A-Level
Computer Science

Binary representation

Lesson Objectives

Students will learn about:

- The binary representation of numbers
- Understand the units of computer memory
- Converting denary numbers to a binary system and vice versa
- Adding binary numbers
- Binary shifts
- Bitwise operations
- Representing negative numbers in a binary system
- Binary Coded Decimal (BCD)

1.

Content

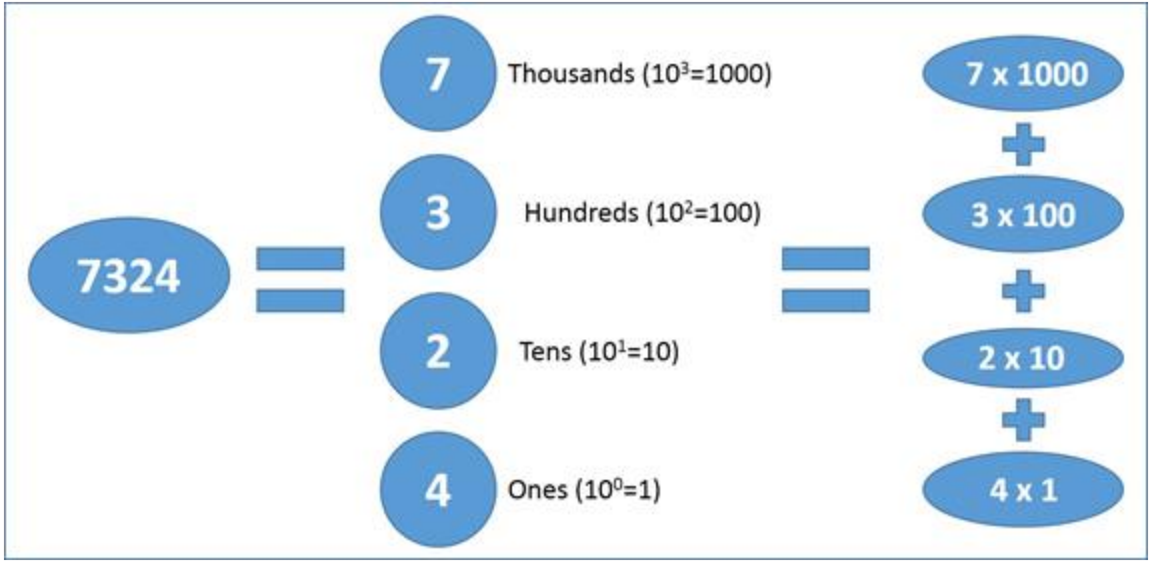
Introduction

- A computer has many electronic components that work as switches.
- These components have two logics as input and output: ON and OFF.
- A similar logic is used to represent data in binary form. ON is represented as 1 and OFF is represented as 0.

In the history of culture, the discovery of zero will always stand out as one of the greatest single achievements of the human race.

–*Tobias Dantzig, in Number: the Language of Science*, 1930

Place value of denary system



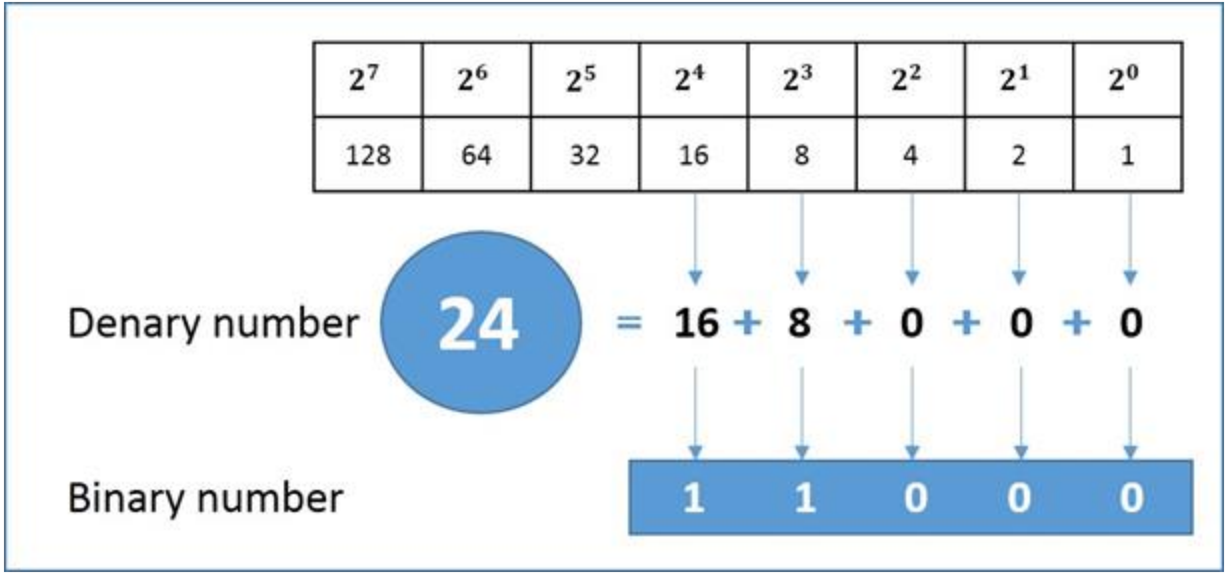
- The denary system has a base value of 10.
- It counts in multiples of 10.

Place value of binary system

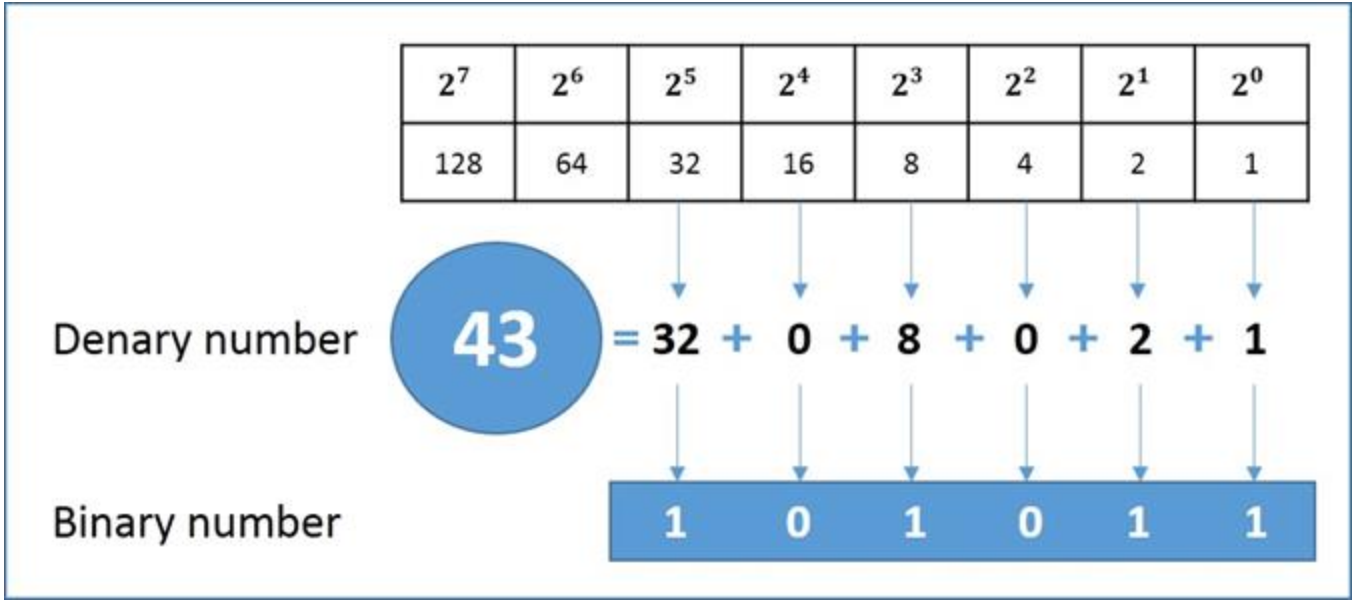
- The place values of binary numbers are of base 2.

128	64	32	16	8	4	2	1

Place value of binary system



Place value of binary system



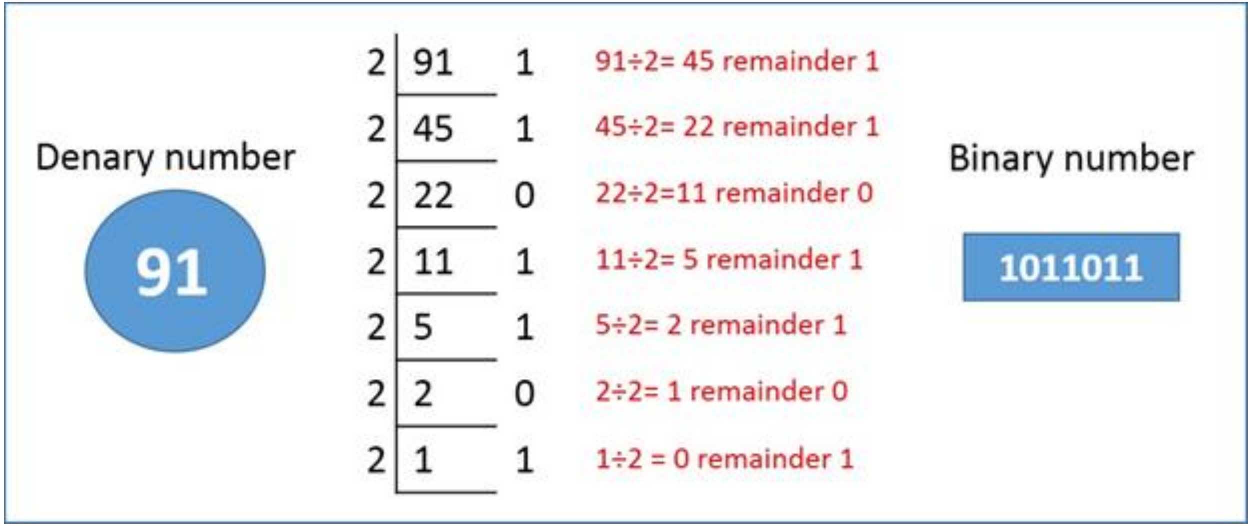
Converting denary to binary

1. Divide the number by 2 and write down the remainder
2. Continue dividing the quotients by 2 and write down the remainder
3. List all the remainders in reverse order

Converting denary to binary

i. A denary number is converted to binary by dividing it by 2 and calculating the remainders.

ii. The binary equivalent is obtained by arranging the remainders in reverse order.



How to check your answer?

- The binary equivalent of denary number 91:

0	1	0	1	1	0	1	1

- The answer can be checked by:

$$(0\times128)+(1\times64)+(0\times32)+(1\times16)+(1\times8)+(0\times4)+(1\times2)+(1\times1)=91$$

Activity-1

Duration: 15 minutes

1. Find the binary equivalent of denary number 113.
2. Check your answer by using the appropriate place values.

Size of computer memory

- A binary digit is referred to as a bit.
- A nibble consists of 4 bits.
- A byte consists of 8 bits.
- A byte is the smallest unit of memory of the computer system.
- The memory sizes available with computers are in multiples of 8 such as 16-bit systems, 32-bit systems, etc.

Size of computer memory

- The memory sizes were originally standardised using the base-2 representation.
- In this system, the prefixes kibi-, mebi-, gibi-, tebi- are used to avoid conflicts with the base-10 system.
- This representation is now used for representing the size of RAM modules only.

New name of the memory size	Number of bytes	Equivalent to
1 kibibyte (1 kiB)		1024 bytes
1 mebibyte (1 MiB)		1024^2 bytes
1 gibibyte (1 GiB)		1024^3 bytes
1 tebibyte (1 TiB)		1024^4 bytes

Size of computer memory

- After the standardisation of base-10 representation, the memory sizes are now represented as given.

Name of the memory size	Number of bytes	Equivalent to
1 kilobyte (1 kB)		1000 bytes
1 megabyte (1 MB)		1000000 bytes or 1000 kB
1 gigabyte (1 GB)		1000 MB
1 terabyte (1 TB)		1000 GB
1 petabyte (1 PB)		1000 TB

Binary combinations

- A one-bit system has a one-place value and can have 2 possible combinations: 0 or 1.
- Similarly, a 2-bit system has two-place values and has 4 possible combinations as shown in the table.

Place value 2	Place value 1	Binary number	Denary number
0	0	00	0
0	1	01	1
1	0	10	2
1	1	11	3

Binary combinations

Similarly, a 3-bit system has three-place values and has 8 possible combinations.

Place value 3	Place value 2	Place value 1	Binary number	Denary number
0	0	0	000	0
0	0	1	001	1
0	1	0	010	2
0	1	1	011	3
1	0	0	100	4
1	0	1	101	5
1	1	0	110	6
1	1	1	111	7

Representing numbers

- Programmers use many arithmetic operations in a program.
- The numbers are either represented as integers or floating point numbers.
- Integers are whole numbers and floating point numbers are used to represent numbers with decimal points.
- A 16-bit system can represent integers up to $2^{16}-1=65535$.
- 8-bit, 16-bit, 32-bit and 64-bit are the most common bit lengths.

Adding binary numbers

- Binary numbers are added in a column method as the denary numbers are added.
- Adding 0101 and 1011 in the table.
- Adding binary numbers
  - ✓ 0+0=0
  - ✓ 1+0=1
  - ✓ 1+1=10 (1 is carried over)

	Place value 4	Place value 2	Place value 3	Place value 1
Carry			1	
Number 1	0	1	0	1
Number 2	1	0	0	1
Sum	1	1	1	0

How to check your answer?

- 0101 and 1001 represent the denary numbers 5 and 9.
- The sum of 5 and 9 is 14.
- Convert the sum obtained to denary number.
- $(8\times1)+(4\times1)+(2\times1)+(0\times1)=14$

	Place value 4	Place value 2	Place value 3	Place value 1
Carry			1	
Number 1	0	1	0	1
Number 2	1	0	0	1
Sum	1	1	1	0

Overflow error

- A CPU with an 8-bit register has a capacity of up to 11111111 in binary. If an extra bit is added, it is said to be an overflow error.
- The number of bits a register can hold is called the word size. Exceeding the capacity of the word size in a register results in an overflow error.

Overflow error

- Consider the addition of two binary numbers 11101101 and 10000100.
- The sum of these two numbers is bigger than 8 bits (an extra bit than the register can hold).
- The computer thinks that 11101101+10000100=01110001 as it does not have space to store the extra bit.

Carry	1				1	1			
Number 1		1	1	1	0	1	1	0	1
Number 2		1	0	0	0	0	1	0	0
Sum	1	0	1	1	1	0	0	0	1

Activity-2

Duration: 15 minutes

1. What is the binary equivalent of denary numbers 11 and 14. Show your working.
2. Add the two binary numbers obtained in question 1. Show your working.
3. Find the denary equivalent of the sum obtained in question 2. Show your working.
4. Play game:

https://learningcontent.cisco.com/games/binary/index.html

Binary shifts

Shifting Right:

The Least Significant Bit (LSB) shifts to the carry and the MSB is occupied by a zero.

Before shifting:

								C
1	0	1	0	1	0	1	1	0

After shifting:

(msb)							(lsb)	C
0	1	0	1	0	1	0	1	1

Binary shifts

Shifting Left:

The Most Significant Bit (MSB) shifts to the carry and the LSB is occupied by a zero.

Before shifting:

C	(msb)							(lsb)
0	1	0	1	0	1	0	1	1

After shifting:

C	(msb)							(lsb)
1	0	1	0	1	0	1	1	0

Binary shifts

- Shifting 132 to the right:
- The decimal equivalent of the number obtained when the binary equivalent of 132 is shifted to the right is: 66.
- Therefore, shifting a number to the right is equivalent to dividing a number by 2.

Before shifting (denary number 132):

(msb)							(lsb)	C
1	0	0	0	0	1	0	0	0

After shifting to the right (denary number 66):

(msb)							(lsb)	C
0	1	0	0	0	0	1	0	0

Binary shifts

- Shifting the number 132 to the left.
- This binary number is equivalent to denary number 264.
- Therefore, shifting a number to the left is equivalent to multiplying a number by 2.

Before shifting (denary number 132):

C								
0	1	0	0	0	0	1	0	0

←

After shifting to the left (denary number 264):

C	(msb)							(lsb)
1	0	0	0	0	1	0	0	0

Binary shifts

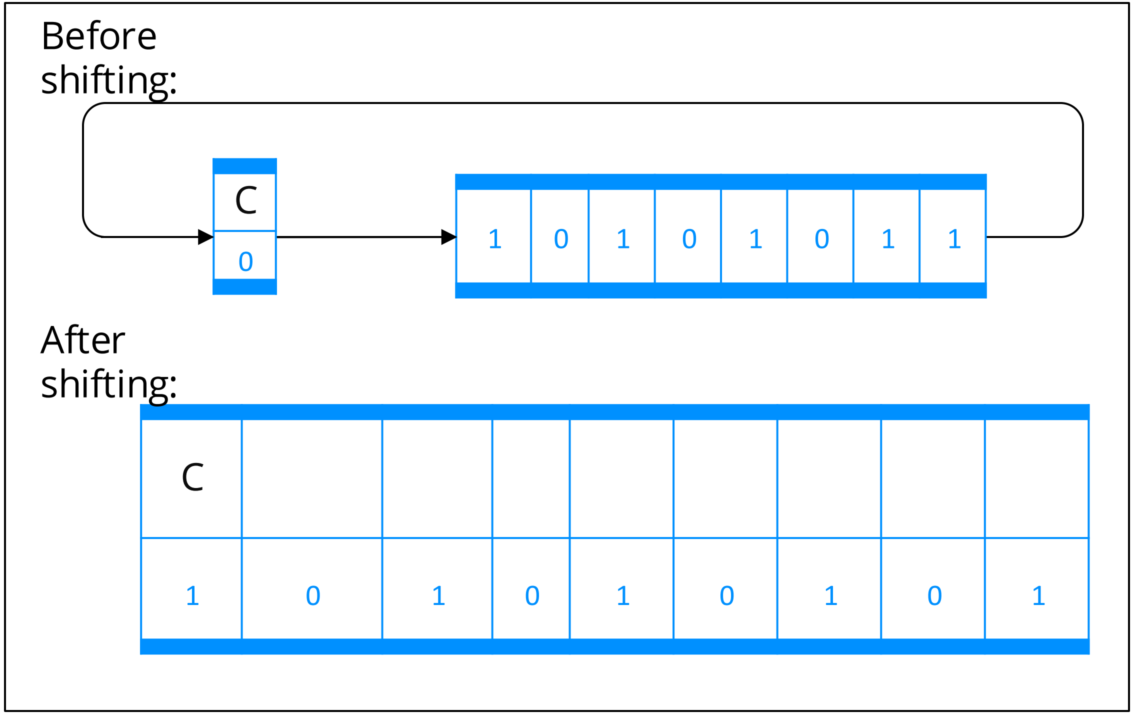
- It can be summarised that when a denary number is multiplied by 2, its binary equivalent shifts left by 1 place.
- Also, multiplication by 4 results in a shift of the binary equivalent by 2 places.
- Multiplication by 8 results in a shift of the binary equivalent by 3 places, and so on.

Binary shifts

- Sometimes, multiple byte operations are performed.
- For example: Dividing a 16-bit number by 2.
- Some processors only support an 8-bit register. In order to hold a 16-bit number, we require two 8-bit registers.
- Shifting operations in such processors are done in a circular form.

Binary shifts

- Circular right shift



Binary shifts

- Consider that a number is stored in two 8-bit registers, A and B with the upper half in register A.

Reg. A							
1	1	1	0	1	0	1	0

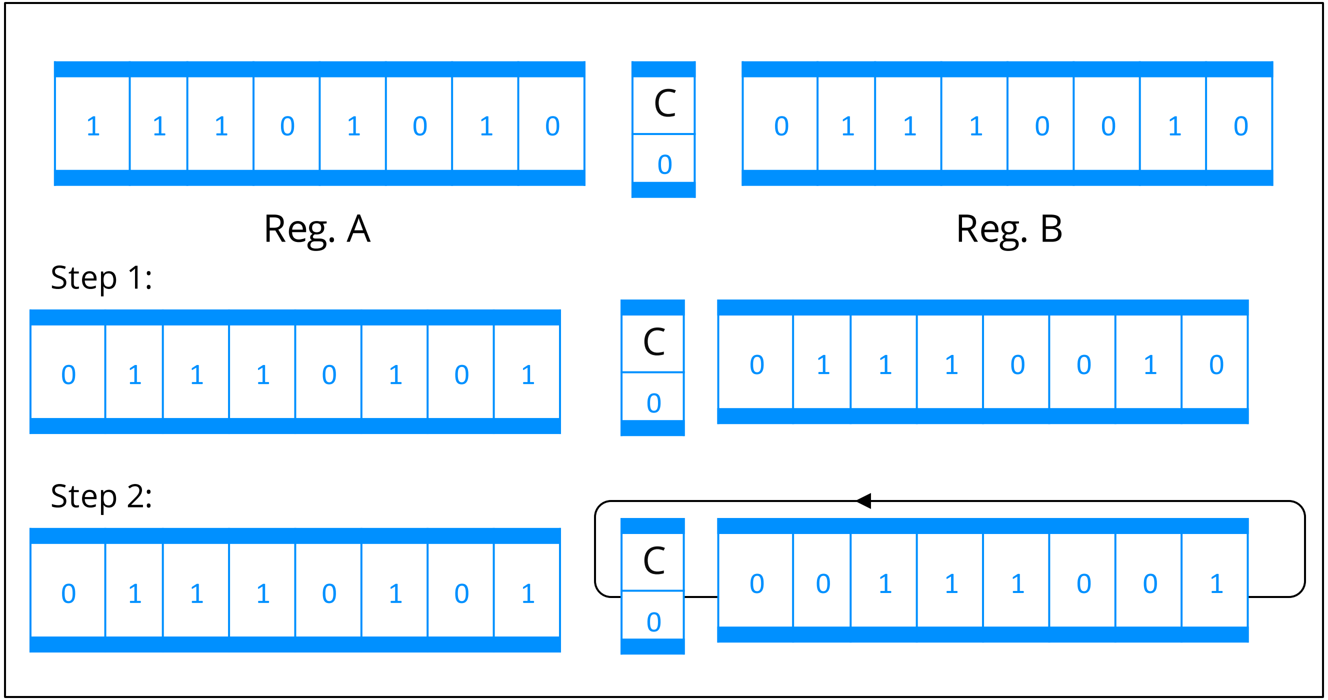
C
0

Reg. B							
0	1	1	1	0	0	1	0

Binary shifts

Dividing 16-bit number by 2:

- Step 1: Shifting the contents of Reg. A to the right.
- Step 2: Performing a circular right shift in reg. B.



Bitwise operations

The logic operations are:

- NOT: Complements the binary value
- AND: Produces output ‘1’ only when both the inputs are ‘1’
- OR: Produces output ‘1’ when at least one of the input is ‘1’
- XOR: Produces output ‘1’ when both the outputs are different, otherwise produces ‘0’

Bitwise operations

A	1	1	0	1
Not A	0	0	1	0

A	1	1	0	1
B	0	1	1	1
A and B	0	1	0	1

A	1	1	0	1
B	0	1	1	1
A or B	1	1	1	1

A	1	1	0	1
B	0	1	1	1
A xor B	1	0	1	0

Bitwise operations

These logical operations are used to manipulate bits in a number.

- For example:
- An OR function is used to convert some bits to ‘1’ without affecting the other bits.
- Similarly, the AND function is used to convert some bits to ‘0’ without affecting the other bits.
- The XOR function is used to invert selected bits.

This concept of manipulating bits, that is, setting selected bits true or false is called masking.

Bitwise operations

Consider a 16-bit register that holds two bytes. To set the second byte to zero, the AND function is used.

	First byte								Second byte							
A	1	1	1	0	1	0	1	0	0	1	1	1	0	0	1	0
B	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
A and B	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0

Bitwise operations

To complement the first byte for a value stored in 16-bit register, the XOR function is used.

	First byte								Second byte							
A	1	1	1	0	1	0	1	0	0	1	1	1	0	0	1	0
B	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
A xor B	0	0	0	1	0	1	0	0	0	1	1	1	0	0	1	0

Bitwise operations

To set the first 4 bits of a number to ‘1’, the OR function is used.

	First byte								Second byte							
A	1	0	1	0	1	0	1	0	0	1	1	1	0	0	1	0
B	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
A or B	1	1	1	1	1	0	1	0	0	1	1	1	0	0	1	0

Representing negative numbers

- Consider the 8-bit binary number 10001101.

	Sign bit	Magnitude						
Binary	1	0	0	0	1	1	0	1
Denary	-13							

- The smallest number that can be represented using 8 bits is 11111111 (-127) and the largest number is 01111111 (+127).
- Similarly, the signed number can be represented in 32-bits, 64-bits and so on.

Finding two's complement

- Finding two's complement is an alternate method to represent negative numbers. This method is used by most computers to perform mathematical operations.
- Let us consider an example of representing -5. The binary value of 5 is 101. The leftmost bit is added to represent the positive sign. +5 is 0101.
- Each bit is inverted and, hence, the 0101 becomes 1010. 1 is added to this number. 1010 + 1=1011.

Finding two's complement

Some examples

Sign bit (-2^3)	(2^3)	(2^2)	(2^1)	Denary number
1	0	1	1	-5
0	1	1	1	7
1	0	0	1	-7
1	1	0	1	-3

Subtracting numbers: Using two’s complement

- Let us consider adding -4 and 3 using two’s complement.
- Converting the sum 1111 into denary number,

  -8+4+2+1=-1

	Sign bit (-2^3)	(2^3)	(2^2)	(2^1)
Carry				
Number 1 = -4	1	1	0	0
Number 2 = 3	0	0	1	1
Sum	1	1	1	1

Subtracting numbers: Using two’s complement

Another method to convert a negative number in 2’s complement to denary is:

- Invert the bits and add 1.
- Convert the obtained number to denary.
- For example: 1111→ (0000) & 0000 +1 → - 0001 → -1

Activity-3

Duration: 15 minutes

1. Convert the denary numbers -15 and 11 to 8-bit signed numbers. Show your working in the box below.
2. Add the two binary numbers obtained in question 1. Show your working in the box below.
3. Find the denary equivalent of the sum obtained in question 2. Show your working in the box below.

Binary Coded Decimal (BCD)

- In some applications, single denary digits are stored and transmitted. In such cases, Binary Coded Decimal (BCD) is used.
- In BCD, four bits are used to represent a denary number.
- If there are more than two digits in a denary number, then there are two possible solutions:
  - One BCD digit per byte: In this method, the 4 least significant bits of a byte are used to store the denary digit. The other 4 bits remain unused.
  - Packed BCD: In this method, two denary digits are grouped together in a byte.

Binary Coded Decimal (BCD)

- Representing the denary digits 9276 in these two methods,

	9	2	7	6
One BCD digit per byte	00001001	00000010	00000111	00000110
Packed BCD	1001 0010		0111 0110	

Binary Coded Decimal (BCD)

- Consider two fixed-point numbers 0.25 and 0.75 represented using BCD, the first byte represents the whole number part, and the second byte represents the two digits of the fractional part.
- Adding the numbers, we get a wrong answer.
- The result represents the number point nine ten, which has no meaning.

	whole number	fractional part	
0.25	00000000	0010 0101	
0.75	00000000	0111 0101	+
error	00000000	1001 1010	

- To rectify this, an additional step is included when adding numbers in BCD format.
- The remedy is to add 0110 (denary number 6) whenever an invalid BCD number is obtained. In a nibble, numbers 0-9 are valid in BCD and the 10-15 are invalid.
- Using this method, the correct result of 1.00 is obtained.

0.25	00000000	0010 0101	
0.75	00000000	0111 0101	+
	00000000	1001 1010	
Add 0110 to LSB		0110	
		1 0000	
	00000000	1001 0000	
Add 0110 + carry to next nibble		0111 0000	+
		1 0000 0000	
Add carry to the next nibble	0000 0001	0000 0000	

Let’s review some concepts

Binary number system

The place values have a base 2.

Binary combinations

An n-bit system has 2^n binary combinations.

Computer memory

A binary digit is referred to as a bit.

A nibble consists of 4 bits.

A byte consists of 8 bits.

Adding binary numbers

0+0=0

1+0=1

1+1=10 (1 is carried over)

Converting denary to binary

Divide the number by 2 and write down the remainder.
Keep dividing the quotient by 2 and write down the remainders.
List the remainders in reverse order.

Overflow error

The number of bits a register can hold is called the word size.
Exceeding the capacity of the word size in a register results in an overflow error.

Let’s review some concepts

Binary shifts

Shifting left: Multiplying by 2

Shifting right: Dividing by 2

Bitwise operations

NOT, OR, AND & XOR

Masking:

OR: set selected bits to ‘1’

AND: set selected bits to ‘0’

XOR: invert selected bits

Signed numbers

An extra bit is used to represent the sign of a number in binary representation.

2’s complement

Add 0 to the leftmost bit. Invert the digits of a positive binary number, add 1 to it

Used for subtracting numbers

Binary Coded Decimal

In a nibble, numbers 0-9 are valid in BCD and the 10-15 are invalid.

When a nibble is greater than 9, add 6 (0110) to it to convert to BCD.

3.

End of topic questions

End of topic questions

1. Convert the following denary numbers to binary.
   a) 13
   b) 52
   c) 145
2. Convert the following binary numbers to denary.
   a) 1010
   b) 111000
   c) 11110111
3. How many megabytes are there in 3 terabytes?

End of topic questions

4. Add the following binary numbers. Show the necessary working.
   a) 1011 + 1001
   b) 1011 0110 + 1010 0011
5. An 8-bit register holds the binary value 1110 0101.
   a) What is the contents of this register after a left shift of 2 bits?
   b) What is the contents of this register after a right shift of 2 bits?

End of topic questions

6. Register A is an 8-bit register with MSB as bit 7 and LSB as bit 0. What mask and logical operator will you use for the following operations?
   a) Complementing bit 3 and bit 6?
   b) Setting bit 0, bit 1, bit 2 and bit 3 to 1?

7. This question is about BCD addition.
   a) How are the numbers 1.28 and 3.74 represented in BCD?
   b) Show how the two numbers in answer (a) are added in BCD format?