（American Standard Code for Information Interchange），简称为 ASCII 码，发音很像 ASS-key。从 1967 年正式公布至今，它一直是计算机产业中最重要的标准。不过还有一个大的例外（后面会讲到），无论何时，当你在计算机上处理文本时，总会在不经意间使用到 ASCII 码。

ASCII 码是 7 位编码，它的二进制取值范围为 0000000～1111111，对应于十六进制就是 00h～7Fh。现在我们一起来讨论下 ASCII 码，但我不建议从开始学起，因为相对于后面的编码，前 32 个编码理解起来还有一点难度。所以我们从第 2 组 32 个编码开始学习，它包括标点符号和 10 个数字。下表列出了这 32 个字符及相应的十六进制编码。

十六进制编码	ASCII 字符	十六进制编码	ASCII 字符
20	空格	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	?

值得注意的是 20h 代表空格符，它的作用是将单词或句子隔开。

接下来的 32 个编码是大写字母和一些附加的标点符号的编码。除了@符号和下画线之外，其余的符号很难在打字机上找到。它们真正出现的地方是标准计算机键盘，下表列出了这些字符及相应的十六进制编码。

十六进制编码	ASCII 字符	十六进制编码	ASCII 字符
40	@	50	P
41	A	51	Q

续表

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	_

再接下来的 32 个编码是所有小写字母和一些附加的标点符号及其对应的十六进制编码，这些字符也很少在打字机上出现。

十六进制编码	ASCII 字符	十六进制编码	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	i	7C	|
6D	m	7D	}
6E	n	7E	~
6F	o		

注意，表的最后不包括 7Fh 及其对应的字符。如果你统计一下，就会发现这三张表共涵盖了 95 个字符。由于 ASCII 码的编码长度为 7 位，所以最多可以表示 128 个编码，

这样算下来还剩 33 个编码可用。下面我们通过几个简短的例子学习一下编码。

像这样一段字符串：

Hello, you!

转换成 ASCII 码，用十六进制数表示如下：

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

这段编码中，除了普通的字符，逗号（编码 2C）、空格（编码 20）和感叹号（编码 21）容易遗漏，需要额外注意。我们再来看一个例子：

I am 12 years old.

它用 ASCII 码表示为：

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

有意思的是数字 12 的表示方法。在这段编码串中，它被表示成十六进制数 31h 和 32h，也就是数字 1 和 2 的 ASCII 码的组合。当数字 12 以文本流的身份出现时，不应该用十六进制码 01h 和 02h，或者 BCD 码 12h，或者 0Ch 来表示。因为这些编码在 ASCII 码中表示其他的意思。

在 ASCII 码中，一个大写字母与其对应的小写字母的 ASCII 码值相差 20h。这种规律大大简化了程序代码的编写，例如一段将特定的字符串变成大写的程序。假设有一个字符串存放在内存的某个区域，每个字符占据一个字节。下面是一段 8080 子程序，初始状态下字符串的首地址存放在寄存器 HL 中；寄存器 C 存放字符串的长度，也就是字符的个数。

```
Capitalize: MOV A, C        ; c 表示剩余的字符数
            CPI A, 00h      ; 与 0 进行比较
            JZ AllDone      ; 如果剩余的字符数为 0，程序结束
            MOV A, [HL]     ; 取得下一个字符
            CPI A, 61h      ; 判断 A 代表的字符的 ASCII 码是否比'a'小
            JC Skipit       ; 如果比'a'小，就跳过
            CPI A, 78h      ; 判断是否比'z'大
            JNC SkipIt      ; 如果是，则跳过
            SBI A, 20h      ; 判断是否是小写，如果是，则减 20h
            MOV [HL], A     ; 保存修改过的字符
SkipIt:     INX HL          ; 指向下一个字符
            DCR C           ; 计数器减一
```

```
            JMP Capitalize  ; 返回到程序起始处
AllDone:    RET
```

还有另外一种方法也可以将小写字母减去20h而转换成大写字母，如下所示：

```
ANI A, DFh
```

ANI指令（AND Immediate）用来“与”一个立即数。在上面这个例子中，累加器中的数值与DFh执行“按位与”操作，其中DFh转换成二进制数就是11011111。“按位与”操作就是把两个数分别转换成二进制，然后将对应的位进行“与”操作。这个例子中，除了自左向右数的第3位被置成0外，A中的其他位均被保留。通过将这一位设置为0，我们实现了将小写字母的ASCII码转换成大写字母的目的。

十六进制编码	缩　写	控制字符的含义
00	NUL	空字符
01	SOH	标题开始
02	STX	文本开始
03	ETX	文本结束
04	EOT	传输中止
05	ENQ	询问
06	ACK	应答
07	BEL	响铃
08	BS	回退
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

续表

17	ETB	块传输结束
18	CAN	取消
19	EM	媒介取消
1A	SUB	替代字符
1B	ESC	跳出
1C	FS	文件分割或信息分割 4
1D	GS	组分割或信息分割 3
1E	RS	记录分割或信息分割 2
1F	US	单元分割或信息分割 1
7F	DEL	删除

前面讲到的 95 个编码也被称为图形文字（graphic characters），因为它们可以被显示出来。其实 ASCII 码还包含 33 个控制字符（control characters），它们用来执行某一特定功能，因而不用显示出来。为了完整地讨论 ASCII 编码，下面将这 33 个控制字符也列举了出来，有一些的确很难理解，不过不用在意。其实在 ASCII 码公布以后，当时人们更多的是想把它用在电传打字机上，所以，如今其中的许多编码已经渐渐离开了人们的视线。

人们最初的想法是可以在图形字符中使用控制字符，以便对文本格式进行基本的调整。举个例子或许会帮助你更好地理解这种做法：假如有一台电传打字机或者打印机，它负责解析 ASCII 码，解析之后做出相应的操作，最后在纸上打印出字符。设备的打印头每打印一个字符就向右移动一格，通过这种方式来对 ASCII 码做出响应。而要实现这些操作就需要用到控制字符。

举个例子来讲，看看下面这个十六进制字符串：

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

编码 09 代表水平制表符，简写为 Tab。假设打印的过程中，所有水平排列字符的起始位置都为 0，Tab 的作用是在下一个水平位置即在距前一个字符的间距为字符长度 8 倍的位置打印下一个字符，如下所示：

```
A       B       C
```

这种简单有效方法使得字符可以保持按列对齐。

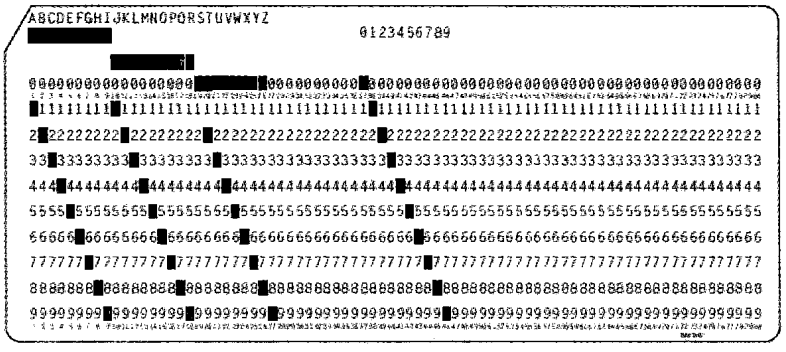
有一些控制字符甚至沿用至今，例如换页符（12h），它使得打印机跳出当前页，并

开始准备打印下一页。

回退符可以用来打印复合字符，尤其是在一些旧的打印机上。假设计算机要控制电传打字机，使其不仅打印小写字母 e，还要将其重音标记出来，即 è。我们可以使用回退符来实现，十六进制码为：65 08 60。

计算机中的回车和换行与 Baudot 码中表示的意思相同，它们可以算得上是控制符中最重要的两个符号。在打印机中，回车符使得打印头换行并转移至当前页面的最左端，换行符使打印头转移至当前位置下一行。这两种操作都使得打印头移至新的一行。回车符通常用来另起一行继续打印，换行符通常在不需要移到页面最左端而换行时使用。

尽管 ASCII 码在计算机领域可以说是一统江湖，但许多 IBM 大型机上却没有采用这种标准。例如，System /360 产品内部采用的是 IBM 自发研制的 8 位字符编码系统，也被称为扩展的 BCD 交换码（Extended BCD Interchange Code），或 EBCDIC（英文中的发音为 EBB-see-dick）。EBCDIC 是早期的 6 位 BCDIC 编码的扩展形式，BCDIC 的起源于 IBM 的打孔卡。一张打孔卡——存储容量为 80 个文本字符——1928 年由 IBM 首创并沿用了将近 50 年，它的外观如下图所示。



在考虑打孔卡与 8 位 EBCDIC 字符码的关系时，需要知道，在几代技术的影响下，这种编码也历经几十年的演变。因此，打孔卡与 EBCDIC 之间的逻辑性和一致性也逐渐消失了。

打孔卡上每一列穿出的一个或多个矩形孔代表一个字符，而这些字符一般也打印在卡片的顶部。最下面的 10 行由数字标识，自上向下分别为第 0 行、第 1 行直到第 9 行。第 0 行上面的一行通常不出现数字，称为第 11 行，顶端为第 12 行，这里没有第 10 行。

以下面列举一些 IBM 打孔卡的常用术语：第 0～9 行称做数字行（digit rows）或数字

穿孔(digit punches), 第 11 和 12 行被称做区域行(zone rows)或区域穿孔(zone punches)。由于不统一，IBM 打孔卡用起来有时会有些混乱：比如有的卡片把第 0 和第 9 行看做是区域行而不是数字行。

一个 EBCDIC 字符码由 8 位比特组成，进一步可以细分为高半字节（4 比特）与低半字节。低半字节是 BCD 码，与字符的数字穿孔保持一致，高半字节与区域穿孔的编码保持一致（而且与区域穿孔一一对应）。回忆一下第 19 章的 BCD 编码原理，其本质是采用二进制数对十进制数进行编码（binary-coded decimal）——其中数字 0～9 都利用不同的 4 位二进制数进行编码。

数字 0～9 并不需要区域穿孔进行额外表示，它们的 EBCDIC 编码的高半字节是 1111，代表了区域穿孔不起作用，而 0～9 的 EBCDIC 编码的低半字节是数字穿孔的 BCD 码，如下所示。

十六进制编码	EBCDIC 字符
F0	0
F1	1
F2	2
F3	3
F4	4
F5	5
F6	6
F7	7
F8	8
F9	9

大写字母有一些有趣的规律，如果区域穿孔只出现在第 12 行，则高半字节标识为 1100；如果只出现在第 11 行，则高半字节标识为 1101；如果出现在第 0 行，则高半字节标识 1110。下表给出了大写字母及其对应的 EBCDIC 编码。

十六进制编码	EBCDIC 字符	十六进制编码	EBCDIC 字符	十六进制编码	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

续表

C7	G	D7	P	E7	X
C8	H	D8	Q	E8	Y
C9	I	D9	R	E9	Z

值得注意的是 R 与 S 之间编号有跳变。有时在编写程序的时候，尤其是程序中用到 EBCDIC 编码时，这个容易被忽视的小细节往往会令人抓狂。

小写与大写字母的数字穿孔是相同的，但它们的区域穿孔不同。在 a～i 的小写字母，穿孔位于第 12 行和第 0 行，高半字节对应的编码为 1000；在 j～r 的小写字母，穿孔位于第 12 行和第 11 行，高半字节对应的编码为 1001；在 s～z 的小写字母，穿孔位于第 11 行和第 0 行，高半字节对应的编码为 1010。小写字母的 EBCDIC 字符及其对应的十六进制编码如下表所示。

十六进制编码	EBCDIC 字符	十六进制编码	EBCDIC 字符	十六进制编码	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都有自己的 EBCDIC 编码，但对于这些字符的编码系统没有必要去深究。

仔细观察 IBM 打孔卡，其中每一列细细数下共有 12 个孔，每个孔代表 1 位，也就是说可以提供 12 位的编码信息，不是吗？我们其实可以用打孔卡上每一列 12 孔中的 7 个来表示 ASCII 码。但是，这种方案有一个非技术方面的缺陷，那就是太多的穿孔将使得卡片变得很脆弱，容易折断。

采用 8 位编码的 EBCDIC 中其实还有很多编码未定义，这也说明当年 ASCII 码采用了 7 位编码也是合乎情理的。在 ASCII 码刚刚问世的那个年代，存储器的价格贵得令人咋舌，有一些观点认为 ASCII 码可以用 6 位编码并配合转义字符来使用，这样既可以区

分大小写又节约了存储器。这种方案并没有被采纳，当时还有一些人认为 ASCII 码应采用 8 位编码，他们对计算机的体系结构有了一个大胆的推测，即计算机应该按字节存储，7 位存储是不合适的。今天来看，8 位的字节存储已经作为了一项标准。尽管 ASCII 码从技术的本质上来看是 7 位编码，但仍以 8 位的形式存储。

在字节与字符之间建立一种等价关系大大简化了我们的工作，举例来讲，如果要粗略估计一个文本文件所需要的存储空间，只要统计字符数就可以了。这时前面学过的 K（kilos）和 M （Megas）就派上了用场，用它们来表示文本所占据的计算机存储空间更加通俗易懂。

传统的排版格式是：一张大小为 8.5×11 英寸的打印纸，采用双倍行距，1 英寸的页边距，每页可以容纳约 27 行的正文。每行宽度约为 6.5 英寸，每英寸可容纳 10 个字符，通过计算可以知道每页共包含约 1750 个字节。如果页面采用单倍行距，那么打印纸的容量约为原先的 2 倍，即 3.5 KB。

翻开一本《纽约客》（*The New Yorker*）杂志，可以看到杂志每页有 3 栏，每栏包含 60 行，每行约有 40 个字符，这样算下来每页大致包含 7200 个字符（也可以说成字节）。

《纽约时报》（*New York Times*）每一页包含 6 栏。假如页面都是文字而不包含标题和图片（这其实是不大可能的），那么可以认为每栏包含 155 行，每行大约容纳 35 个字符，这样算下来整个页面共包含 32,550 个字符，即 32 KB。

一般来讲精装书每页大约包含 500 个单词。根据统计，每个单词平均占用 5 个字母——更确切地来讲应该是 6 个字母，因为单词与单词之间是通过空格来分隔的，所以要一并统计在内。这样算下来，书的每一页大约包含 3000 个字符。假设每本书平均页数为 333，这个估计或许和实际不符，但如果这样估算的话，每本书平均容量为 1 MB。

不得不承认的是，书与书之间千差万别，所以上面这些也只是估算，下面列举出一些实际数据。

斯科特·菲茨杰拉德（F. Scott Fitzgerald）的《了不起的盖茨比》（*The Great Gatsby*）大约 300 KB。

塞林格（J. D. Salinger）的《麦田守望者》（*Catcher in the Rye*）大约 400 KB。

马克·吐温（Mark Twain）的《哈克贝里·弗恩历险记》（*The Adventures of Huckleberry*

Finn）大约 540 KB。

约翰·斯坦贝克（John Steinbeck） 的《愤怒的葡萄 / 怒火之花》（*The Grapes of Wrath*）大约 1MB。

赫尔曼·梅尔维尔（Herman Melville）的《白鲸》（*Moby Dick*）大约 1.3 MB。

亨利·菲尔丁（Henry Fielding）的《弃儿汤姆·琼斯的历史》（*The History of Tom Jones*）大约 2.25 MB。

玛格丽特·米切尔（Margaret Mitchell）的《乱世佳人》（*Gone With the Wind*）大约 2.5 MB。

斯蒂芬·金（Stephen King）的《末日逼近》（*The Stand*）大约 2.7 MB。

列夫·托尔斯泰（Leo Tolstoy）的《战争与和平》（*War and Peace*）大约 3.9 MB。

马塞尔·普鲁斯特（Marcel Proust）的《追忆似水年华》（*Remembrance of Things Post*）大约 7.7 MB。

美国国会图书馆（The United States Library of Congress）藏书约为 2000 万本，大概有 20 万亿字符，从存储器角度来说，数据总量为 20 TB（这还不包括图书馆中的大量珍贵照片和录音资料）。

尽管 ASCII 码是计算机领域最重要的标准，但它并不是十全十美的。它的问题就蕴含在它的全称中——American Standard Code for Information Interchange，它是太美国化了！即使那些以英语为主要语言的国家，ASCII 码也并不适用。ASCII 码中包含美元符号，而英镑符号怎么找不到呢？还有西欧国家语言中用到的重音符号在哪里？更别说使用非拉丁字母的希腊文（Greek）、阿拉伯文（Arabic）、希伯来文（Hebrew）和西里尔文（Cyrillic）等欧洲国家了。此外，印度及东南亚地区用到的婆罗门手记、北印度的 Devanagari 方言、孟加拉语、泰语、西藏语也并没有在 ASCII 码中出现。简单的 7 位编码在面对数以万计的中国、日本、韩国的象形文字，以及奇怪的朝鲜文音节时也显得力不从心。

在 ASCII 码的发展历程中，尽管没有在引入非拉丁字母方面做过工作，但开发者也一直在积极思考与改进编码系统，使其适用于其他国家。根据公布的 ASCII 码标准，有 10 个 ASCII 码保留位（40h、5Bh、5Ch、5Dh、5Eh、60h、7Bh、7Ch、7Dh 和 7Eh）可被重新定

义，这样就便于特定国家的使用。另外，英镑符号（£）可以在需要时替换特殊符号（#），通用货币符号（¤）可以在需要时替换美元符号（$）。当然，为使得这一替换过程不发生混淆，如果在文本文件中使用了这些重定义的符号，相关人员都必须知道这些变化。

大多数计算机系统采用 8 位编码来存储字符，我们也自然地想到设计一种扩展的 ASCII 字符集，这样可以包含 256 个字符，比原先扩展了一倍。在这种字符集中，编码 00h～7Fh 与原 ASCII 码保持一致；编码 80h～FFh 可以用来引入其他字符。这项技术已经被用来定义附加的字符编码，比如前面提到过的重音字母以及非拉丁字母。下面这个例子是对 96 个额外字符的 ASCII 码扩展，称为第 1 号拉丁字母表（Latin Alphabet No. 1），其中包括 A0h～FFh 字符编码。在该表中，每个字符的十六进制编码的高半字节由第一行给出，低半字节由第一列给出，如下表所示。

	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

编码 A0h 对应的字符为不中断空格（No-Break Space）。通常计算机在对文本进行排版时，会将其划分为行和段，行与行之间以空格符号区分（空格所对应的 ASCII 码为 20h）。编码 A0h 显示为空格，但是并不表示行与行之间被断开。比如在“WW II”这样一段文

字中就可以使用不中断空格。编码 ADh 被定义为软连字符（soft hyphen），它的用途是连接同一单词之间的音节，在一个单词被不得已划分在两行时就会用到它。

只可惜问题也随之而来，近几十年来出现了许多不同版本的扩展的 ASCII 码，多个不同的版本严重影响了编码的一致性，导致了混淆和不兼容。ASCII 码被扩展到极致，有的甚至可以对中文、日文和韩文进行编码。其中有一种流行的编码——Shift-JIS，即日本工业标准（Japanese Industrial Standard），利用 81h～9Fh 表示双字节字符编码的初始字节。通过这种手段，Shift -JIS 可对额外的约 6000 个字符进行编码。只可惜 Shift-JIS 并不是唯一的采用这种技术的编码系统。在亚洲地区，还有三个类似的双字节字符编码系统（double-byte character sets，DBCS）同样也很流行。

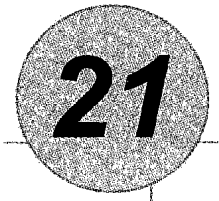
双字节字符集的确有很多版本，但兼容性并不是它最主要的问题。它的另一个缺陷是，一些字符，特别是通用的 ASCII 码字符，是用单个字节编码表示的，相比而言，成千上万的象形文字则是双字节编码，这在无形之中增加了使用这种字符集的难度。

业界一直有一个目标，那就是建立一个独一无二的字符编码系统，它可以用于世界上所有语言文字，从 1988 年开始，几大著名计算机公司合作研究出一种用来替代 ASCII 码的编码系统，取名为 Unicode（统一化字符编码标准）。相对于 ASCII 的 7 位编码，Unicode 采用了 16 位编码，每一个字符需要 2 个字节。也就是说 Unicode 的字符编码范围为 0000h～FFFFh，总共可以表示 65,536 个不同字符。全世界所有的人类语言，尤其是经常出现在计算机通信过程中的语言，都可以使用同一个编码系统，而且这种系统还具备很高的扩展性。

Unicode 编码其实并不是从零开始设计的，前 128 个字符编码——即 0000h～007Fh——与 ASCII 码是一致的。Unicode 编码中的 00A0h～00FFh 与先前讲到的第 1 号拉丁字母表是一致的。全世界很多标准也被一同收录在 Unicode 中。

尽管相对于之前讲过的一些字符编码系统，可以说 Unicode 做出了有效地改进，但这也不能确保它被全世界广泛采纳。ASCII 码，包括数不清的有一点小缺陷的扩展 ASCII 码已经在计算机领域根深蒂固，想一下子就取代它们并不是轻而易举的。

对于 Unicode 来讲，它唯一的问题，就是它改变了字符与存储空间之间“单字符，单字节”的等价对应关系。采用 ASCII 编码方式存储的著作《怒火之花》，其所占据的存储空间约为 1 MB。而如果采用 Unicode 编码，约占 2 MB。为了使编码系统兼容，Unicode 在存储空间上付出了相应的代价。



总 线

在一台计算机中，中央处理器无疑是最重要的部件，但它并不是唯一的部件。随机访问存储器（Random Access Memory，RAM）也是计算机不可或缺的部件，它存放着处理器要执行的机器代码指令。通过怎样的方法才能把指令加载到 RAM 中？怎样才能把程序的结果变得可见呢？或许你一下子就想到了输入设备（Input Device）和输出设备（Output Device）。回想一下前面讲过的内容，RAM 是易失性存储器——换言之，当掉电的时候其中的内容就会丢失。所以，长期存储设备也是一台计算机必不可少的部件，只有这样，代码和数据才能够被永久保存，不会因为掉电而丢失重要的数据。

搭建一台完整的计算机还需要很多集成电路，这些集成电路都必须挂载（mounted）到电路板上。在一些小型的机器中，一块电路板足以容纳所有的集成电路，但这种情况并不常见。我们通常所看到的是另一种情况：计算机中各部件按照功能被分别安装在两个或更多的电路板上。这些电路板之间通过总线（bus）通信。如果对总线做一个简单的概括，可以认为总线就是数字信号的集合，而这些信号被提供给计算机上的每块电路板。通常把这些信号划分为如下四类。

- 地址信号。这些信号是由微处理器产生，通常用来对 RAM 进行寻址操作，当然也可以用来对连接到计算机的其他设备进行寻址操作。

- 数据输出信号。这些信号也是由微处理器产生的，用来把数据写入到 RAM 或其他设备。这里特别要注意区分术语输入（input）和输出（output），来自微处理器的数据输出信号会变成 RAM 和其他设备的数据输入信号。
- 数据输入信号。这些信号是由计算机的其他部分提供的，并由微处理器读取。通常情况下，数据输入信号由 RAM 输出，这就解释了微处理器是怎样从内存中读取内容的。其实，其他部件也可以给微处理器提供数据输入信号。
- 控制信号。这些信号是多种多样的，通常与计算机内所用的特定的微处理器相对应。控制信号可以产生于微处理器，也可以由与微处理器通信的其他设备产生。比如，当微处理器要把一些数据写入到特定内存单元时，它所使用的信号就是控制信号。

还有一点需要说明：总线还可以为计算机上不同电路板供电。

回顾一下总线的发展历程。在家用计算机领域，早期比较流行的就是 S-100 总线，1975 年第一台家用计算机 MITS Altair 就率先采用了这种总线。尽管一开始，S-100 总线只是基于 8080 微处理器的，后来经过改进，也开始适用于其他处理器，例如 6800。一块 S-100 电路板的规格是 5.3×10 英寸，其中有一边是要插到一个插槽上的，这个插槽有 100 个连接器（这就是名为 S-100 的原因）。

每台 S-100 计算机都有一块很大的被称为母板（motherboard 或 mainboard）的电路板，它有若干相互连接的 S-100 总线插槽（可能有 12 个）。有时候，这些插槽也被称为扩展插槽（expansion slots）。S-100 电路板（也称为扩展板，expansion boards）就插在这些插槽中。8080 微处理器及支持芯片（第 19 章提到过的其中的一些）分布在一块 S-100 电路板上，而 RAM 分布在一块或多块其他电路板上。

S-100 总线是专门为 8080 芯片而设计的，有 16 个地址信号，8 个数据输入信号及 8 个数据输出信号（仔细回忆一下，8080 本身并不区分数据输入和输出信号，这项工作是由电路板上的其他支持芯片完成的）。总线上也含有 8 个中断信号，其他设备需要 CPU 立即做出响应时，便会产生这些信号。下面我们看一个例子（本章的后面也要讲到），当某个按键按下时，键盘可能就会产生一个中断信号。接下来 8080 会执行一段小程序，检测出是什么按键被按下，并做出响应。通常，在安装了 8080 的电路板上有一个被称为 Intel 8214 优先级中断控制单元的芯片，就是专门用来处理中断的。当中断发生时，这个芯片

会产生一个中断信号并送给 8080。8080 识别出这个中断后，此芯片就会提供一个 RST（Restart，重启）指令，在这条指令的作用下，微处理器会把当前程序计数器的值保存下来，并依据中断类型，跳转到地址 0000h、0008h、0010h、0018h、0020h、0028h、0030h 或 0038h 处执行。

如果你在设计一个新的计算机系统，而这个系统中采用新的总线类型，你可以选择把总线规范公布于众（也可以通过其他方式发布出去）或者使其保密，决定权在于你。

一旦一条指定总线的规范公布开来，其他制造商——称为第三方（third-party）制造商——就可以设计并销售采用了这种总线的扩展板了。这些额外扩展板不仅加强了计算机的实用性，还使其更加满足实际需求。计算机的销售情况越好，扩展板的市场前景也就越好。正是由于这个原因，设计者在设计多数小型计算机系统时，都会坚持开放体系结构（Open Architecture）的原则，这样一来，其他制造商就可以生产计算机的外设。最终会有一条总线成为工业标准。在今天，“标准”已经成为个人计算机产业的一个重要组成部分。

1981 年秋，最著名的开放体系结构个人计算机——IBM 的 PC 问世。IBM 公布了 PC 的技术参考资料（technical reference），里面包含了整台计算机的全部电路图，IBM 为其制造的扩展板的资料也在其中。这个手册可是很重要的资料，它的出现使得很多制造商可以生产自己的 PC 扩展板，实际上，这创造出了整个 PC 的克隆体——其实与 IBM 的 PC 几乎完全相同，运行的软件也一样。

在如今的桌面计算机领域，从起初的 IBM 的 PC 发展而来的计算机数量庞大，占据约 90%的市场份额。尽管 IBM 本身只占很小一部分，但事实上，如果起初的 PC 采用的是封闭体系结构（closed architecture）且设计是私有化（proprietary）的，其所占的市场份额会更少。苹果公司的麦金托什（Macintosh）起初采用的是封闭体系结构，尽管也曾部分考虑过开放的问题，这也就解释了为什么 Macintosh 在如今的桌面计算机市场上只占有不到 10%的份额。（请记住这一点：一个计算机系统可以是在开放体系结构下设计的，也可以是在封闭体系结构下设计的，无论是哪种情况，其他公司都可以为其设计软件。但也有例外的情况，某些视频游戏的开发商会限制其他公司开发其专用系统上的软件）。

最初的 IBM PC 采用的是 Intel 8088 微处理器，可以寻址 1 MB 的存储单元。虽然 8088 内部是一个 16 位的微处理器，但外部却只能寻址 8 位的存储器。工业标准体系结构