摩尔斯电码和布莱叶盲文的早期研究中，可以看到如何将字母字符表示成二进制的形式。尽管这些系统在特定的场合应用地很好，但用到计算机里都有一些问题。例如：摩尔斯电码是宽度可变的编码：对常用的字符采用短编码，对不常用的字符采用长编码。这样的编码系统适用于电报，但对计算机来说却不合适。另外，摩尔斯电码对字母的大小写没有区分。

布莱叶盲文是宽度固定的编码，很适合计算机。每一个字符由 6位表示，也可以区分大小写，尽管它是用特殊的escape码来区分的，该代码表明下一个字符为大写。这也就是说，每个首部字符需要两个代码而不是一个。数字用 shift码表示，在这个特定的代码后紧跟的代码被看作表示数字，直到又一个shift码将其转换到字符状态。

我们的目标是开发一个字符编码集，使得像如下的句子

```
I have 27 sisters。
```

可以用一串代码来表示，每一个代码具有一定的位数。一些代码用来表示字母，一些表示标点符号，一些表示数字。甚至有代码来表示字间的空格。上面的句子中有 18个字符（包括字间空格），这样一个句子的连续字符代码常称作文本串。

在文本串里，用代码来表示数字(如27)似乎很奇怪，因为前面许多章里已讲过用位来表示数字。我们可能会用简单的二进制数10和111作为该句中2和7的代码，但用在这里是不合适的。该句中，字符2和7可像英文作品中出现的任何一种字符一样来看待，它们可能具有与它们的实际值毫不相干的字符代码。

也许最经济的字符编码是5位编码，它首先用于1874年的电报机，是由法国电报服务公司职员Emile Baudot 发明的。他的编码1877年被服务公司采纳，后来由Donald Murray修改并在1931年被 CCITT，即现在的国际电联 (ITU) 标准化。该编码的正式名称是国际电报字母表NO.2或ITA-2，在美国通常称为Baudot，尽管更科学的叫法为Murray编码。

在20世纪，Baudot经常用于电传打字机。 Baudot电传打字机有一个键盘，除了只有 30个键和一个间隔棒外，有些像打字机。电传打字机的键实际上是转换器，它产生二进制代码并且通过电传打字机的输出电缆一位紧接一位地传送出去。电传打字机也有打印机制，从电传打字机的输入电缆输入的代码触发电磁铁在纸上打印出字符 。

由于Baudot是5位编码，所以总共只有32个代码，这些代码的十六进制值范围从00h～1Fh。下表是32个 代码所对应的字母表中的字符：

十六进制码	Baudet字符	十六进制码	Baudet字符
00		10	E
01	T	11	Z
02	*Carriage Return*(回车)	12	D
03	O	13	B
04	*Space*(空格)	14	S
05	H	15	Y
06	N	16	F
07	M	17	X
08	*Line Feed*(换行)	18	A
09	L	19	W
0A	R	1A	J
0B	G	1B	*Figure Shift*(数字转义)
0C	I	1C	U
0D	P	1D	Q
0E	C	1E	K
0F	V	1F	*Letter Shift*(字符转义)

代码00h没有指定。其余的31个代码中，26个指定给字母表中的字符， 5个用斜体字或短语表示出来了。

代码04h是空格代码，用来分隔不同的字；代码 02h和08h表示回车和换行。这些术语来自于电传打字机。当在电传打字机上打字并且到了一行的末尾时，按下一个杠杆或按钮来完成两件事情。第一，使打印头回到开始处，以便从纸的左边开始打印下一行，这是回车。第二，移动打印头紧接至刚完成的那一行的下一行，这是换行。在 Baudot中，独立的键产生这两个代码。打印的时候，Baudot电传打字机响应这两个代码，完成相应动作。

在Baudot系统里，如何表示数字和标点符号呢？这就是代码 1Bh的作用，在表中标识为数字转义。在数字转义代码之后，所有的代码序列被看作是数字或标点符号，直到遇到字符转

义代码(1Fh)再返回到字符状态。下表是数字和标点符号的代码。

十六进制码	Baudot字符	十六进制码	Baudot字符
00		10	3
01	5	11	+
02	*Carriage Return*	12	*Who Are You?*
03	9	13	?
04	*Space*	14	‘
05	#	15	6
06	,	16	$
07	。	17	/
08	*Line Feed*	18	-
09	)	19	2
0A	4	1A	*Bel* (响铃)
0B	&	1B	*Figure Shift*
0C	8	1C	7
0D	0	1D	1
0E	：	1E	(
0F	=	1F	*Letter Shift*

实际上，ITU没有定义代码05h、0Bh和16h，而是保留为“国家使用”，这个表里列出的是美国的用法。这些代码在某些欧洲国家语言中用作重音符号。响铃代码用来敲响电传打字机上能听见的铃声；“Who Are You”代码激活一种机制，用它电传打字机能识别自己。

像摩尔斯电码一样，这5位编码不能区别大、小写。语句

```
I SPENT $25 TODAY.
```

由下面的十六进制数据流来表示：

```
0C 04 14 0D 10 06 01 04 1B 16 19 01 1F 04 01 03 12 18 15 1B 07 02 08
```

注意三个转义代码:1Bh在数字的前面，1Fh在数字的后面，最后一部分之前又有1Bh。该行代码用回车、换行代码来结束。

然而，如果一行两次传送该数据流到电传打印机，将会出现以下情形：

```
I SPENT $25 TODAY.
8‘03,5 $25 TODAY.
```

这是怎么回事？打印机接收到的上一行的最后一个转义代码是数字代码，所以第二行开始的代码被解释成数字。

类似这样的问题是采用转义代码所产生的典型的令人烦恼的结果。尽管Baudot是很经济的编码，但人们可能更想采用能唯一表示字符或标点符号且对大、小写进行区分的代码。

如果想确定比Baudot更好的编码系统需要多少位，只需把各种符号加起来：大小写字母需52个代码，0～9数字需10个代码，这已经有62个，加上一些标点符号，则超过了64个代码，这意味着需要多于6位的编码。但是距离128个字符数，似乎还有足够的余地。如果超过128个字符，则需要8位编码。

所以答案应该是7。如果不用转换代码来区分大、小写，那么英文里应该用7位来表示字符。

这些字符编码都是什么呢？当然，我们可以随心所欲地编码。如果打算自己制造计算机且计算机的每一个硬件都由自己制造，自己编程且不把自己所造的计算机去与任何其他计算机连接，则可以构造自己的编码，所要做的就是给每一个字符一个唯一的编码。

但是因为很少有独立制造和使用计算机这种情形发生，所以通常是大家遵循并使用同一

编码。这样制造出来的计算机就可以与其他计算机兼容，并且可以交换文本信息。

我们可能也不应该随意编码，例如，当在计算机上处理文本时，如果字母表上的字符是按顺序进行编码的，则会带来很多好处，其码这样的顺序使得按字母排序和分类更容易一些。

幸运的是，我们已经有了这样一个标准，即美国信息交换标准代码，简写为 ASCII码。它1967年正式公布，此后一直是计算机工业界最为重要的标准。除了一个大的例外（在后面讲到），可以肯定的是，无论什么时候处理文本，总会以某种方式涉及到 ASCII码。

ASCII码是7位编码，用二进制代码 0000000～1111111,即十六进制代码 00h～7Fh来表示。让我们来看 ASCII码，但不要从最开始看，因为前 32个代码比其余代码理解起来要困难一些。从第二批的32个代码开始讲起，它包括标点符号和 10个数字。下表列出了它们的十六进制代码及对应的字符：

十六进制码	ASCII字符	十六进制码	ASCII字符
20	*space*	30	0
21	!	31	1
22	“	32	2
23	#	33	3
24	$	34	4
25	%	35	5
26	&	36	6
27	‘	37	7
28	(	38	8
29	)	39	9
2A	*	3A	:
2B	+	3B	;
2C	,	3C	<
2D	-	3D	=
2E	.	3E	>
2F	/	3F	?

注意20h是空格符，用来分隔单词和句子。

接下来的32个代码包括大写字母和一些附加的标点符号。除 @符号和下划线之外，这些符号在打字机上不经常出现，它们出现在标准的计算机键盘上：

十六进制码	ASCII字符	十六进制码	ASCII字符
40	@	50	P
41	A	51	Q
42	B	52	R
43	C	53	S
44	D	54	T
45	E	55	U
46	F	56	V
47	G	57	W
48	H	58	X
49	I	59	Y
4A	J	5A	Z
4B	K	5B	[
4C	L	5C	\
4D	M	5D	]
4E	N	5E	^
4F	O	5F	-

接下来的32个字符包括所有小写字母和一些附加的标点符号，也是在打字机上不常出现的：

十六进制码	ASCII字符	十六进制码	ASCII字符
60	`	70	p
61	a	71	q
62	b	72	r
63	c	73	s
64	d	74	t
65	e	75	u
66	f	76	v
67	g	77	w
68	h	78	x
69	i	79	y
6A	j	7A	z
6B	k	7B	{
6C	l	7C	|
6D	m	7D	}
6E	n	7E	~
6F	o		

注意该表中少了与7Fh对应的最后一个字符。如果你一直在统计，这三个表总共列出了95个字符。因为ASCII码是7位编码，可以有128个代码，所以还有33个代码可用。下面简单地讲一下这些代码。

文本串：

```
Hello, you!
```

可以表示成ASCII码的十六进制形式

```
48 65 6C 6C 6F 2C 20 79 6F 75 21
```

注意除了字母代码以外，还有逗号（代码2C）、空格（代码20）和感叹号（代码21）。下面是另一短句：

```
I am 12 years old.
```

用ASCII码表示为：

```
49 20 61 6D 20 31 32 20 79 65 61 72 73 20 6F 6C 64 2E
```

注意句中数字12表示成十六进制数31h和32h，分别是数字1和2的ASCII码。当数字12作为文本流的一部分时，它不应该被表示成十六进制码01h和02h，或者BCD码12h，或者十六进制代码0Ch。这些代码在ASCII码里都表示的是其他意思。

ASCII码表示的大写字母与其对应的小写字母的ASCII码值相差20h，这使得编写某些程序代码更为容易，如：把一个字符串变成大写。假设在内存的某个区域存放有字符串，一个字节放一个字符。下面的8080子程序假设字符串的首地址存放在寄存器HL中；寄存器C存放有字符串的长度，即字符串中的字符个数：

```
Capitalize: MOV A,C        ;C=number of characters left (C为余下的字符数)
            CPI A,00h      ;Compare with 0 (与0比较)
            JZ  AllDone    ;If C is 0, we're finished (如果C为0，则结束)

            MOV A ,[HL]    ;Get the next character (取下一个字符)
            CPI A,61h      ;Check if it's less than 'a' (判断是否小于'a')
```

```
            JC SkipIt       ;If so,ignore it(如果是，则跳过)

            CPI A,7Bh       ;Check if its greater than 'z'（判断是否大于'z'）
            JNC SkipIt      ;If so ,ignore it(如果是，则跳过)

            SBI A,20h       ;It's lowercase,so subtract 20h（是小写，则减20h）
            MOV [HL],A      ;Store the character(保存字符)

SkipIt:  INX HL             ;Increment the text address（字符地址加1）
            DCR C           ;Decrement the counter（计数器减1）
            JMP Capitalize ;Go back to the top（返回到程序开始处）

AllDone:   RET
```

从小写字母减去20h转换成大写字母的语句可以用下面的语句代替：

```
ANI A,DFh
```

ANI指令是一个“与”立即数的操作，它把累加器中的数值与DFh（即二进制数11011111）执行按位“与”操作，即把两个数的对应位进行“与”操作“与”操作保留A中的所有位，除了从左边数第3位被置成0。把这个位设置为0也即把ASCII码表示的小写字母转换成大写字母。

上面列出的95个代码也称作图形字符，因为它们可以显示出来。ASCII码还包括33个控制字符，它们不能显示出来但表示执行某一特定功能。鉴于完整性，这里列出了33个控制字符，即使它们很难理解也不要担心。在ASCII码公布以后，更多地是想把它们用在电传打字机上，现在许多代码已经很少见到了。

十六进制码	缩写词	控制字符名称
00	NUL	空
01	SOH	标题开始
02	STX	文本开始
03	ETX	文本结束
04	EOT	传输结束
05	ENQ	询问
06	ACK	应答
07	BEL	响铃
08	BS	退格
09	HT	水平制表
0A	LF	换行
0B	VT	垂直制表
0C	FF	换页
0D	CR	回车
0E	SO	移出
0F	SI	移入
10	DLE	转义
11	DC1	设备控制1
12	DC2	设备控制2
13	DC3	设备控制3
14	DC4	设备控制4
15	NAK	否定应答
16	SYN	同步
17	ETB	传输块结束

(续)

十六进制码	缩写词	控制字符名称
18	CAN	作废
19	EM	载体结束
1A	SUB	替代字符
1B	ESC	扩展
1C	FS	文件分隔或信息分隔4
1D	GS	组分隔或信息分隔3
1E	RS	记录分隔或信息分隔2
1F	US	单元分隔或信息分隔1
7F	DEL	删除

控制字符可以与图形字符混合使用来设置一些基本的文本格式。这很容易理解，想像一下诸如电传打字机或简单打印机之类的设备，它们对ASCII码流作出的响应是在纸上打印出字符。设备的打印头通过打印一个字符并向右移动一格来对ASCII码作出响应。上面这些很重要的控制字符就用来改变这种通常的动作。

例如：看以下的十六进制字符串

```
41  09  42  09  43  09
```

09字符是一个水平制表符，简称Tab。假设打印页面上所有的水平字符位置是从0开始，Tab的作用是在下一个水平位置即8的倍数处开始打印下一个字符，如下所示：

```
A       B       C
```

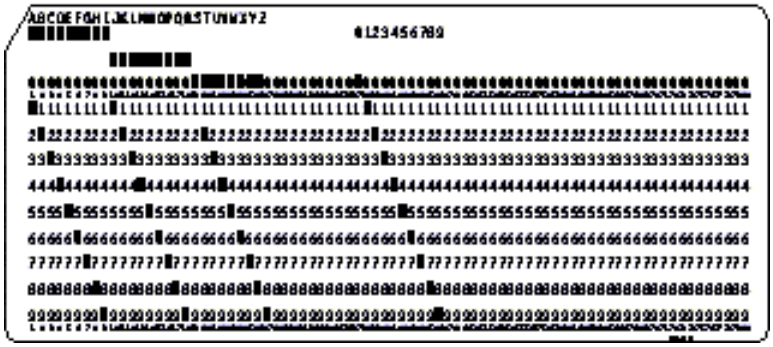
这是保持字符按列对齐的简便方法。

换页符（12h）的作用是使打印机跳过当前页开始打印下一页。

退格符用来在一些旧的打印机上打印复合字符，例如，假设计算机要控制电传打字机以重音标记来打印小写字母e，即è，可以通过用十六进制码65 08 60来实现。

最重要的控制字符是回车和换行，它们与Baudot码中的回车换行符意义相同。在打印机中，回车符使打印头移到打印页面的左边，换行符使打印头移到下一行，用两个代码通常表示从新的一行开始。单独使用回车符可以用来在一个已有的行上打印，单独使用换行符可以用来跳到当前位置的下一行而不移到左边。

尽管ASCII码是计算机世界的主要标准，但在许多IBM大型机系统上却没有采用。在System/360系统中，IBM研制了自己的8位字符编码，即扩展的BCD交换代码EBCDIC。该编码是早期的BCDIC 6位编码的扩展，从IBM穿孔卡片使用的代码演变而来。穿孔卡片可以存放80个文本字符，这种模式由IBM 1928年首先引入并且用了50多年。



当考察穿孔卡片与相关的8位EBCDIC字符编码的关系时，需要记住的是，在若干种不同技术的支持下这种代码已经经历了好几代的演变。正因为如此，不要指望从中发现太多的逻辑性和一致性。

穿孔卡片中，字符编码由一列上穿出的一个或多个矩形孔的组合而形成，字符本身通常在接近卡片的上边沿处打印出来。下面的10行由数字标识，分别是第0行、第1行直到第9行。在第0行之上没有数字的行为第11行，最上边的行为第12行，没有第10行。

以下是一些常用的IBM穿孔卡片术语：行0～9称作数字行或数字穿孔，行11和12称作区域行或区域穿孔。有一些IBM穿孔卡片也会带来混淆，把行0和9看作是区域行而不是数字行。

一个8位EBCDIC字符编码由高半字节（4位）与低半字节组成。低半字节是与字符的数字穿孔一致的BCD码，高半字节是与区域穿孔一致的编码。回忆一下第19章的BCD编码标准，它是用二进制编码十进制数，即用4位编码来代表十进制的0～9。

对数字0～9，没有区域穿孔，没有区域穿孔对应的高半字节是1111，低半字节是数字穿孔的BCD码。下面是0～9的EBCDIC编码：

十六进制码	EBCDIC字符
F0	O
F1	1
F2	2
F3	3
F4	4
F5	5
F6	6
F7	7
F8	8
F9	9

对大写字母，如果只有第12行有穿孔，则用1100来标识；如果只有第11行有穿孔，则用1101来标识；如果只有第0行有穿孔，则用1110来标识。大写字母的EBCDIC编码为：

十六进制码	EBCDIC字符	十六进制码	EBCDIC字符	十六进制码	EBCDIC字符
C1	A	D1	J		
C2	B	D2	K	E2	S
C3	C	D3	L	E3	T
C4	D	D4	M	E4	U
C5	E	D5	N	E5	V
C6	F	D6	O	E6	W
C7	G	D7	P	E7	X
C8	H	D8	Q	E8	Y
C9	I	D9	R	E9	Z

注意这些编码的编号次序。在一些场合，当用EBCDIC文本编写程序的时候，这些次序有时还真令人头痛。

小写字母与大写字母的数字穿孔相同但区域穿孔不同。小写a～i的第12行和第0行有穿孔，相应的区域代码为1000；j～r的第12行和第11行有穿孔，区域代码为1001；s～z的第11行和第0行有穿孔，区域代码为1010。小写字母的EBCDIC编码为：

十六进制码	EBCDIC字符	十六进制码	EBCDIC字符	十六进制码	EBCDIC字符
81	a	91	j		
82	b	92	k	A2	s
83	c	93	l	A3	t
84	d	94	m	A4	u
85	e	95	n	A5	v
86	f	96	o	A6	w
87	g	97	p	A7	x
88	h	98	q	A8	y
89	i	99	r	A9	z

当然，标点符号和控制字符也有EBCDIC编码，但对该编码系统的全面考察并不需要。

似乎IBM穿孔卡片上的每一列就足以提供12位的编码信息，每个孔代表1位，是这样吗？果真如此的话，可以用穿孔卡片上每一列12个位置中的7个来表示ASCII码的字符代码。但是，实际上，这并不合适，太多的穿孔将会影响到卡片物理状态的平直。

EBCDIC中的许多8位码没有定义，建议采用ASCII码的7位编码是合理的。当ASCII码研制出来的时候，存储器非常昂贵，于是一些人感到ASCII码应该用6位码并且采用转义字符来区分大小写用以节约存储器。这种观点没有被接受，相反，人们认为ASCII码应该是8位编码，因为即使在当时人们也普遍认为计算机应该是按8位存储，而不是7位。当然，8位字节现在是标准的。因此，尽管ASCII码在技术上是7位编码，但它普遍是接8位值来存储的。

字节与字符之间的等价关系的确很方便，我们只需简单地通过统计字符数就可以粗略估计一个文本文件所需要的存储空间。当然，用K和M来表示计算机存储空间用得更为广泛一些。

例如，传统的8.5×11英寸的双倍空隙打印页面有1英寸的页边空白和大约27行的正文。每行约6.5英寸宽，每英寸有10个字符，这样一页共有1750个字节。单倍空隙打印页面大约是它的2倍，约3.5KB。

《NEW Yorker》杂志每页有3列，每列有60行，每行大约有40个字符，这样每页有7200个字符（或字节）。《纽约时代》每页有6列。如果页面都是文字而没有标题和图片（这是不常有的）,则每列有155行，每行大约有35个字符，从而整个页面有32 550个字符，即32KB。

硬面书每页大约500个字，平均每个字有5个字母，确切的说，应该是6个字符，因为要把字间空格统计在内，这样书的每页约3000个字符。假设平均每本书有333页(这可能是一个估计较高的数字)，则意味着平均每本书大约是1MB。

当然，每本书的差别很大：

F. Scott Fitzgerald的《The Great Gatsby》大约300KB。
J. D. Salinger的《Catcher in the Rye》大约400KB。
Mark Twain的《The Adventures of Huckleberry Finn》大约540KB。
John Steinbeck的《The Grapes of Wrath》大约1MB。
Herman Melville的《Moby Dick》大约1.3MB。
Henry Fielding的《The History of Tom Jones》大约2.25MB。
Margaret Mitchell的《Gone With the Wind》大约2.5MB。
Stephen King的《The Stand》大约2.7MB。
Leo Tolstoy的《War and Peace》大约3.9MB。
Marcel Proust的《Remembrance of Things Post》大约7.7MB。

美国国会图书馆大约有20 000万本书，总共有20万亿字符，即20TB的文本数据。（图书馆还有大量的照片和录音。）

尽管ASCII码是计算机世界里最重要的标准，但它并不是完美的。 ASCII码的最大问题在于它太倾向于美国！的确， ASCII码即使对那些以英语为主要语言的国家也几乎是不合适的。尽管ASCII码包含有美元符号，但英镑符号呢？还有许多西欧国家语言中用到的重音符号呢？更不用说在欧洲一些国家里使用的非拉丁字母，包括希腊文、阿拉伯文、希伯来文和西里尔文。此外，还有印度及东南亚国家用到的婆罗门教的手迹。而一个 7位编码又如何来处理成千上万的中文、日文、韩文笔画以及韩语音节？

在研究ASCII码的时候，也一直在考虑其他国家的需要 ,尽管没有充分考虑非拉丁字母。根据公开的ASCII码标准，10个ASCII码代码（40h、5Bh、5Ch、5Dh、5Eh、60h、7Bh、7Ch、7Dh和7Eh）可用来重新定义而为某一国家使用。另外，如果需要，数字符号（ #）可用英镑符号（£）替换，美元符号（$）可用通用货币符号（¤）替换。显而易见，只有使用包含这些替换符号的特定文本文档的所有人都知道这些变化的时候，替换符号才有意义。

由于许多计算机系统按8位来存储字符，则可以设计扩展的 ASCII码字符集来包含256个字符而不仅仅是128个。在这样的字符集里，代码 00h～7Fh定义成与ASCII码一致；代码80h～FFh可定义成表示另外的字符。这种技术已被用来定义附加的字符代码，包含重音字母及非拉丁字母。作为例子，这里有一个 96个字符的ASCII码的扩展，称之为第 1号拉丁字母表，定义的字符编码从A0h～FFh。在该表里，十六进制字符编码的高半字节由最高行给出，低半字节由左边列给出：

	A-	B-	C-	D-	E-	F-
-0		°	À	Ð	à	ð
-1	¡	±	Á	Ñ	á	ñ
-2	¢	²	Â	Ò	â	ò
-3	£	³	Ã	Ó	ã	ó
-4	¤	´	Ä	Ô	ä	ô
-5	¥	µ	Å	Õ	å	õ
-6	¦	¶	Æ	Ö	æ	ö
-7	§	·	Ç	×	ç	÷
-8	¨	¸	È	Ø	è	ø
-9	©	¹	É	Ù	é	ù
-A	ª	º	Ê	Ú	ê	ú
-B	«	»	Ë	Û	ë	û
-C	¬	¼	Ì	Ü	ì	ü
-D	-	½	Í	Ý	í	ý
-E	®	¾	Î	Þ	î	þ
-F	¯	¿	Ï	ß	ï	ÿ

代码A0h对应的字符为非断开空格。通常计算机处理格式文本是按照行和段来编排的，每一行以空格符号断开，对应的ASCII码为20h。代码A0h用来显示一个空格，但不能用来断开一行。非断开空格可以用在如“WW II”这样的文本中。代码ADh定义成软连字符，该连字符用来分开一个字中间的音节，且只在需要连接被两行分开的一个单词时才使用。

遗憾的是，近几十年来出现了许多不同的ASCII码的扩展，导致了混淆和不兼容性。ASCII码通过扩展甚至可以编码中文、日文和韩文的笔画。有一个流行的编码叫作Shift-JIS（日本工业标准），其代码81h～9Fh用来表示2字节字符编码的起始字节。以这种方法，Shift-JIS可编码约6000个额外字符。遗憾的是，Shift-JIS不是使用这种技术的唯一系统。在亚洲，还有三个很流行的双字节字符集。

双字节字符集有许多问题，不兼容性只是其中之一。另一个问题是，一些字符——特别是普通的ASCII码字符——是用1个字节编码来表示的，而成千上万的笔画则是由双字节编码来表示，从而导致使用这样的字符集很困难。

在假定会有一个特定的字符编码系统能适用于世界上所有语言的前提下，1988年，几个主要的计算机公司一起开始研究一种替换ASCII码的编码，称为Unicode。鉴于ASCII码是7位编码，Unicode采用16位编码，每一个字符需要2个字节。这意味着Unicode的字符编码范围从0000h～FFFFh，可以表示65 536个不同字符。对世界上所有可用计算机进行来通信的语言来说，有足够的扩展空间。

Unicode编码不是从零开始的，开始的128个字符编码0000h～007Fh与ASCII码字符一致。Unicode编码00A0h～00FFh也与前面讲到的对ASCII码扩展的第1号拉丁字母表一致。其他世界范围的标准也收编在Unicode中。

尽管Unicode对现有的字符编码做了明显改进，但并不能保证它能很快被人们接受。ASCII码和无数的有缺陷的扩展ASCII码已经在计算机世界中占有一席之地，要把它们逐出计算机世界并不是件容易的事。

对Unicode来说的一个实实在在的问题是，它改变了一个文本字符与1个字节存储器之间的等效关系。用ASCII码编码，《The Grapes of Wrath》这本书的大小约为1M字节；而用Unicode编码，约是2MB，这也算是采用Unicode编码付出的代价吧。