Parul® University

# Fundamental of Programming

**Prof. Digvijaysinh Mahida,** Assistant Professor
Computer Science & Engineering

# CHAPTER-1

# Number system

# Common number system

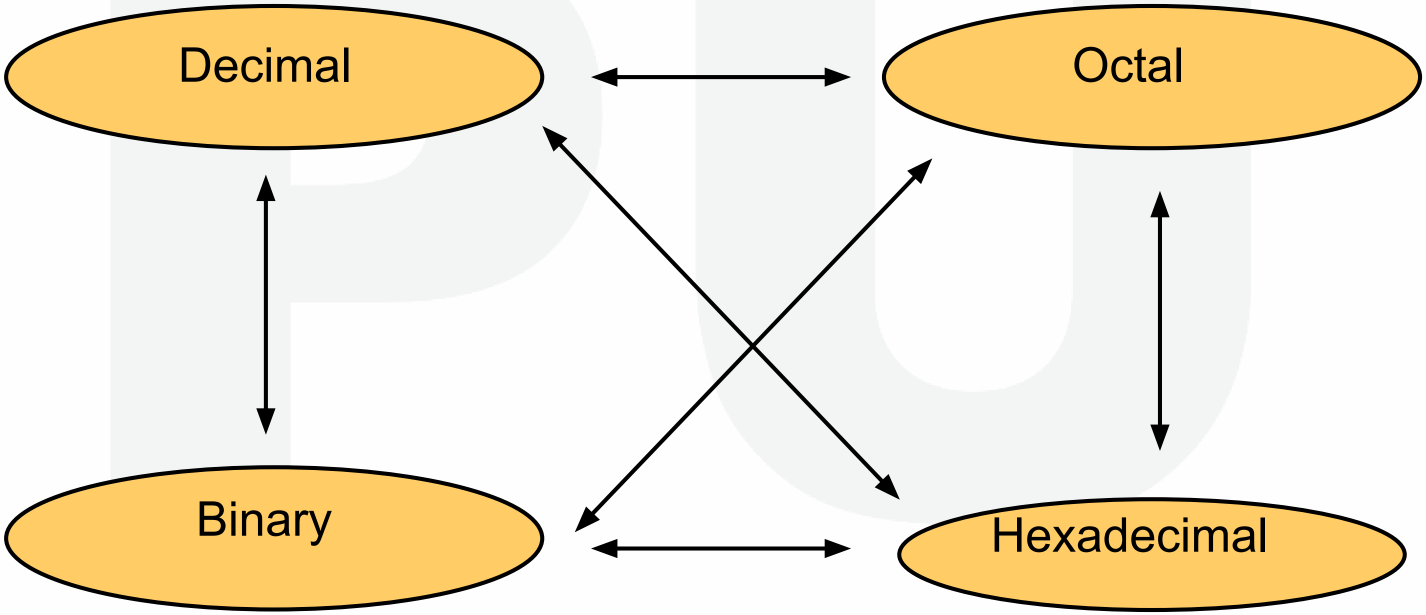
| System | Base | Symbols | Used by humans? | Used in computers? |
|---|---|---|---|---|
| Decimal | 10 | 0, 1, … 9 | Yes | No |
| Binary | 2 | 0, 1 | No | Yes |
| Octal | 8 | 0, 1, … 7 | No | No |
| Hexadecimal | 16 | 0, 1, … 9, A, B, … F | No | No |

# Counting

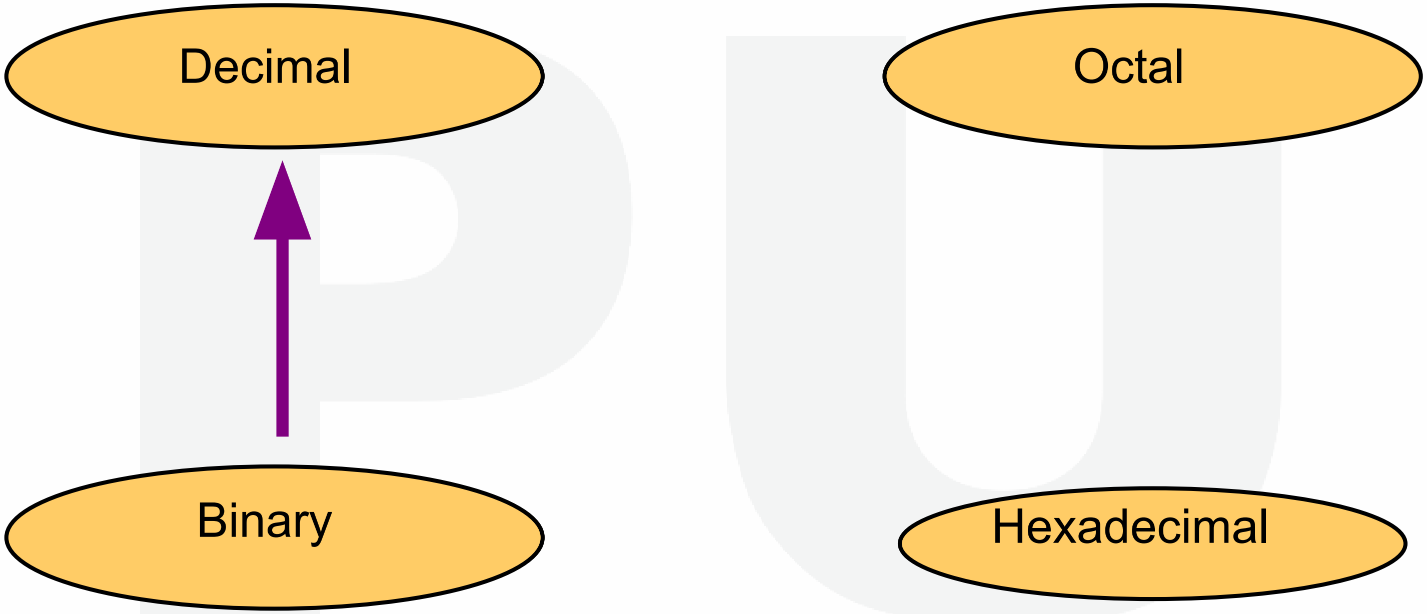
| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 4 | 4 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |

# Conversion Among Bases

- The possible conversion



# Binary to Decimal



# Technique for conversion

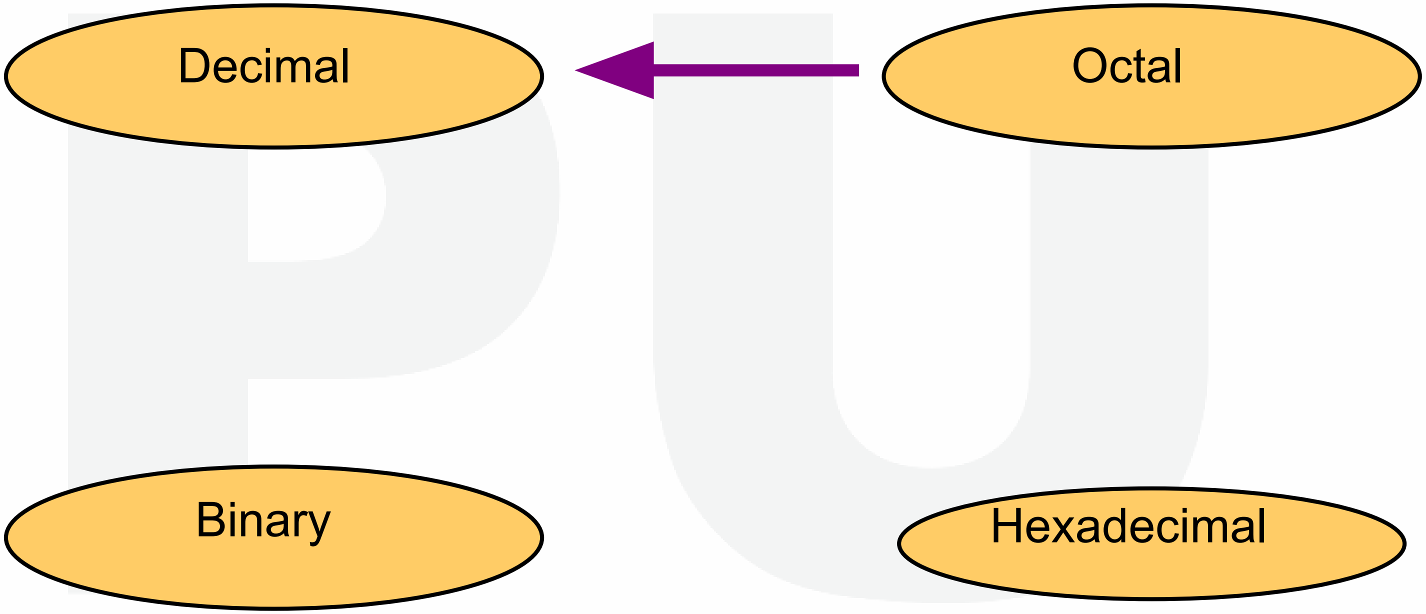
- Multiply each bit by $2^n$, where $n$ is the “weight” of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

# Example

.

$$
\begin{array}{lrcllr}
101011_2 \;\; => & 1 & \text{x} & 2^0 & = & 1 \\
 & 1 & \text{x} & 2^1 & = & 2 \\
 & 0 & \text{x} & 2^2 & = & 0 \\
 & 1 & \text{x} & 2^3 & = & 8 \\
 & 0 & \text{x} & 2^4 & = & 0 \\
 & 1 & \text{x} & 2^5 & = & 32 \\
 & & & & & 43_{10}
\end{array}
$$

# Octal to Decimal



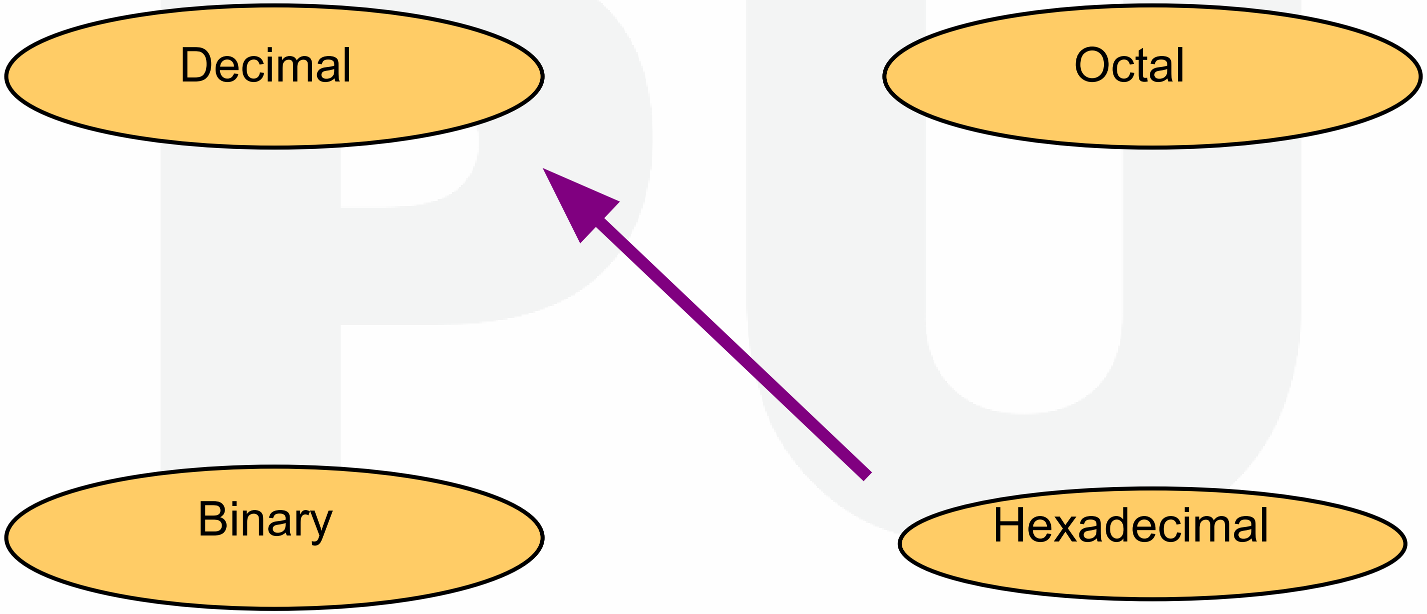
# Technique for conversion

- Multiply each bit by $8^n$, where *n* is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

# Example

$$
\begin{array}{llcl}
724_8 \Rightarrow & 4 \times 8^0 & = & 4 \\
& 2 \times 8^1 & = & 16 \\
& 7 \times 8^2 & = & 448 \\
& & & 468_{10}
\end{array}
$$

# Hexadecimal to Decimal



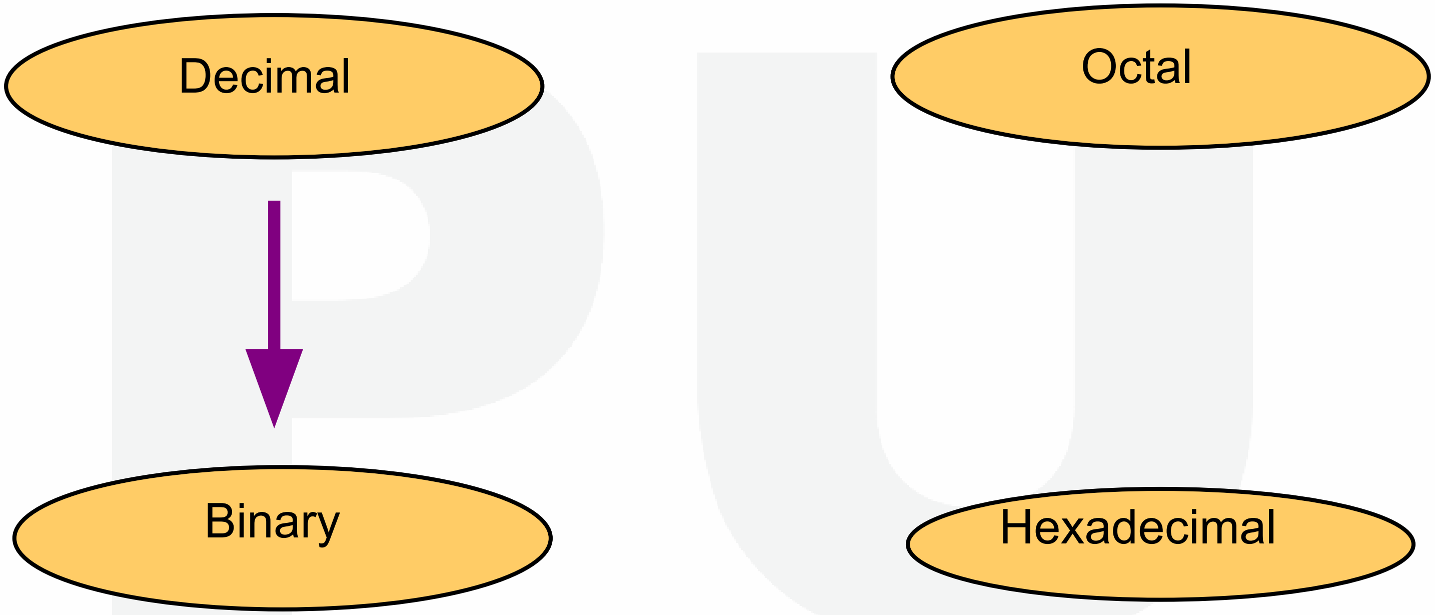
# Technique for conversion

- Multiply each bit by $16^n$, where $n$ is the “weight” of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

# Example

$$\begin{array}{lllllllllll} ABC_{16} & \Rightarrow & C & \times & 16^{0} & = & 12 & \times & 1 & = & 12 \\ & & B & \times & 16^{1} & = & 11 & \times & 16 & = & 176 \\ & & A & \times & 16^{2} & = & 10 & \times & 256 & = & 2560 \\ & & & & & & & & & & 2748_{10} \end{array}$$

# Decimal to Binary



Decimal

Octal

Binary

Hexadecimal

# Technique for conversion

- Divide by two, keep track of the remainder
- First remainder is bit 0 (LSB, least-significant bit)
- Second remainder is bit 1
- Etc.

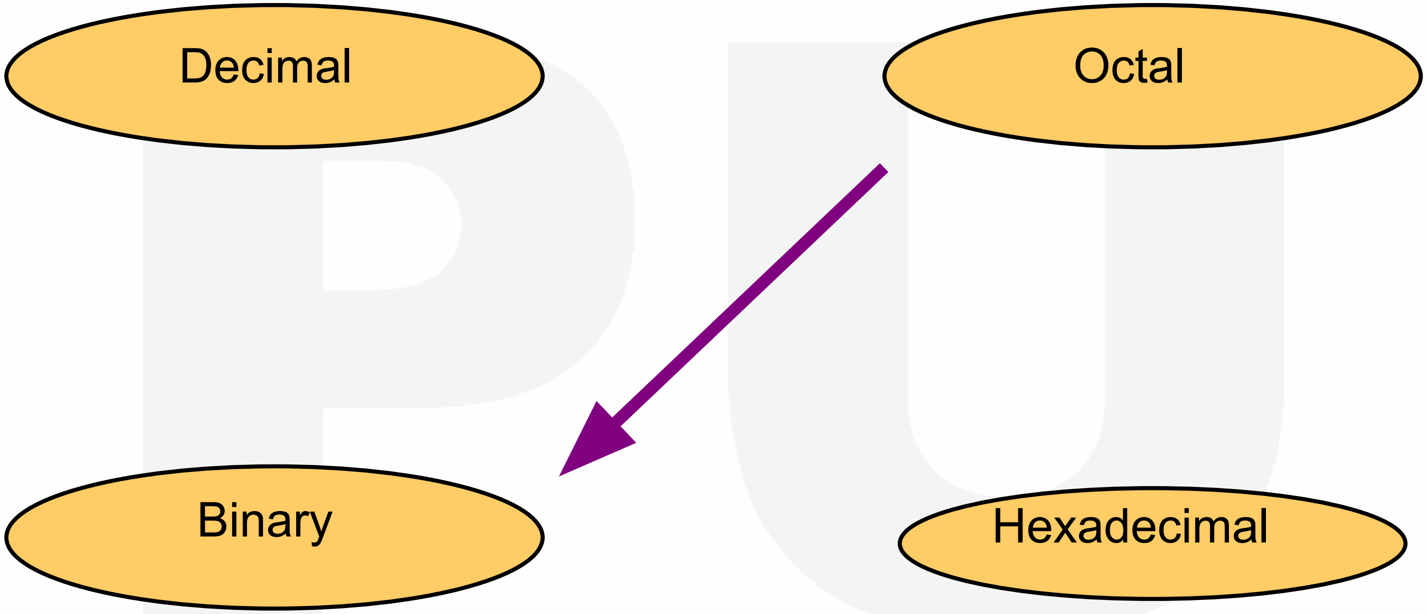
# Example

$125_{10} = ?$

| | | |
|---|---|---|
| 2 | 125 | |
| 2 | 62 | 1 |
| 2 | 31 | 0 |
| 2 | 15 | 1 |
| 2 | 7 | 1 |
| 2 | 3 | 1 |
| 2 | 1 | 1 |
| | 0 | 1 |

$125_{10} = 1111101_2$

# Octal to Binary



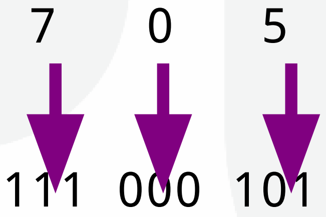
# Technique for conversion

- Convert each octal digit to a 3-bit equivalent binary representation

macOS Source: Google

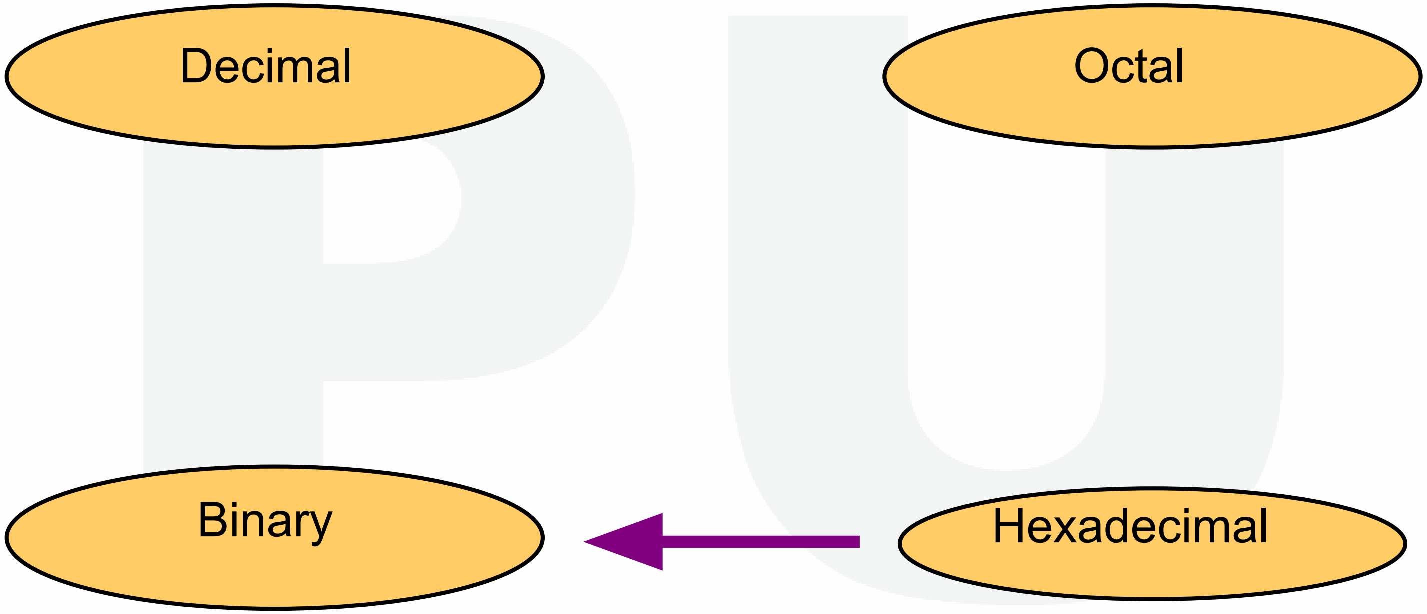
# Example

$705_8 = ?_2$



$705_8 = 111000101_2$

# Hexadecimal to Binary

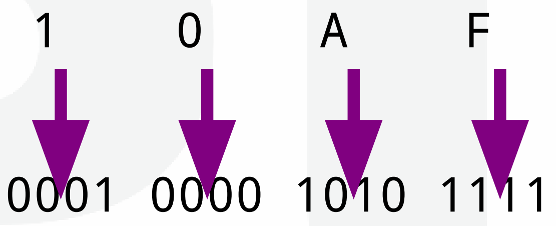


# Technique for conversion

- Convert each hexadecimal digit to a 4-bit equivalent binary representation

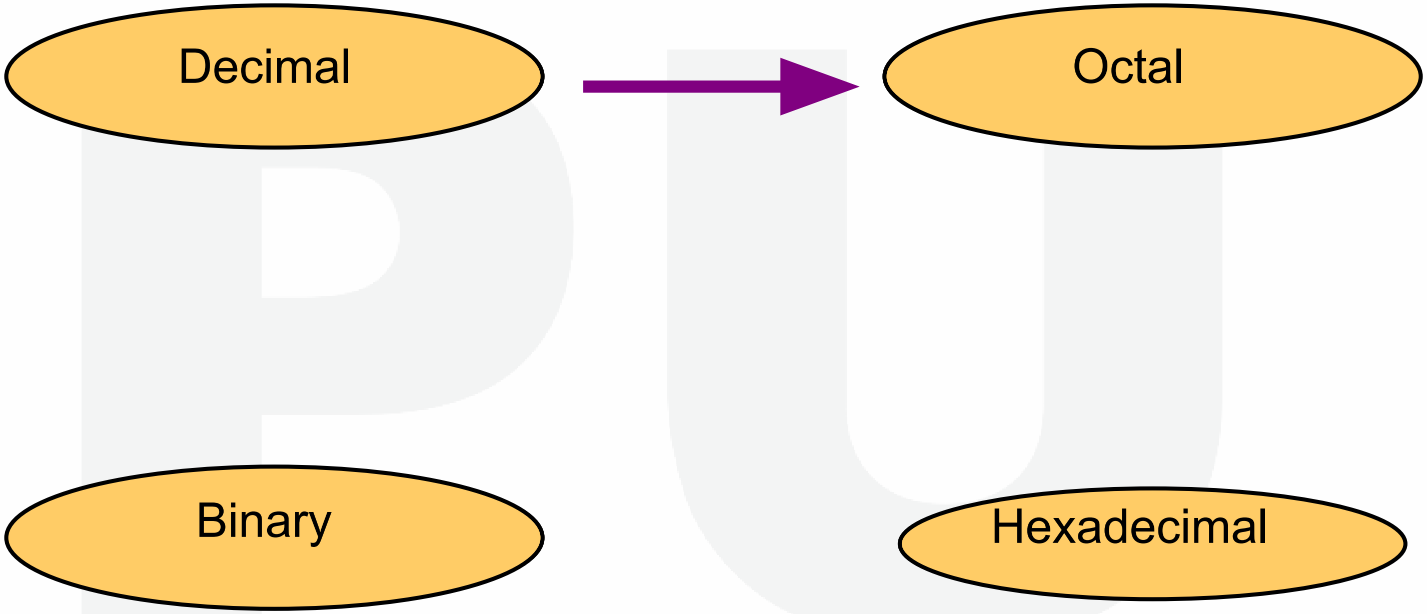
# Example

$10AF_{16} = ?_2$



$10AF_{16} = 0001000010101111_2$

# Decimal to Octal

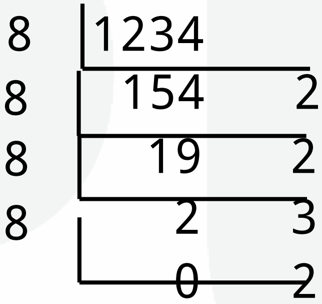


# Technique of conversion

- Divide by 8
- Keep track of the remainder

# Example

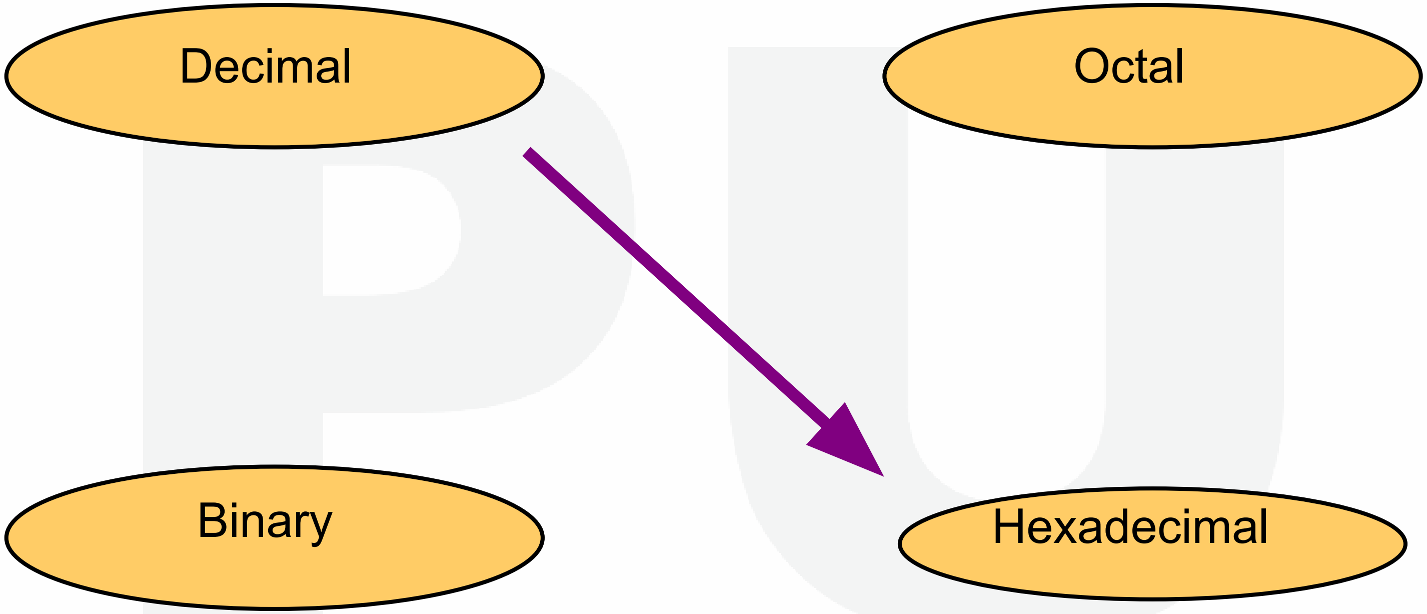
$1234_{10} = ?_8$



$$\begin{array}{r|rr} 8 & 1234 & \\ \hline 8 & 154 & 2 \\ \hline 8 & 19 & 2 \\ \hline 8 & 2 & 3 \\ & 0 & 2 \end{array}$$

$1234_{10} = 2322_8$

# Decimal to Hexadecimal



# Technique of conversion

- Divide by 16
- Keep track of the remainder

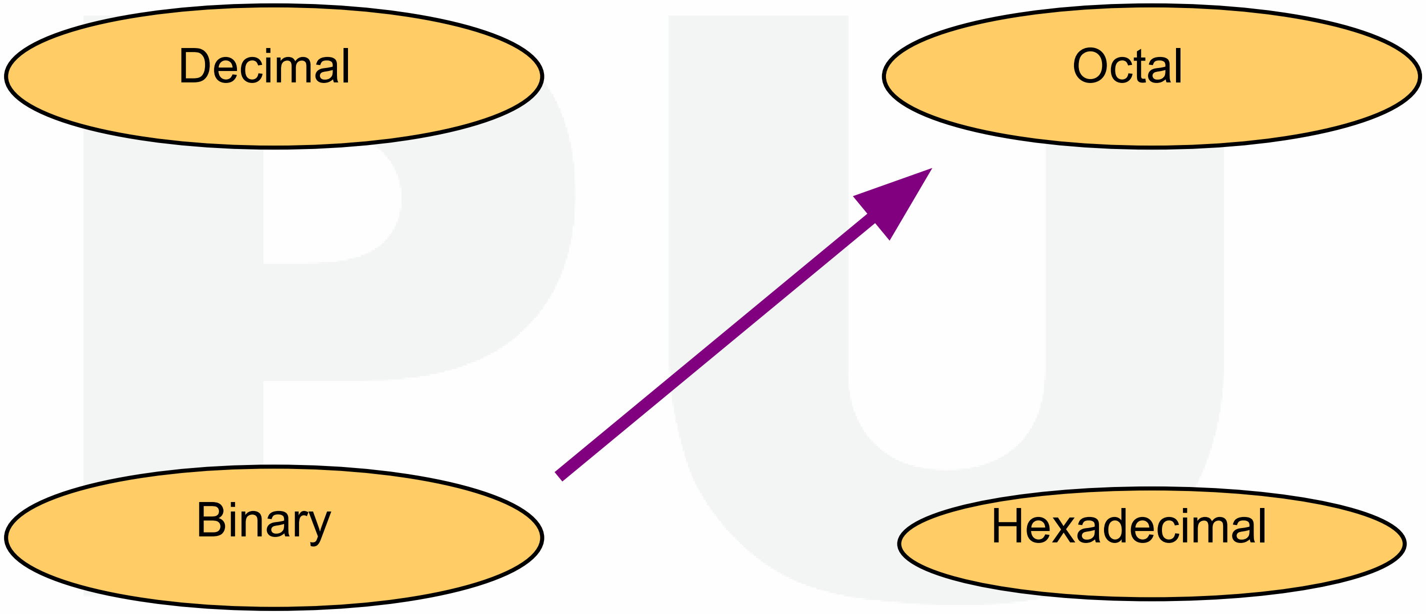
# Example

$1234_{10} = ?_{16}$

|  |  |  |
|---|---|---|
| 16 | 1234 |  |
| 16 | 77 | 2 |
| 16 | 4 | 13 = D |
|  | 0 | 4 |

$1234_{10} = 4D2_{16}$

# Binary to Octal

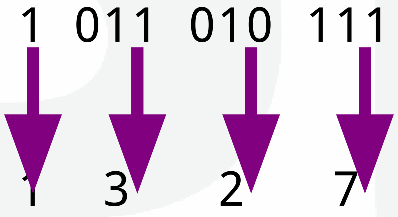


# Technique for conversion

- Group bits in threes, starting on right
- Convert to octal digits

# Example

$1011010111_2 = ?_8$

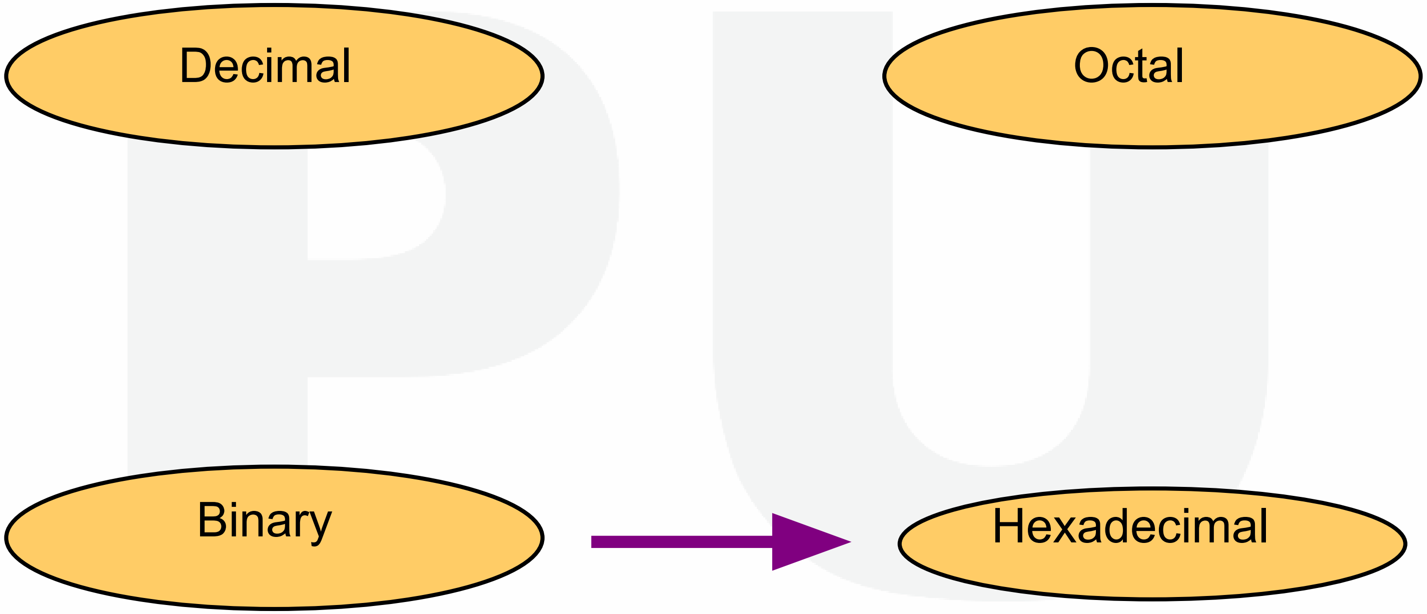


1 011 010 111

1 3 2 7

$1011010111_2 = 1327_8$

# Binary to Hexadecimal

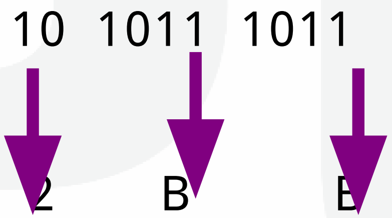


# Technique for conversion

- Group bits in fours, starting on right
- Convert to hexadecimal digits

# Example

$1010111011_2 = ?_{16}$

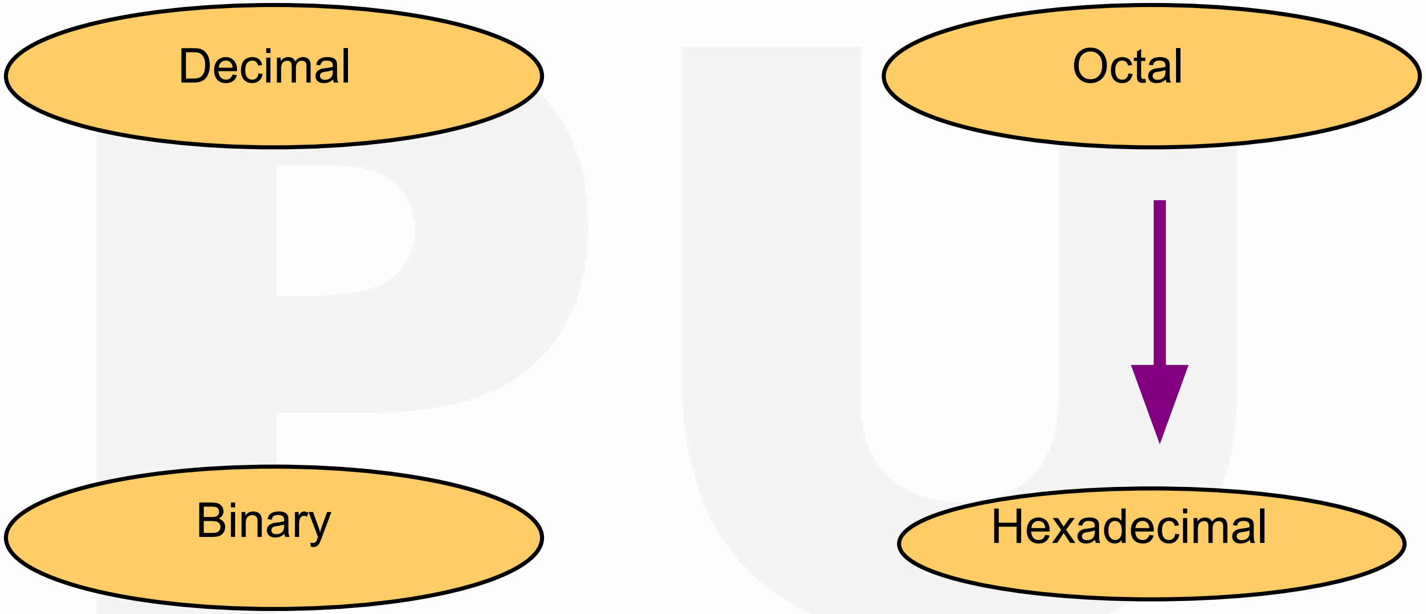


10 1011 1011

2 B B

$1010111011_2 = 2BB_{16}$

# Octal to Hexadecimal



Decimal

Octal

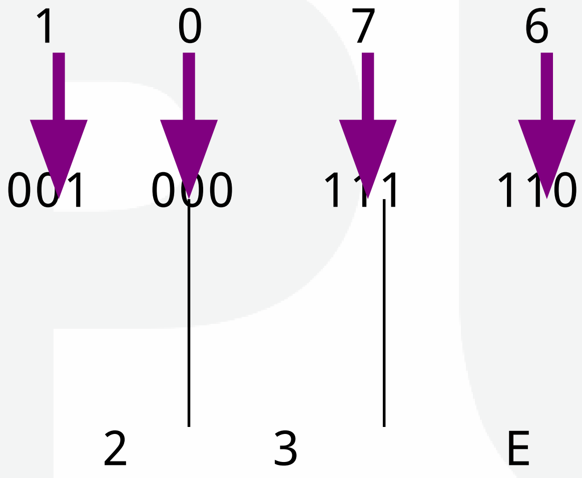
Binary

Hexadecimal

# Technique for conversion

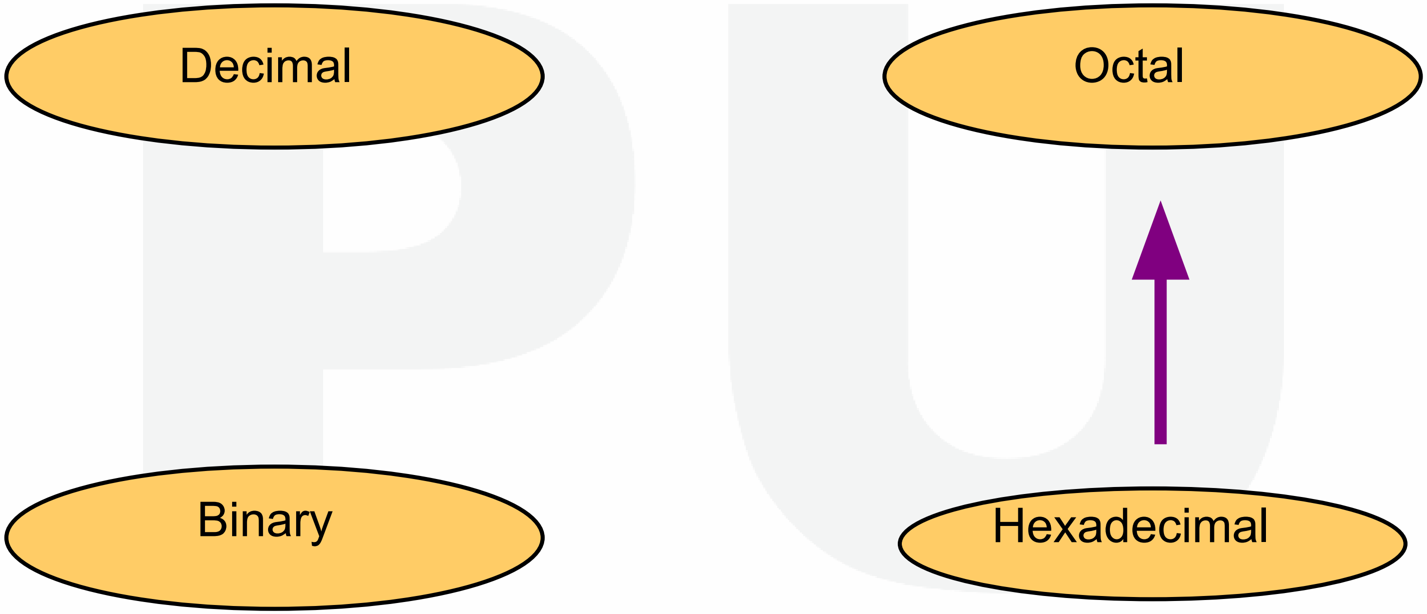
- Use binary as an intermediary

$1076_8 = ?_{16}$



$1076_8 = 23E_{16}$

# Hexadecimal to Octal

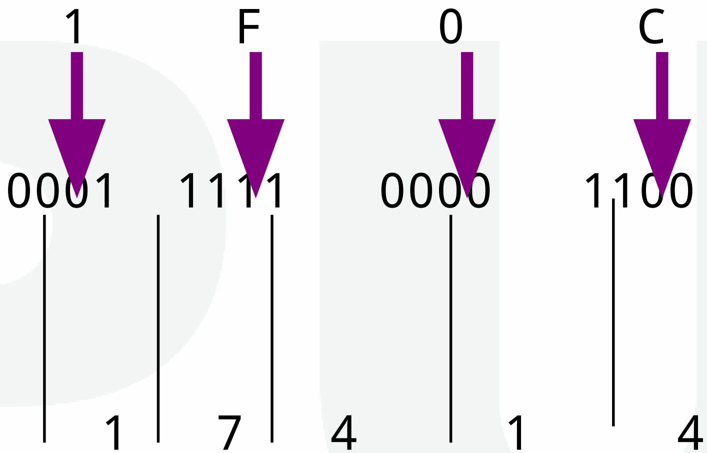


# Technique for conversion

- Use binary as an intermediary

# Example

$1F0C_{16} = ?_8$



$1F0C_{16} = 17414_8$

# Binary Addition

- Two 1-bit values

| A | B | A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 10 |

# Binary Addition

- Two *n*-bit values
  - Add individual bits
  - Propagate carries
  - E.g.,

$$\begin{array}{rrcrr} & 10101 & \qquad & & 21 \\ + & 11001 & & + & 25 \\ & 101110 & & & 46 \end{array}$$

# Multiplication

- Binary, two 1-bit values

| A | B | A × B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Example of multiplication

- Binary, two *n*-bit values
  - As with decimal values
  - E.g.,

```
    1110
x   1011
    1110
   1110
  0000
 1110
10011010
```

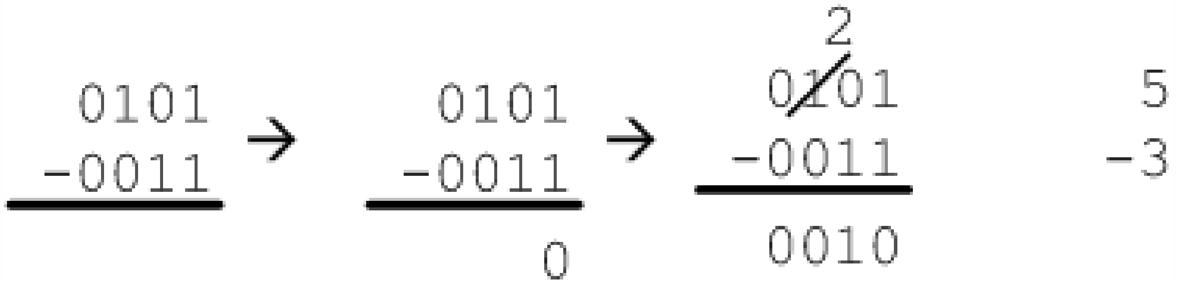
# Binary Subtraction

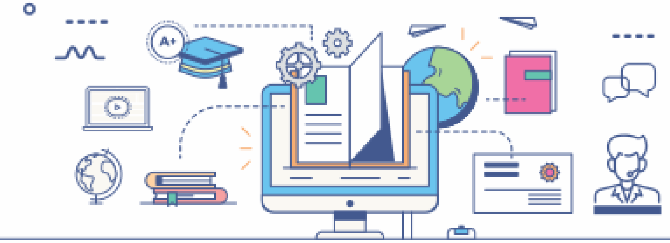
Have previously looked at the subtraction operation.  A quick review.

Just like subtraction in any other base

| | |
|---|---|
| Minuend | 10110 |
| Subtrahend | - 10010 |
| Difference | 00100 |

And when a borrow is needed.  Note that the borrow gives us 2 in the current bit position.





# Binary Division

Binary division is also performed in the same way as we perform decimal division. Like decimal division, we also need to follow the binary subtraction rules while performing the binary division. The dividend involved in binary division should be greater than the divisor. The following are the two important points, which need to be remembered while performing the binary division.

- If the remainder obtained by the division process is greater than or equal to the divisor, put 1 in the quotient and perform the binary subtraction.
- If the remainder obtained by the division process is less than the divisor, put 0 in the quotient and append the next most significant digit from the dividend the remainder

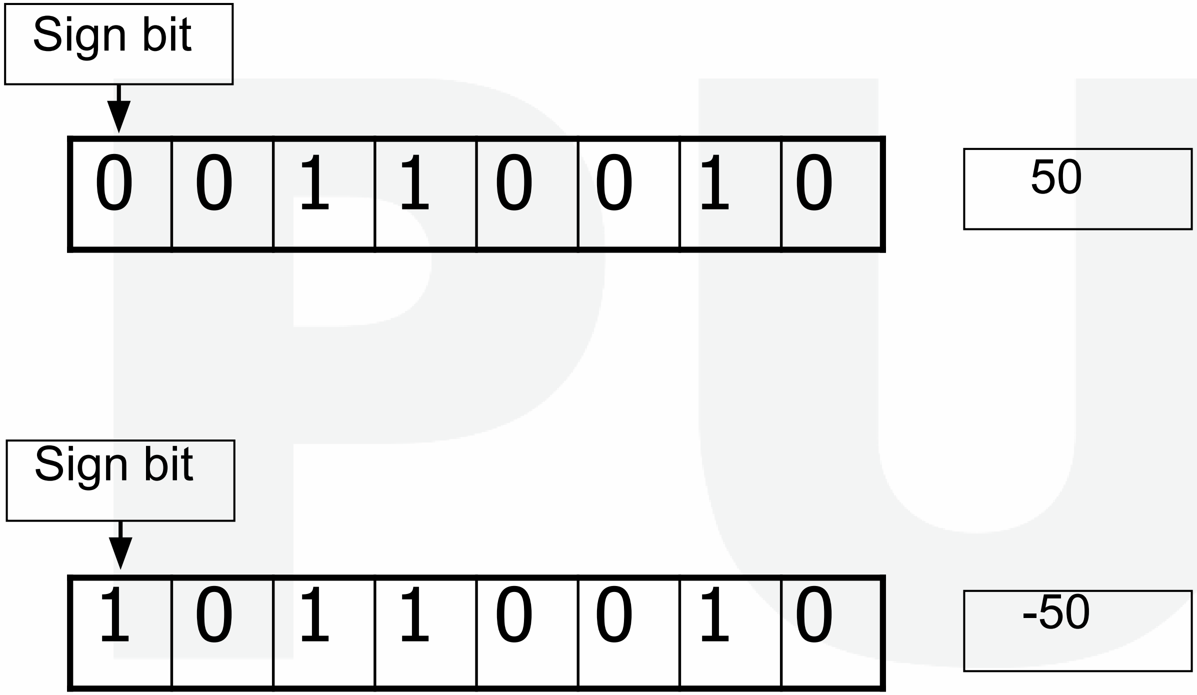
# Binary Division

Perform the binary division of the decimal numbers 18 and 8.

The equivalent binary representation of the decimal number 18 is 10010.
The equivalent binary representation of the decimal number 8 is 1000.

$$
\begin{array}{r@{\ }l}
1\,0\,0\,0\;) & 1\,0\,0\,1\,0\;(\;1\,0 \longrightarrow \text{Quotient} \\
 & 1\,0\,0\,0 \\ \cline{2-2}
 & 0\,0\,0\,1\,0 \\
 & 0\,0\,0\,0\,0 \\ \cline{2-2}
 & 0\,0\,0\,1\,0 \longrightarrow \text{Remainder}
\end{array}
$$

# Signed /Unsigned Number



Sign bit

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

50

Sign bit

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

-50

# Signed/Unsigned Numbers

Advantages of the signed-magnitude representation:

It is very easy to represent and understand positive as well as negative numbers using this representation.

The binary multiplication and the binary division of the signed binary numbers can be easily performed.

Represent equal number of positive and negative quantities that makes it a very symmetrical method of representation.

# Signed/Unsigned Numbers

1

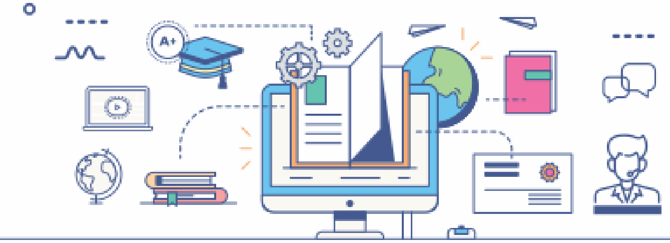
Disadvantages of the signed-magnitude representation:

2

It is not an easy task to perform the binary addition and the binary subtraction using this representation.

3

It provides two different representations of zero, one for plus zero and another for negative zero but actually they are the same values. This could lead to some confusion while performing various arithmetic operations.



# Complements of Binary Numbers

The complement system can also be used to represent the signed binary numbers apart from the signed-magnitude representation method.

In the complement system, the positive integers are represented in a similar manner as they are represented in the signed-magnitude representation. The following are the two most popular complement methods used in the computer system:

One’s complement

Two’s complement

# One’s Complement

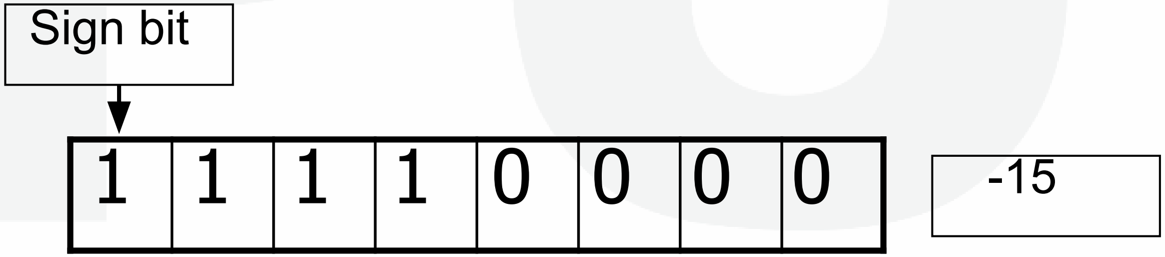
One’s complement method can be used to represent negative binary numbers.

A negative number can be represented using one’s complement method by first computing the binary equivalent of the number and then changing all the zeros with ones and all the ones with zeros.

For example, the binary equivalent of the decimal number 15 is 00001111. Therefore, -15 can be represented using one’s complement method as 11110000.

# One’s Complement

The one’s complement method also uses the left most bit as the sign bit to indicate the sign of the number.



Sign bit

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

-15

# One’s Complement

| Integers | One’s complement representation |
|---|---|
| -7 | 1000 |
| -6 | 1001 |
| -5 | 1010 |
| -4 | 1011 |
| -3 | 1100 |
| -2 | 1101 |
| -1 | 1110 |
| -0 | 1111 |
| +0 | 0000 |
| +1 | 0001 |
| +2 | 0010 |
| +3 | 0011 |
| +4 | 0100 |
| +5 | 0101 |
| +6 | 0110 |
| +7 | 0111 |
| | |

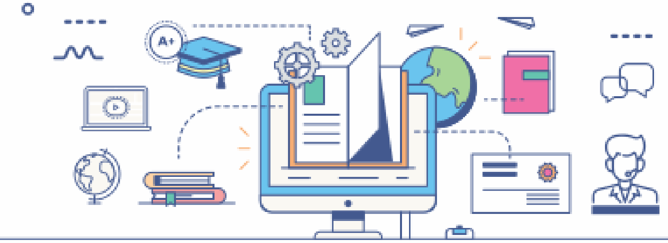
The one’s complement method of representing signed numbers also has two different representations for the number, zero.

# One’s Complement

The equivalent binary representation of 25 is in byte size is 00011001.

Now, change all the zeros to ones and all the ones to zeros in order to obtain the ones complement representation: 11100110

Therefore, the one’s complement representation of -25 is 11100110.



# Two’s Complement

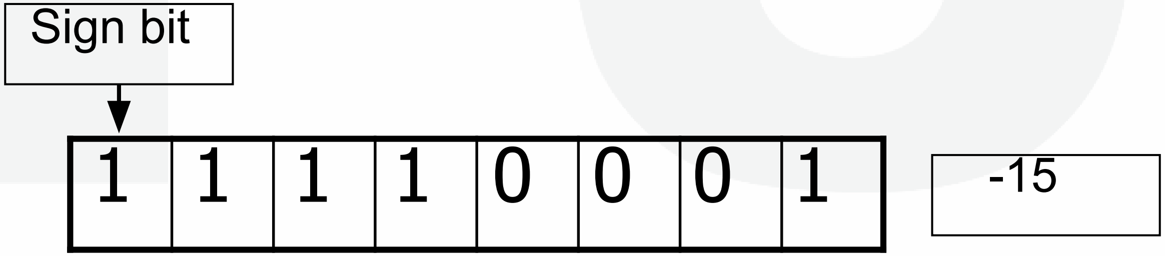
Two’s complement is the most widely used method for representing negative numbers in the computer system.

The two’s complement of the given integer can be obtained by adding 1 to the one’s complement of that number.

For example, the two’s complement representation of -15 can be obtained by adding 1 to 11110000, which is the one’s complement representation of -15. Therefore, the two’s complement representation of -15 is 11110001.

# Two’s Complement

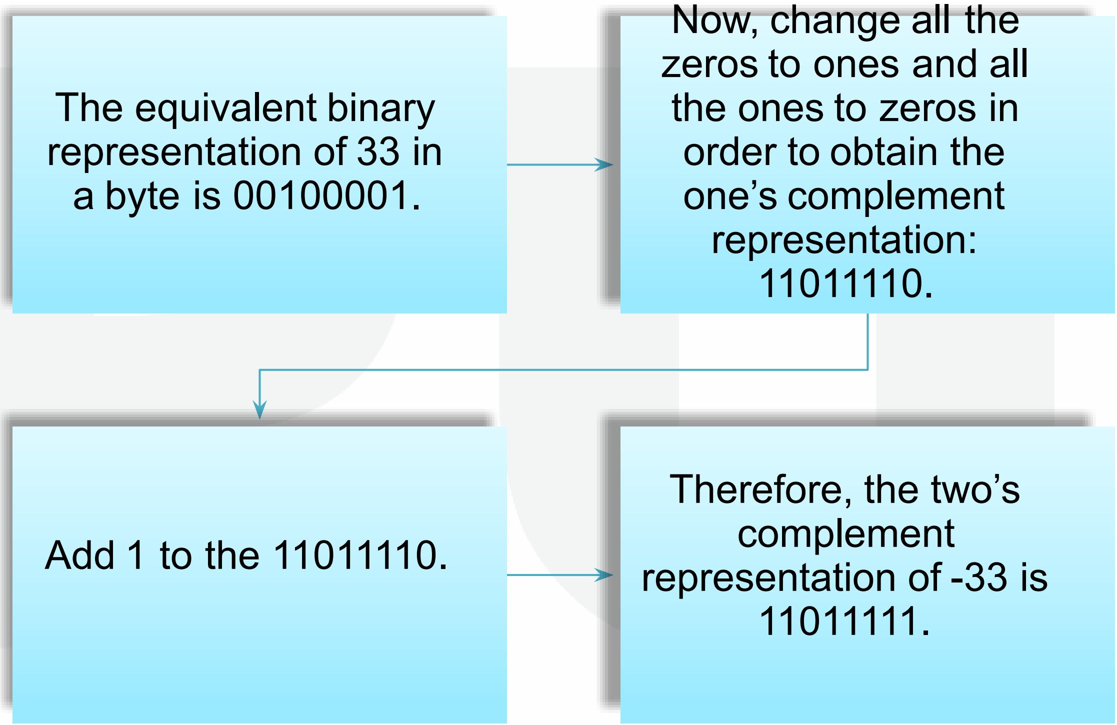
The two’s complement method also uses the left most bit as the sign bit to indicate the sign of the number.



# Two’s Complement

| Integers | Two’s complement representation |
|---|---|
| -7 | 1001 |
| -6 | 1010 |
| -5 | 1011 |
| -4 | 1100 |
| -3 | 1101 |
| -2 | 1110 |
| -1 | 1111 |
| -0 | 0000 |
| +0 | 0000 |
| +1 | 0001 |
| +2 | 0010 |
| +3 | 0011 |
| +4 | 0100 |
| +5 | 0101 |
| +6 | 0110 |
| +7 | 0111 |

# Two’s Complement

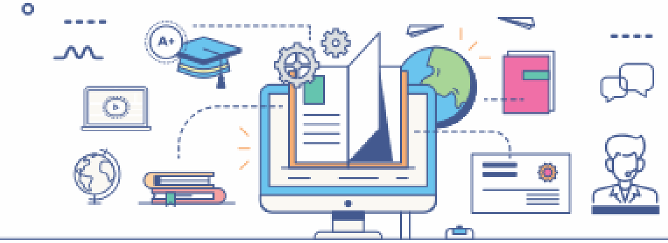


The equivalent binary representation of 33 in a byte is 00100001.

Now, change all the zeros to ones and all the ones to zeros in order to obtain the one’s complement representation: 11011110.

Add 1 to the 11011110.

Therefore, the two’s complement representation of -33 is 11011111.



# Fraction

- Binary to decimal

10.1011 => $1 \times 2^{-4} = 0.0625$

$1 \times 2^{-3} = 0.125$

$0 \times 2^{-2} = 0.0$

$1 \times 2^{-1} = 0.5$

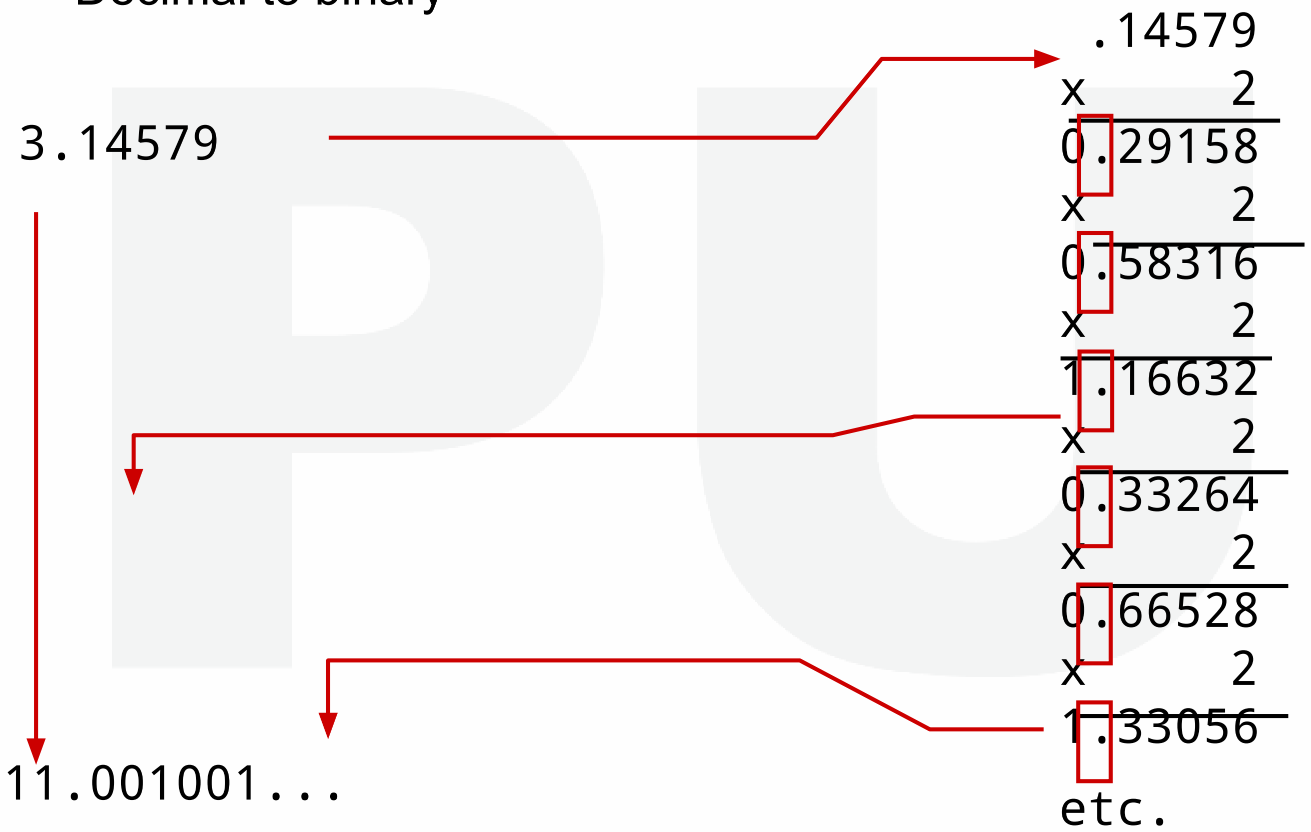
$0 \times 2^{0} = 0.0$

$1 \times 2^{1} = 2.0$

2.6875

# Fraction

- Decimal to binary



3.14579

```
  .14579
x      2
0.29158
x      2
0.58316
x      2
1.16632
x      2
0.33264
x      2
0.66528
x      2
1.33056
etc.
```

11.001001...

# DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in