第3章 chapter 3

数据表示

本章主要介绍进位记数制及不同记数制之间数值的相互转换。本章还详细讲解字符编码、汉字编码等编码方案以及计算机中的数据校验。


数据表示及运算

本章学习目的：

(1) 了解进位记数制的概念以及二进制、八进制、十进制、十六进制等常见进位记数制的表示。

(2) 理解计算机中数值数据和非数值数据的表示。

(3) 掌握二进制、八进制、十进制、十六进制等常见进位记数制之间的相互转换。

(4) 了解数据校验的原理。

3.1 进位记数制


进位记数制及其表示

在人类的生产和生活中，经常要遇到数的表示问题，人们通常采用从低位向高位进位的方式进行记数，这种表示数据的方法称为进位记数制，简称数制。讨论数制涉及两个基本概念：基数(radix)和权(weight)。

1. 十进制

在数制中，每个数位用到的数码符号的个数叫作基数。十进制是人们最熟悉的一种数制，每个数位允许选用 0～9 共 10 个不同数码符号中的某一个，因此十进制的基数为 10。每个数位满 10 就向高位进位，即逢 10 进 1。

在一个数中，数码在不同的数位上表示的数值是不同的。每个数码表示的数值就等于该数码本身乘以一个与它所在数位有关的常数，这个常数叫作权。例如，对于十进制数 6543.21，数码 6 所在数位的权为 1000，这一位代表的数值为 $6\times10^3=6000$；5 所在数位的权为 100，这一位代表的数值为 $5\times10^2=500$……

所以，一个数的数值就是它的各位数码按权相加之和。例如：

$$(6543.21)_{10}=6\times10^3+5\times10^2+4\times10^1+3\times10^0+2\times10^{-1}+1\times10^{-2}$$

由此可见，任何一个十进制数都可以用一个多项式表示：

$$(N)_{10}=k_n\times10^n+k_{n-1}10^{n-1}+\cdots+k_0\times10^0+\cdots+k_{-m}\times10^{-m}=\sum_{i=n}^{-m}k_i\times10^i$$ 其中，

k_i 的取值是 0 ～ 9 中的一个数码，m 和 n 为正整数。

推而广之，一个基数为 R 的 R 进制数可表示为

$$(N)_R = k_n \times R^n + k_{n-1}R^{n-1} + \cdots + k_0 \times R^0 + \cdots + k_{-m} \times R^{-m} = \sum_{i=n}^{-m} k_i \times R^i$$

其中，R^i 是第 i 位的权，k_i 的取值可以是 0，1，…，$R-1$ 共 R 个数码中的任意一个。R 进制数的进位原则是逢 R 进 1。

计算机为什么采用二进制

2. 二进制

计算机中信息的存储、处理和传送采用的都是二进制。不论是数据还是多媒体信息（声音、图形、图像等），都必须采用二进制编码形式才能存入计算机中。

二进制是一种最简单的数制，它只有 0 和 1 两个不同的数码，即基数为 2，逢 2 进 1。任意数位的权是 2^i。

因此，任何一个二进制数都可表示为

$$(N)_2 = \sum_{i=n}^{-m} k_i \times 2^i$$

3. 十六进制

十六进制数的基数为 16，逢 16 进 1，每个数位可取 0，1，…，9，A，B，…，F 共 16 个不同的数码符号中的任意一个，其中 A～F 分别表示十进制数 10～15。

任何一个十六进制数都表示为

$$(N)_{16} = \sum_{i=n}^{-m} k_i \times 16^i$$

既然有不同的数制，在给出一个数的同时，就必须指明它是哪种数制的数。例如，$(1010)_2$、$(1010)_{10}$、$(1010)_{16}$ 代表的数值完全不同，如果不用下标加以标注，就会产生歧义。除了用下标表示之外，还可以用后缀字母表示不同的数制，后缀 B 表示该数是二进制（Binary）数，后缀 H 表示该数是十六进制（Hexadecimal）数，而后缀 D 表示该数是十进制（Decimal）数。十进制数在书写时可以省略后缀 D，其他进制的数在书写时一般不能省略后缀。例如，有 3 个数分别为 375D、101B 和 AFEH，从后缀就可以知道它们分别是十进制数、二进制数和十六进制数。

大多数计算机都采用十六进制描述计算机中的指令和数据。表 3-1 给出了 3 种常用数制的数值对应关系。

表 3-1　3 种常用数制的数值对应关系

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0000	0	3	0011	3
1	0001	1	4	0100	4
2	0010	2	5	0101	5

续表

十进制	二进制	十六进制	十进制	二进制	十六进制
6	0110	6	11	1011	B
7	0111	7	12	1100	C
8	1000	8	13	1101	D
9	1001	9	14	1110	E
10	1010	A	15	1111	F

数据的表示方法有很多种，不同的表示方法对计算机的结构和性能都会产生不同的影响。人们日常生活中一般采用十进制数进行计数和计算，但十进制数难以在计算机内直接存储与运算。为了简化计算机的设计，方便计算机对数据进行处理，在计算机系统中，通常将十进制数用于人机交互，而数据则以二进制数的形式存储和运算。计算机采用二进制的主要原因有以下几点：

(1) 易于物理实现。二进制在技术上最容易实现。这是因为具有两种稳定状态的物理器件有很多，如门电路的导通与截止、电压的高与低等，而它们恰好可以对应 1 和 0 这两个数码。假如采用十进制，那么就要制造具有 10 种稳定状态的物理电路，而这是非常困难的。

(2) 运算规则简单。数学推导已经证明，对 R 进制数进行算术求和或求积运算，其运算规则各有 $R(R+1)/2$ 种。如果采用十进制，则 $R=10$，就有 55 种求和或求积的运算规则；而如果采用二进制，则 $R=2$，仅有 3 种求和或求积的运算规则。以二进制加法为例：0+0=0，0+1=1(1+0=1)，1+1=10。因而二进制可以大大简化运算器等物理器件的设计。

(3) 可靠性高。由于电压的高和低，电流的有和无等都是一种质的变化，两种物理状态稳定、分明，因此二进制码传输的抗干扰能力强，鉴别信息的可靠性高。

(4) 逻辑判断方便。采用二进制后仅有的两个符号 1 和 0 正好可以与逻辑命题的两个值真和假相对应，能够方便地使用逻辑代数这一有力工具分析和设计计算机的逻辑电路。

3.2 进制转换

计算机中采用的是二进制，因为二进制具有运算简单、易实现且可靠、便于逻辑设计等优点。但是，用二进制表示一个数使用的位数要比用十进制表示长得多，书写和阅读都不方便，也不容易理解。为了书写和阅读的方便，人们通常使用十六进制弥补二进制的这一不足，因此常常需要在十进制数、二进制数和十六进制数之间进行转换。本节主要介绍常用的几种转换方法。

1. 二进制数转换为十六进制数

将一个二进制数转换为十六进制数的方法是将二进制数的整数部分和小数部分分别进行转换。即，以小数点为界，整数部分从小数点开始往左数，每 4 位分成一组，当最左边的一组数不足 4 位时，可根据需要在数的最左边添加若干个 0 以补足 4 位；对于小数部分，从小数点开始往右数，每 4 位分成一组，当最右边的一组数不足 4 位时，可根据需要在数的最右边添加若干个 0 以补足 4 位。最终使二进制数的总位数是 4 的倍数，然后用相应的十六进制数取而代之。例如：

$$(111011.1010011011)_2=(00111011.101001101100)_2=(3B.A6C)_H$$

2. 任意进制数转换为十进制数

将任意进制的数各位数码与它们的权值相乘，再把乘积相加，就得到了对应的十进制数。这种方法称为按权展开相加法。例如：

$$(11011.1)_2=1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}=(27.5)_{10}$$

3. 十进制数转换为任意进制数

将十进制数转换为任意进制数，常采用基数乘除法。这种转换方法对十进制数的整数部分和小数部分分别进行处理，对于整数部分用除基取余法，对于小数部分用乘基取整法，最后将整数部分与小数部分的转换结果拼接起来。

除基取余法（整数部分的转换）：整数部分除基取余，最先取得的余数为数的最低位，最后取得的余数为数的最高位，商为 0 时结束。

乘基取整法（小数部分的转换）：小数部分乘基取整，最先取得的整数为数的最高位，最后取得的整数为数的最低位，乘积为 0（或满足精度要求）时结束。

例如，将十进制数 123.6875 转换成二进制数。


十进制数转换为二进制数

整数部分：

除基	取余	
2⌊123	1	最低位
2⌊61	1	
2⌊30	0	
2⌊15	1	
2⌊7	1	
2⌊3	1	
2⌊1	1	最高位
0		

故整数部分 $(123)_{10}=1111011_2$。

小数部分：

乘基取整		
0.6875		
× 2		
1.3750	1	最高位
0.3750		
× 2		
0.7500	0	
× 2		
1.5000	1	
0.5000		
× 2		
1.0000	1	最低位

故小数部分$(0.6875)_{10}=0.1011_2$。

所以$(123.6875)_{10}=1111011.1011_2$。

3.3 十进制数据编码

二进制数的实现方案简单、可靠，因此在计算机内部采用二进制数进行工作，但如果直接使用二进制数进行输入和输出则非常不直观，难以被用户接受。因此，在计算机进行输入输出处理时，一般还是用十进制数表示，这就要求对十进制数进行编码，使计算机能够接收并处理这些数据信息。

由于十进制有0～9共10个数码，需要使用$\log_2 10$位二进制数进行编码，向上取整为4，因此一般用4位二进制数表示1位十进制数。由于4位二进制数有16种组合，从中选出的10种组合表示0～9这10个数码，可以产生多种方案，如8421BCD码、2421码、余3码等。这里介绍使用最普遍的8421BCD码。

BCD(Binary-Coded Decimal)码用4位二进制数表示1位十进制数的0～9这10个数码，是一种二进制的数字编码形式。在该种编码方案中，4个二进制数位的权值从低到高分别是1、2、4、8，使用000，0001，…，1001对应十进制中的0～9。在运算时，每个数位内满足二进制规则，而数位之间满足十进制规则。

8421BCD码的优点在于直观，而且与数字的ASCII码的转换非常方便，但是直接使用8421BCD码进行算术运算时要复杂一些，在某些情况下，需要对加法运算的结果进行修正。修正规则是：如果两个8421BCD码数相加之和小于或等于1001，即十进制的9，无须修正；如果结果为十进制的10～15，需要主动向高位产生一个进位，本位进行加6修正；如果结果为十进制的16～18，会自动向高位产生一个进位，本位仍需进行加6修正。十进制数、二进制数和8421BCD码的对应关系如表3-2所示。

表 3-2　十进制数、二进制数和 8421BCD 码的对应关系

十进制数	二进制数	8421BCD 码	十进制数	二进制数	8421BCD 码
0	0000	0000	8	1000	1000
1	0001	0001	9	1001	1001
2	0010	0010	10	1010	0001　0000
3	0011	0011	11	1011	0001　0001
4	0100	0100	12	1100	0001　0010
5	0101	0101	13	1101	0001　0011
6	0110	0110	14	1110	0001　0100
7	0111	0111	15	1111	0001　0101

3.4　ASCII 码

目前计算机中用得最广泛的是 ASCII 码。ASCII 码是由 128 个字符组成的字符集，其中包括 10 个十进制数码、26 个英文字母（区分大小写），以及其他专用符号和控制符号。使用 7 位二进制数可以给出 128 个编码，表示 128 个不同的字符。其中 95 个编码对应计算机终端能输入和显示的 95 个字符，如大小写各 26 个英文字母、0～9 这 10 个数字、通用的运算符和标点符号等（包括空格），打印机设备也能打印这 95 个字符。编码值 0～31 不对应任何一个可以显示或打印的实际字符，通常称为控制字符，被用作控制码，控制计算机某些外固设备的工作特性和某些计算机软件的运行情况。编码值为 127 的是刷除控制 DEL 码。ASCII 码规定 8 个二进制位的最高一位为 0，在实际使用中，最高位可以根据需求来存放奇偶校验的结果，称为校验位。表 3-3 列出了 7 位的 ASCII 码字符编码表。

表 3-3　ASCII 码表

$b_3b_2b_1b_0$		$b_6b_5b_4$							
		0	1	2	3	4	5	6	7
		000	001	010	0011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	'	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v

续表

$b_3b_2b_1b_0$		$b_6b_5b_4$							
		0	1	2	3	4	5	6	7
		000	001	010	0011	100	101	110	111
7	0111	BEL	ETB	/	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	l
D	1101	CR	GS	—	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	Si	US	/	?	O	—	o	DEL

观察表 3-3 可以发现以下两个基本规律：

(1) 数字 0～9 的高 3 位编码均为 011，低 4 位编码为 0000～1001，正好是 0～9 的 8421BCD 码，这有利于 ASCII 码与 8421BCD 码的相互转换。

(2) 同一英文字母的大小写编码的差别仅在于第 6 位是 0 还是 1，这也方便了大小写字母的相互转换。

在 IBM 计算机中采用了另一种字符编码，即 EBCDIC 编码。它采用 8 位二进制，可以表示 256 个编码状态，但只选用了其中的一部分。0～9 这 10 个数字编码的高 4 位为 1111，低 4 位仍为 0000～1001。大小写英文字母的编码同样满足正常的排序要求，而且有简单的对应关系，易于转换和识别。

3.5 汉字编码

区位码、国标码和机内码

1. 汉字的输入编码

为了能够使用西文标准键盘将汉字提供给计算机信息处理系统，需要为汉字设计相应的输入编码方法。目前使用的汉字输入编码主要有以下 3 类：

(1) 数字编码。常用的是区位码，每个汉字对应一个唯一的数字串。其优点是无重码，且输入码与内部码的转换较方便；其缺点是编码难记。

(2) 拼音码。这是一种以汉语拼音为基础的输入方法。其优点是熟悉汉语拼音的用户可以轻松掌握，无须特殊的训练和记忆；其缺点是重码率高，需进行同音字选择，影响输入速度。

其他汉字编码方式

(3) 字形码。这种编码通过分析汉字的字形，将汉字的笔画用字母或数字进行编码。

目前最常用的字形码是五笔字形码。

2. 国标码

国标码是国家标准汉字编码的简称，其全称是《信息交换用汉字编码字符集：基本集》，是我国在1980年颁布的用于汉字信息处理使用的代码依据（GB 2312—1980）。国标码依据使用频度把汉字划分为高频字（约100个）、常用字（约3000个）、次常用字（近4000个）、罕见字（约8000个）和死字（约45 000个）。把高频字、常用字和次常用字归结为汉字字符集（共6763个）。其中，一级汉字3755个，以汉语拼音为序排列；二级汉字3008个，以偏旁部首为序排列。再加上682个图形符号以及西文字母、数字等，在一般情况下已足够使用。国标码规定：一个汉字用2字节表示，每字节只使用低7位，最高位未做定义。为书写方便，常用4位十六进制数表示一个汉字。

国标码是一种机器内部编码，用于统一不同系统的各种编码。通过将不同系统使用的不同编码统一转化成国标码，不同系统的汉字信息就可以相互交换。

3. 汉字机内码

汉字机内码是计算机内部对汉字进行存储、处理和传输使用的编码，简称机内码。机内码是根据GB 2312—1980进行编码的。在计算机中，一般采用2字节表示一个汉字。为区分汉字和英文字符，规定汉字机内码两字节的最高位均为1，以避免造成混乱。

例如，汉字“文”的国标码为4E44H（010011100100100），每字节的最高位变为1，得到1100111011000100，即“文”的机内码为CEC4H。

区位码、国标码和机内码之间存在一定的转换规则：将区位码用十六进制表示后，加上2020H即可得到国标码，在国标码的基础上加上8080H即可得到机内码。

4. 汉字字模码

字模码是用点阵表示的汉字字形编码，是汉字的输出形式。

字模码一般采用点阵式编码，即把一个汉字按一定的字形需要写在一定规格的点阵格纸中。根据汉字输出要求的不同，点阵的大小也不同。简易型汉字为16×16点阵，提高型汉字为24×24点阵、32×32点阵或更高。点阵中每个点的信息用1位二进制码表示，用1表示该位置是黑色，用0表示该位置是白色。随着点阵规模的增加，其存储量也相应地提高。例如，16×16点阵的汉字要占用32字节（256位），24×24点阵的汉字要占用72字节（576位），32×32点阵的汉字要占用128字节（1024位）。因此字模码只能用来构成汉字库，而不能用于机内存储。汉字库中存储了每个汉字的点阵编码，当显示或打印输出时才检索汉字库，输出字模点阵，得到字形。

3.6 数据校检

数据校验的基本原理

数据在存取和传送的过程中可能会产生错误，其原因可能有很多种，如设备的临界工作状态、外界高频干扰、收发设备中的间歇性故障以及电源偶然的瞬变现象等。为减

少和避免错误，除了提高硬件本身的可靠性之外，还可以对数据采用专门的逻辑电路进行编码，以检测错误，甚至校正错误。

通常的方法是，在每个字上添加一些校验位，用来确定字中出现错误的位置。计算机中常用这种检错技术进行存储器读写或信息传输正确性的检验。

1. 奇偶校验的概念


奇偶校验

奇偶校验码是一种最简单且应用最广泛的数据校验码，它的硬件成本很低，可以检测出一位或奇数位错误，但不能确定出错的位置，也不能检测出偶数位错误。事实上，一位出错的概率比多位同时出错的概率要高得多，因此，虽然奇偶校验的检错能力很低，但仍然是一种很有效的校验方法，常用于存储器读写检查或 ASCII 字符传送过程检查。

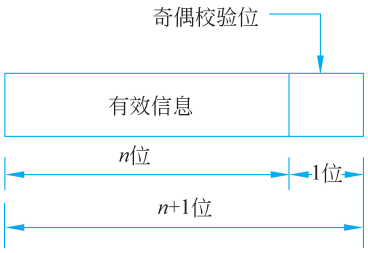

图 3-1 奇偶校验码

奇偶校验的实现方法是：由若干位有效信息（如 1 字节）加上 1 位奇偶校验位组成奇偶校验码，如图 3-1 所示。

奇偶校验位的取值（0 或 1）将使整个奇偶校验码中 1 的个数为奇数或偶数，所以有两种可供选择的校验规律：

（1）奇校验。当有效信息位中 1 的个数为奇数时，奇校验位为 0；否则为 1。

（2）偶校验。当有效信息位中 1 的个数为偶数时，偶校验位为 0；否则为 1。

注意： 奇偶校验位可以放在有效信息的右面，也可以放在有效信息的左面。

2. 简单奇偶校验

简单奇偶校验仅能实现横向的奇偶校验。

在实际应用中，多采用奇校验，因为奇校验中不存在全 0 代码，在某些场合下更便于判别正确性。

1）奇偶校验位形成

奇偶校验码的编码和校验是由专门的逻辑电路实现的。当把一字节的代码 $d_7 \sim d_0$ 写入主存时，也同时将它们送往奇偶校验电路，该电路将产生奇形成、偶形成、奇校验出错和偶校验出错等信号。奇形成和偶形成信号是奇偶校验位，并将与 8 位代码一起作为奇偶校验码写入主存。

$$\text{奇校验位（奇形成信号）} = \overline{d_7 \oplus d_6 \oplus d_5 \oplus d_4 \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0}$$

$$\text{偶校验位（偶形成信号）} = d_7 \oplus d_6 \oplus d_5 \oplus d_4 \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0$$

若 $d_7 \sim d_0$ 中有奇数个 1，则奇校验位为 0，偶校验位为 1。

若 $d_7 \sim d_0$ 中有偶数个 1，则奇校验位为 1，偶校验位为 0。

注意： ⊕表示逻辑异或（或称按位加）。

2）奇偶校验检测

读出数据时，将读出的 9 位编码（8 位有效信息和 1 位奇偶校验位）同时送入奇偶校验电路进行检测。

$$奇校验出错信号=d_7\oplus d_6\oplus d_5\oplus d_4\oplus d_3\oplus d_2\oplus d_1\oplus d_0\oplus d_{奇}$$

$$偶校验出错信号=d_7\oplus d_6\oplus d_5\oplus d_4\oplus d_3\oplus d_2\oplus d_1\oplus d_0\oplus d_{偶}$$

若读出代码无错误，则奇校验出错/偶校验出错信号为＝0。

若读出代码出现错误，则奇校验出错/偶校验出错信号为 1，从而指示这个 9 位编码中一定有某一位出现了错误，但具体的错误位置无法确定。

【例 3-1】 已知有如下 5 字节的数据：

请分别用奇校验和偶校验进行编码。

1 0 1 0 1 0 1 0

0 1 0 1 0 1 0 0

0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

解：假定最低一位为校验位，其余高 8 位为数据位，结果如表 3-4 所示。

表 3-4 例 3-1 的结果

数 据	1 的个数	偶 校 验 码	奇 校 验 码
1 0 1 0 1 0 1 0	4	1 0 1 0 1 0 1 0 0	1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 0	3	0 1 0 1 0 1 0 0 1	0 1 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1	7	0 1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1	8	1 1 1 1 1 1 1 1 0	1 1 1 1 1 1 1 1 1

从表 3-4 中可以看出，校验位的值取 0 还是取 1，是由数据位中 1 的个数决定的。

3. 交叉奇偶校验

计算机在进行大量字节（数据块）传送时，不仅每字节有一个奇偶校验位用于横向校验，而且全部字节的同一位也设置一个奇偶校验位用于纵向校验，这种横向、纵向同时校验的方法称为交叉奇偶校验。

例如，对于一个由 4 字节组成的信息块，纵向、横向均采用偶校验，各校验位取值如下所示：

	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	横向校验位
第 1 字节	1	1	0	0	1	0	1	1	1
第 2 字节	0	1	0	1	1	1	0	0	0
第 3 字节	1	0	0	1	1	0	1	0	0
第 4 字节	1	0	0	1	0	1	0	1	0
纵向校验位	1	0	0	1	1	0	0	0	

交叉奇偶校验可以发现两位同时出错的情况。假设第 2 字节的 d_6、d_4 两位均出错，横向校验位无法检出错误，但是 d_6、d_4 位所在列的纵向校验位会显示出错，这与前述的

简单奇偶校验相比要可靠得多。

3.7 偶校验码生成仿真实验

1. 实验目的

(1) 熟悉数据校验码的生成电路。
(2) 掌握偶校验位的计算方法。

2. 实验要求

(1) 补全实验代码,构建偶校验位生成电路。
(2) 实现功能:输入 8 个 1 位二进制数,通过异或生成偶校验位。
(3) 思考:如何生成奇校验位,需要增加何种门电路。

3. 实验原理

程序中包含或(OR)、异或(XOR)两个模块,模拟逻辑或门和异或门的作用。使用这两个模块构建偶校验位生成电路。

4. 实验代码

```
def AND(A, B):
    #与
    return (int(A) and int(B))

def OR(A, B):
    #或
    return (int(A) or int(B))

def XOR(A, B):
    #异或
    if int(A)==1 and int(B)==1:
        return 0
    else:
        return OR(A, B)

if __name__ == "__main__":
    data = []
    for i in range(8):
        s = "input data"+str(i)+":"
        data.append(input(s))
    #补全代码,实现偶校验位的生成
```

```
'''
print("生成的偶校验位为:",result)
```

习　题

（1）下列数中最小的数为（　　）。

A. $(101001)_2$　　B. $(52)_8$　　C. $(101001)_{BCD}$　　D. $(233)_{16}$

（2）下列数中最大的数为（　　）。

A. $(10010101)_2$　　B. $(227)_8$　　C. $(96)_{16}$　　D. $(143)_5$

（3）某数在计算机中用 8421BCD 码表示为 11110001001，其真值为（　　）。

A. 789　　B. 789H　　C. 1929　　D. 01110000

（4）在小型或微型计算机里普遍采用的字符编码是（　　）。

A. 8421BCD 码　　B. 十六进制　　C. 格雷码　　D. ASCII 码

（5）$(20000)_{10}$转换成十六进制数是（　　）。

A. $(7CD)_{16}$　　B. $(7D0)_{16}$　　C. $(7E0)_{16}$　　D. $(7F0)_{16}$

（6）根据国标码的规定，每个汉字在计算机内存储时占用（　　）。

A. 1 字节　　B. 2 字节　　C. 3 字节　　D. 4 字节