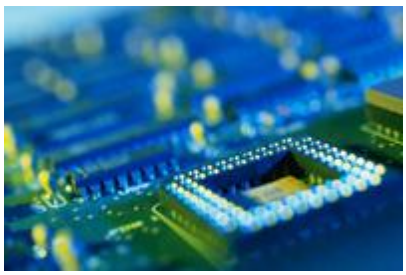
1. Binary Number Representation

1.1 Introduction to Digital Systems

- Digital systems (such as computers, electronic games, electronic appliances, electro-medical equipment, industrial instrumentation, etc.) are those that handle information encoded in binary form.



- Binary form is information that has only **two** possible values such as TRUE and FALSE, ON and OFF, or in more useful form, 1 or 0.

- It is easy to build a simple circuit that has only two states, the classic example being a switch. The switch is either on or off.

- Thus, if we can represent things, such as numbers, with some sort of combination of binary symbols, then we should be able to represent the same concepts with our simple circuits.

1.2 Digital Circuits and Waveforms

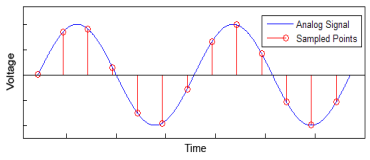
- Electronic circuits can be divided into two broad categories, namely analog and digital.

- An analog quantity is a variable that has continuous values, eg. people's heights.

- A digital quantity is a variable that has only a discrete set of values, eg. shoe sizes.

- Most variables that occur naturally are in analog form, eg. air temperature.

- However, if we measure the air temperature at discrete points in time, say every minute, to produce a sampled-value representation and represent each of these samples with a suitable digital code, then it becomes a digital quantity.



- The advantages of doing this (i.e. of digital) are:
  - digital data can be processed and transmitted more efficiently than analog data
  - digital data can be stored more efficiently
  - digital data is less affected by noise.

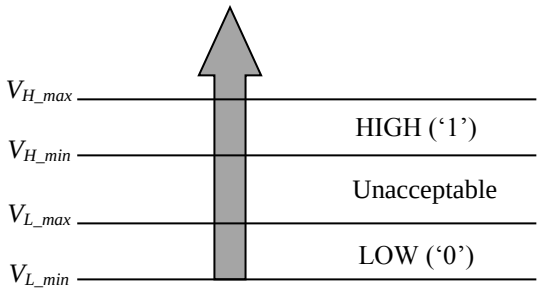
- In binary, there are two states, namely logic '1' and logic '0'.

- These are implemented in digital circuits using two different voltage levels where a '1' is represented by a higher voltage (HIGH) and '0' is represented by a lower voltage (LOW).

- This convention is known as ***Positive Logic***.

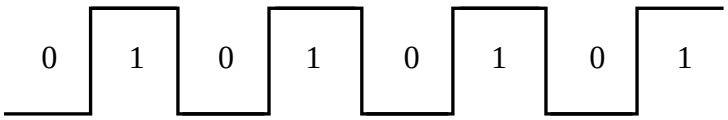
- The voltages used to represent the logic '1' and logic '0' are referred to as the ***Logic Levels***.

- In practical circuits, we have a range for these levels but there can be no overlap between them, as shown in the diagram.



Digital waveforms …

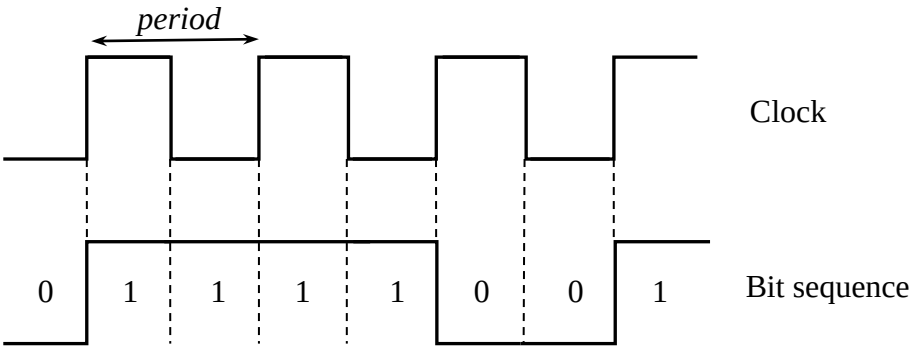
- Digital waveforms consist of voltage levels that are changing back and forth between HIGH and LOW levels of states.



- The pulses shown here are ideal, as in reality such waveforms do not have instantaneous changes between states, as illustrated.

- Binary information that is handled by digital systems appears as waveforms that represent sequences of bits. When the waveform is HIGH, a binary '1' is present and when the waveform is LOW a binary '0' is present.

- In digital systems, all waveforms are generally synchronized with a basic timing waveform known as the **clock**.



- The period of the clock waveform dictates the time for one bit.

- Having briefly introduced the concept of digital logic and digital systems, it should be apparent that it is essential to understand the world of binary and to see how we can represent numbers in this form. We will now do so …

1.3 The Binary Number System

The decimal system …

- First, let us consider the well known decimal system, which works to represent any number we need.
- As we know, it uses ten different symbols, namely 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
- So how does the decimal representation work exactly?
- By way of example, consider the decimal number 360. This really means that the value is represented by the sum of three one-hundreds and six tens.
- More generally, a 4-digit decimal number ***wxyz*** actually stands for:

$$\boldsymbol{wxyz} = (\boldsymbol{w} \times 10^3) + (\boldsymbol{x} \times 10^2) + (\boldsymbol{y} \times 10^1) + (\boldsymbol{z} \times 10^0)$$

- By way of illustration, consider the decimal number 4354:

$$4354 = (4 \times 10^3) + (3 \times 10^2) + (5 \times 10^1) + (4 \times 10^0)$$
$$= 4000 + 300 + 5 + 4 = 4354$$

- Likewise, a 5-digit decimal number ***vwxyz*** stands for:

$$\boldsymbol{vwxyz} = (\boldsymbol{v} \times 10^4) + (\boldsymbol{w} \times 10^3) + (\boldsymbol{x} \times 10^2) + (\boldsymbol{y} \times 10^1) + (\boldsymbol{z} \times 10^0)$$

  and so on …

- Note that the '10' here refers to the numbers of allowed symbols and is often referred to as the **base** or **radix** of the number system
- This means that decimal is considered to be counting using "base 10", i.e. that there are 10 different symbols.
- Note also, as numerous different base values are possible, we should, in theory, clearly identify which base we are using. As such, the decimal number 360 should in fact be expressed as:

$$360_{10}$$

- However, as the decimal system is so widely used and known, the subscript '10' is typically omitted.
- Now let's consider the binary representation, where only 2 symbols are possible, i.e. a "base 2" or binary counting system.

The binary system …

- Now, a 4-digit **binary** number ***wxyz*** means:

$$\boldsymbol{wxyz} = (\boldsymbol{w} \times 2^3) + (\boldsymbol{x} \times 2^2) + (\boldsymbol{y} \times 2^1) + (\boldsymbol{z} \times 2^0)$$

- For example:

$$1001_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$
$$= 8 + 0 + 0 + 1 = 9 \text{ in decimal}$$

- Similarly:

$$111_2 = (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 4 + 2 + 1 = 7 \text{ in decimal}$$

$$1_2 = (1 \times 2^0) = 1 \text{ in decimal}$$

$$10_2 = (1 \times 2^1) + (0 \times 2^0) = 2 + 0 = 2 \text{ in decimal}$$

- Note the importance of the base:

  1001 is one thousand and one (taking the default base as decimal)
  1001_2 is binary 1, 0, 0, 1 and actually has a value of decimal 9.

- *However, that said, as we will be focusing primarily on binary numbers (for a digital systems module), we will omit the base 2 for the sake of convenience.*

Counting in binary …

- Counting in binary follows the same principle as in decimal. With decimal, we count from 0 to 9 at which point we have exhausted all available symbols (or digits). Thus, we then start a digit position to the left and continue counting 10 through to 99.

- Once again, we have exhausted all available symbols and now add another digit position to the left and continue counting from 100, etc.

- Binary works in the same fashion. Hence:

  0, 1, **1**0, 11, **1**00, 101, 110, 111, **1**000, 1001, ...

A 'Bit' of information! …

- A digit in binary is often referred to as a **Bit**.

- A single bit can represent at most 2 numbers (0 to 1), i.e. 0 and 1.

- Two bits can represent up to 4 numbers (0 to 3), i.e. 00, 01, 10 and 11.
- Three bits can represent up to 8 numbers (0 to 7), i.e. 000, 001, 010, 011, 100, 101, 110 and 111.
- And so on for higher number of bits.
- In general, using *n*-bits, we can represent 2^n numbers from 0 to $2^n - 1$. In other words, the largest decimal number represented by *n*-bits is $2^n - 1$.
- For example, consider using 3 bits, then:

  $2^n - 1 = 2^3 - 1 = 7$ is the highest decimal value that can be represented, i.e. $7 = 111_2$

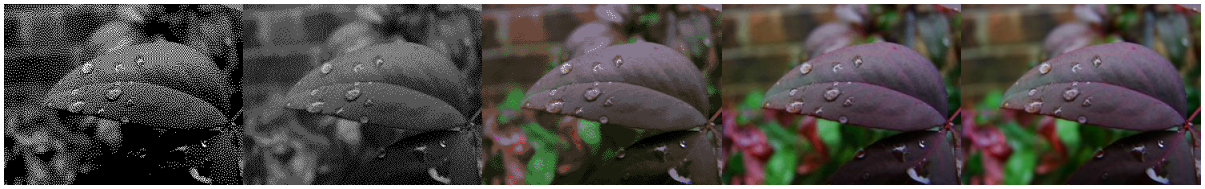
  So, in this case, we can represent 8 numbers, from 0 up to 7, as we have seen already.

- Likewise, using 4 bits:

  $2^n - 1 = 2^4 - 1 = 15$

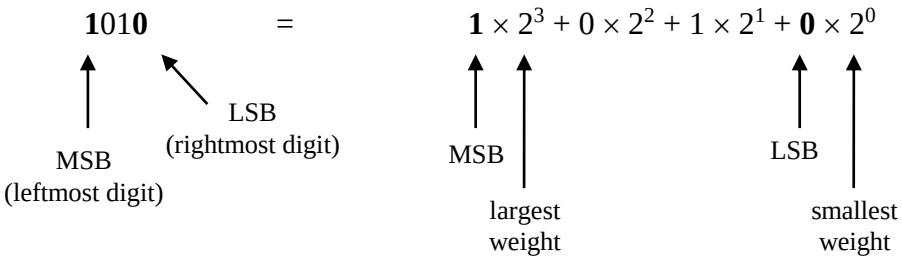
  So, in this case, we can represent 16 numbers, from 0 up to 15.

- Clearly, the greater the number of bits used, the more information that can be represented.
- This concept is neatly conveyed by the following illustration on colour depth:



1-bit 2-bit 4-bit 8-bit 16-bit

- In a given binary number, we have two specific bits called the **Least Significant Bit (LSB)** and the **Most Significant Bit (MSB).**
- The LSB refers to the binary digit with the smallest weight (the rightmost digit) and the MSB refers to the binary digit with the largest weight (the leftmost digit).
- By way of illustration, consider the binary number 1010:



1.4 Binary-decimal conversion

- It's important that we can readily convert between decimal and binary. We have already seen how to convert from **binary to decimal**.

- ***Ex. 1.1 Convert 1100 to decimal***

  $1100 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
  $= 8 + 4 + 0 + 0 =$ **12**

- ***Ex. 1.2 Convert 1101101 to decimal***

  $1101101 = 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
  $= 64 + 32 + 0 + 8 + 4 + 0 + 1 =$ **109**

- Note, in practice, we only need to sum the weights (i.e. the powers of 2 in this case) of each '1' bit. We can simply ignore those for the '0' bits. Thus the last example would be carried out as follows:

  1101101 has weights $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

  Decimal value is $= 2^6 + 2^5 + 2^3 + 2^2 + 2^0 = 64 + 32 + 8 + 4 + 1 =$ **109**

- To convert from **decimal to binary**, we carry out successive **divisions** by 2 as follows:

  Consider a binary number $N = (a_3\ a_2\ a_1\ a_0)_2 = a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0$

  Divide by 2 to give:

	$N/2 = a_3 2^2 + a_2 2^1 + a_1 = P_1$,	*remainder* a_0	= LSB
again:	$P_1/2 = a_3 2^1 + a_2 = P_2$,	*remainder* a_1	
again:	$P_2/2 = a_3 = P_3$,	*remainder* a_2	
again:	$P_3/2 = 0 = P_4$,	*remainder* a_3	= MSB

  The process continues until we are left with 0.

- ***Ex. 1.3 Convert 12 to binary***

12 /2 = 6,	rem = 0	= LSB
6 /2 = 3,	rem = 0	
3 /2 = 1,	rem = 1	
1 /2 = 0,	rem = 1	= MSB

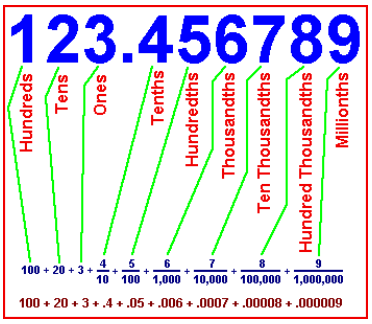
  Hence, 12 in binary is: **1100**

- ***Ex. 1.4 Convert 20 to binary***

  20 /2 = 10, rem = 0 = LSB
  10 /2 = 5, rem = 0
  5 /2 = 2, rem = 1
  2 /2 = 1, rem = 0
  1 /2 = 0, rem = 1 = MSB

  Hence, 20 in binary is: **10100**

1.5 Dealing with Fractions



- So how do we represent fractional numbers in binary?
- In decimal representation:

  $54.23 = 5 \times 10^1 + 4 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2}$
  $= 50 + 4 + 0.2 + 0.03$

- We do exactly the same thing with binary fractions!
- For example:

  $10.01 = 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 2 + 0 + 0 + 0.25 = 2.25$ in decimal

  $11.1 = 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} = 2 + 1 + 0.5 = 3.5$ in decimal

- In general, a 5 bit number, for example, with 3 bits to the right of the decimal point can be expressed as:

  $$\boldsymbol{vw.xyz} = \boldsymbol{v} \times 2^1 + \boldsymbol{w} \times 2^0 + \boldsymbol{x} \times 2^{-1} + \boldsymbol{y} \times 2^{-2} + \boldsymbol{z} \times 2^{-3}$$

- ***Ex. 1.5 Convert 0.1011 to decimal***

  $0.1011 = 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$
  $= 0 + 0.5 + 0.125 + 0 + 0.0625 =$ **0.6875**

  alternatively:

binary number	0.	1	0	1	1
weights	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

  Hence: $0.1011 = 2^{-1} + 2^{-3} + 2^{-4} = 0.5 + 0.125 + 0.0625 =$ **0.6875**

- To convert from **fractional decimal to binary**, we carry out successive **multiplications** by 2.

- The fractional part is carried over the next stage and the whole number forms part of the binary number. The process is repeated until the fractional part is zero. This is best illustrated using an example.

- Consider the decimal fraction 0.625:

  $0.625 \times 2 = 1.25$, so the whole number 1 = MSB

  $0.25 \times 2 = 0.5$, so the whole number 0

  $0.5 \times 2 = 1.0$, so the whole number 1 = LSB

  The process continues until the fractional part is 0.

  Hence, 0.625 in binary is: **0.101**

- ***Ex. 1.6 Convert 0.6875 to binary***

  $0.6875 \times 2 = 1.375$, so the whole number 1 = MSB

  $0.375 \times 2 = 0.75$, so the whole number 0

  $0.75 \times 2 = 1.5$, so the whole number 1

  $0.5 \times 2 = 1.0$, so the whole number 1 = LSB

  Hence, 0.6875 in binary is: **0.1011**

- ***Ex. 1.7 Convert 5.25 to binary***

  We split this into two parts, namely 5 and 0.25

  5 in binary is given by 101 *... verify this for yourself*

  0.25 in binary is given by 0.01 *... verify this for yourself*

  Hence 5.25 in binary is 101.01

  Quick check: $101.01 = 2^2 + 2^0 + 2^{-2} = 4 + 1 + 0.25 = 5.25$

1.6 Binary Arithmetic

- We will now look at adding and subtracting binary numbers. The rules for binary are similar to those used in the decimal system.

- *Similarly, multiplying and dividing binary numbers follow the same principles as the decimal equivalent. However, we are not going to consider these in detail in this module – refer to a textbook if need be.*

THERE ARE 10 KINDS OF PEOPLE IN THE WORLD...

THOSE WHO UNDERSTAND BINARY, AND THOSE WHO DON'T.

Addition ...

- The four rules of **binary addition** are:

$0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 1$ $1 + 1 = 10$

- ***Ex. 1.8 Add the binary numbers 11 and 01***

$$\begin{array}{r} 1\ \ 1 \\ +\ {}_{1}\ 0_{1}\ 1 \\ \hline \mathbf{1\ \ 0\ \ 0} \end{array}$$

Note – the subscript shows the 'carry'

(Quick check – in decimal, we have 3 + 1 = 4)

- ***Ex. 1.9 Add the binary numbers 011 and 011***

$$\begin{array}{r} 0\ \ 1\ \ 1 \\ +\ 0_{1}\ 1_{1}\ 1 \\ \hline \mathbf{1\ \ 1\ \ 0} \end{array}$$

(Quick check – in decimal, we have 3 + 3 = 6)

Subtraction ...

- The four rules of **binary subtraction** are:

$0 - 0 = 0$ $1 - 1 = 0$ $1 - 0 = 1$ $0 - 1 = 1$ with a borrow of 1

- ***Ex. 1.10 Subtract the binary number 01 from 11***

$$\begin{array}{r} 1\ \ 1 \\ -\ \ 0\ \ 1 \\ \hline \mathbf{1\ \ 0} \end{array}$$

(Quick check – in decimal, we have 3 - 1 = 2)

- ***Ex. 1.11 Carry out the binary subtraction 101 – 011***

$$\begin{array}{rrrr} & 1 & {}^{1}0 & 1 \\ - & 0_{1} & 1 & 1 \\ \hline & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{array}$$

Note – the superscript shows the 'borrow'

(Quick check – in decimal, we have 5 – 3 = 2)

Multiplication ...

- The four rules of **binary multiplication** are:

$0 \times 0 = 0$ $0 \times 1 = 0$ $1 \times 0 = 0$ $1 \times 1 = 1$

- It is useful to know this when we look at logic gates later in the notes.

1.7 Dealing with Negative Numbers

- How do we deal with negative numbers in binary form?
- In decimal, we simply add in the '–' sign to indicate a negative number. However, digital circuits have only two states and hence, this is not possible in such a binary system.
- Instead, we devise a CODE using the existing symbols of '1' and '0' to represent negative binary numbers.
- There are two common codes used, namely 1's complement and 2's complement.
- To find the **1's complement** of a binary number we simply *invert each bit*, as follows:

$$\begin{array}{rr} 1\,0\,1\,0\,1\,0\,0\,1\,1 & \ldots \text{ binary number} \\ 0\,1\,0\,1\,0\,1\,1\,0\,0 & \ldots \text{ 1's complement} \end{array}$$

- To go from 1's complement back to normal form we simply invert each bit again.
- To find the **2's complement** of a binary number we simply *add 1 to the LSB of the 1's complement*, as follows:

$$\begin{array}{rrr} & 1\,0\,1\,0\,1\,0\,0\,1\,1 & \ldots \text{ binary number} \\ & 0\,1\,0\,1\,0\,1\,1\,0\,0 & \ldots \text{ 1's complement} \\ + & \underline{\qquad\qquad\qquad 1} & \\ & 0\,1\,0\,1\,0\,1\,1\,0\,1 & \ldots \text{ 2's complement} \end{array}$$

- To go from 2's complement back to normal form we invert all the bits and add 1 to the LSB, i.e. the exact same process as obtaining the 2's complement in the first place!

- There are 3 ways of representing negative numbers, by using the:

  - sign-magnitude system
  - 1's complement system
  - 2's complement system

- Under all 3 systems, the left-most bit is designated the **sign bit**. This informs us whether the number is positive or negative as follows:

  - a '0' implies a positive number
  - a '1' implies a negative number

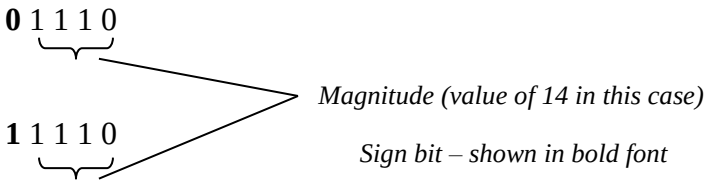
Sign-magnitude system …

- In the sign-magnitude system, the left-most bit is the sign bit and the remaining bits are the magnitude bits.

- Thus, for example:

  +14 is given in binary as:

  while –14 is:



1's complement system …

- In the 1's complement system, positive numbers are simply the same as the positive sign-magnitude numbers.

- Negative numbers are then the 1's complement of the corresponding positive number.

- Hence:

+14	→	**0** 1 1 1 0	*(Note sign bit '0')*
–14	→	**1** 0 0 0 1	*(the 1's complement)*

2's complement system …

- In the 2's complement system, positive numbers are, once again, simply the same as the positive sign-magnitude numbers.
- Negative numbers are then the 2's complement of the corresponding positive number.
- Hence:

+14 → **0** 1 1 1 0

–14 → **1** 0 0 1 0 *(the 2's complement ≡ 1's complement + '1')*

- ***Ex. 1.12 Express the decimal number –39 as an 8-bit binary number using:***

(i) the sign-magnitude system,

(ii) the 1's complement system and

(iii) the 2's complement system

Firstly, the 8-bit binary representation of + 39 is:

0 0 1 0 0 1 1 1 *(don't forget… 1 bit for sign and 7 bits for magnitude)*

(i) Hence, using the sign-magnitude system, we get:

–39 → **1 0 1 0 0 1 1 1**

(ii) Using the 1's complement system, we get:

–39 → **1 1 0 1 1 0 0 0**

(iii) Using the 2's complement system, we get:

–39 → **1 1 0 1 1 0 0 1**

1.8 Converting Signed Numbers to Decimal

Sign-magnitude system …

- The decimal value of a signed number expressed using the **sign-magnitude system** is found by summing the weights of the ***magnitude*** bits that have a value of '1'.
- The sign is determined by checking to see of the sign bit is a '1' for '–' or a '0' for '+'.
- For example, consider the following binary number expressed using the sign-magnitude system:

  1 0 0 1 0 1 0 1

  weights: x 2^6 2^5 2^4 2^3 2^2 2^1 2^0 = 16 + 4 + 1 = 21

  sign-bit is '1' ⇒ negative. Hence, we have **– 21**

1's complement system …

- The decimal value of a signed number expressed using the **1's complement system** is found by summing the weights of ***all*** the bits that have a value of '1'.
- We ***assign a negative value to the weight of the sign bit during the summing***.
- If we have a negative value (i.e. sign bit is '1') we add 1 to the decimal result to get our final answer. If the sign bit is '0' we don't do anything.
- For example, find the decimal values of the following signed binary numbers, given that they are expressed in 1's complement form: (i) 00010111 and (ii) 11101000

  (i) 0 0 0 1 0 1 1 1

  weights: -2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 = 16 + 4 + 2 + 1 = +23

  sign-bit is '0' ⇒ no change required. Hence, we have **+ 23**

  (ii) 1 1 1 0 1 0 0 0

  weights: -2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 = –128 + 64 + 32 + 8 = –24

  sign-bit is '1' ⇒ **add '1' to decimal value**. Hence, we have –24 + 1 = **– 23**

2's complement system …

- The decimal value of a signed number expressed using the **2's complement system** is found by summing the weights of ***all*** the bits that have a value of '1'.
- We ***assign a negative value to the weight of the sign bit during the summing***.
- Unlike the 1's complement system, there is no further action required.
- For example, find the decimal values of the following signed binary numbers, given that they are expressed in 2's complement form: (i) 01010110 and (ii) 10101010

(i) 0 1 0 1 0 1 1 0

weights: $-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ = 64 + 16 + 4 + 2 = **+86**

(ii) 1 0 1 0 1 0 1 0

weights: $-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ = –128 + 32 + 8 + 2 = **–86**

Which system to use …

- The 2's complement system is the preferred system because the conversion process requires just a simple summation irrespective of whether the number is positive or negative.
- Furthermore, the addition of the positive and negative representations of such a number gives a zero.
- *Note – in an electronic system, it is important that the system chosen is clearly specified prior to use.*
- ***Ex. 1.13 Find the decimal equivalent of the 8-bit binary number 10000101 given that it is expressed using:***

(i) the sign-magnitude system,
(ii) the 1's complement system and
(iii) the 2's complement system

(i) 1 0 0 0 0 1 0 1

weights: x $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ = 4 + 1 = 5

sign-bit is '1' ⇒ negative. Hence, we have **– 5**

(ii) 1 0 0 0 0 1 0 1

weights: $-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ $= -128 + 4 + 1 = -123$

sign-bit is '1' $\Rightarrow$ add '1' to decimal value. Hence, we have $-123 + 1 =$ **-122**

(iii) 1 0 0 0 0 1 0 1

weights: $-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ $= -128 + 4 + 1 =$ **-123**

Range of signed integer numbers …

- Consider an *n*-bit binary number. As we have seen already, this has 2^n different combination of 1's and 0's.
- For standard (unsigned) binary, the possible represented number ranges from 0 to $2^n - 1$. So for an 8-bit binary number, for example, the number would exist in the range 0 to 255.
- In the case of an *n*-bit **2's complement signed binary number**, the range becomes:

  -2^{n-1} to $2^{n-1} - 1$

- So, for an 8-bit example, we have the range: –128 to +127
- Note that for an *n*-bit **1's complement signed binary number**, the range is:

  $-2^{n-1} + 1$ to $2^{n-1} - 1$

- So, for an 8-bit example, we have the range: –127 to +127
- Similarly, for an *n*-bit **sign-magnitude binary number**, the range is:

  $-(2^{n-1} - 1)$ to $2^{n-1} - 1$

- So, for an 8-bit example, we have the range: –127 to +127
- *Exercise - Verify all these ranges by determining the decimal equivalent of the following 8-bit binary numbers, given that they are expressed using (i) the sign-magnitude system, (ii) the 1's complement system and (iii) the 2's complement system:*

  (a) 00000000 (b) 11111111 (c) 10000000 (d) 01111111

1.9 Hexadecimal Representation

Why hexadecimal? …

- Binary numbers can be cumbersome to write down, especially when we are working with a large number of bits.

- A more convenient notation is that of hexadecimal, which uses a base 16. This cuts down the number of digits by 4. Furthermore, it is very easy to convert between binary and hexadecimal.

- Note that hexadecimal is simply used for the ease of notation (i.e. for the sake of convenience). Digital computers only understand binary regardless.

- Also note – each set of four bits is referred to as a **nibble**.

The hexadecimal symbol set …

- The hexadecimal system is base 16 and hence has 16 symbols. These are the conventional ones for decimal plus ‘A’, ‘B’, ‘C’, ‘D’, ‘E’ and ’F’.

- The following table shows a count from 0 to 15 in binary, decimal and hexadecimal for comparison purposes.

Decimal	Binary (4-bit)	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

- We denote a hexadecimal number using the subscript 16 (sometimes 'h' is used) for example F_{16} or F_h.

- Once we reach F_{16}, we simply add another column to the left and start again (as we do when counting in decimal and in binary). Hence:

Decimal	Binary (5-bit)	Hexadecimal
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14
21	10101	15
22	10110	16
23	10111	17
24	11000	18
25	11001	19
26	11010	1A
27	11011	1B
28	11100	1C
29	11101	1D
30	11110	1E
31	11111	1F

Converting binary to hexadecimal …

- Since a 4-bit binary number has the same number of count representations as a 1-digit hexadecimal number, we can easily convert a binary number into hexadecimal by breaking it into 4-bit groups starting from the right. We then replace each if the groups with the corresponding hexadecimal (hex) symbol.

- ***Ex. 1.14 Convert the binary numbers (i) 1100 1010 0101 0111 and (ii) 11 1111 0001 0110 1001 to hex***

(i) $\underbrace{1100}_{C}\ \underbrace{1010}_{A}\ \underbrace{0101}_{5}\ \underbrace{0111}_{7}$ $\Rightarrow$ **$CA57_{16}$**

(ii) $\underbrace{\mathbf{00}11}_{3}\ \underbrace{1111}_{F}\ \underbrace{0001}_{1}\ \underbrace{0110}_{6}\ \underbrace{1001}_{9}$ $\Rightarrow$ **$3F169_{16}$**

Converting hex to binary …

- To convert from hex to binary, we simply replace each hex symbol with the appropriate 4-bits.

- ***Ex. 1.15 Convert the hex numbers (i) 10A4 and (ii) CF8E to binary***

(i) $\underbrace{1}_{1}\ \underbrace{0}_{0000}\ \underbrace{A}_{1010}\ \underbrace{4}_{0100}$ $\Rightarrow$ **$1\ 0000\ 1010\ 0100_2$**

(ii) $\underbrace{C}_{1100}\ \underbrace{F}_{1111}\ \underbrace{8}_{1000}\ \underbrace{E}_{1110}$ $\Rightarrow$ **$1100\ 1111\ 1000\ 1110_2$**

Converting between hex and decimal …

- There are two obvious ways of converting between hex and decimal.

- The first way is to convert the decimal to binary and then convert the binary to hex and vice versa for hex to decimal.

- The second, more direct, method is to use the same conversion procedure as was used between binary and decimal but using weights of '16' instead of '2'.

- For example, converting $B2F8_{16}$ to decimal gives:

$$B2F8_{16} = (B \times 16^3) + (2 \times 16^2) + (F \times 16^1) + (8 \times 16^0)$$

$$= 45056 + 512 + 240 + 8 \qquad = \mathbf{45816_{10}}$$

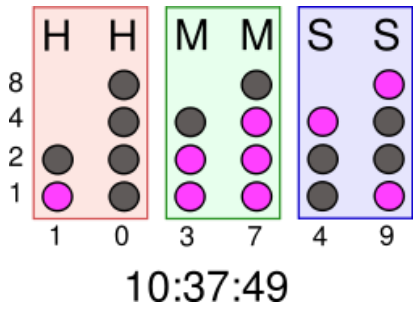
1.10 Binary Codes

- In this section, we are going to look at three different, but commonly used, binary codes. These are BCD (binary coded decimal), Gray code and ASCII code.

Binary Coded Decimal …

- Binary Coded Decimal (BCD) is a simple way to express EACH of the decimal digits with a binary code.

- The use of BCD, in this way, makes it very easy to convert between binary and decimal.



- Using the 8421 BCD code**, each decimal digit 0 to 9 is represented by a binary code of 4 bits**.

- For example: Decimal 1 0 3
  BCD **0001 0000 0011**

- In conventional binary, the number would be 1100111_2.

- To convert from BCD back to decimal, we simply break the binary number into groups of four bits, starting from the right, and replace each group of 4 bits with their decimal equivalent.

- For example: BCD 1000 0010
  Decimal **8 2**

- By applying different weighting, the following sample codes can be derived including *Binary Coded Decimal* (BCD):

Decimal	8421 (BCD)	84-2-1	2421
0	0000	0000	0000
1	0001	0111	0001
2	0010	0110	0010
3	0011	0101	0011
4	0100	0100	0100
5	0101	1011	1011
6	0110	1010	1100
7	0111	1001	1101
8	1000	1000	1110
9	1001	1111	1111

- *Note that the 84-2-1 and 2421 codes are known as the 9's complement codes because if you invert the binary numbers for 0 to 4 you get 9 to 5!*

Gray Code …

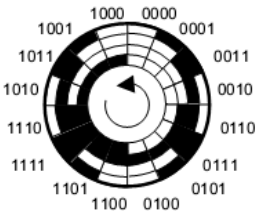
- The Gray code is yet another binary code but it does not use weights, i.e. it does not have weights assigned to the bits position.

- Instead, it is designed so that successive decimal digits differ by exactly one bit. This property is very useful in some applications, particularly where mechanical switches are involved.

- With the Gray code, transition between consecutive states requires only a single switch, whereas with natural codes, multiple switches may need to occur at the same time and this can cause 'glitch' issues (i.e. it is difficult to ensure that the switches are properly synchronized).

- The 4-bit Gray code is shown below:

Decimal	8421 (BCD)	Gray
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**



Gray code

ASCII Code …

- Binary codes can also be used to convey letters and other symbols (and not just numbers). The most commonly used alphanumeric code is the ASCII code.

- ASCII stands for the American Standard Code for Information Interchange and it is the standard alphanumeric code used in computers today.

- Every time you press a key on the keyboard of your computer, the corresponding ASCII binary code is sent to the processor.

- ASCII has 128 characters and symbols which is represented in a 7-bit code. In fact, it is actually an 8-bit code but the MSB is always '0' to allow for possible extendibility.

- The actual ASCII code for each character and symbol is available in a look-up table, a sample of which is shown below:

Symbol	Decimal	Binary	Hex
A	65	1000001	41
B	66	1000010	42
C	67	1000011	43
D	68	1000100	44
E	69	1000101	45

ASCII