Digital Design is concerned with the design of digital electronic circuits.

Digital circuits are employed in the design of systems such as digital computers, control systems, data communications, and many other applications that require electronic digital hardware.

This course presents the basic tools for the design of digital circuits and provides methods and procedures suitable for a variety of digital design applications.

In this course, we will cover combinational circuits and sequential circuits with various aspects of commercially available integrated circuits.

Chapter 1: Binary Systems

1.1 Digital Computers and Digital Systems

1.2 Binary Numbers

1.3 Number Base Conversions

1.4 Octal and Hexadecimal Numbers

1.5 Complements

1.6 Signed Binary Numbers

1.7 Binary Codes

1.1 Digital Computers and Digital Systems

Digital computers have made possible many scientific, industrial, and commercial advances that would have been unattainable otherwise.

Computers are used in scientific calculations, commercial and business data processing, air traffic control, space guidance, the educational fields and many other areas.

The most amazing property of a digital computer is its generality. It can follow a sequence of instructions, called a program that operates on a given data.

The user can specify and change programs and /or data according to the specific need.

As a result of this flexibility, general-purpose digital computers can perform a wide variety of information-processing tasks.

The general-purpose digital computer is best-known example of digital system. Other examples include electronic calculators, digital voltmeters and the like.

Characteristic of a digital system is its manipulation of discrete elements of information.

Such discrete elements may be electric impulses, the decimal digits, the letters of the alphabet, arithmetic operations, punctuation marks, or any other set of meaningful symbols.

The placing next to each other of discrete elements of information represents a quantity of information.

For example, the letters d, o, and g form the word dog. The digits 237 form a number.

Thus, a sequence of discrete elements forms a language, that is, a discipline that conveys information.

Early digital computers were used mostly for numerical computations.

In this case, the discrete elements used are the digits.

From this application, the term digital computer has emerged.

Discrete elements of information are represented in a digital system by physical quantities called signals.

Electrical signals such as voltages and currents are the most common.

The signals in all present-day electronic digital systems have only two discrete values and are said to be binary.

The digital-system designer is restricted to the use of binary signals because of the lower reliability of many-valued electronic circuits.

In other words, a circuit with ten states, using one discrete voltage value for each sate, can be designed, but it would possess a very low reliability of operation.

In contrast, a transistor circuit that is either on or off has two possible signal values and can be constructed to be extremely reliable.

Because of this physical restriction of components, and because human logic tends to be binary, digital systems that are constrained to take discrete values are further constrained to take binary values.

To simulate a physical process in a digital computer, the quantities must be quantized.

When the variables of the process are presented by real-time continuous signals, the latter are quantized by an analog-to-digital conversion device.

A physical system whose behavior is described by mathematical equations is simulated in a digital computer by means of numerical methods.

When the problem to be processed is inherently discrete, as in commercial applications, the digital computer manipulates the variables in their natural form.

Block diagram of the digital computer is shown in Fig 1-1.

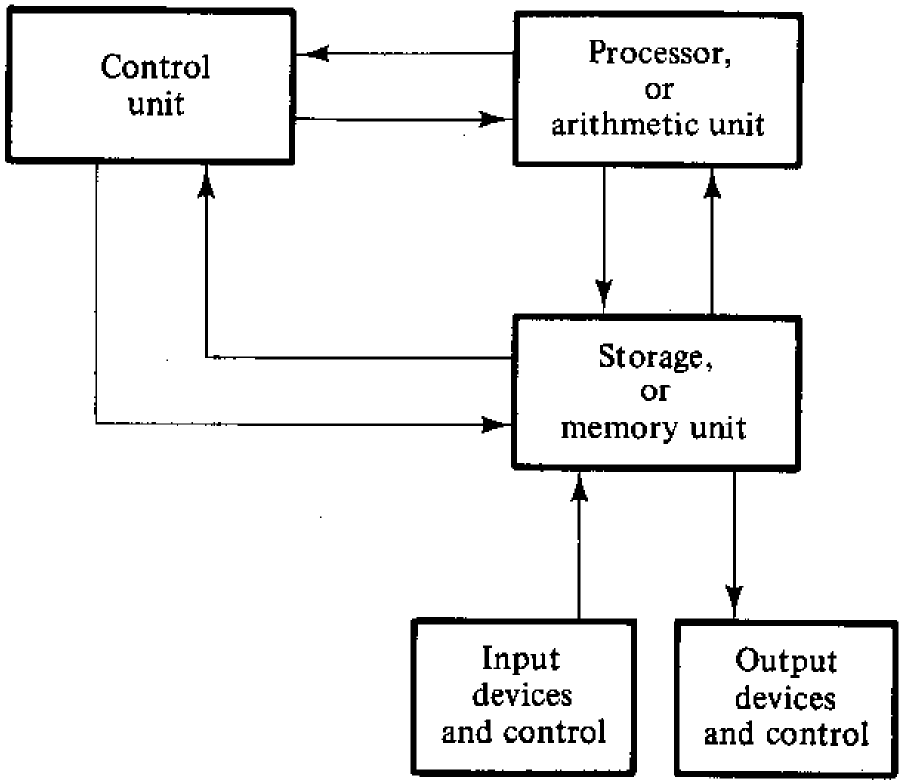

FIGURE 1-1
Block diagram of a digital computer

The memory unit stores programs as well as input, output, and intermediate data.

The processor unit stores programs as well as input, output, and intermediate data.

The processor unit performs arithmetic and other data-processing tasks as specified by a program.

The control unit supervises the flow of information between the various units.

The control unit retrieves the instructions, one by one, from the program that is stored in memory.

For each instruction, the control unit informs the processor to execute the operation specified by the instruction, and the processor manipulates the data as specified by the program.

The program and data prepared by the user are transferred into the memory unit by means of an input device such as a keyboard.

An output device, such as a printer, receives the result of the computations and the printed results are presented to the user.

The input and output devices are special digital systems driven by electro-mechanical parts and controlled by electronic digital circuits.

A digital computer is an interconnection of digital modules.

To understand the operation of each digital module, it is necessary to have a basic knowledge of digital systems and their general behavior.

It has already been mentioned that a digital computer manipulates discrete elements of information and that these elements are represented in the binary form.

Operands used for calculations may be expressed in the binary number system.

Other discrete elements, including the decimal digits, are represented in binary codes.

Data processing is carried out by means of binary logic elements using binary signals.

Quantities are stored in binary storage elements.

The purpose of this chapter is to introduce the various binary concepts as a frame of reference for further detailed study in the succeeding chapters.

1-2 Binary Numbers

A decimal number such as 7392 represents a quantity equal to 7 thousands plus 3 hundreds, plus 9 tens, plus 2 units.

The thousands, hundreds, etc are powers of 10 implied by the position of the coefficients.

To be more exact, 7392 should be written as

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

However, the convention is to write only the coefficients and from their position deduce the necessary powers of 10. In general, a number with a decimal point is represented by a series of coefficients as follows.

$$a_5 a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3}$$

The a_j coefficients are one of the ten digits (0, 1, 2,... 9), and the subscript value *j* gives the place value and hence, the power of 10 by which the coefficient must be multiplied.

$$10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^{-1} a_{-1} + 10^{-2} a_{-2} + 10^{-3} a_{-3}$$

The decimal number system is said to be of base, or radix, 10 because it used ten digits and the coefficients are multiplied by powers of 10.

The binary system is a different number system.

The coefficients of the binary numbers system have two possible values: 0 and 1.

Each coefficient a_j is multiplied by 2^j.

For example, the decimal equivalent of the binary number 11010.11 is 26.75, as shown from the multiplication of the coefficients by powers of 2.

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

In general, a number expressed in base-r system has coefficients multiplied by powers of r:

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r + a_0 \\ + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

The coefficients a_j range in value from 0 to r-1.

To distinguish between numbers of different bases, we enclose the coefficients in parentheses and write a subscript equal to the base used (except sometimes for decimal numbers, where the content makes it obvious that it is decimal). An example of a base-5 number is

$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$

Note that the coefficient values for base 5 can only be 0, 1, 2, 3, 4.

It is customary to borrow the needed r digits for the coefficients from the decimal system when the base of the number is less than 10.

The letters of the alphabet are used to supplement the ten decimal digits when the base of the number is greater than 10.

For example, the hexadecimal (base 16) number system, the first ten digits are borrowed from the decimal system.

The letters A, B, C, D, E, and F are used for digits 10, 11, 12, 13, 14, 15, respectively.

An example of a hexadecimal number is

$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16 + 15 = (46687)_{10}$

The first 16 numbers in decimal, binary, octal, and hexadecimal are listed in Table 1-1.

TABLE 1-1
Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

Arithmetic operations with numbers in base r follow the same rules as for decimal numbers.

When other than the familiar base 10 is used, one must be careful to use only the r allowable digits.

Examples of addition, subtraction, and multiplication of two binary numbers are as follows:

```
augend:      101101    minuend:       101101    multiplicand:    1011
addend:     +100111    subtrahend:   -100111    multiplier:     × 101
           --------                  -------                   ------
sum:        1010100    difference:    000110                     1011
                                                                0000
                                                               1011
                                                               ------
                                                product:       110111
```

The sum of two binary numbers is calculated by the same rules as in decimal, except that the digits of the sum in any significant position can be only 0 or 1.

Any carry obtained in a given significant position is used by the pair of digits used by the pair of digits one significant position higher.

The subtraction is slightly more complicated.

The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit. (A borrow in the decimal system adds 10 to a minuend digit.)

Multiplication is very simple. The multiplier digits are always 1 or 0. Therefore, the partial products are equal either to the multiplicand or to 0.

1-3 Number Base Conversions

A binary number can be converted to decimal by forming the sum of the powers of 2 of those coefficients whose value is 1. For example

$(1010.011)_2 = 2^3 + 2^1 + 2^{-2} + 2^{-3} = (10.375)_{10}$

The binary number has four 1's and the decimal equivalent is found from the sum of four powers of 2.

Similarly, a number expressed in base r can be converted to its decimal equivalent by multiplying each coefficient with the corresponding power of r and adding.

The following is an example of octal-to-decimal conversion.

$(630.4)_8 = 6 \times 8^2 + 3 \times 8 + 4 \times 8^{-1} = (408.5)_{10}$

The conversion from decimal to binary or to any other base-r system is more convenient if the number is separated into an integer part and a fraction part and the conversion of each part is done separately.

The conversion of an integer from decimal binary is best explained by example.

Example 1-1: Convert decimal 41 to binary.

First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of 1/2. The quotient is again divided by 2 to give a new quotient and remainder.

This process is continued until the integer quotient becomes 0.

The coefficients of the desired binary number are obtained from the remainders as follows:

answer: $(41)_{10} = (a_5a_4a_3a_2a_1a_0) = (101001)_2$

The arithmetic process can be manipulated more conveniently as follows:

Integer	Remainder	
41		
20	1	
10	0	
5	0	
2	1	
1	0	
0	1	101001 = answer

The conversion from decimal integers to any base-r system is similar to the example, except that division is done by r instead of 2.

Example 1-2 Convert decimal 153 to octal.

The required base r is 8. First 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3.

Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently manipulated as follows:

$$\begin{array}{r|l} 153 & \\ 19 & 1 \\ 2 & 3 \\ 0 & 2 \end{array} \qquad = (231)_8$$

The conversion of a decimal fraction to binary is accomplished by a method similar to that used for integers.

However, multiplication is used instead of division, and integers are accumulated instead of remainders.

Again, the method is best explained by example.

Example 1-3: Convert $(0.6875)_{10}$ to binary.

First, 0.6875 is multiplied by 2 to give an integer and a fraction. The new fraction is multiplied by 2 to give a new integer and a new fraction. This process is continued until the fraction becomes 0 or until the numbers of digits have sufficient accuracy.

The coefficients of the binary number are obtained from the integers as follows:

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

Answer: $(0.6875)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$

To convert a decimal fraction to a number expressed in base r, a similar procedure is used.

Multiplication is by r instead of 2, and the coefficients found from the integers may range in value from 0 to r-1 instead of 0 and 1.

Example 1-4 Convert $(0.513)_{10}$ to octal.

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

The answer, to seven significant figures, is obtained from the integer part of the products:

$$(0.513)_{10} = (0.406517\ldots)_8$$

The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and fraction separately and then combining the two answers.

Using the results of Examples 1-1 and 1-3, we obtain

$$(41.6875)_{10} = (101001.1011)_2$$

From Examples 1-2 and 1-4, we have

$$(153.513)_{10} = (231.406517)_8$$

1-4 OCTAL AND HEXADECIMAL NUMBERS

The conversion from and to binary, octal, and hexadecimal plays an important part in digital computers.

Since $2^3=8$ and $2^4=16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits.

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three digits each, starting from the binary point and proceeding to the left and the right.

The corresponding octal digit is then assigned to each group.

The following example illustrates the procedure:

$$(\underbrace{10}_{2}\ \underbrace{110}_{6}\ \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{011}_{3}\ .\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{000}_{0}\ \underbrace{110}_{6})_2 = (26153.7460)_8$$

Conversion from binary to hexadecimal is similar, except that the binary number is divided into groups of four digits:

$$(\underbrace{10}_{2}\ \underbrace{1100}_{C}\ \underbrace{0110}_{6}\ \underbrace{1011}_{B}\ .\ \underbrace{1111}_{F}\ \underbrace{0010}_{2})_2 = (2C6B.F2)_{16}$$

The corresponding hexadecimal (or octal) digit for each group of binary digits is easily remembered after studying the values listed in Table 1-1.

Conversion from octal or hexadecimal to binary is done by a procedure reverse to the above.

Each octal digits is converted to its three-digit binary equivalent.

Similarly, each hexadecimal digit is converted to its four-digit binary equivalent.

This is illustrated in the following examples:

$$(673.124)_8 = (\underbrace{110}_{6}\ \underbrace{111}_{7}\ \underbrace{011}_{3}\ .\ \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{100}_{4})_2$$

$$(306.D)_{16} = (\underbrace{0011}_{3}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6}\ .\ \underbrace{1101}_{D})_2$$

Binary numbers are difficult to work with because they require more space to write on compared to octal or hexadecimal numbers.

However, machines use binary numbers because they are easier to represent electrical signals.

For our use we use octal or hexadecimal numbers because they require less space, but when we communicate with machines we convert these octal or hexadecimal numbers to binary.

1-5 Complements

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation.

There are two types of complements for each base-r system: the radix complement and the diminished radix complement.

The first is referred to as the r's complement and the second as the (r-1)'s complement.

When the value of the base r is substituted in the name, the two types are referred to as the 2's complement and 1's complement for binary numbers, and 10's complement and 9's complement for decimal numbers.

Diminished Radix Complement

Given a number N in a base r having n digits, the (r-1)'s complement of N is defined as $(r^n - 1) - N$.

For decimal numbers, r=10 and r-1=9, so the 9's complement of N is (10^n-1) -N.

Now, 10^n represents a number that consists of a single 1 followed by n 0's. 10^n-1 is a number represented by n 9's.

For example, if n=4 we have 10^4=10,000 and 10^4-1 = 9999.

It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9.

Some numerical examples follow:

The 9's complement of 546700 is 999999 − 546700 = 453299.

The 9's complement of 012398 is 999999 − 012398 = 987601.

For binary numbers, r=2 and r-1=1, so the 1's complement of N is (2^n-1)-N.

Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's.

2^n -1 is a binary number represented by n 1's.

For example, if n=4, we have $2^4 = (10000)_2$ and $2_4-1 = (1111)_{2.}$

Thus the 1's complement of binary number is obtained by subtracting each digit from 1.

However, when subtracting binary digits from 1, we can have either 1-0 =1 or 1-1 =0, which causes the bit to change from 0 to 1 or from 1 to 0.

Therefore, the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's.

The following are some numerical examples.

The 1's complement of 1011000 is 0100111.

The 1's complement of 0101101 is 1010010.

The (r-1)'s complement of octal or hexadecimal numbers is obtained by subtracting each digit from 7 or F (decimal 15), respectively.

Radix Complement

The r's complement of an n-digit number N in base r is defined as r^n-N for N ≠0 and 0 for N=0.

Comparing with the (r-1)'s complement, we note that the r's complement is obtained by adding 1 to the (r-1)'s complement since $r^n-N=[(r^n-1)-N] + 1$= 010100 and is obtained by adding 1 to the 1's complement value.

Since 10n is a number represented by a 1 followed by n 0's , 10n-N, which is 10's complement of N, can be formed also by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9.

The 10's complement of 012398 is 987602.

The 10's complement of 246700 is 753300.

Similarily, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged, and replacing 1's with 0's and 0's with 1's in all other higher significant digits.

The 2's complement of 1101100 is 0010100.

The 2's complement of 0110111 is 1001001.

In the previous definitions, it was assumed that the numbers do not have a radix point.

If the original number N contains a radix point, the point should be removed temporarily in order to form the r's or (r-1)'s complement.

The radix point is then restored to the complemented number in the same relative position.

It is also worth mentioning that the complement of the complement restores the number to its original value.

The r's complement of N is r^n-N.

The complement of the complement is $r^n-(r^n-N)=N$, giving back the original number.

Subtraction with Complements

The direct method of subtraction taught in elementary schools uses the borrow concept.

In this method, we borrow a 1 from a higher significant position when the minuend digit is smaller than the subtrahend digit.

This seems to be easiest when people perform subtraction with paper and pencil.

When subtraction is implemented with digital hardware, this method is found to be less efficient than the method that uses complements.

The subtraction of two n-digit unsigned numbers M-N in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. This performs $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry, r^n, which is discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

The following examples illustrate the procedure.

Example: Using 10's complement, subtract 72532-3250.

$$
\begin{aligned}
M &= 72532 \\
\text{10's complement of } N &= +\ \underline{96750} \\
\text{Sum} &= 169282 \\
\text{Discard end carry } 10^5 &= -\underline{100000} \\
\text{Answer} &= 69282
\end{aligned}
$$

Note that M has 5 digits and N has only 4 digits. Both numbers must have the same number of digits: so we can write N as 03250.

Taking the 10's complement of N produces a 9 in the most significant position.

The occurrence of the end carry signifies that M≥N and the result are positive.

Example 1-6:

Using 10's complement, subtract 3250 − 72532.

$$
\begin{aligned}
M &= 03250 \\
\text{10's complement of } N &= +\ \underline{27468} \\
\text{Sum} &= 30718
\end{aligned}
$$

There is no end carry.

Answer: −(10's complement of 30718) = −69282

Note that since 3250 < 72532, the result is negative. Since we are dealing with unsigned numbers, there is really no way to get an unsigned result for this case.

When subtracting with complements, the negative answer is recognized from the absence of the end carry and the complemented result.

When working with paper and pencil, we can change the answer to a signed negative number in order to put it in a familiar form.

Subtraction with complements is done with binary numbers in a similar manner using the same procedure outlined before.

Example 1-7: Given the two binary numbers X=1010100 and Y=1000011, perform the subtraction (a) X-Y and (b) Y-X using 2's complements.

(a)

$$\begin{aligned} X &= \quad 1010100 \\ \text{2's complement of } Y &= \quad +\underline{0111101} \\ \text{Sum} &= \quad 10010001 \\ \text{Discard end carry } 2^7 &= \quad -\underline{10000000} \\ \text{Answer: } X - Y &= \quad 0010001 \end{aligned}$$

(b)

$$\begin{aligned} Y &= \quad 1000011 \\ \text{2's complement of } X &= \quad +\underline{0101100} \\ \text{Sum} &= \quad 1101111 \end{aligned}$$

There is no end carry.

Answer: $Y - X = -(\text{2's complement of } 1101111) = -0010001$

Subtraction of unsigned numbers can be done also by means of the (r-1)'s complement.

Remember that the (r-1)'s complement is one less than the r's complement. Because of this, the result of adding the minuend to the complement of the subtrahend produces a sum that is 1 less than the correct difference when an end carry occurs.

Removing the end carry and adding 1 to the sum is referred to as an end-around carry.

Example 1-8:

Repeat Example 1-7 using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$$
\begin{array}{rr}
X = & 1010100 \\
\text{1's complement of } Y = & +\ \underline{0111100} \\
\text{Sum} = & 10010000 \\
\text{End-around carry} & \underline{+\ 1} \\
\text{Answer: } X - Y = & 0010001
\end{array}
$$

(b) $Y - X = 1000011 - 1010100$

$$
\begin{array}{rr}
Y = & 1000011 \\
\text{1's complement of } X = & +\ \underline{0101011} \\
\text{Sum} = & 1101110
\end{array}
$$

There is no end carry.

Answer: $Y - X = -(\text{1's complement of } 1101110) = -0010001$

Note that the negative result is obtained by taking 1's complement of the sum since this is the type of complement used.

The procedure with end-around carry is also applicable for subtracting unsigned decimals numbers with 9's complement.

1-6 Signed Binary Numbers

Positive integers including zero can be represented as unsigned numbers.

However, to represent negative integers, we need a notation for negative values.

In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign.

Because of hardware limitations, computers must represent everything with binary digits, commonly referred to as bits.

It is customary to represent the sign with a bit placed in the leftmost position of the number.

The convention is to make the sign bit 0 for positive a 1 for negative.

It is important to realize that both signed and unsigned binary numbers consist of a string of bits when represented in a computer.

The user determines whether the number is signed or unsigned.

If the binary number is signed, then the leftmost bit represents the sign and the rest of the bits represent the number.

If the binary number is assumed to be unsigned, the leftmost bit is the most significant bit of the number.

For example, the string of bits 01001 can be considered as 9 (unsigned binary) or a +9 (signed binary) because the leftmost bit is 0.

The string of bits 11001 represent the binary equivalent of 25 when considered as an unsigned number of as -9 when considered as a signed number because of the 1 in the leftmost position, which designates negative, and the other four bits, which represent binary 9.

Usually, there is no confusion in identifying the bits if the type of representation for the number is known in advance.

The representation of the signed numbers in the last example is referred to as the signed-magnitude convention.

In this notation, the number consists of a magnitude and a symbol (+ or -) or a bit (0 or 1) indicating the sign.

This is the representation of signed numbers used in ordinary arithmetic.

When arithmetic operations are implemented in a computer, it is more convenient to use a different system for representing negative numbers, referred to as the signed-complement system.

In this system, a negative number is indicated by its complement.

Whereas the signed-magnitude system negates a number by changing its sign, the signed-complement system negates a number by taking its complement.

Since positive numbers always start with 0 (plus) in the leftmost position, the complement will always start with a 1, indicating a negative number.

The signed-complement system can use either the 1's complement or the 2's complement, but the 2's complement is the most common.

As an example, consider the number 9 represented in binary with eight bits.

+9 is represented with a sign bit of 0 in the leftmost position followed by the binary equivalent of 9 to give 00001001.

Note that all eight bits must have a value and, therefore, 0's are inserted following the sign bit up to the first 1.

Although there is only one way to represent +9, there are three different ways to represent -9 with eight bits:

In signed-magnitude representation:	10001001
In signed-1's-complement representation:	11110110
In signed-2's-complement representation:	11110111

In signed magnitude, -9 is obtained from +9 by changing the sign bit in the leftmost position from 0 to 1.

In signed-1's complement, -9 is obtained by complementing all the bits of +9, including the sign bit.

The signed-2's complement representation of -9 is obtained by taking the 2's complement of the positive number, including the sign bit.

The signed magnitude system is used in ordinary arithmetic, but is awkward when employed in computer arithmetic.

Therefore, the signed-complement is normally used.

The 1's complement imposes some difficulties and is seldom used for arithmetic operations except in some older computers.

The 1's complement is useful as logical operation since the change of 1 to 0 or 0 to 1 is equivalent to a logical complement operation, as will be shown in the next chapter.

The following discussion of signed binary arithmetic deals exclusively with the signed-2's complement representation of negative numbers.

Arithmetic Addition

The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic.

If the signs are the same, we add the two magnitudes and give the sum the common sign.

If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.

For example, (+25) + (-37) = - (37 - 25) = -12 and is done by subtracting the smaller magnitude 25 from the larger magnitude 37 and using the sign of 37 for the sign of the result.

This is a process that requires the comparison of the signs and the magnitudes and then performing either addition or subtraction.

The same procedure applies to binary numbers in signed-magnitude representation.

In contrast, the rule for adding numbers in the signed-complement system does not require a comparison or subtraction, but only addition.

The procedure is very simple and can be stated as follows for binary numbers.

The addition of two signed binary numbers with negative numbers represented in signed-2's complement form is obtained from the addition of the two numbers, including their sign bits. A carry out of the sign-bit position is discarded.

Numerical examples for addition follow.

Note that negative numbers must be initially in 2's complement and that the sum obtained after the addition if negative is in 2's-complement form.

$$
\begin{array}{rr}
+\ 6 & 00000110 \\
+13 & 00001101 \\
\hline
+19 & 00010011
\end{array}
\qquad\qquad
\begin{array}{rr}
-\ 6 & 11111010 \\
+13 & 00001101 \\
\hline
+\ 7 & 00000111
\end{array}
$$

$$
\begin{array}{rr}
+\ 6 & 00000110 \\
-13 & 11110011 \\
\hline
-\ 7 & 11111001
\end{array}
\qquad\qquad
\begin{array}{rr}
-\ 6 & 11111010 \\
-13 & 11110011 \\
\hline
-19 & 11101101
\end{array}
$$

In each of the four cases, the operation performed is addition with the sign bit included.

Any carry out of the sign-bit position is discarded, and negative results are automatically in 2's-complement form.

In order to obtain a correct answer, we must ensure that the result has a sufficient number of bits to accommodate the sum.

If we start with two n-bit numbers and the sum occupies n+ 1 bit, we say that an overflow occurs.

When one performs the addition with paper and pencil, an overflow is not a problem since we are not limited by the width of the page.

We just add another 0 to a positive number and another 1 to a negative number in the most-significant position to extend them to n+1 bits and then perform the addition.

Overflow is a problem in computer because the number of bits that hold a number is finite, and a result that exceeds the finite value by 1 cannot be accommodated.

The complement form of representing negative number is unfamiliar to those used to the signed-magnitude system.

To determine the value of a negative number when in signed-2's complement, it is necessary to convert it to a positive number to place it in a more familiar form.

For example, the signed binary number 11111001 is negative because the leftmost bit is 1.

Its 2's complement is 00000111, which is the binary equivalent of +7.

We therefore recognize the original negative number to be equal to -7.

Arithmetic Subtraction

Subtraction of two signed binary number when negative numbers are in 2's complement form is very simple and can be stated as follows:

Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign-bit position is discarded.

This procedure occurs because a subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed.

This is demonstrated by the following relationship.

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

But changing a positive number to a negative number is easily done by taking its 2's complement.

The reverse is also true because the complement of a negative number in complement form produces the equivalent positive number.

Consider the subtraction of (-6)- (-13) = +7.

In binary with eight bits, this is written as (11111010 -111011).

The subtraction is changed to addition by taking the 2's complement of the subtrahend (-13) to give (+13).

In binary, this is 11111010 + 00001101 = 100000111.

Removing the end carry, we obtain the correct answer 00000111 (+7).

It is worth noting that binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers.

1-7 Binary Codes

Electronic digital systems use signals that have two distinct values and circuit elements that have two stable states.

There is a direct analogy among binary signals, binary circuit elements, and binary digits.

A binary number of n digits, for example, may be represented by n binary circuit elements, each having an output signal equivalent to a 0 or a 1.

Any discrete element of information distinct among a group of quantities can be represented by a binary code.

Binary codes play an important role in digital computers.

The codes must be in binary because computers can only hold 1's and 0's.

It must be realized that binary codes merely change the symbols, not the meaning of elements of information

Digital systems represent and manipulate not only binary numbers, but also many other discrete elements of information.

Any discrete element of information distinct among a group of quantities can be represented by a binary code.

Binary codes play an important role in digital computers.

The codes must be in binary because computers can only hold 1's and 0's.

It must be realized that binary codes merely change the symbols, not the meaning of elements of information that they represent.

If we inspect the bits of a computer at random, we will find that most of the time they represent some type of coded information rather than binary numbers.

A bit, by definition, is a binary digit.

When used in conjunction with a binary code, it is better to think of it as denoting a binary quantity equal to 0 or 1.

To represent a group of 2^n distinct elements in a binary code requires a minimum of n bits.

This is because it is possible to arrange n bits in 2^n distinct ways.

For example, a group of four distinct quantities can be represented by a two-bit code, with each quantity assigned one of the following bit combinations: 00, 01, 10, 11.

A group of eight elements requires a three-bit code, with each element assigned to one and only one of the following: 000, 001, 010, 011, 100, 101, 110, 111.

The examples show that the distinct bit combinations of an n-bit code can be found by counting in binary from 0 to (2^n-1).

Some bit combinations are unassigned when the number of elements of the group to be coded is not a multiple of the power of 2.

The ten decimal digits 0,1,2,.....,9 are an example of such a group.

A binary code that distinguishes among ten elements must contain at least four bits; three bits can distinguish a maximum of eight elements.

Four bits can form 16 distinct combinations, but since only ten digits are codes, the remaining six combinations are unassigned and not used.

Decimal Codes

Binary codes for decimal digits require a minimum of four bits.

Numerous different codes can be obtained by arranging four or more bits in ten distinct possible combinations. A few possibilities are shown in Table 1-2.

TABLE 1-2
Binary codes for the decimal digits

Decimal digit	(BCD) 8421	Excess-3	84-2-1	2421	(Biquinary) 5043210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

The BCD (binary-code decimal) is a straight assignment of the binary equivalent.

It is possible to assign weights to the binary bits according to their positions.

The weights in the BCD code are 8, 4,2,1.

The bit assignment of 0110, for example, can be interpreted by the weights to represent the decimal digits 6 because 0 x 8 + 1 x 4 + 1 x 2 + 0 x1 =6.

It is also possible to assign negative weights to a decimal code, as shown by the 8, 4, -2, -1 code.

In this case , the bit combination 0110 is interpreted as the decimal digit 2, as obtained from 0x8 + 1 x4 + 1x(-2) + 0 x (-1) =2.

Two other weighted codes shown in the table are the 2421 and the 5043210.

A decimal code that has been used in some old computers is the excess-3 code.

This is an unweighted code; its code assignment is obtained from the corresponding value of BCD after the addition of 3.

It is very important to understand the difference between conversion of a decimal number to binary and the binary coding of a decimal number.

In each case, the final result is a series of bits.

The bits obtained from conversion are binary digits.

Bits obtained from coding are combinations of 1's and 0's arranged according to the rules of the code used.

Therefore, it is extremely important to realize that a series of 1's and 0's in a digital system may sometimes represent a binary number and at other times represent some other discrete quantity of information as specified a given binary code.

The BCD code, for example, has been chosen to be both a code and a direct binary conversion, as long as the decimal numbers are integers from 0 to 9.

For numbers greater than 9, the conversion and the coding are completely different.

This concept is so important that it is worth demonstrating with another example.

The binary conversion of decimal 13 is 1101; the coding of the decimal 13 with BCD is 00010011.

From the five binary codes listed in Table 1-2, the BCD seems the most natural to use and is indeed the one most commonly encountered.

The other four-bit codes listed have one characteristic in common that is not found in BCD.

The excess-3, the 2,4,2,1, and the 8, 4, -2,-1 are self-complementing codes, that is, the 9's complement of the decimal number is easily obtained by changing 1's to 0's and 0's to 1's.

For example, the decimal 395 is represented by 2, 4, 2, 1 code by 001111111011.

Its 9's complement 604 is represented by 11000000100, which is easily obtained from the replacement of 1's by 0's and 0's by 1's.

This property is useful when arithmetic operations are internally done with decimal numbers (in a binary code) and subtraction is calculated by means of 9's complement.

The biquinary code shown in Table 1-2 is an example of seven-bit code with error-detection properties.

Each decimal digit consists of five 0's and two 1's placed in the corresponding weighted columns.

The error-detection property of this code may be understood if one realizes that digital systems represent binary 1 by one distinct signal and binary 0 by a second distinct signal.

During transmissions of signals from one location to another, an error may occur.

One of more bits may change value.

A circuit in the receiving side can detect the presence of more (or less) than two 1's and if the received combination of bits does not agree with the allowable combination, an error is detected.

Error-Detection Code

Binary information can be transmitted from one location to another by electric wires or other communication medium.

Any external noise introduced into the physical communication medium may change some of the bits from 0 to 1 or vice versa.

The purpose of an error-detection code is to detect such bit-reversal errors.

One of the most common ways to achieve error detection is by means of a parity bit.

A parity bit is an extra bit included with a message to make the total number of 1's transmitted either odd or even.

A message of four bits and a parity bit P are shown in Table 1-3.

TABLE 1-3
Parity bit

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

If an odd parity is adopted, the P bit is chosen such that the total number of 1's is odd in the five bits that constitute the message and P.

If an even parity is adopted, the P bit is chosen so that the total number of 1's in the five bits is even.

In a particular situation, one or the other parity is adopted, with even parity being more common.

The parity bit is helpful in detecting errors during the transmission of information from one location to another.

This is done in the following manner.

An even parity bit is generated in the sending end for each message transmission.

The message, together with the parity bit, is transmitted to its destination.

The parity of the received data is checked in the receiving end.

If the parity of the received information is not even, it means that at least one bit has changed value during the transmission.

This method detects one, three, or any odd combination of errors in each message that is transmitted.

An even combination of errors in undetected.

Additional error-detection schemes may be needed to take care of an even combination of errors.

What is done after an error is detected depends on the particular application.

One possibility is to request retransmission of the message on the assumption that the error was random and will not occur again.

Thus, if the receiver detects a parity error, it sends back a negative acknowledge message.

If no error is detected, the receiver sends back an acknowledge message.

The sending end will respond to a previous error by transmitting the message again until the correct parity is received.

If, after a number of attempts, the transmission is still in error, a message can be sent to the human operator to check for malfunctions in the transmission path.

Gray Code

Digital systems can be designed to process data in discrete forms only.

Many physical systems supply continuous output data.

These data must be converted into digital form before they are applied to a digital system.

Continuous or analog information is converted into digital form by means of an analog-to-digital converter.

It is sometimes convenient to use the Gray code shown in Table 1-4 to represent the digital data when it is converted from analog data.

TABLE 1-4
Four-bit Gray code

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

The advantage of the Gray code over binary numbers is that only one bit in the code group changes when going from one number to the next.

For example, in going from 7 to 8, the Gray code changes from 0100 to 1100.

Only the first bit from the left changes from 0 to 1; the other three bits remain the same.

When comparing this with binary numbers, the change from 7 to 8 will be from 0111 to 1000, which causes all four bits to change values.

The Gray code is used in applications where the normal sequence of binary numbers may produce an error or ambiguity during the transition from one number to the next.

If binary numbers are used, a change from 0111 to 1000 may produce an intermediate erroneous number 1001 if the rightmost bit takes more time to change than the other three bits.

The Gray code eliminates this problem since only one bit changes in value during any transition between two numbers.

ASCII Character Code

Many applications of digital computers require the handling of data not only of numbers, but also of letters.

For instance, an insurance company with thousands of policy holders will use a computer to process its files.

To represent the names and other pertinent information, it is necessary to formulate a binary code for the letters of the alphabet.

In addition, the same binary code must represent numerals and special characters such as $.

An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters o the alphabet, and a number of special characters.

Such a set contains between 36 and 64 elements if only capital letters are included, or between 64 and 128 elements if both uppercase and lowercase letters are included.

In the first case, we need a binary code of six bits, and in the second we need a binary code of seven bits.

The standard binary code for alphanumeric characters is ASCII (American Standard Code for Information Interchange).

It uses seven bits to code 128 characters, as shown in Table 1-5.

TABLE 1-5
American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	″	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	¦
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	–	o	DEL

Control characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

The seven bits of the code are designated by b_1 through b_7, with b_7 being the most-significant bit.

The letter A, for example, is represented in ASCII as 1000001 (column 100, row 0001).

The ASCII code contains 94 graphic characters that can be printed and 34 nonprinting characters used for various control functions.

The graphic characters consist of the 26 uppercase letters (A through Z), the 26 lowercase letters (a through z), the 10 numerals (0 through 9), and 32 special printable characters such as %, *, and $.

The 34 control characters are designated in the ASCII table with abbreviated names.

They are listed in the table with their full functional names.

The control characters are used for routing data and arranging the printed text into a prescribed format.

There are three types of control characters: format effectors, information separators, and communication-control characters.

Format effectors are characters that control the layout of printing.

They include the familiar typewriter controls such as backspace (BS), horizontal tabulation (HT), and carriage return (CR).

Information separators are used to separate data into divisions such as paragraphs and pages.

They include characters such as record separator (RS) and file separator (FS).

The communication-control characters are useful during the transmission of text between remote terminals.

Examples of communication-control characters are STX (start of text) and ETX (end of text), which are used to frame a text message when transmitted through telephone wires.

ASCII is a 7-bit code, but most computers manipulate an 8-bit quantity as a single unit called a byte.

Therefore, ASCII characters most often are stored one per byte.

The extra bit is sometimes used for other purposes, depending on the application.

For example, some printers recognize 8-bit ASCII characters with the most significant bit set to 0.

Additional 128 8-bit characters with most significant bit set to 1 are used for other symbols such as the Greek alphabet or italic type font.

When used in data communication, the eighth bit may be employed to indicate the parity of the character.

Other Alphanumeric Codes

Another alphanumeric code used in IBM equipment is the EBCDIC (Extended Binary Coded Decimal Interchange Code).

It uses eight bits for each character.

EBCDIC has the same character symbols as ASCII, but the bit assignment for characters is different.

As the name implies, the binary code for the letters and numerals is an extension of the binary-coded decimal (BCD) code.

This means that the last four bits of the code range from 0000 through 1001 as in BCD.

When characters are used internally in a computer for data processing (not for transmission purposes), it is sometimes convenient to use a 6-bit code to represent 64 characters.

A 6-bit code can specify 64 characters consisting of the 26 capital letters, the 10 numerals, and up to 28 special characters.

This set of characters is usually sufficient for data-processing purposes.

Using fewer bits to code characters has the advantages of reducing the space needed to store large quantities of alphanumeric data.

A code developed in the early stages of teletype transmission is the 5-bit Baudot code.

Although five bits can specify only 32 characters, the Baudot code represents 58 characters by using two modes of operation.

In the mode called letters, the five bits encode the 26 letters of the alphabet.

In the mode figures, the five bits encode the numerals and other characters.

There are two special characters that are recognized by both modes and used to shift from one mode to the other.

The letter-shift character places the reception station in the letters mode, after which all subsequent character codes are interpreted as letters.

The figure-shift character places the system in the figures mode.

The shift operation is analogous to the shifting operation on a typewriter with a shift lock key.