13. NUMBERS AND DATA

Topics:

- Number bases; binary, octal, decimal, hexadecimal
- Binary calculations; 2s compliments, addition, subtraction and Boolean operations
- Encoded values; BCD and ASCII
- Error detection; parity, gray code and checksums

Objectives:

- To be familiar with binary, octal and hexadecimal numbering systems.
- To be able to convert between different numbering systems.
- To understand 2s compliment negative numbers.
- To be able to convert ASCII and BCD values.
- To be aware of basic error detection techniques.

13.1 INTRODUCTION

Base 10 (decimal) numbers developed naturally because the original developers (probably) had ten fingers, or 10 digits. Now consider logical systems that only have wires that can be on or off. When counting with a wire the only digits are 0 and 1, giving a base 2 numbering system. Numbering systems for computers are often based on base 2 numbers, but base 4, 8, 16 and 32 are commonly used. A list of numbering systems is give in Figure 13.1. An example of counting in these different numbering systems is shown in Figure 13.2.

Base	Name	Data Unit
2	Binary	Bit
8	Octal	Nibble
10	Decimal	Digit
16	Hexadecimal	Byte

Figure 13.1 Numbering Systems

decimal	binary	octal	hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Note: As with all numbering systems

most significant digits are at left,
least significant digits are at right.

Figure 13.2 Numbers in Decimal, Binary, Octal and Hexadecimal

The effect of changing the base of a number does not change the actual value, only how it is written. The basic rules of mathematics still apply, but many beginners will feel disoriented. This chapter will cover basic topics that are needed to use more complex programming instructions later in the book. These will include the basic number systems, conversion between different number bases, and some data oriented topics.

13.2 NUMERICAL VALUES

13.2.1 Binary

Binary numbers are the most fundamental numbering system in all computers. A single binary digit (a bit) corresponds to the condition of a single wire. If the voltage on the wire is true the bit value is *1*. If the voltage is off the bit value is *0*. If two or more wires are used then each new wire adds another significant digit. Each binary number will have an equivalent digital value. Figure 13.3 shows how to convert a binary number to a decimal equivalent. Consider the digits, starting at the right. The least significant digit is *1*, and

is in the 0th position. To convert this to a decimal equivalent the number base (2) is raised to the position of the digit, and multiplied by the digit. In this case the least significant digit is a trivial conversion. Consider the most significant digit, with a value of *1* in the 6th position. This is converted by the number base to the exponent 6 and multiplying by the digit value of 1. This method can also be used for converting the other number system to decimal.

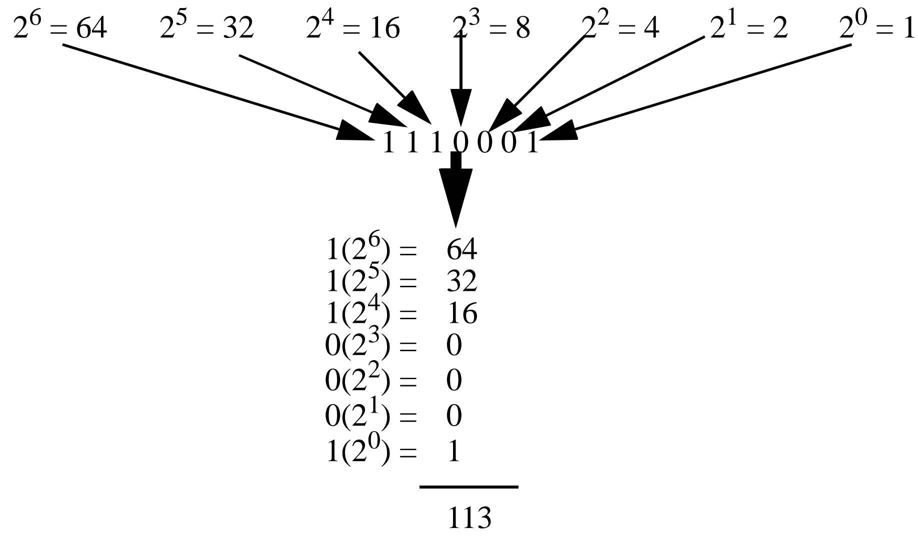


Figure 13.3 Conversion of a Binary Number to a Decimal Number

Decimal numbers can be converted to binary numbers using division, as shown in Figure 13.4. This technique begins by dividing the decimal number by the base of the new number. The fraction after the decimal gives the least significant digit of the new number when it is multiplied by the number base. The whole part of the number is now divided again. This process continues until the whole number is zero. This method will also work for conversion to other number bases.

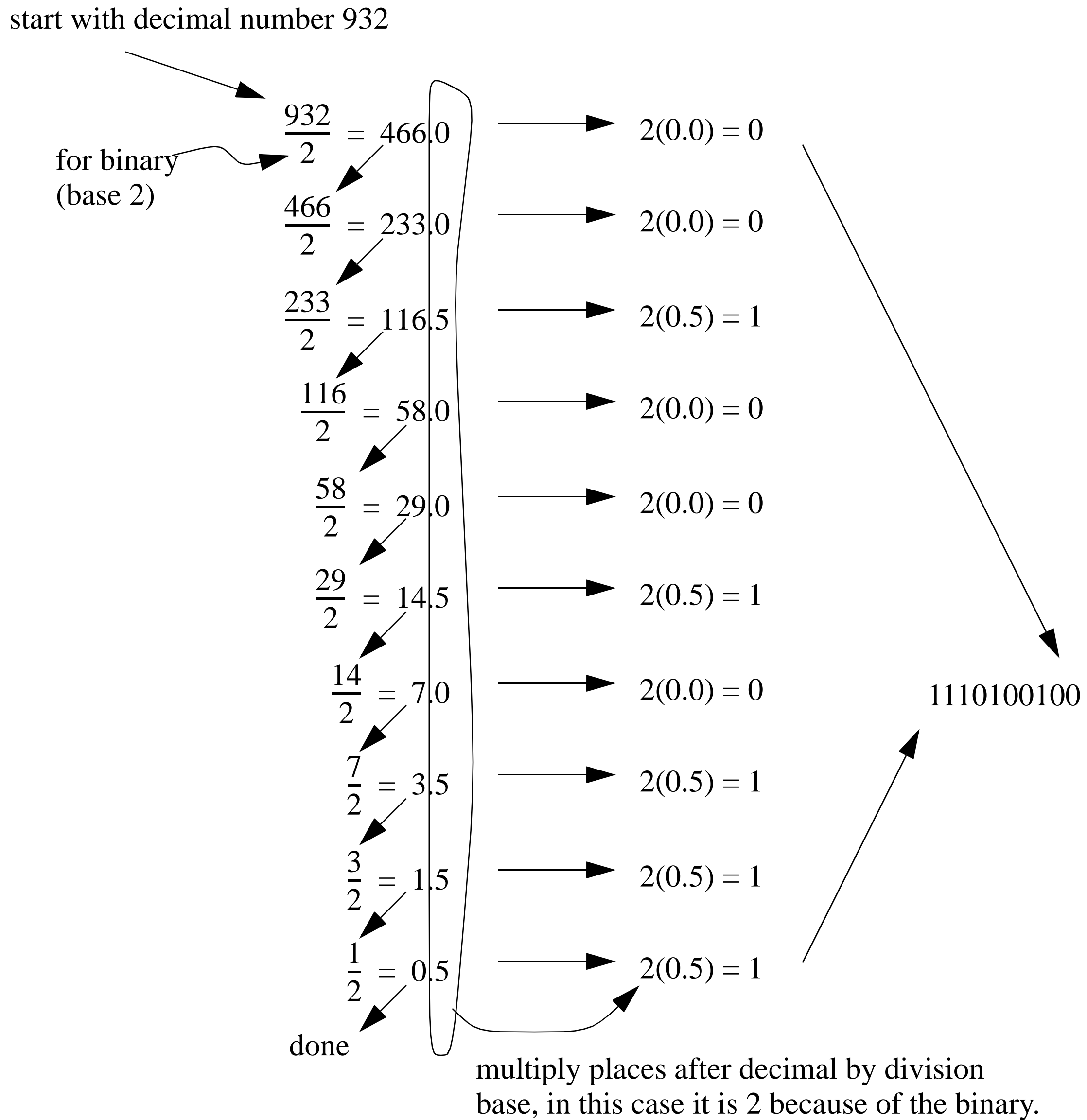


Figure 13.4 Conversion from Decimal to Binary

Most scientific calculators will convert between number bases. But, it is important to understand the conversions between number bases. And, when used frequently enough the conversions can be done in your head.

Binary numbers come in three basic forms - a bit, a byte and a word. A bit is a single binary digit, a byte is eight binary digits, and a word is 16 digits. Words and bytes are

shown in Figure 13.5. Notice that on both numbers the least significant digit is on the right hand side of the numbers. And, in the word there are two bytes, and the right hand one is the least significant byte.

Figure 13.5 Bytes and Words

Binary numbers can also represent fractions, as shown in Figure 13.6. The conversion to and from binary is identical to the previous techniques, except that for values to the right of the decimal the equivalents are fractions.

binary: 101.011

$$1(2^2) = 4 \quad 0(2^1) = 0 \quad 1(2^0) = 1 \quad 0(2^{-1}) = 0 \quad 1(2^{-2}) = \frac{1}{4} \quad 1(2^{-3}) = \frac{1}{8}$$

$$= 4 + 0 + 1 + 0 + \frac{1}{4} + \frac{1}{8} = 5.375 \text{ decimal}$$

Figure 13.6 A Binary Decimal Number

13.2.1.1 - Boolean Operations

In the next chapter you will learn that entire blocks of inputs and outputs can be used as a single binary number (typically a word). Each bit of the number would correspond to an output or input as shown in Figure 13.7.

There are three motors M_1, M_2 and M_3 represented with three bits in a binary number. When any bit is on the corresponding motor is on.

100 = Motor 1 is the only one on
111 = All three motors are on
in total there are 2^n or 2^3 possible combinations of motors on.

Figure 13.7 Motor Outputs Represented with a Binary Number

We can then manipulate the inputs or outputs using Boolean operations. Boolean algebra has been discussed before for variables with single values, but it is the same for multiple bits. Common operations that use multiple bits in numbers are shown in Figure 13.8. These operations compare only one bit at a time in the number, except the shift instructions that move all the bits one place left or right.

Name	Example	Result	
AND	0010 * 1010	0010	
OR	0010 + 1010	1010	
NOT	$\overline{0010}$	1101	
EOR	0010 eor 1010	1000	
NAND	$0010 \overline{*} 1010$	1101	
shift left	111000	110001	(other results are possible)
shift right	111000	011100	(other results are possible)
etc.			

Figure 13.8 Boolean Operations on Binary Numbers

13.2.1.2 - Binary Mathematics

Negative numbers are a particular problem with binary numbers. As a result there are three common numbering systems used as shown in Figure 13.9. Unsigned binary numbers are common, but they can only be used for positive values. Both signed and 2s compliment numbers allow positive and negative values, but the maximum positive values is reduced by half. 2s compliment numbers are very popular because the hardware and software to add and subtract is simpler and faster. All three types of numbers will be found in PLCs.

Type	Description	Range for Byte
unsigned	binary numbers can only have positive values.	0 to 255
signed	the most significant bit (MSB) of the binary number is used to indicate positive/negative.	-127 to 127
2s compliment	negative numbers are represented by complimenting the binary number and then adding 1.	-128 to 127

Figure 13.9 Binary (Integer) Number Types

Examples of signed binary numbers are shown in Figure 13.10. These numbers use the most significant bit to indicate when a number is negative.

decimal	binary byte
2	00000010
1	00000001
0	00000000
-0	10000000
-1	10000001
-2	10000010

Note: there are two zeros

Figure 13.10 Signed Binary Numbers

An example of 2s compliment numbers are shown in Figure 13.11. Basically, if the number is positive, it will be a regular binary number. If the number is to be negative, we start the positive number, compliment it (reverse all the bits), then add 1. Basically when these numbers are negative, then the most significant bit is set. To convert from a negative 2s compliment number, subtract 1, and then invert the number.

decimal	binary byte
2	00000010
1	00000001
0	00000000
-1	11111111
-2	11111110

METHOD FOR MAKING A NEGATIVE NUMBER

1. write the binary number for the positive

   for -30 we write 30 = 00011110

2. Invert (compliment) the number

   00011110 becomes 11100001

3. Add 1

   11100001 + 00000001 = 11100010

Figure 13.11 2s Compliment Numbers

Using 2s compliments for negative numbers eliminates the redundant zeros of signed binaries, and makes the hardware and software easier to implement. As a result most of the integer operations in a PLC will do addition and subtraction using 2s compliment numbers. When adding 2s compliment numbers, we don't need to pay special attention to negative values. And, if we want to subtract one number from another, we apply the twos compliment to the value to be subtracted, and then apply it to the other value.

Figure 13.12 shows the addition of numbers using 2s compliment numbers. The three operations result in zero, positive and negative values. Notice that in all three operation the top number is positive, while the bottom operation is negative (this is easy to see because the MSB of the numbers is set). All three of the additions are using bytes, this is important for considering the results of the calculations. In the left and right hand calculations the additions result in a 9th bit - when dealing with 8 bit numbers we call this bit the carry *C*. If the calculation started with a positive and negative value, and ended up with a carry bit, there is no problem, and the carry bit should be ignored. If doing the calculation on a calculator you will see the carry bit, but when using a PLC you must look elsewhere to find it.

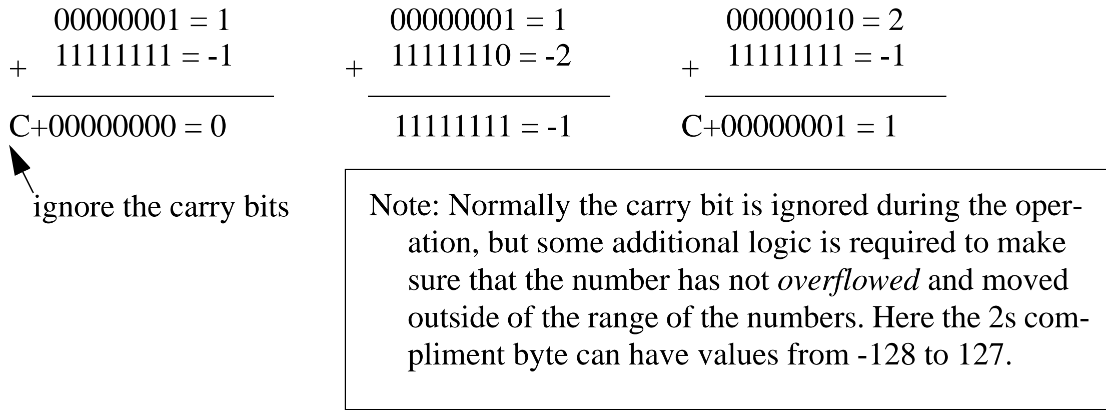
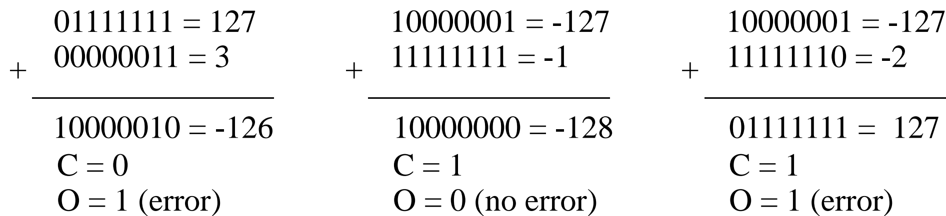


Figure 13.12 Adding 2s Compliment Numbers

The integers have limited value ranges, for example a 16 bit word ranges from -32,768 to 32,767. In some cases calculations will give results outside this range, and the Overflow *O* bit will be set. (Note: an overflow condition is a major error, and the PLC will probably halt when this happens.) For an addition operation the Overflow bit will be set when the sign of both numbers is the same, but the sign of the result is opposite. When the signs of the numbers are opposite an overflow cannot occur. This can be seen in Figure 13.13 where the numbers two of the three calculations are outside the range. When this happens the result goes from positive to negative, or the other way.



Note: If an overflow bit is set this indicates that a calculation is outside and acceptable range. When this error occurs the PLC will halt. Do not ignore the limitations of the numbers.

Figure 13.13 Carry and Overflow Bits

These bits also apply to multiplication and division operations. In addition the PLC will also have bits to indicate when the result of an operation is zero *Z* and negative *N*.

13.2.2 Other Base Number Systems

Other number bases are typically converted to and from binary for storage and mathematical operations. Hexadecimal numbers are popular for representing binary values because they are quite compact compared to binary. (Note: large binary numbers with a long string of 1s and 0s are next to impossible to read.) Octal numbers are also popular for inputs and outputs because they work in counts of eight; inputs and outputs are in counts of eight.

An example of conversion to, and from, hexadecimal is shown in Figure 13.14 and Figure 13.15. Note that both of these conversions are identical to the methods used for binary numbers, and the same techniques extend to octal numbers also.

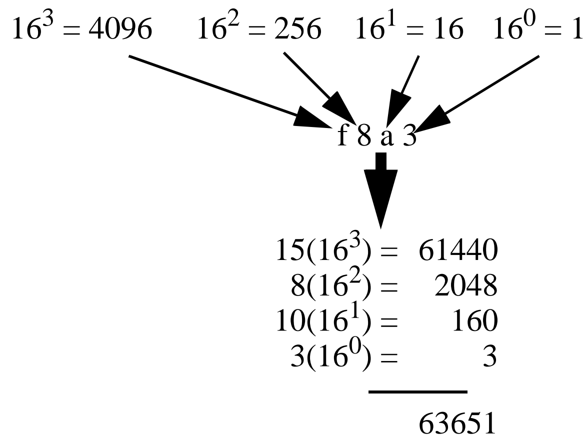


Figure 13.14 Conversion of a Hexadecimal Number to a Decimal Number

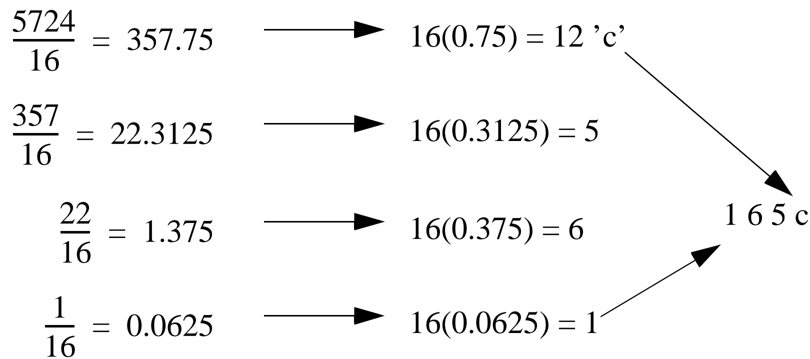


Figure 13.15 Conversion from Decimal to Hexadecimal

13.2.3 BCD (Binary Coded Decimal)

Binary Coded Decimal (BCD) numbers use four binary bits (a nibble) for each digit. (Note: this is not a base number system, but it only represents decimal digits.) This means that one byte can hold two digits from *00* to *99*, whereas in binary it could hold from 0 to 255. A separate bit must be assigned for negative numbers. This method is very popular when numbers are to be output or input to the computer. An example of a BCD number is shown in Figure 13.16. In the example there are four digits, therefore 16 bits are required. Note that the most significant digit and bits are both on the left hand side. The BCD number is the binary equivalent of each digit.

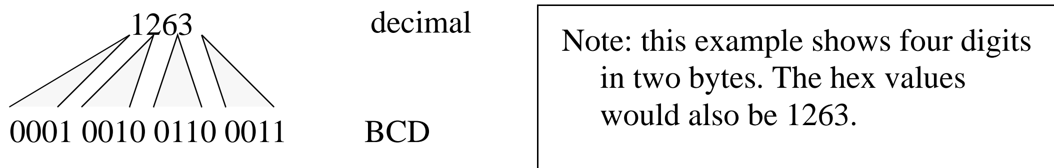


Figure 13.16 A BCD Encoded Number

Most PLCs store BCD numbers in words, allowing values between *0000* and *9999*. They also provide functions to convert to and from BCD. It is also possible to calculations with BCD numbers, but this is uncommon, and when necessary most PLCs have functions to do the calculations. But, when doing calculations you should probably avoid BCD and use integer mathematics instead. Try to be aware when your numbers are BCD values and convert them to *integer* or binary value before doing any calculations.

13.3 DATA CHARACTERIZATION

13.3.1 ASCII (American Standard Code for Information Interchange)

When dealing with non-numerical values or data we can use plain text characters and strings. Each character is given a unique identifier and we can use these to store and interpret data. The ASCII (American Standard Code for Information Interchange) is a very common character encryption system is shown in Figure 13.17 and Figure 13.18. The table includes the basic written characters, as well as some special characters, and some control codes. Each one is given a unique number. Consider the letter *A*, it is readily recognized by most computers world-wide when they see the number 65.

decimal	hexadecimal	binary	ASCII	decimal	hexadecimal	binary	ASCII
0	0	00000000	NUL	32	20	00100000	space
1	1	00000001	SOH	33	21	00100001	!
2	2	00000010	STX	34	22	00100010	“
3	3	00000011	ETX	35	23	00100011	#
4	4	00000100	EOT	36	24	00100100	$
5	5	00000101	ENQ	37	25	00100101	%
6	6	00000110	ACK	38	26	00100110	&
7	7	00000111	BEL	39	27	00100111	‘
8	8	00001000	BS	40	28	00101000	(
9	9	00001001	HT	41	29	00101001	)
10	A	00001010	LF	42	2A	00101010	*
11	B	00001011	VT	43	2B	00101011	+
12	C	00001100	FF	44	2C	00101100	,
13	D	00001101	CR	45	2D	00101101	-
14	E	00001110	S0	46	2E	00101110	.
15	F	00001111	S1	47	2F	00101111	/
16	10	00010000	DLE	48	30	00110000	0
17	11	00010001	DC1	49	31	00110001	1
18	12	00010010	DC2	50	32	00110010	2
19	13	00010011	DC3	51	33	00110011	3
20	14	00010100	DC4	52	34	00110100	4
21	15	00010101	NAK	53	35	00110101	5
22	16	00010110	SYN	54	36	00110110	6
23	17	00010111	ETB	55	37	00110111	7
24	18	00011000	CAN	56	38	00111000	8
25	19	00011001	EM	57	39	00111001	9
26	1A	00011010	SUB	58	3A	00111010	:
27	1B	00011011	ESC	59	3B	00111011	;
28	1C	00011100	FS	60	3C	00111100	<
29	1D	00011101	GS	61	3D	00111101	=
30	1E	00011110	RS	62	3E	00111110	>
31	1F	00011111	US	63	3F	00111111	?

Figure 13.17 ASCII Character Table

decimal	hexadecimal	binary	ASCII	decimal	hexadecimal	binary	ASCII
64	40	01000000	@	96	60	01100000	‘
65	41	01000001	A	97	61	01100001	a
66	42	01000010	B	98	62	01100010	b
67	43	01000011	C	99	63	01100011	c
68	44	01000100	D	100	64	01100100	d
69	45	01000101	E	101	65	01100101	e
70	46	01000110	F	102	66	01100110	f
71	47	01000111	G	103	67	01100111	g
72	48	01001000	H	104	68	01101000	h
73	49	01001001	I	105	69	01101001	i
74	4A	01001010	J	106	6A	01101010	j
75	4B	01001011	K	107	6B	01101011	k
76	4C	01001100	L	108	6C	01101100	l
77	4D	01001101	M	109	6D	01101101	m
78	4E	01001110	N	110	6E	01101110	n
79	4F	01001111	O	111	6F	01101111	o
80	50	01010000	P	112	70	01110000	p
81	51	01010001	Q	113	71	01110001	q
82	52	01010010	R	114	72	01110010	r
83	53	01010011	S	115	73	01110011	s
84	54	01010100	T	116	74	01110100	t
85	55	01010101	U	117	75	01110101	u
86	56	01010110	V	118	76	01110110	v
87	57	01010111	W	119	77	01110111	w
88	58	01011000	X	120	78	01111000	x
89	59	01011001	Y	121	79	01111001	y
90	5A	01011010	Z	122	7A	01111010	z
91	5B	01011011	[	123	7B	01111011	{
92	5C	01011100	yen	124	7C	01111100	|
93	5D	01011101	]	125	7D	01111101	}
94	5E	01011110	^	126	7E	01111110	r arr.
95	5F	01011111	_	127	7F	01111111	l arr.

Figure 13.18 ASCII Character Table

This table has the codes from 0 to 127, but there are more extensive tables that contain special graphics symbols, international characters, etc. It is best to use the basic codes, as they are supported widely, and should suffice for all controls tasks.

An example of a string of characters encoded in ASCII is shown in Figure 13.19.

e.g. The sequence of numbers below will convert to

A W e e T e s t

A	65
space	32
W	87
e	101
e	101
space	32
T	84
e	101
s	115
t	116

Figure 13.19 A String of Characters Encoded in ASCII

When the characters are organized into a string to be transmitted and *LF* and/or *CR* code are often put at the end to indicate the end of a line. When stored in a computer an ASCII value of zero is used to end the string.

13.3.2 Parity

Errors often occur when data is transmitted or stored. This is very important when transmitting data in noisy factories, over phone lines, etc. Parity bits can be added to data as a simple check of transmitted data for errors. If the data contains error it can be retransmitted, or ignored.

A parity bit is normally a 9th bit added onto an 8 bit byte. When the data is encoded the number of true bits are counted. The parity bit is then set to indicate if there are an even or odd number of true bits. When the byte is decoded the parity bit is checked to make sure it that there are an even or odd number of data bits true. If the parity bit is not satisfied, then the byte is judged to be in error. There are two types of parity, even or odd. These are both based upon an even or odd number of data bits being true. The odd parity bit is true if there are an odd number of bits on in a binary number. On the other hand the Even parity is set if there are an even number of true bits. This is illustrated in Figure 13.20.

	data bits	parity bit
Odd Parity	10101110	1
	10111000	0
Even Parity	00101010	0
	10111101	1

Figure 13.20 Parity Bits on a Byte

Parity bits are normally suitable for single bytes, but are not reliable for data with a number of bits.

Note: Control systems perform important tasks that can be dangerous in certain circumstances. If an error occurs there could be serious consequences. As a result error detection methods are very important for control system. When error detection occurs the system should either be *robust* enough to recover from the error, or the system should *fail-safe*. If you ignore these design concepts you will eventually cause an accident.

13.3.3 Checksums

Parity bits are suitable for a few bits of data, but checksums are better for larger data transmissions. These are simply an algebraic sum of all of the data transmitted. Before data is transmitted the numeric values of all of the bytes are added. This sum is then transmitted with the data. At the receiving end the data values are summed again, and the total is compared to the checksum. If they match the data is accepted as good. An example of this method is shown in Figure 13.21.

```
        DATA
                124
                43
                255
                9
                27
                47
--------------------
CHECKSUM
                505
```

Figure 13.21 A Checksum

Checksums are very common in data transmission, but these are also hidden from the average user. If you plan to transmit data to or from a PLC you will need to consider parity and checksum values to verify the data. Small errors in data can have major consequences in received data. Consider an oven temperature transmitted as a binary integer (1023d = 0000 0100 0000 0000b). If a single bit were to be changed, and was not detected the temperature might become (0000 0110 0000 0000b = 1535d) This small change would dramatically change the process.

13.3.4 Gray Code

Parity bits and checksums are for checking data that may have any value. Gray code is used for checking data that must follow a binary sequence. This is common for devices such as angular encoders. The concept is that as the binary number counts up or down, only one bit changes at a time. Thus making it easier to detect erroneous bit changes. An example of a gray code sequence is shown in Figure 13.22. Notice that only one bit changes from one number to the next. If more than a single bit changes between numbers, then an error can be detected.

> ASIDE: When the signal level in a wire rises or drops, it induces a magnetic pulse that excites a signal in other nearby lines. This phenomenon is known as *cross-talk*. This signal is often too small to be noticed, but several simultaneous changes, coupled with background noise could result in erroneous values.

decimal	gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Figure 13.22 Gray Code for a Nibble

13.4 SUMMARY

- Binary, octal, decimal and hexadecimal numbers were all discussed.
- 2s compliments allow negative binary numbers.
- BCD numbers encode digits in nibbles.
- ASCII values are numerical equivalents for common alphanumeric characters.
- Gray code, parity bits and checksums can be used for error detection.

13.5 PRACTICE PROBLEMS

1. Why are binary, octal and hexadecimal used for computer applications?

2. Is a word is 3 nibbles?

3. What are the specific purpose for Gray code and parity?

4. Convert the following numbers to/from binary

a) from base 10: 54,321   b) from base 2: 110000101101

5. Convert the BCD number below to a decimal number,

0110 0010 0111 1001

6. Convert the following binary number to a BCD number,

0100 1011

7. Convert the following binary number to a Hexadecimal value,

0100 1011

8. Convert the following binary number to a octal,

0100 1011

9. Convert the decimal value below to a binary byte, and then determine the odd parity bit,

97

10. Convert the following from binary to decimal, hexadecimal, BCD and octal.

a) 101101   c) 10000000001

b) 11011011   d) 0010110110101

11. Convert the following from decimal to binary, hexadecimal, BCD and octal.

a) 1 c) 20456
b) 17 d) -10

12. Convert the following from hexadecimal to binary, decimal, BCD and octal.

a) 1 c) ABC
b) 17 d) -A

13. Convert the following from BCD to binary, decimal, hexadecimal and octal.

a) 1001 c) 0011 0110 0001
b) 1001 0011 d) 0000 0101 0111 0100

14. Convert the following from octal to binary, decimal, hexadecimal and BCD.

a) 7 c) 777
b) 17 d) 32634

15.
a) Represent the decimal value thumb wheel input, 3532, as a Binary Coded Decimal (BCD) and a Hexadecimal Value (without using a calculator).
i) BCD
ii) Hexadecimal
b) What is the corresponding decimal value of the BCD value, 1001111010011011?

16. Add/subtract/multiply/divide the following numbers.

a) binary 101101101 + 01010101111011
b) hexadecimal 101 + ABC
c) octal 123 + 777
d) binary 110110111 - 0101111
e) hexadecimal ABC - 123
f) octal 777 - 123
g) binary 0101111 - 110110111
h) hexadecimal 123-ABC
i) octal 123 - 777
j) 2s complement bytes 10111011 + 00000011
k) 2s complement bytes 00111011 + 00000011
l) binary 101101101 * 10101
m) octal 123 * 777
n) octal 777 / 123
o) binary 101101101 / 10101
p) hexadecimal ABC / 123

17. Do the following operations with 8 bit bytes, and indicate the condition of the overflow and carry bits.
    a) 10111011 + 00000011
    b) 00111011 + 00000011
    c) 11011011 + 11011111
    d) 110110111 - 01011111
    e) 01101011 + 01111011
    f) 10110110 - 11101110

18. Consider the three BCD numbers listed below.

    1001 0110 0101 0001
    0010 0100 0011 1000
    0100 0011 0101 0001

    a) Convert these numbers to their decimal values.
    b) Convert the decimal values to binary.
    c) Calculate a checksum for all three binary numbers.
    d) What would the even parity bits be for the binary words found in b).

19. Is the 2nd bit set in the hexadecimal value F49?

20. Explain where grey code occurs when creating Karnaugh maps.

21. Convert the decimal number 1000 to a binary number, and then to hexadecimal.

13.6 PRACTICE PROBLEM SOLUTIONS

1. base 2, 4, 8, and 16 numbers translate more naturally to the numbers stored in the computer.

2. no, it is four nibbles

3. Both of these are coding schemes designed to increase immunity to noise. A parity bit can be used to check for a changed bit in a byte. Gray code can be used to check for a value error in a stream of continuous values.

4. a) 1101 0100 0011 0001, b) 3117

5. 6279

6. 0111 0101

7. 4B

8. 113

9. 1100001 odd parity bit = 1

10.

binary	101101	11011011	10000000001	0010110110101
BCD	0100 0101	0010 0001 1001	0001 0000 0010 0101	0001 0100 0110 0001
decimal	45	219	1025	1461
hex	2D	5D	401	5B5
octal	55	333	2001	2665

11.

decimal	1	17	20456	-10
BCD	0001	0001 0111	0010 0000 0100 0101 0110	-0001 0000
binary	1	10001	0100 1111 1110 1000	1111 1111 1111 0110
hex	1	11	4FE8	FFF6
octal	1	21	47750	177766

12.

hex	1	17	ABC	-A
BCD	0001	0010 0011	0010 0111 0100 1000	-0001 0000
binary	1	10111	0000 1010 1011 1100	1111 1111 1111 0110
decimal	1	23	2748	-10
octal	1	27	5274	177766

13.

BCD	1001	1001 0011	0011 0110 0001	0000 0101 0111 0100
binary	1001	101 1101	1 0110 1001	10 0011 1110
decimal	9	93	361	0574
hex	9	5D	169	23E
octal	11	135	551	1076

14.

octal	7	17	777	32634
binary	111	1111	1 1111 1111	0011 0101 1001 1100
decimal	7	15	511	13724
hex	7	F	1FF	359C
BCD	0111	0001 0101	0101 0001 0001	0001 0011 0111 0010 0100

15. a) 3532 = 0011 0101 0011 0010 = DCC, b0 the number is not a valid BCD

16.

a) 0001 0110 1110 1000

b) BBD

c) 1122

d) 0000 0001 1000 1000

e) 999

f) 654

g) 1111 1110 0111 1000

h) -999

i) -654

j) 0000 0001 0111 1010

k) 0000 0000 0011 1110

l) 0001 1101 1111 0001

m) 122655

n) 6

o) 0000 0000 0001 0001

p) 9

17.

a) 10111011 + 00000011=1011 1110

b) 00111011 + 00000011=0011 1110

c) 11011011 + 11011111=1011 1010+C+O

d) 110110111 - 01011111=0101 1000+C+O

e) 01101011 + 01111011=1110 0110

f) 10110110 - 11101110=1100 1000

18. a) 9651, 2438, 4351, b) 0010 0101 1011 0011, 0000 1001 1000 0110, 0001 0000 1111 1111, c) 16440, d) 1, 0, 0

19. The binary value is 1111 0100 1001, so the second bit is 0

20. when selecting the sequence of bit changes for Karnaugh maps, only one bit is changed at a time. This is the same method used for grey code number sequences. By using the code the bits in the map are naturally grouped.

21.

$$1000_{10} = 1111101000_2 = 3e8_{16}$$

13.7 ASSIGNMENT PROBLEMS

1. Why are hexadecimal numbers useful when working with PLCs?