CHAPTER 2

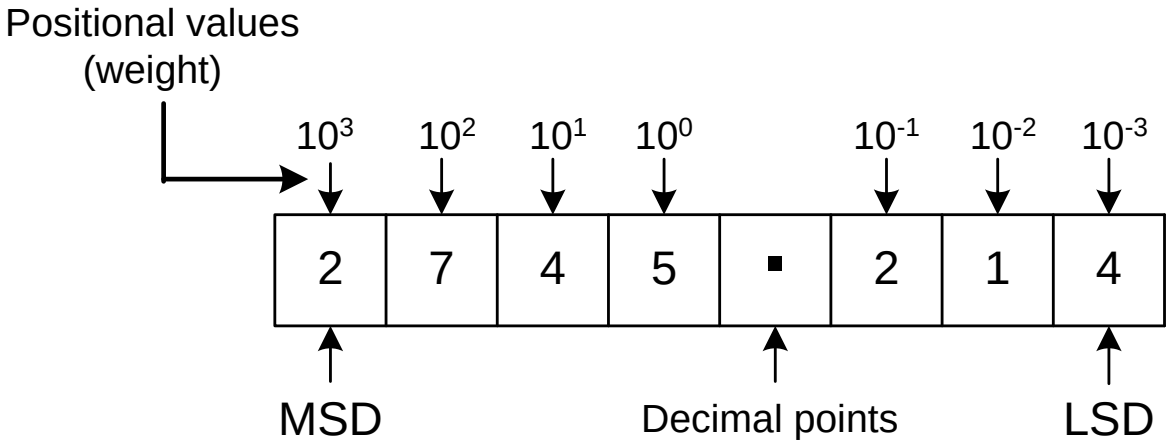
NUMBER SYSTEMS AND CODES

Introduction

- A number system is nothing more than a code that uses symbols to refer to a number of items.
- The binary number system and digital codes are fundamental to computers and to digital electronics in general.
- In this chapter, the binary number system and its relationship to other number systems, such as decimal , hexadecimal, and octal is the principal focus.
- You will learn to make conversion between binary and octal and decimal and octal.
- Also, digital codes such as binary coded decimal (BCD), the Gray code, the Excess-3, and the ASCII are covered.
- The arithmetic operations with binary numbers are also included.

Decimal Systems

- **The decimal number system uses the symbols 0,1,2,3,4,5,6,7,8 and 9.**
- **The decimal number system contains 10 symbols and is sometimes called the base 10 system.**
- **The decimal system is a positional value system in which the value of a digit depends on its position.**
- **In general ‘’ Any number is simply the sum of the products of each digit value and its positional value ’’**



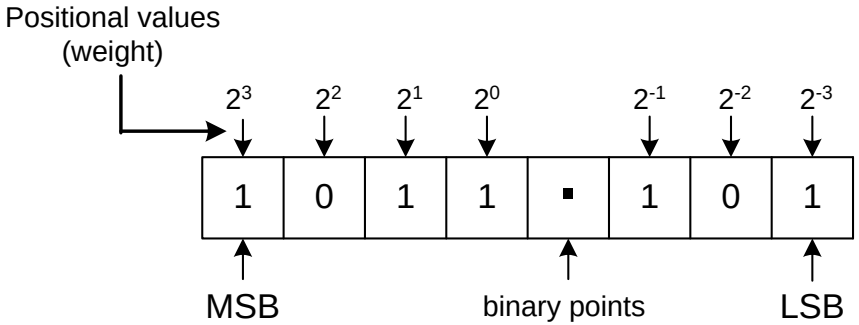
$$(2*10^{3})+(7*10^{2})+(4*10^{1})+(5*10^{0})+(2*10^{-1})+(1*10^{-2})+(4*10^{-3})$$

Decimal Counting

- When counting in decimal system, we start with 0 in the unit's position and take each symbol (digit) in progression until we reach 9.
- Then we add a 1 to the next higher position and start over with zero in the first position. This process continues until the count of 99 is reached.
- Then we add a 1 to third position and start over with zeros in first two positions.
- The same pattern is followed continuously as high as we wish to count.
- It is important to note that in decimal counting
  - ✓ the units position (LSD) changes up ward with each step in the count,
  - ✓ the tens position changes up ward every 10 steps in the count,
  - ✓ the hundreds position changes upward every 100 steps in the count and so on.

Binary System

- A binary system is a code that uses only two basis symbols, 0 &1 and is sometimes called the base 2 system.
- This base 2 system can be used to represent any quantity that can be represented in decimal or other number systems.
- All the statements made earlier concerning the decimal system are equally applicable to the binary system.
- The binary system is also a positional value system, where in each binary digit has its own value or weight expressed as a power of 2.



- The subscripts 2 & 10 were used to indicate the base in which the particular number is expressed.
- This convention is used to avoid confusion whenever more than one number system is being employed.
- In binary system, the term binary digit is often abbreviated to the term bit, which we will use henceforth.
- The most significant bit (MSB) is the left most bit (largest weight). The least significant bit (LSB) is the right most bit (Smallest weight). These are indicated in previous example.

Binary Counting

- Let us use 4 bit binary numbers to illustrate the method for counting in binary. The sequence (shown on the right side) begins with all bits at 0, this is called the Zero count.
- For each successive count the units (2^0) position toggles; that is, it changes from one binary value to the other.
- Each time the units bits changes from a 1 to 0, the twos (2^1) position will toggle (change states).
- Each time the twos position changes from 1 to 0, the fours (2^2) position will toggle (change states).
- Like wise, each time the fours position goes from 1to 0, the eights (2^3) position toggles.

Weights	2^3=8	2^2=4	2^1=2	2^0=1	Decimal Equivalent
	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

Hexadecimal Number Systems

- The hexadecimal system uses base 16.
- Thus, it has 16 possible digit symbols.
- It uses the digits
  - 0 through 9 plus
  - the letters A, B, C, D, E and F as the 16 digit symbols.
- In table below the relationships between hexadecimal, decimal and binary.
- Note that each hexadecimal digit represents a group of four binary digits.
- It is important to remember that hex (abbreviation for hexadecimal) digits A through F are equivalent to the decimal values 10 though 15.

Counting in Hexadecimal

- When counting in hex, each digit position can be incremented (increase by 1) from 0 to F.
- Once a digit position reaches the value F, it is reset to 0 and the next digit position is incremented.
- This is illustrated in the following hex counting sequences.

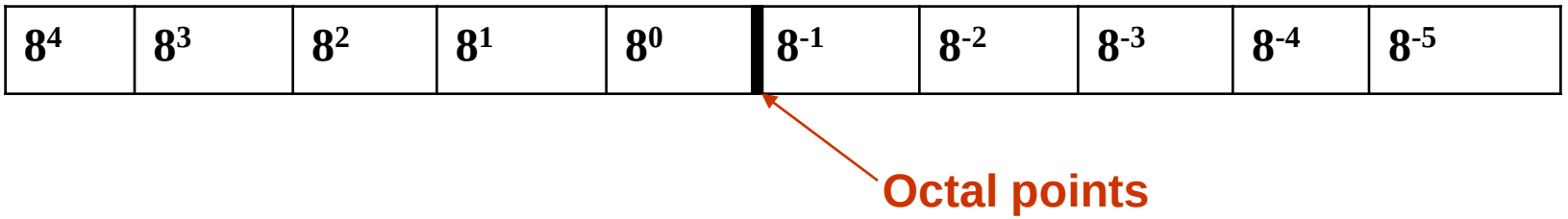
28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31, …

6F9, 6FA, 6FB, 6FC, 6FD, 6FE, 6FF, 700 …

Decimal	Binary	Hexadecimal
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
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11

Octal Number System

- The octal number system is very important in digital Computer work.
- The octal number system has a base of eight, meaning that it has eight possible digits: 0,1,2,3,4,5,6 and 7.
- Thus, each digit of an octal number can have any value from 0 to 7.
- The advantage of the octal system is its usefulness in converting directly from a 3 bit binary number.
- The digit positions in an octal number have weights as follows.



- The equivalent binary and octal representations for decimal numbers 0 through 17 is shown below

Counting in Octal

- The largest octal digit is 7, so that in counting in octal, a digit position is incremented up ward form 0 to 7.
- Once it reaches 7, it recycles to 0 on the next count and causes the next higher digit position to be incremented.
- *For example:*

65, 66, 67, 70, 71, 72, …

275, 276, 277, 300, 301, …

Decimal	Binary	Octal
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7
8	001 000	10
9	001 001	11
10	001 010	12
11	001 011	13
12	001 100	14
13	001 101	15
14	001 110	16
15	001 111	17
16	010 000	20
17	010 001	21

Conversion of one Number System to Another

Binary-to-decimal Conversions:

- Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number, which contain a 1.
- For Example

a) $1\ 1\ 0\ 0\ 1\ 1_2$

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$32 + 16 + 0 + 0 + 2 + 1 = 51_{10}$

b) $1\ 0\ 1\ 1\ 0\ 0\ 1\ 0_2$

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$128 + 0 + 32 + 16 + 0 + 0 + 2 + 0 = 178_{10}$

- Note that the procedure is to find the weights (i.e. powers of 2) for each bit position that contains a 1, and then to add them up.
- Also note that the MSB has a weight of 2^7 even though it is the eighth bit this is because the LSB is the first bit and has a weight of 2^0.

Decimal – to – Binary conversions

- There are two ways to convert a decimal whole number to its equivalent binary system representation.
- **The 1st method (Sum-of-Weights)**

- The decimal number is simply expressed as a sum of power of 2 and then 1s and 0s are written in the appropriate bit positions.

Examples

a) $57_{10} = 32 + 16 + 8 + 1 = 2^5 + 2^4 + 2^3 + 0 + 0 + 2^0 = 111001_2$

b) $132_{10} = 128 + 4 = 2^7 + 0 + 0 + 0 + 0 + 2^2 + 0 + 0 = 10000100_2$

Exercises

a) 398_{10} =????

b) 4153_{10}= ????

The 2nd method (''Repeated Division'')

- This method uses repeated division by 2.
- Requires repeatedly dividing the decimal number by 2 and writing down the remainder after division until the quotient of 0 is obtained.
- Note that the binary result is obtained by writing the first remainder as the LSB and the last remainder as the MSB.
- **Examples**

a) $37_{10}=?_2$

$37 \div 2 = 18$ with remainder of 1 1s → LSB

$18 \div 2 = 9$ with remainder of 0 2s

$9 \div 2 = 4$ with remainder of 1 4s

$4 \div 2 = 2$ with remainder of 0 8s

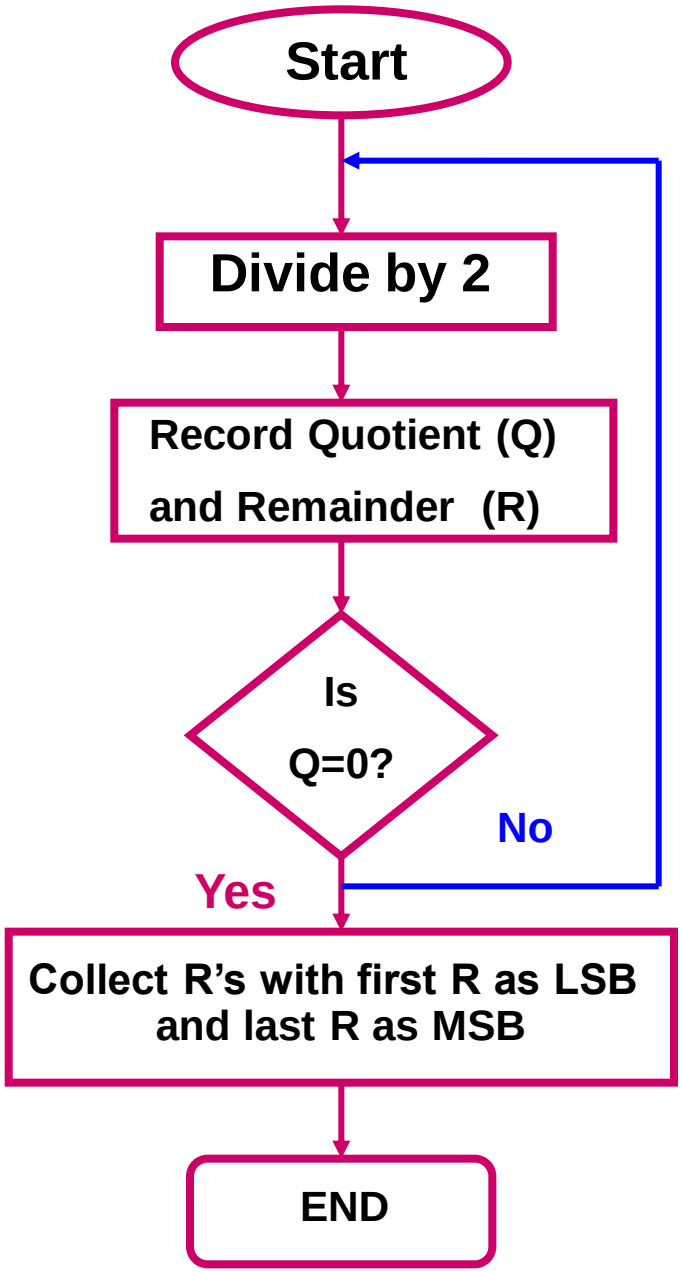
$2 \div 2 = 1$ with remainder of 0 16s

$1 \div 2 = 0$ with remainder of 1 32s → MSB

$\therefore$ $37_{10} = 100101_2$

b) $398_{10} = ?_2$

Flow chart for Repeated Division



Hex –to- decimal Conversion

- A hex number can be converted to its decimal equivalent by using the fact tat each hex digit position has a weight that is a power of 16.
- The LSD has a weight of $16^0 = 1$,the next higher digit position has a weight of $16^1 = 16$, the next has a weight of $16^2 = 256$, and so on.

Examples

a) 356_{16}

$= 3*16^2 + 5*16^1 + 6*16^0 = 768 + 80 + 6 = 854_{10}$

b) $A3F_{16}$

$= 10*16^2 + 3*16^1 + 15*16^0 = 2560 + 48 + 15 = 2623_{10}$

Exercises

- $1BD5_{16} = ???_{10}$
- $C9A_{16} = ???_{10}$

Decimal-to-Hex conversion

- Recall that we did decimal –to- binary conversion using repeated division by 2. Like wise decimal-to-hex conversion can be done using repeated division by 16.

Examples

a) Convert 47_{10} to hex

47 ÷ 16 = 2 remainder of 15 → F

2 ÷ 16 = 0 remainder of 2

∴ $47_{10} = 2F_{16}$

b) Convert 234_{10} to hex

234 ÷ 16 = 14 remainder of 10 → A

14 ÷ 16 = 0 remainder of 14 → E

∴ $234_{10} = EA_{16}$

Exercise

a) Convert 100_{10} to hex

b) Convert 445_{10} to hex

Hex-to-Binary conversion

- The Hexadecimal number system is used primarily as a ‘’Shorthand’’ method for representing binary numbers.
- It is a relatively simple method to convert a hex number to binary.
- Each hex digit is converted to its 4 bit binary equivalent. (see table ‘’Hexadecimal Number system’’).

Examples

a) $C3_{16}$ = C 3

1100 0011 = 11000011_2

b) $7E8_{16}$ = 7 E 8

0111 1110 1000 = 011111101000_2

Exercise

- Convert $A56B_{16}$ to binary

Binary-to- Hex conversion

- Conversion from binary to hex is just the reverse of the above process.
- The binary number is grouped in to groups of *four* bits, and each group is converted to its equivalent hex digit.
- Zeros are added, as needed, to complete a 4-bit group.

Examples

a) Convert 101101001110_2 to Hex

$1011\ 0100\ 1110_2 = 1011\ \ 0100\ \ 1110$

$\Downarrow \quad \Downarrow \quad \Downarrow$

$B \quad 4 \quad E_{16}$

b) Convert 10100101011_2 to Hex

$0101\ 0010\ 1011_2 = 0101\ \ 0010\ \ 1011$

$\Downarrow \quad \Downarrow \quad \Downarrow$

$5 \quad 2 \quad B_{16}$

- This is why hex (and octal) are so useful in representing large binary numbers.

Exercise

Convert 1001110101010101_2 to Hex

Octal to decimal Conversion

- An octal number, can easily be converted to its decimal equivalent by multiplying each octal digit by it positional weight i.e. a power of 8.

Examples

a) $415_8 = 4*8^2 + 1*8^1 + 5*8^0 = 256 + 8 + 5 = 269_{10}$

b) $730_8 = 7*8^2 + 3*8^1 + 0*8^0 = 448 + 24 + 0 = 472_{10}$

Decimal –to – Octal Conversion

- Decimal integer can be converted to octal by using repeated division with a division factor of 8.

Examples

- Convert 35_{10} to octal

35 ÷ 8 = 4 remainder of 3 → LSD

4 ÷ 8 = 0 remainder of 4 → MSD

$\therefore$ $35_{10} = 43_8$

- Note that the first remainder becomes the least significant digit (LSD) of the octal number and the last remainder becomes the most significant digit (MSD)

Octal to Binary Conversion

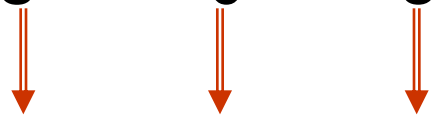
- The conversion from octal to binary is presented by converting each octal digit to its 3- bit binary equivalent.

Examples

a) 47_8 = 4 7

100 111 = 100111_2

b) 305_8 = 3 0 5



011 000 101 = 011000101_2

Binary to Octal Conversion

- Converting from binary to octal integers is simply the reverse of the foregoing process.
- The bits of the binary number are grouped into groups of *three* bits starting at the LSB.
- The each group is converted to its octal equivalent.

Examples

a) Convert 101111001_2 to Octal

$101\ 111\ 001_2$ = 101 111 001

5 7 1_8

b) Convert 10011110_2 to Octal

$010\ 011\ 110_2$ = 010 011 110

2 3 6_8

Exercise:

Convert $B\ 2\ F_{16}$ to Octal

Fractions

- As far as fractions are concerned, you multiply by 2 and record a carry in the integer position.
- The carries taken in forward order are the binary fraction.

Examples

a) Convert 0.625_{10} to a binary fraction

$0.625 * 2 = 1.25 \rightarrow 0.25$ with carry of 1

$0.25 * 2 = 0.5$ with carry of 0

$0.5 * 2 = 1.0$ with carry of 1

$\therefore \quad 0.625_{10} = 0.101_2$

b) Convert 0.23_{10} into an octal fraction

$0.23 * 8 = 1.84 \rightarrow 0.84$ with carry of 1

$0.84 * 8 = 6.72 \rightarrow 0.72$ with carry of 6

$0.72 * 8 = 5.76 \rightarrow 0.76$ with carry of 5

:

:

$\therefore \quad 0.23_{10} = 0.165_8$

Exercises

$67.82_{10} = \text{---------}_{2}$

$= \text{---------}_{8}$

$= \text{---------}_{16}$

Summary of Conversions

The following summery should help you in doing the different conversion.

- ✓ **When converting from binary [or octal, or hex] to decimal, use the method of taking the weight sum of each digit position.**
- ✓ **When converting from decimal to binary [or octal or hex] use the method of repeatedly dividing by 2 [or 8 or 16] and collecting remainders [refer fig. flow chart]**
- ✓ **When converting from binary to octal [or hex], group the bits in groups of the three [or four], and convert each group into the correct octal [or hex] digit.**
- ✓ **When converting from octal [or hex] into binary, convert each digit in to its 3-bit [or 4-bit] equivalent.**
- ✓ **When converting from octal to hex [or vice versa, first convert to binary, then convert the binary into the desired number system.**

SIGNED NUMBERS

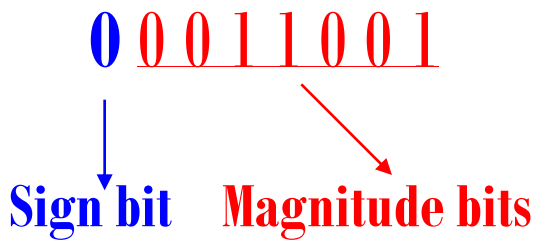
- Digital systems, such as the computer, must be able to handle both positive and negative numbers.
- A signed binary number consists of both sign and magnitude information.
- The sign indicates whether a number is positive or negative and
- the magnitude is the value of the number.
- There are three forms in which signed integer (whole) numbers can be represented in binary:
  - **Sign-magnitude,**
  - **1's complement, and**
  - **2's complement.**
- Of these, the 2's complement is the most important and the sign-magnitude is rarely used.

The Sign Bit

- The left-most bit in a signed binary number is the **sign bit**, which tells you whether the number is positive or negative.
  - ***a 0 is for positive, and***
  - ***a 1 is for negative***

Sign-Magnitude Form

- When a signed binary number is represented in sign-magnitude, the left-most bit is the sign bit and the remaining bits are the magnitude bits.
- The magnitude bits are in true (uncomplemented) binary for both positive and negative numbers.
- For example, the decimal number **+25** is expressed as an 8-bit signed binary number using the sign-magnitude form as



- The decimal number **−25** is expressed as **10011001**
- Notice that the only difference between +25 and -25 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers
- **In the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number but the sign bit is a 1 rather than a zero.**

1's and 2's Complements of Binary Numbers

- The 1's complement and the 2's complement of a binary number are important because they permit the representation of negative numbers.
- The method of 2's complement , arithmetic is commonly used in computers to handle negative numbers.

Finding the 1's Complement of a Binary Number

- The 1's complement of a binary number is found by changing all 1s to 0s and all 0s to 1s, illustrated below:

1 0 1 1 0 0 1 0	Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 1 0 1	1's complement

Finding the 2's Complement of a Binary Number

- The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

2's complement = (1's complement) + 1

Example

Find the 2's complement of 10110010.

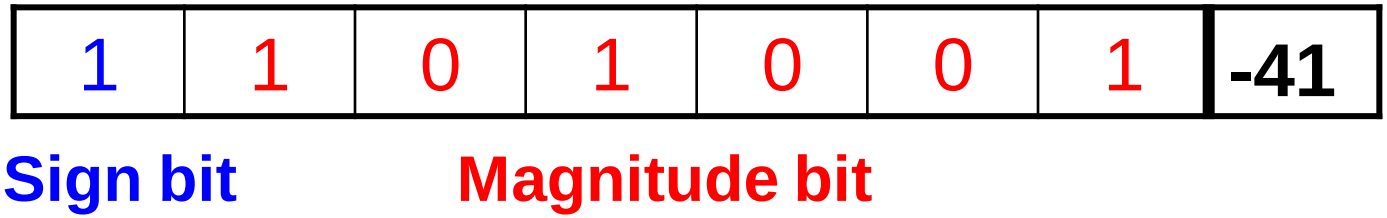
Solution

10110010	**Binary number**
01001101	**1's complement**
+ 1	**Add 1**
01001110	**2's complement**

REPRESENTATION OF SIGNED NUMBERS AND BINARY ARITHMETIC IN COMPUTERS

- So far, we have considered only positive numbers. The representation of negative is also equally important. There are two ways of representing signed numbers:
  - **sign magnitude form and**
  - **complement form.**
- There are two complement forms:
  - **1's complement form and**
  - **2's complement form.**
- Most digital computers do subtraction by the 2's complement method, but some do it by the 1's complement method.
- **The advantage of performing subtraction by the complement method is reduction in the hardware**.
- Instead of having separate digital circuits for addition and subtraction, only adding circuits are needed.
- That is, subtraction is also performed by adders only. Instead of subtracting one number from the other, the complement of the subtrahend is added to the minuend.
- In sign-magnitude form, an additional bit called the sign bit is placed in front of the number.
- If the sign bit is a 0, the number is positive. If it is a 1, the number is negative.

- Example of Signed-magnitude form



- Under the signed-magnitude system, a great deal of manipulation is necessary to add a positive number to a negative number.
- Thus, though the signed-magnitude number system is possible, it is impractical.

Representation of Signed Numbers Using the 2's (or 1's) Complement Method

The 2's (or 1's) complement system for representing signed numbers works like this:

1. If the number is positive, the magnitude is represented in its true binary form and a sign bit 0 is placed in front of the MSB.
2. If the number is negative, the magnitude is represented in its complement form and a sign bit 1 is placed in front of the MSB.

- That is, to represent the numbers in sign 2's (or 1's) complement form, determine the 2's (or 1's) complement of the magnitude of the number and then attach the sign bit.
- The conversion of complement to true binary is the same as the process used to convert true binary to complement.

❖ Example on representation of + 51 and - 51 in both 2's and 1's complement forms is shown below:

0	1	1	0	0	1	1	+51 (in signed magnitude form) (in 2's complement form) (In 1's complement form)
Sign bit			Magnitude				

1	1	1	0	0	1	1	-51 (in signed magnitude form)
Sign bit			Magnitude				

1	0	0	1	1	0	0	-51 (In 1's complement form)
Sign bit			Magnitude				

1	0	0	1	1	0	1	-51 (in 2's complement form)
Sign bit			Magnitude				

- To subtract using the 2's (or 1's) complement method; represent both the subtrahend and the minuend by the same number of bits.
- Take the 2's (or 1's) complement of the subtrahend including the sign bit.
- Keep the minuend in its original form and add the 2's (or 1's) complement of the subtrahend to it.
- The choice of 0 for positive sign and 1 for negative sign is not arbitrary.
- In fact, this choice makes it possible to add the sign bits in binary addition just as other bits are added.
- When the sign bit is a 0, the remaining bits represent magnitude, and
- when the sign bit is a 1, the remaining bits represent 2's or 1's complement of the number.
- The polarity of the signed number can be changed simply by performing the complement on the complete number.

Special case in 2's complement representation:

- Whenever a signed number has a 1 in sign bit and all 0s for the magnitude bits, the decimal equivalent is -2^n, where n number of bits in the magnitude. For example, 1000 = - 8 and 10000 = - 16.

Characteristics of the 2's complement numbers.

- The 2's complement numbers have the following properties:
- There is one unique zero.
- The 2's complement of 0 is 0.
- The left most bit cannot be used to express a quantity. It is a sign bit. If it is a 1, the number is negative and if it is a 0, the number is positive.
- For an n-bit word which includes the sign bit, there are $2^{n-1} - 1$ positive integers, 2^{n-1} negative integers and one 0, for a total of 2^n unique states.
- Significant information is contained in the 1s of the positive numbers and 0s of the negative numbers.
- A negative number may be converted into a positive number by finding its 2's complement.

Methods of obtaining the 2's complement of a number:

- The 2's complement of a number can be obtained in three ways as given below.
- By obtaining the 1's complement of the given number (by changing all 0s to 1s and 1s to 0s) and then adding 1.
- By subtracting the given n-bit number N from 2^n.
- Starting at the LSB, copying down each bit up to and including the first 1 bit encountered, and complementing the remaining bits.

Example: *Express -45 in 8-bit 2's complement form.*

Solution

+45 in 8-bit form is 00101101.

First method

Obtain the 1's complement of 00101101 and then add 1.

Positive expression of the given number	00101101
1's complement of it	11010010
Add 1	+1
Thus, the 2's complement form of -45 is	**11010011**

Second method

Subtract the given number N from 2^n

2^n =	100000000
Subtract 45 =	– 00101101
Thus, the 2's complement form of -45 is	**11010011**

Third method

Copy down the bits starting from LSB up to and including the first 1, and then complement the remaining bits.

Original number	00101101
Copy up to first 1 bit	1
Complement the remaining bits	1101001
Thus, the 2's complement form of -45 is	**11010011**

Two's Complement Arithmetic

- The 2's complement system is used to represent negative numbers using modulus arithmetic.
- The word length of a computer is fixed.
- That means if a 4-bit number is added to another 4-bit number, the result will be only of 4 bits. Carry, if any, from the fourth bit will overflow.
- This is called the ***modulus arithmetic***.

  For example: 1100 + 1111 = 1011.
- In the 2's complement subtraction, add the 2's complement of the subtrahend to the minuend.
- If there is a carry out, ignore it. Look at the sign bit, i.e. MSB of the sum term.
- If the MSB is a 0, the result is positive and is in true binary form.
- If the MSB is a 1 (whether there is a carry or no carry at all) the result is negative and is in its 2's complement form.
- Take its 2's complement to find its magnitude in binary.

***Example*:** subtract 14 from 46 using the 8-bit 2's complement arithmetic.

Solution

+14	=	00001110	
–14	=	11110010	(in 2's complement form)
+46		00101110	
–14	⇒	+11110010	(2's complement form of–14)
+32		1**00100000**	(Ignore the carry)

- There is a carry, ignore it.
- The MSB is 0; so, the result is positive and is in normal binary form.
- Therefore, the result is +00100000=+32

Example: add –75 to +26 using the 8-bit 2's complement arithmetic

Solution

+75	=	01001011	
–75	=	10110101	(in 2's complement form)
+26		00011010	
–75	⇒	+10110101	(2's complement form of -75)
–49		11001111	(No carry)

- There is no carry, the MSB is a 1.
- So, the result is negative and is in 2's complement form.
- The magnitude is 2's complement of 11001111, that is, 00110001=49.
- Therefore, the result is –49.

One's Complement Arithmetic

- The 1's complement of a number is obtained by simply complementing each bit of the number, that is, by changing all the 0s to 1 s and all the 1 s to 0s.
- We can also say that the 1's complement of a number is obtained by subtracting each bit of the number from l.
- This complemented value represents the negative of the original number.
- This system is very easy to implement in hardware by simply feeding all bits through inverters.
- One of the difficulties of using 1's complement is its representation of zero.
- Both 00000000 and its 1's complement 11111111 represent zero.
- The 00000000 is called positive zero and the 11111111 is called negative zero.
- In 1's complement subtraction, add the 1's complement of the subtrahend to the minuend.
- If there is a carry out, bring the carry around and add it to the LSB. *This is called the end around carry.*
- Look at the sign bit (MSB); if this is a 0, the result is positive and is in true binary.
- If the MSB is a 1 (whether there is a carry or no carry at all), the result is negative and is in its 1's complement form. Take its 1's complement to get the magnitude in binary.

***Example*:** subtract 14 from 25 using the 8-bit 1's complement arithmetic

Solution

+25		00011001	
–14	⇒	+11110001	(1's complement form of –14)
+11		100001010	
		+1	(Add the end around carry)
		00001011	$=+ 11_{10}$

Example: Add –25 to 14 using the 8-bit 1's complement method.

Solution

+14 00001110

–25 $\Rightarrow$ +11100110 (1's complement form of –25)

–11 11110100 (No carry)

- There is no carry, the MSB is a 1.
- So, the result is negative and is in its 1's complement form.
- The 1's complement of 11110100 is 00001011.
- The result is, therefore, -11_{10}.

- The two numbers in addition are the **addend** and the **augend**. The result is the **sum**. The 2's compliment will be used to represent negative numbers.
- There are four cases that must be considered when adding two numbers:

1. Both numbers positive
2. Positive number and smaller negative number
3. Positive number and larger negative number
4. Both numbers negative

We will take one case at a time. Eight bits are used to represent each number

1. Both numbers positive:

- In this case, both sign bits are zero and a 2's compliment is not required. To illustrate, we will add +7 and +4:

7	00000111
+ 4	00000100
11	00001011

2. Positive number and smaller negative number:

- In this case, the true binary form of the positive number is added to the 2's complement of the negative number.
- The sign bits are included in the addition, and the result will be positive. To illustrate we will add + 15 and -6:

15	00001111
+ -6	11111010
9	100001001

Discard carry

Notice that the sign of the sum is positive (0) as it should be.

3. Positive number and larger negative number:

- Again, the true binary form of positive number is added to the 2's complement of the negative number.
- The sign bits are included in the addition, and the result will be negative. To illustrate will add + 16 and - 24:

16	00010000	
+ -24	11101000	(2's complement of -24)
-8	11111000	(2's complement of -8)

- Notice that the result automatically comes out in 2's complement because it is a negative number.

4. Both numbers negative:

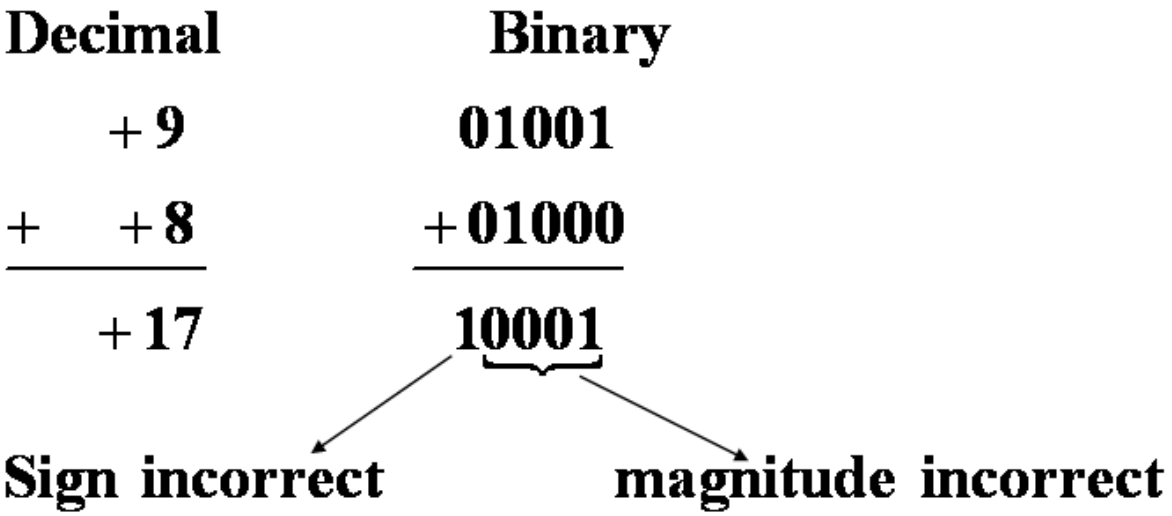
- In this case, the 2's complements of both numbers added and, of course, the sum is a negative number in 2's complement form illustrate, we will add - 5 and -9:

-5	11111011	(2's complement of -5)
+ -9	11110111	(2's complement of -9)
-14	111110010	(2's complement of -14)

Discard carry

Overflow

- When the number of bits in the sum exceeds the number of bits in each of numbers added, *overflow results, as* illustrated by the following example.

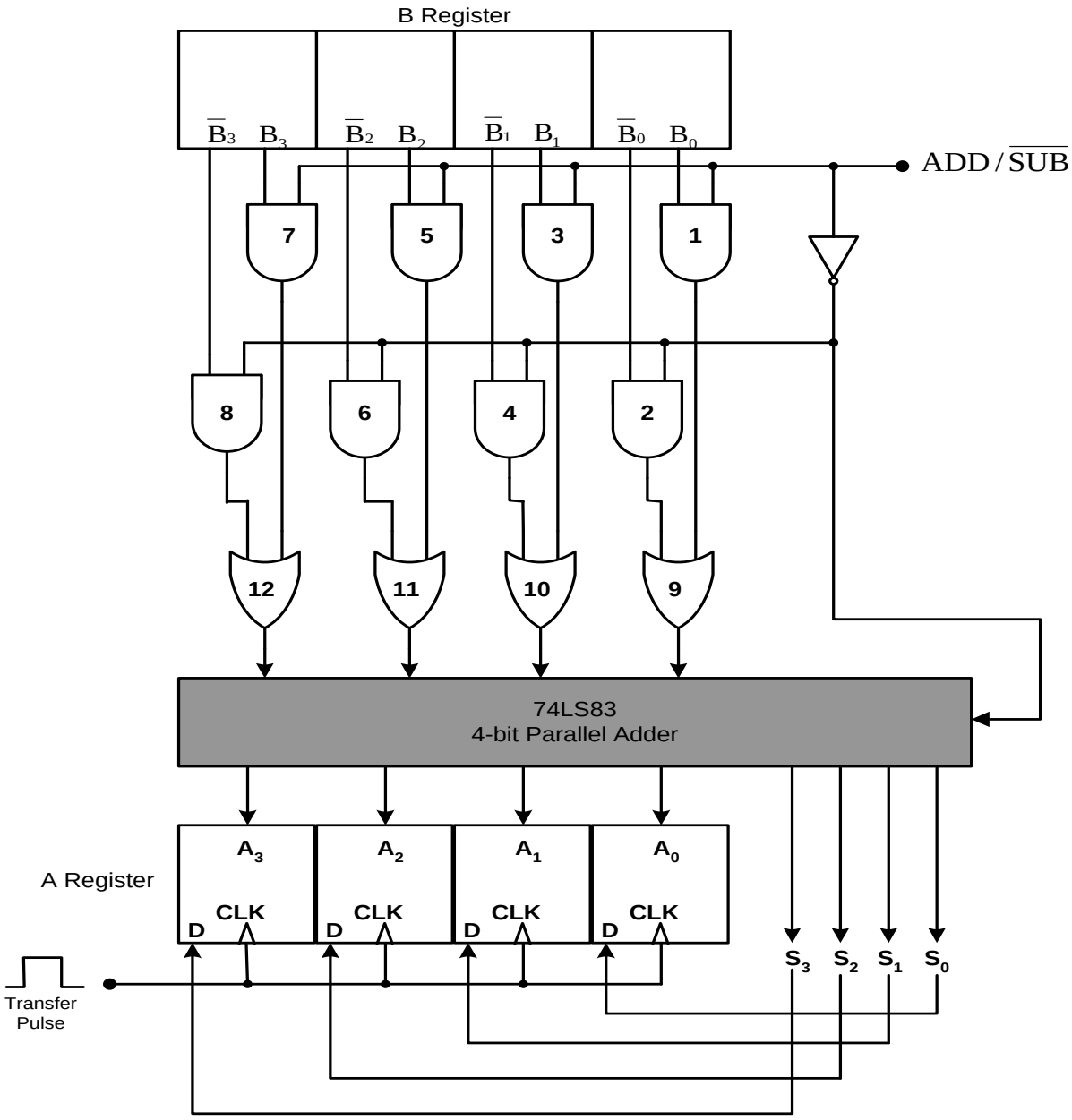


- The overflow condition can occur only when both numbers are positive or both numbers are negative. An overflow result is *indicated by an incorrect sign bit.*

Binary Subtraction

- Subtraction is a special case of addition.
- For example, subtracting +6 (subtrahend) from +9 (minuend) is equivalent to *adding* -6 to +9.
- Basically the subtraction operation *changes the sign* of *the subtrahend and adds it to the minuend.*
- The 2's complement method can be used in subtraction that all operations require only addition.
- The four cases that were discussed relation to the addition of signed numbers apply to the subtraction process because subtraction can be essentially reduced to an addition process.

Parallel adder/Subtractor using 2's Complement system



CODES

Introduction

- **When numbers, letters, or words are represented by Special group of symbols, we say that they are being encoded, and the group of symbols is called a code.**
- **The group of 0s and 1s in the binary number can be thought of as a code representing the decimal number.**
- **When a decimal number is represented by its equivalent binary number, we call it *straight binary coding*.**
- **We have seen that the conversing between decimal and binary can be come long and complicated for large numbers.**
- **For this reason, a means of encoding decimal numbers that combines some features of both the decimal and binary system is used in certain situations.**

Binary-Coded- Decimal Code

- If each digit of a decimal number is represented by its 4- bit binary equivalent the result is a code called binary-coded-decimal (hereafter abbreviated BCD).
- Since a decimal digit can be as large as 9, four bits are required to code each digit (the binary code for 9 is 1001).

Examples

a) The decimal number 437 is changed to its BCD equivalent as follows:

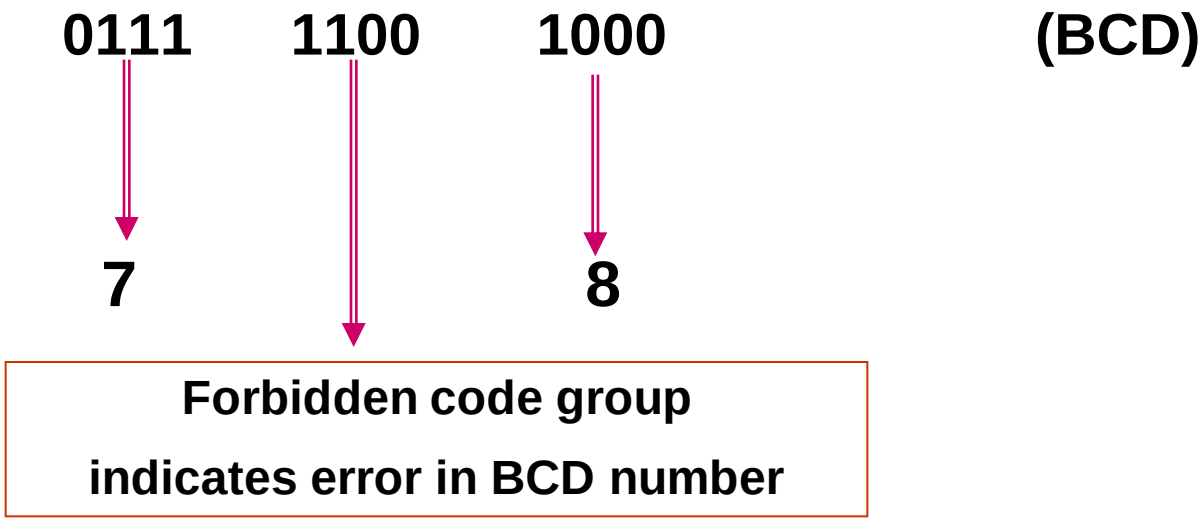
4	3	7	(decimal)
↓	↓	↓	
0100	0011	0111	(BCD)

b)

9	5	8	0	(decimal)
↓	↓	↓	↓	
1001	0101	1000	0000	(BCD)

- **Clearly, only the 4- bit binary numbers from 0000 through 1001 are used.**
- **The BCD Code does not use the numbers 1010, 1011, 1100, 1101, 1110 and 1111.**
- **In other words only 10 of the 16 possible 4 bit binary code groups are used.**
- **If any of the ‘’forbidden’’ 4 bit numbers ever occurs in machine using the BCD code, it is usually an indication that an error has occurred.**

➢ *Convert the BCD number 011111001000 to its decimal equivalent*



Comparison of BCD and Binary

- It is important to realize that BCD is not another number system like binary, octal, decimal, and hexadecimal.
- It is, in fact, the decimal system with each digit encoded in its binary equivalent.
- It is also important to understand that a BCD number is not the same as a straight binary number.
- A straight binary code takes the complete decimal number and represents it in binary;
- But the BCD code converts each decimal digit to binary individually.
- To illustrate, take the number 253 and compare its straight binary and BCD codes:

$253_{10} = 11111101_2$ (Straight binary)

$253_{10} = 0010\ 0101\ 0011$ (BCD)

- The BCD code requires 12 bits while the straight binary code requires only 8 bits to represent 253.
- This is because BCD does not use all possible 4-bit groups, as pointed out earlier, and is therefore somewhat inefficient.
- The main advantage of the BCD code is the relative ease of converting to and from decimal only the 4 bit code groups for the decimal digits 0 through 9 need to be remembered.

Gray Code

- **The Gray code belongs to a class of codes called minimum change codes, in which *only one bit in the code group changes* when going from one step to the next.**
- **The gray code is an *unweighted* code, meaning that the bit positions in the code groups do not have any specific weight assigned to them.**
- **Because of this, the gray code is not suitable for arithmetic operations but finds application in input/out put devices and some types of analog-to-digital converters.**
- **Table below shows the gray code representation for the decimal number 0 through 15,together with straight binary code.**

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

- If we examine the Gray code groups for each decimal number, it can be seen that in going from any one decimal number to the next, only one bit of Gray code changes.
- For example:

Decimal	Gray code
3 to 4	0010 to 0110
14 to 15	1001 to 1000

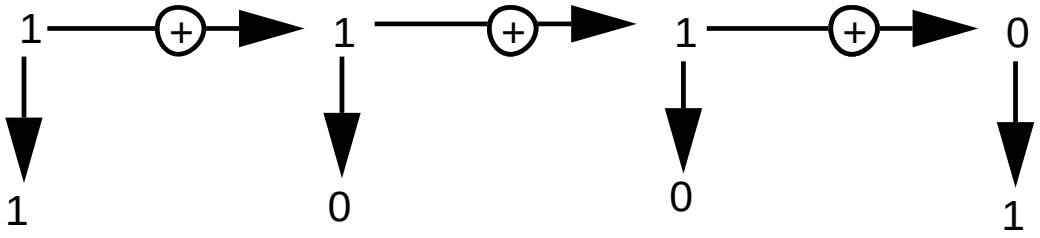
- Compare this with the binary code, where anywhere from one to all of the bits changes in going from one step to the next.
- For example:

Decimal	Binary code	Gray code
7 to 8	0111 to 1000	0100 to 1100

- The Gray code is often used in situations where other codes, such as binary, might produce erroneous or ambiguous results during those translations in which more that one bit of the code is changing.
- Obviously, using the Gray code would eliminate this problem, since only one bit changes occurs per transition and no "race" between bits can occur.

Binary-to-Gray code Conversion

- Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:
- The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.
- Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.
- For example, the conversion of the binary number 1110 to Gray code is as follows:



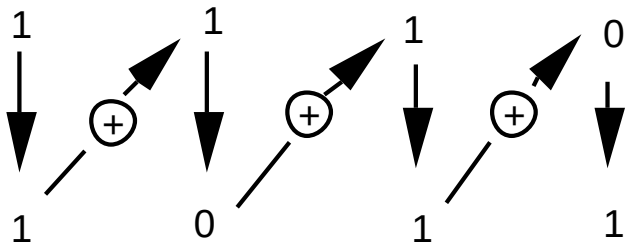
= 1001 (Gray code)

- The Gray code is 1001.

Gray -to- Binary Conversion

- The following rules explain how to convert from gray code to a binary number :
- The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
- Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.
- For example, the conversion of the Gray code word 1110 to binary is as follows:

Eg : 1110 (Gray Code)



= 1011_2

The binary number is 1011.

The Excess –3 code

- The excess-3 code is another important BCD code. To encode a decimal number to the excess –3 form, we add 3 to each digit of the decimal number and convert to binary form.
- For example

a) Convert 4 to an excess –3 number

Decimal number		Excess-3 codes number
4 + 3 = 7 (Add 3)	(Convert to 4-bit binary) →	0111

b) Convert 39 to an excess –3 number

3	9	
+ 3	+ 3	→ Add 3 to each digit
6	12	
↓	↓	→ Convert to 4-bit binary code
0110	1100	

- ✓ Note that both BCD and Excess-3 use only 10 of the 16 possible 4-bit code groups.
- ✓ The Excess-3 code, however, does not use the same code groups.
- ✓ For Excess-3 code, the invalid code groups are 0000, 0001, 0010, 1101,1110, and 1111.

ALPHANUMERIC CODES

- Computer should recognize codes that represent letters of the alphabet, punctuation marks, and other special characters as well as numbers.
- These codes are called ''alphanumeric codes''.
- A complete alphanumeric code would include:
  - ✓ The 26 lower case letters;
  - ✓ 26 upper case letters,
  - ✓ 10 numeric digits,
  - ✓ 7 punctuation marks, and
  - ✓ anywhere from 20 to 40 other characters, such as +, /, #, %, *, and so on.
- A complete alphanumeric characters and function are found on a standard typewriter (or computer) keyboard.

The ASCII Code

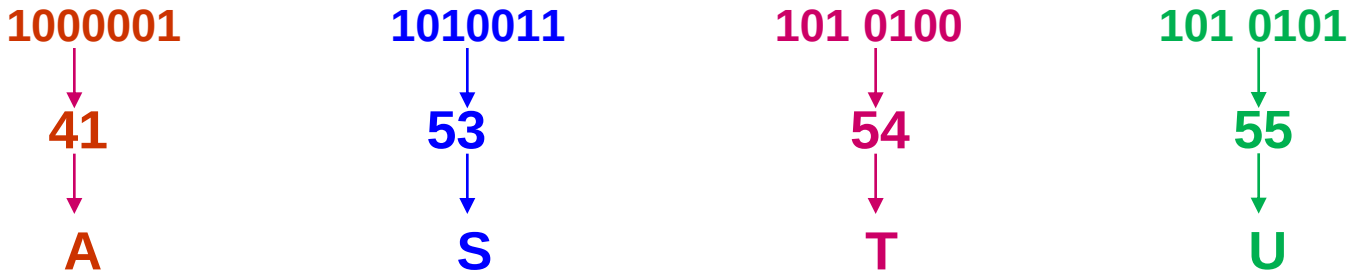
- The most widely used alphanumeric code, the America standard code for Information Interchange (ASC II), is used in most micro computers and mini computers, and in many mainframes.
- The ASCII code (pronounced ‘’ask-ee’’) is a 7- bit code, and so it has 2^7= 128 possible code groups.
- This is more than enough to represent all of the standard keyboard characters as well as control functions such as the <RETURN> AND < LINEFEED> Functions.

Character	7-Bit ASCII	Octal	Hex	Character	7-Bit ASCII	Octal	Hex
A	100 0001	101	41	Y	101 1001	131	59
B	100 0010	102	42	Z	101 1010	132	5A
C	100 0011	103	43	0	011 0000	060	30
D	100 0100	104	44	1	011 0001	061	31
E	100 0101	105	45	2	011 0010	062	32
F	100 0110	106	46	3	011 0011	063	33
G	100 0111	107	47	4	011 0100	064	34
H	100 1000	110	48	5	011 0101	065	35
I	100 1001	111	49	6	011 0110	066	36
J	100 1010	112	4A	7	011 0111	067	37
K	100 1011	113	4B	8	011 1000	070	38
L	100 1100	114	4C	9	011 1001	071	39
M	100 1101	115	4D	blank	010 0000	040	20
N	100 1110	116	4E	.	010 1110	056	2E
O	100 1111	117	4F	(	010 1000	050	28
P	101 0000	120	50	+	010 1011	053	2B
Q	101 0001	121	51	$	010 0100	044	24
R	101 0010	122	52	*	010 1010	052	2A
S	101 0011	123	53	)	010 1001	051	29
T	101 0100	124	54	-	010 1101	055	2D
U	101 0101	125	55	/	010 1111	057	2F
V	101 0110	126	56	,	010 1100	054	2C
W	101 0111	127	57	=	011 1101	075	3D
X	101 1000	130	58	<RETURN>	000 1101	015	0D
				<LINEFEED>	0001010	012	0A

***Example 1*:**

The following is a message encoded in ASCII code. What is the message?
1000001 1010011 1010100 1010101

Solution: **convert each 7- bit code to its hex equivalent. The results are**



- **The ASCII code is used for the transfer of alphanumeric information between a computer and input / output devices such as video terminals or printers.**

Example 2

Determine the codes that will be entered in to memory when the operator types in the following BASIC statement:

GOTO 25

***Solution*:**

Locate each character (including the space) in table and record its ASCII code

G 1000111
O 1001111
T 1010100
O 1001111
(Space) 0100000
2 0110010
5 0110101