Chapter 1. Number Systems and Codes

1.1. INTRODUCTION

Digital computers have brought about the information age that we live in today. Computers are important tools because they can locate and process enormous amounts of information very quickly and efficiently. They allow us to utilize our mathematical disciplines to the fullest. In one second, a computer can perform calculations that would take a person several weeks to do by hand. However, computers are not creative and will only do what we tell them. The list of instructions that tells the computer what to do is called a computer program.

System reliability, fast performance, and efficient information storage and retrieval are major factors in the acceptance and use of digital computer systems. The high reliability of computer systems is due largely to the fact that all data is in a digital format. Computers are designed such that digital formatted data can be processed quickly and efficiently.

1.1.1. Digital Logic States

A computer is made up of many digital circuits that pass information between components in the form of digital signals. These signals may represent either instructions or data to be processed. A digital signal can be considered a *logic variable* that has only one of two possible values at any moment in time. These values are called *logic states*. Figure 1 describes the common notation for logic states.

Figure 1: Logic State Values

TRUE	ON	1	YES
FALSE	OFF	0	NO

The binary (Base 2) number system is often used to represent the states of a group of logic variables at specific instant in time. Each digit of a binary number is called a *bit*. Figure 2 describes the binary equivalent to the decimal (Base 10) numbers 0 through 9. Only two possible values can exist for each binary digit, either "1" or "0". Note that it takes four bits to represent the decimal numbers 8 and 9. A thorough discussion of the binary number system, including conversion methods, is discussed in the next section.

Figure 2

Base 10	Base 2
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

1.2. NUMBER SYSTEMS

The decimal number system (base 10) has become the standard number system used by people for counting and mathematical operations. The decimal (base 10) system is used by most cultures because people have ten fingers. Each finger is used to represent one of ten possible values that a digit can possess. Originally, European origin cultures used Roman numerals. However, today the Arabic numeral symbols (0123456789) have become dominant worldwide because of their efficiency in doing arithmetic and mathematical operations.

Computers do not have 10 fingers. However, they are made up of electronic switches that represent Boolean variables in either a 1 or 0 state. For this reason, the binary (base 2) number system is used to represent the states of Boolean variables. A single binary digit is called a *bit*, which is in either a 1 or 0 state. A group of eight bits is called a *byte*. A half byte, which is a set of four bits, is often called a *nibble*.

Discussed in this chapter are the data formats used by computers to represent numbers and alphanumeric data. The base of a number is denoted in this chapter by the subscript 2 for binary, 10 for decimal, and 16 for hexadecimal (base 16). In this text, commas are used with binary numbers to separate groups of four bits, which make the number more readable.

Hexadecimal (Base 16) = 17_{16}

Decimal (Base 10) = 23_{10}

Binary (Base 2) = $0001,0111_2$

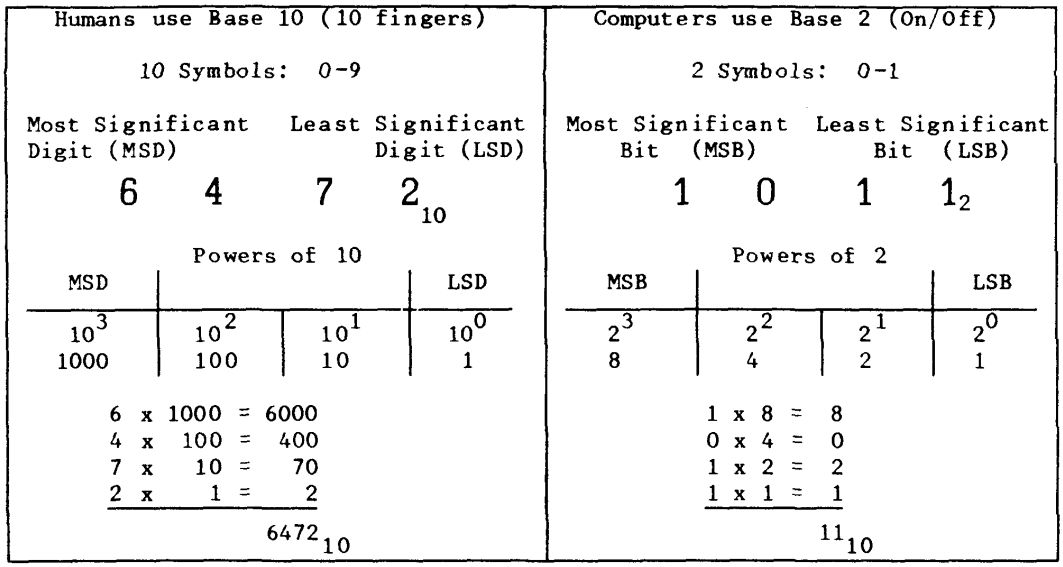
} These numbers are equivalent magnitudes in different bases

1.2.1. Unsigned Binary Numbers

Binary (base-2) numbers can be used to represent various whole number quantities called *integers*. The binary number system operates similar to the decimal system. However, only two symbols (0 and 1) exist for each bit and ten symbols (0 through 9) exist for each digit of the decimal system.

Figure 3 illustrates a decimal and binary equivalent representation for the same number. Both number Systems are right justified. That is, the least significant bit (LSB) of a binary number is always the rightmost bit, just as the least significant digit (LSD) of a decimal number is the rightmost digit. Similarly, the most-significant-bit (MSB) represents the highest power of two required for representing a number and is the left-most bit. Each digit of a decimal number represents a power of 10 and each bit of a binary number represents a power of 2 higher then the bit to its right.

Figure 3 Decimal and Binary Number Systems



1.2.1.1. Unsigned Binary to Decimal Conversion

The four bit binary number of Figure 3 is easily converted to decimal notation. Simply determine the sum of the powers of two for all bits with a 1. Bits with a 0 are not added to this sum. Therefore, the conversion of 1011_2 to decimal is performed by the sum:

$$(1)2^3 + (0)2^2 + (1)2^1 + (1)2^0 = 8 + 2 + 1 = 11_{10}$$

Generally, the minimum data word length for microcomputers is 8 bits or one byte. For an eight-bit number represented by the eight bits $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ the least significant bit, b_0, represents the 2^0 or the 1's place and the most significant bit, b_7, represents the 2^7 or the 128's place. Conversion of an eight-bit number is represented by the following equation and illustrated in example 1:

$$b_7 2^7 + b_6 2^6 + b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0$$

$$= \quad b_7\,(128) + b_6\,(64) + b_5(32) + b_4\,(16) + b_3\,(8) + b_2\,(4) + b_1\,(2) + b_0\,(1)$$

The largest number that can be represented by eight bits would have a 1 for all eight bits b_7 through b_0. This corresponds to $1111,1111_2 = 255_{10}$. The binary number $0000,0000_2$ represent decimal zero. Therefore, eight bits may be used to represent any integer decimal number within the range of 0 to 255 inclusive. Additional bits are required to represent decimal integers greater than 255.

Example 1: Convert $100,1000_2$ to Decimal

MSB						LSB
1	0	0	1	0	0	0_2
2^6	2^5	2^4	2^3	2^2	2^1	2^0
64x1 +	32x0 +	16x0 +	8x1 +	4x0 +	2x0 +	1x0
64 +	0 +	0 +	8 +	0 +	0 +	0
						72_{10}

1.2.1.2. Positive Decimal to Unsigned Binary Conversion

To convert a decimal number to a binary number is more tedious than binary to decimal conversion. Tables such as Figure 4 are often used to facilitate these conversions. As an alternative to tables, a direct mathematical procedure is shown in Example 2. Consider the decimal number 19. Conversion is performed using successive divisions by 2. First 19 is divided by 2, which will generate a quotient of 9 and remainder of 1. Next divide the preceding quotient 9 by 2, which will generate the next quotient of 4 and remainder of 1.

Example 2: Convert 19 to Binary

	Q	R		
19 ÷ 2 =	9	1	(LSB	Least Significant Bit)
9 ÷ 2 =	4	1		
4 ÷ 2 =	2	0		
2 ÷ 2 =	1	0		
1 ÷ 2 =	0	1	(MSB	Most Significant Bit)

Therefore $19_{10} = 10011_2 = 0001,0011_2$

Continue the division by 2 until a quotient of 0 exists. The remainder column represents the binary equivalent number. The final remainder is the most significant bit and the first remainder is the least significant bit. Therefore, 19 decimal is represented by the binary number 10011. Binary numbers are often zero filled to eight bits resulting in 0001,0011 to represent decimal 19.

Figure 4 Unsigned Decimal to Binary Conversions

Decimal	Binary	Decimal	Binary
0	00000000	32	00100000
1	00000001	33	00100001
2	00000010	34	00100010
3	00000011	...	
4	00000100	62	00111110
5	00000101	63	00111111
6	00000110	64	01000000
7	00000111	65	01000001
8	00001000	66	01000010
9	00001001	...	
10	00001010	126	01111110
11	00001011	127	01111111
12	00001100	128	10000000
13	00001101	129	10000001
14	00001110	130	10000010
15	00001111	...	
16	00010000	252	11111100
17	00010001	253	11111101
...		254	11111110
31	00011111	255	11111111

1.2.2. Signed Binary Numbers

Signed binary number representations are used to represent both positive and negative numbers. They also allow signed binary addition and subtraction operations, which may yield negative results.

The *2's complement* data format is utilized to represent signed binary numbers. An abbreviated table of 8-bit 2's complement numbers is shown in Figure 5. For positive numbers within the range of 127_{10} through 0_{10}, the 2's complement representation is identical to the unsigned binary format. Note that all positive numbers have a “0” in the most significant bit.

However, for negative numbers a conversion procedure is required. Note that all negative numbers in Table 5 have a “1” in the most significant bit. Conversion is performed for 2's complement negative numbers using the following three step method:

Step 1) Determine the unsigned binary number magnitude
Step 2) Complement (invert) the state of each bit
Step 3) Add 1 to the result

Figure 5 Some Signed Binary to Decimal Conversions

+	Two's complement code	−	Two's complement code
+ 127	01111111	− 128	10000000
+ 126	01111110	− 127	10000001
+ 125	01111101	− 126	10000010
. . .		− 125	10000011
		. . .	
+ 65	01000001	− 65	10111111
+ 64	01000000	− 64	11000000
+ 63	00111111	− 63	11000001
. . .		. . .	
+ 33	00100001	− 33	11011111
+ 32	00100000	− 32	11100000
+ 31	00011111	− 31	11100001
. . .		. . .	
+ 17	00010001	− 17	11101111
+ 16	00010000	− 16	11110000
+ 15	00001111	− 15	11110001
+ 14	00001110	− 14	11110010
+ 13	00001101	− 13	11110011
+ 12	00001100	− 12	11110100
+ 11	00001011	− 11	11110101
+ 10	00001010	− 10	11110110
+ 9	00001001	− 9	11110111
+ 8	00001000	− 8	11111000
+ 7	00000111	− 7	11111001
+ 6	00000110	− 6	11111010
+ 5	00000101	− 5	11111011
+ 4	00000100	− 4	11111100
+ 3	00000011	− 3	11111101
+ 2	00000010	− 2	11111110
+ 1	00000001	− 1	11111111
+ 0	00000000		

Example 3 illustrates this 2's complement conversion procedure using four different numbers. This procedure is only used for converting negative numbers to 2's complement form. Again, positive numbers will have the same form for both unsigned binary and 2's complement representations.

Example 3: The 2's Complement Representation

Convert the following decimal Numbers:

	-5_{10}	-65_{10}	-126_{10}	$+63_{10}$
Step 1) Determine the unsigned binary number which represents the magnitude;	0000,0101	0100,0001	0111,1110	Positive Number
Step 2) Complement (invert) the state of each bit;	1111,1010	1011,1110	1000,0001	↓
Step 3) Add 1 to the result.	1111,1011	1011,1111	1000,0010	0011,1111

The same three-step procedure can also be used to change the sign of the number. This is useful when converting a negative 2's complement number to decimal and for the subtraction operation. Subtraction can be performed on 2's complement numbers by first performing the three-step 2's complement procedure described to change the sign of the subtrahend. Then add the two numbers together.

The range of 8-bit 2's complement numbers is +127 to -128 inclusive, as shown in Figure 5. The most significant bit is the sign bit. Again if $b_7 = 0$, the number is positive, and if $b_7 = 1$ the number is negative. Note that only one value exists for zero in 2's complement numbers, and zero is considered a positive number. Therefore, the highest value of positive numbers is 127 instead of 128 as one might expect.

1.2.3. Binary Number Magnitude

The number of bits used for the data word restricts the magnitude of decimal numbers that can be represented by binary numbers. Binary numbers with eight bits can represent unsigned numbers in the range from 255 through 0, or signed numbers in the range from +127 through -128. Numbers outside of these ranges cannot be represented unless additional bits are used to increase the data word size.

Binary numbers can represent a maximum of 2^n decimal numbers, where n is the number of bits of the data word. For unsigned numbers the range begins at zero. For signed binary numbers the center of the range is zero.

Consider the case when the data word size is increased from one byte to two bytes (16 bits). A two-byte binary number can be used to represent 65,536 different decimal numbers. For unsigned binary numbers, 16 bits would represent decimal numbers in the range of 65,535 to 0. The signed 2's complement range is from +32,767 to -32,768. Once again, zero is considered a positive number. Therefore, the range of negative numbers appears to be one more than positive numbers.

Computers have a standard word size with all data represented as some multiple of eight bits (i.e. 8, 16, 32, 64). Most personal computers today use 32 bit computer words. Therefore, unsigned integers can represent whole decimal numbers in the range between 0 to 4,294,967,296.

1.3. Floating Point Representations

Signed integer numbers are represented using the 2's complement format of Section 1.2.2. An alternate approach is required to represent real numbers that contain fractional parts.

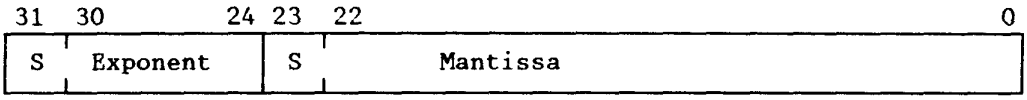
The floating-point representation is used to represent real numbers in much the same way as scientific notation. Consider the number -0.0004772, which can be represented in scientific notation as -0.4772×10^{-3}. The mantissa is normalized to be a number between ±0.9999 and ±0.1000. The exponent -3 is a signed integer quantity representing the power of ten that the mantissa is multiplied. The actual bit format used to represent floating point numbers varies dependent on the data word size of the computer.

Figure 6 shows a typical 32-bit format used to represent floating point numbers. For example both the mantissa and exponent can be represented using 2's complement numbers. In both cases a sign bit of 0 represents a positive number and a sign bit of 1 represents a negative number. The number of bits in the exponent determines the range of powers of ten. An 8 bit exponent can represent powers of ten from 10^{+127} to 10^{-128}. A 24-bit mantissa can represent signed decimal numbers of six significant digits for the range +999,999 to -999,999.

If more significant digits are required for an application, allocating more bits to the mantissa can increase the mantissa size. Most computer programming languages allow a higher precision

(more significant digits) floating point number representation that is often called double precision floating point numbers.

Figure 6 Floating Point Number Representation



1.4. Hexadecimal Numbers

Representing eight bits of data as a string of 1's and 0's can be tedious.

The hexadecimal number system is used to simplify data representation by encoding 4 bits as one symbol. Hexadecimal numbers comprise the base 16 number system. The hexadecimal number system has sixteen different symbols, which represent the value of each hexadecimal digit. As shown in the conversion table of Figure 7a, 0 through 9 are used to represent the first ten hexadecimal symbols, and letters A through F are used to represent the last six symbols. Each hexadecimal symbol represents one of the sixteen possible combinations of four bits. Therefore, eight bits of data is more easily represented as two hexadecimal digits.

1.4.1. Binary to Hexadecimal Conversion

To convert a binary number to hexadecimal is quite easy. Starting from the least significant bit (b_0), group the binary bits into groups of four. Use the table of Figure 7a to determine the hexadecimal symbol that represents each group of four bits.

For example:

$1000,1100,0111,1010_2$ =

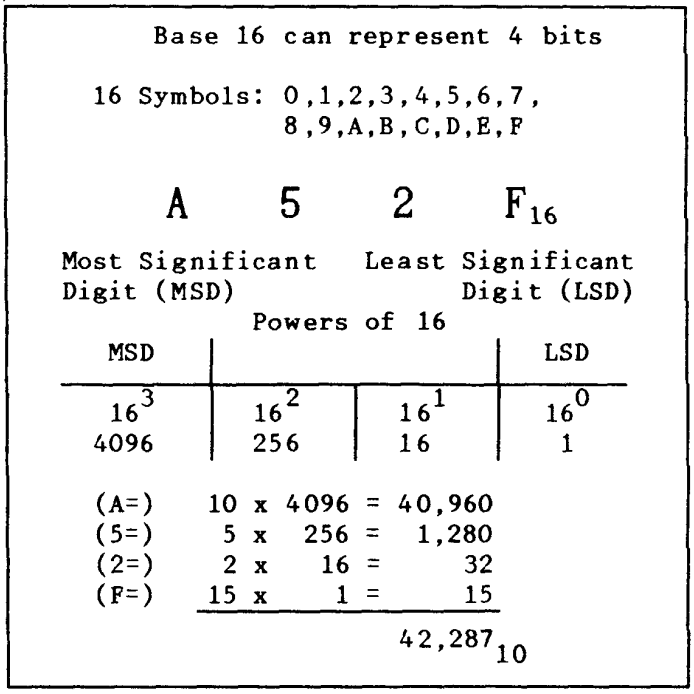
8 C 7 A_{16}

Figure 7 Hexadecimal Numbers

a)

Base 16	Base 2	Base 10
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

b)



1.4.2. Hexadecimal to Binary Conversion

Converting from hexadecimal to binary is also quite simple. Use the table of Figure 7a to determine the group of four bits, which represents each digit of the hexadecimal number.

For example:

A 5 2 F_{16} =
$1010,0101,0010,1111_2$

1.5. Alphanumeric Data Representation

People use written language to communicate among themselves and to give instructions to a computer in the form user keyboard input. Alphanumeric data is represented by a binary code for each letter, number, and symbol commonly associated with a typewriter keyboard. The ASCII (American Standard Code for Information Interchange) code is the most commonly used representation for alphanumeric data. Figure 8 provides a table used to convert from either hexadecimal or binary codes to the ASCII characters represented by these codes. All upper and lower case letters, numbers, and symbols used in the English language are in column 2 through 7. Special computer control characters are found in columns 0 and 1. A definition of each of these control characters is located below the conversion table.

An ASCII character is represented by a byte, with bit 7 (MSB) being the parity bit and bits 6 through 0 determined by the conversion table. The parity bit is used for error detection when transmitting and receiving data. Assume the parity bit will be set for '0' parity so that it is equal to '0'.

1.5.1. Binary String to ASCII Character Conversion

Given a string of binary bits or hexadecimal numbers, conversion to the ASCII characters is performed by separating the string into individual bytes. Again, it is assumed that the parity bit is equal to 0.

To convert bits 6 through 0 to ASCII, use the ASCII conversion table of Figure 8. The least significant nibble, bits 3 through 0, represents the rows of the conversion table. Entries are listed in both binary bits and hexadecimal digits. Bits 6 through 4 represent the columns of the table. Once the specific column and row are located, the ASCII character defined by the byte is found at the specified row and column. Continue this conversion process for each byte of the string. This procedure is illustrated in the example below.

As an example, convert the following hexadecimal representation of a binary string with '0' parity to ASCII characters.

Hexadecimal Representation = 576861743F

57	68	61	74	3F
0101,0111	0110,1000	0110,0001	0111,0100	0011,1111

Convert using ASCII Conversion Table gives you the following answer:

W	h	a	t	?

= What?

1.5.2. ASCII Character to Binary String Conversion

The conversion of a string of characters to a string of bits or hexadecimal digits is performed using the reverse process of the above. Find the ASCII character in the ASCII Conversion table. Determine the byte representing this ASCII character by first determining the row of the character in the table. This specifies the least significant nibble, bits 3 through 0. Then determine the column of the character, which

Figure 8: ASCII Conversion Table

HEX	MSD	0	1	2	3	4	5	6	7
LSD	BINARY	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	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
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	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

Control Characters:

NUL	Null	VT	Vertical	SYN	Synchronous Idle
SOH	Start of Heading	FF	Form Feed	ETB	End Transmission Block
STX	Start of Text	CR	Carriage Return	CAN	Cancel
ETX	End of Text	SO	Shift Out	EM	End of Medium
EOT	End of Transmission	SI	Shift In	SUB	Substitute
ENQ	Enquiry	DLE	Data Link	ESC	Escape
ACK	Acknowledge	DC1	Device Control	FS	File Separator
BEL	Bell	DC2	Device Control	GS	Group Separator
BS	Backspace	DC3	Device Control	RS	Record Separator
HT	Horizontal	DC4	Device Control	US	Unit Separator
LF	Line Feed	NAK	Negative	DEL	Delete

specifies bits 6 through 4. Bit 7 is the parity bit and assumed to be 0. Once all eight bits of the ASCII character byte are determined, they can be converted to hexadecimal, if desired. This procedure is illustrated below.

ASCII Character String = What?

	W	h	a	t	?
Conversion	101,0111	110,1000	110,0001	111,0100	011,1111
Parity Added	0101,0111	0110,1000	0110,0001	0111,0100	0011,1111
Hexadecimal	57	68	61	74	3F

1.5.3. Unicode Characters

To enable globalization of computer technology the Unicode character set has been developed to accommodate different characters used by people on planet earth. Unicode is a 16-bit character format, which can accommodate 65,536 different characters. This is large enough to accommodate all known character sets used by humans. A compressed version of Unicode called UTF-8 allows for variable sized computer bits to represent characters. UTF-8 is identical to the ASCII set when using the Standard English language.

Problem Set

1. Perform the following decimal to binary conversions. Verify your answers by performing binary to decimal conversions. Use a minimum of one byte for the answer.
   a) 25 b) 31 c) 173 e) 320 f) -19 g) -7

2. Convert the following hexadecimal numbers to binary. Verify your answers by converting the binary result back to hexadecimal.
   a) 87 b) 23 c) CF d)73B

3. Write the hexadecimal string for the following ASCII character strings. Assume parity bit is ‘0’.
   a) What If?
   b) P.O. Box 185
   c) Saipan
   d) 49931

4. Convert the following hexadecimal string to ASCII characters.
   43 53 32 35 35 20 49 73 20 46 75 6E 21