

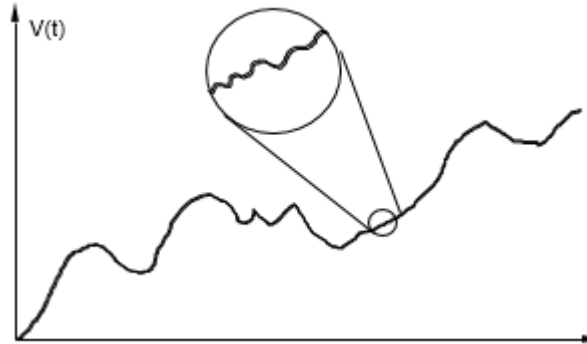
Digital Fundamentals

Computer Engineering
3rd Semester

Fundamentals of Digital systems and Logic families

Analogue Systems

$V(t)$ can have *any value* between its minimum and maximum value



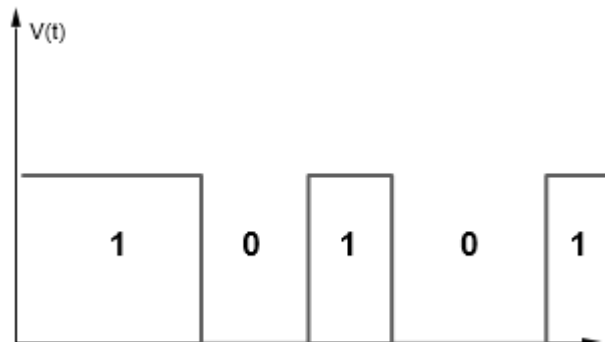
Digital Systems

$V(t)$ must take a value selected from a set of values called an alphabet

Binary digital systems form the basis of almost all hardware systems currently

For example,

Binary Alphabet: 0, 1.

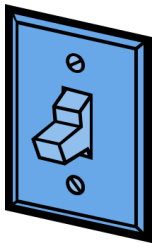


Advantages of Digital Systems

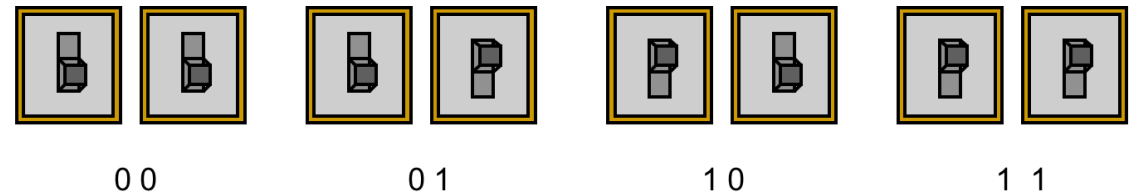
- ☐ Analogue systems: slight error in input yields large error in output
- ☐ Digital systems more accurate and reliable
- ☐ Computers use digital circuits internally
- ☐ Interface circuits (for instance, sensors and actuators) are often analogue

Coding:

- ☐ A single binary input can only have two values: True or False (Yes or No) (1 or 0)



- ☐ More bits = more combinations

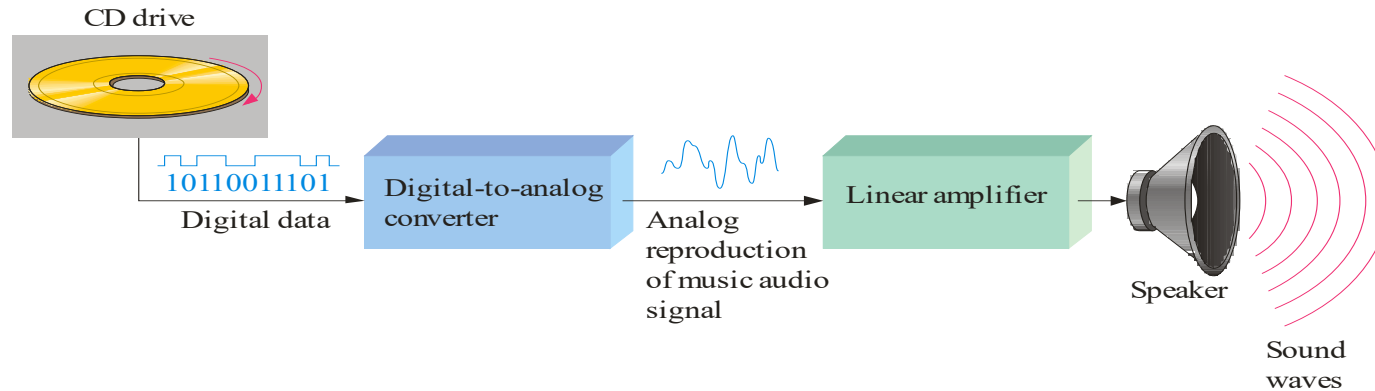


- ☐ Each additional input doubles the number of combinations we can represent
i.e. with n inputs it is possible to represent 2^n combinations



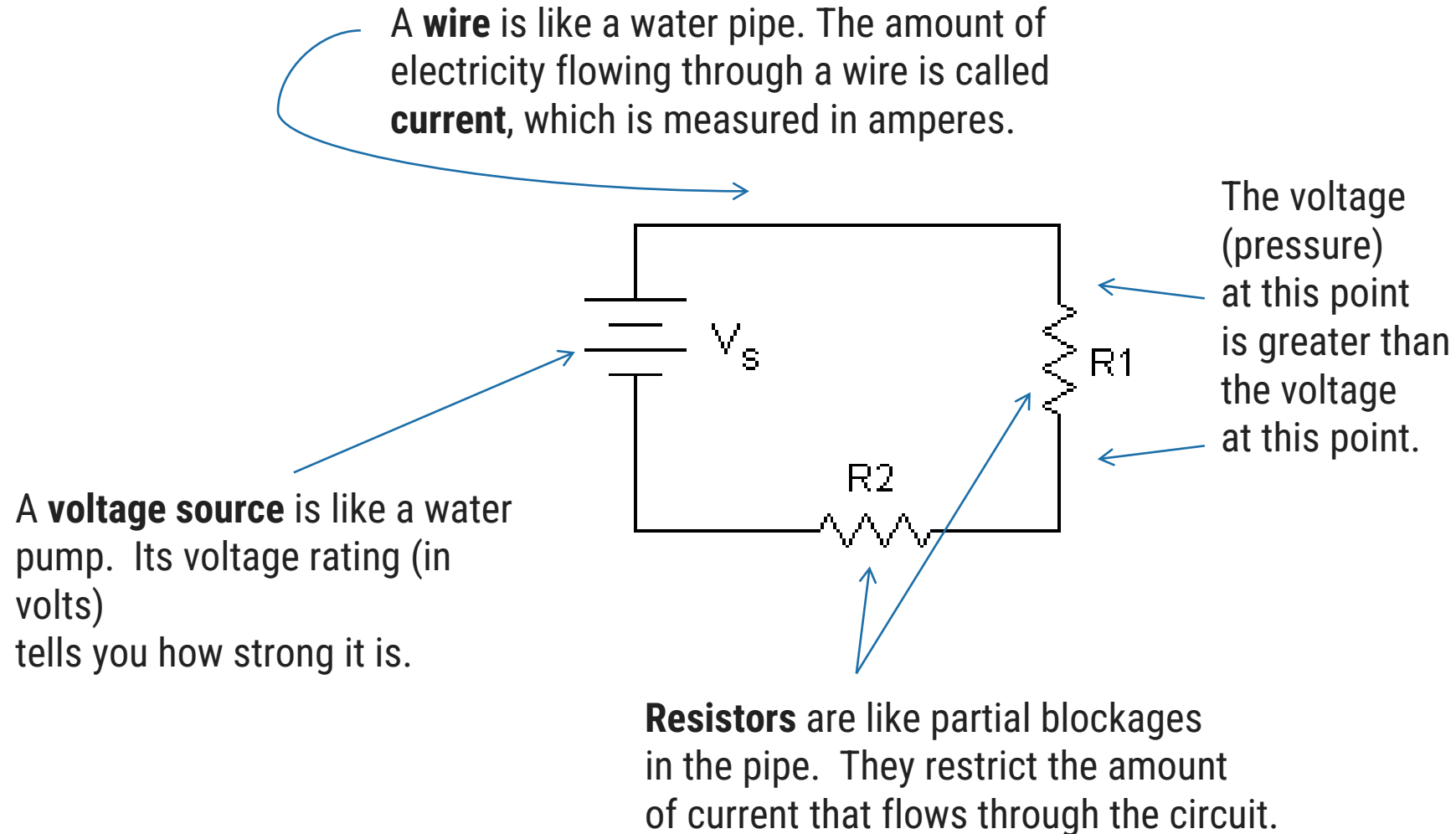
Analog and Digital Systems

- ❑ Many systems use a mix of analog and digital electronics to take advantage of each technology. A typical CD player accepts digital data from the CD drive and converts it to an analog signal for amplification.



- ❑ Voltage is a basic electrical quantity that is important in all circuits (analog or digital).
- ❑ You can think of a circuit as being like a plumbing system, with water flowing through pipes.
- ❑ On this analogy, voltage is like the water pressure in the pipes. Its value will vary at different points in the circuit.

A Simple Circuit



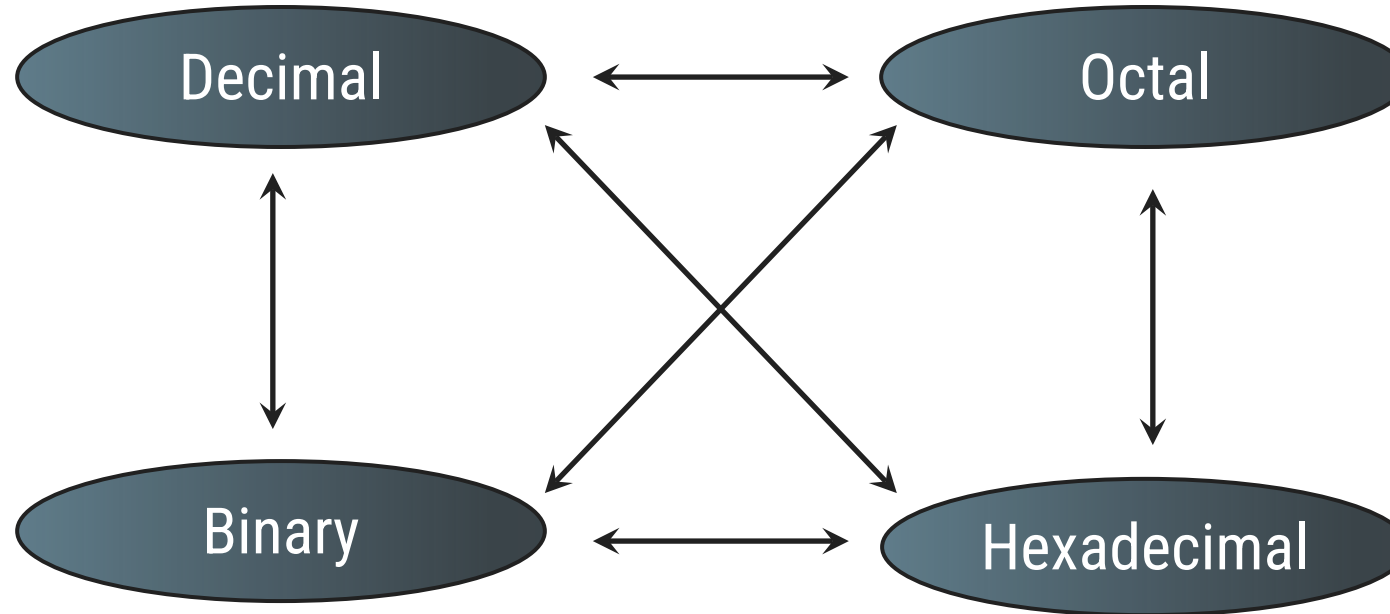
Number System

Common Number Systems

| 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 |
| Hexa- decimal | 16 | 0, 1, ... 9, A, B, ... F | No | No |

Conversion among Bases

► Possibilities



► Example

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$

Base

Decimal to Binary

► Technique

- ➔ Divide by **two**, keep track of the remainder
- ➔ The remainders read from bottom to top give the equivalent binary integer number.

► Example - 1

$$125_{10} = ?_2$$

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

$$125_{10} = 1111101_2$$

► Example - 2

$$0.6875_{10} = ?_2$$

| | <u>integer</u> | | <u>fraction</u> |
|----------------------------|----------------|---|-----------------|
| $0.6875 \times 2 = 1.3750$ | 1 | + | 0.3750 |
| $0.3750 \times 2 = 0.7500$ | 0 | + | 0.7500 |
| $0.7500 \times 2 = 1.5000$ | 1 | + | 0.5000 |
| $0.5000 \times 2 = 1.0000$ | 1 | + | 0.0000 |

$$0.6875_{10} = 0.1011_2$$

Binary to Decimal

► Technique

- ➔ 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. Finally, Add the results.

► Example - 1

$$101011_2 = ?_{10}$$

$$\begin{array}{cccccc} & 1 & 0 & 1 & 0 & 1 & 1 \\ & \swarrow & \swarrow & \swarrow & \swarrow & \swarrow & \swarrow \\ 1 \times 2^5 & + & 0 \times 2^4 & + & 1 \times 2^3 & + & 0 \times 2^2 & + & 1 \times 2^1 & + & 1 \times 2^0 \\ 32 & + & 0 & + & 8 & + & 0 & + & 2 & + & 1 \end{array}$$

$$101011_2 = 43_{10}$$

► Example - 2

$$11.11_2 = ?_{10}$$

$$\begin{array}{cccc} & 1 & 1 & . & 1 & 1 \\ & \swarrow & \swarrow & & \swarrow & \swarrow \\ 1 \times 2^1 & + & 1 \times 2^0 & + & 1 \times 2^{-1} & + & 1 \times 2^{-2} \\ 2 & + & 1 & + & 0.5 & + & 0.25 \end{array}$$

$$11.11_2 = 3.75_{10}$$

Decimal to Octal


► Technique

- ➔ Divide by **eight**, keep track of the remainder
- ➔ The remainders read from bottom to top give the equivalent octal integer number.

► Example - 1

$$125_{10} = ?_8$$

| | | |
|---|-----|---|
| 8 | 125 | 5 |
| 8 | 15 | 7 |
| 8 | 1 | 1 |
| | 0 | |




$$125_{10} = 175_8$$

► Example - 2

$$0.6875_{10} = ?_8$$

| | <u>integer</u> | | <u>fraction</u> |
|----------------------------|----------------|---|-----------------|
| $0.6875 \times 8 = 5.5000$ | 5 | + | 0.5000 |
| $0.5000 \times 8 = 4.0000$ | 4 | + | 0.0000 |



$$0.6875_{10} = 0.54_8$$

Octal to Decimal

► Technique

- ➔ Multiply each digit by 8^n , where n is the “weight” of the digit
- ➔ The weight is the position of the digit, starting from 0 on the right. Finally, Add the results.

► Example - 1

$$724_8 = ?_{10}$$

$$\begin{array}{rcccl} & 7 & & 2 & & 4 \\ & \swarrow & & \downarrow & & \searrow \\ 7 \times 8^2 & + & 2 \times 8^1 & + & 4 \times 8^0 \\ 448 & + & 16 & + & 4 \end{array}$$

$$724_8 = 468_{10}$$

► Example - 2

$$43.25_8 = ?_{10}$$

$$\begin{array}{rcccc} & 4 & & 3 & & . & & 2 & & 5 \\ & \swarrow & & \downarrow & & & & \downarrow & & \searrow \\ 4 \times 8^1 & + & 3 \times 8^0 & + & 2 \times 8^{-1} & + & 5 \times 8^{-2} \\ 32 & + & 3 & + & 0.25 & + & 0.0781 \end{array}$$

$$43.25_8 = 35.3281_{10}$$

Decimal to Hexa-Decimal


► Technique

- ➔ Divide by **sixteen**, keep track of the remainder
- ➔ The remainders read from bottom to top give the equivalent hexadecimal integer number.

► Example - 1

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

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




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

► Example - 2

$$0.03125_{10} = ?_{16}$$

| | <u>integer</u> | | <u>fraction</u> |
|-----------------------|----------------|---|-----------------|
| 0.03125 x 16 = 0.5000 | 0 | + | 0.5000 |
| 0.5000 x 16 = 8.0000 | 8 | + | 0.0000 |



$$0.03125_{10} = 0.08_{16}$$

Hexa-Decimal to Decimal

► Technique

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

► Example - 1

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

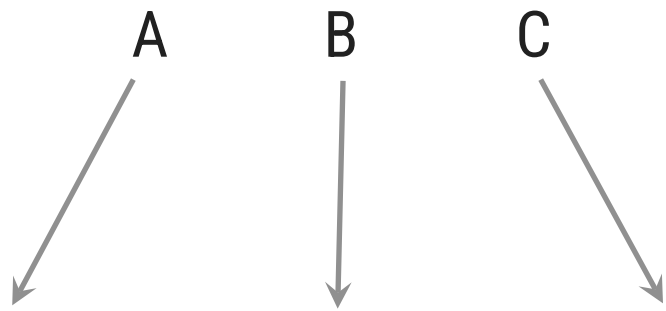


Diagram showing the conversion of hex digits A, B, and C to their decimal equivalents: A → 10, B → 11, and C → 12.

$$\begin{array}{rcl} A \times 16^2 & + & B \times 16^1 + C \times 16^0 \\ 10 \times 16^2 & + & 11 \times 16^1 + 12 \times 16^0 \\ 2560 & + & 176 + 12 \end{array}$$

$$ABC_{16} = 2748_{10}$$

► Example - 2

$$43.25_{16} = ?_{10}$$

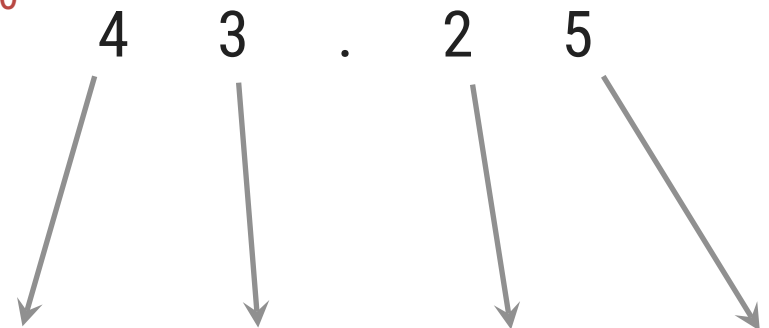


Diagram showing the conversion of hex digits 4, 3, 2, and 5 to their decimal equivalents: 4 → 4, 3 → 3, 2 → 2, and 5 → 5.

$$\begin{array}{rcl} 4 \times 16^1 & + & 3 \times 16^0 + 2 \times 16^{-1} + 5 \times 16^{-2} \\ 64 & + & 3 + 0.125 + 0.0195 \end{array}$$

$$43.25_{16} = 67.1445_{10}$$

Octal to Binary

► Technique

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

► Example

$$705_8 = ?_2$$

| | | |
|-----|-----|-----|
| 7 | 0 | 5 |
| ↓ | ↓ | ↓ |
| 111 | 000 | 101 |

$$705_8 = 111000101_2$$

| Octal | Binary |
|-------|--------|
| 0 | 000 |
| 1 | 001 |
| 2 | 010 |
| 3 | 011 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |

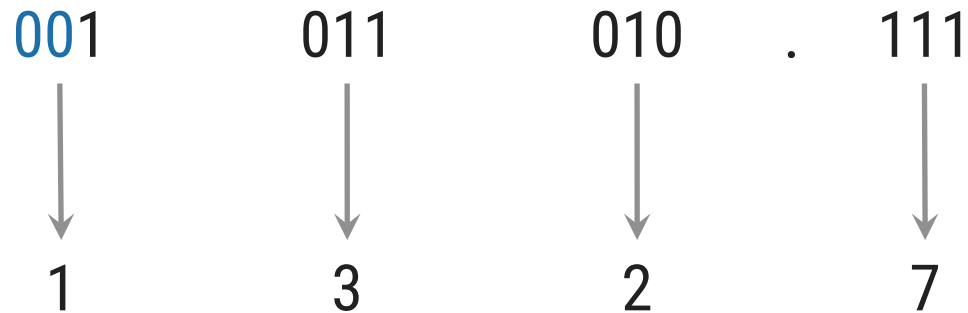
Binary to Octal

► Technique

- ➔ From given fractional **point**, group bits in **threes to right** and group bits in **threes to left**
- ➔ If, left with less than 3 bits at the end then **stuff 0s** to make it group of three
- ➔ Convert to octal digits

► Example

$$1011010.111_2 = ?_8$$



$$1011010111_2 = 132.7_8$$

Hexa-Decimal to Binary

► Technique

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

► Example

$$10AF_{16} = ?_2$$

| | | | |
|------|------|------|------|
| 1 | 0 | A | F |
| ↓ | ↓ | ↓ | ↓ |
| 0001 | 0000 | 1010 | 1111 |

$$10AF_{16} = 1000010101111_2$$

| Hexa-Decimal | Binary | Hexa-Decimal | Binary |
|--------------|--------|--------------|--------|
| 0 | 0000 | 8 | 1000 |
| 1 | 0001 | 9 | 1001 |
| 2 | 0010 | A | 1010 |
| 3 | 0011 | B | 1011 |
| 4 | 0100 | C | 1100 |
| 5 | 0101 | D | 1101 |
| 6 | 0110 | E | 1110 |
| 7 | 0111 | F | 1111 |

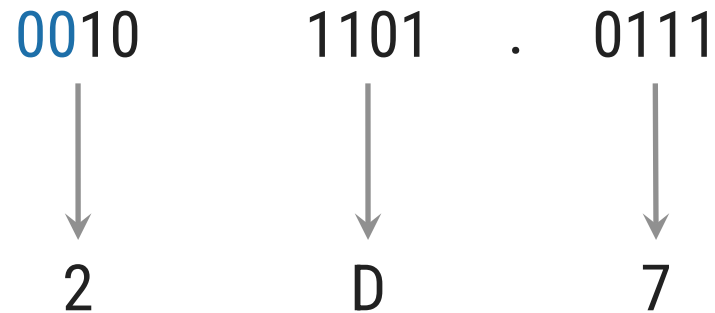
Binary to Hexa-Decimal

► Technique

- ➔ From given fractional **point**, group bits in **fours to right** and group bits in **fours to left**
- ➔ If, left with less than 4 bits at the end then **stuff 0s** to make it group of four
- ➔ Convert to hexadecimal digits

► Example

$$101101.0111_2 = ?_{16}$$



$$1011010111_2 = 2D.7_{16}$$

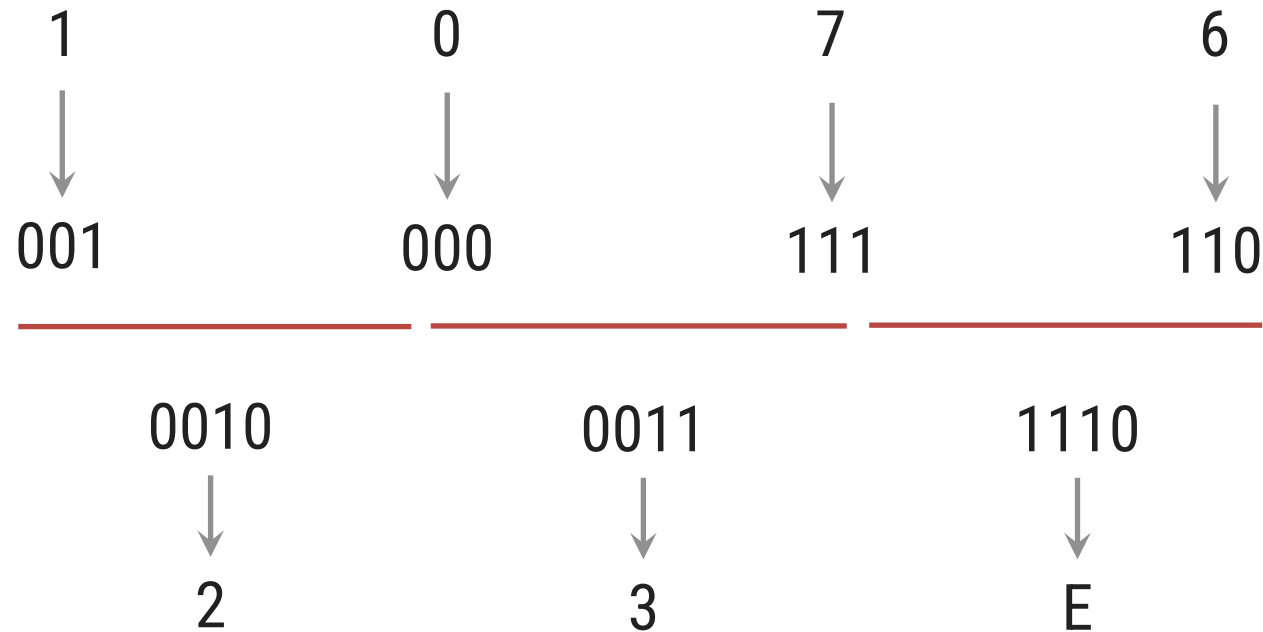
Octal to Hexa-Decimal

► Technique

- ➔ Convert Octal to Binary
- ➔ From given fractional **point**, group bits in **fours to right** and group bits in **fours to left**
- ➔ Convert Binary to Hexa-Decimal

► Example

$$1076_8 = ?_{16}$$



$$1076_8 = 23E_{16}$$

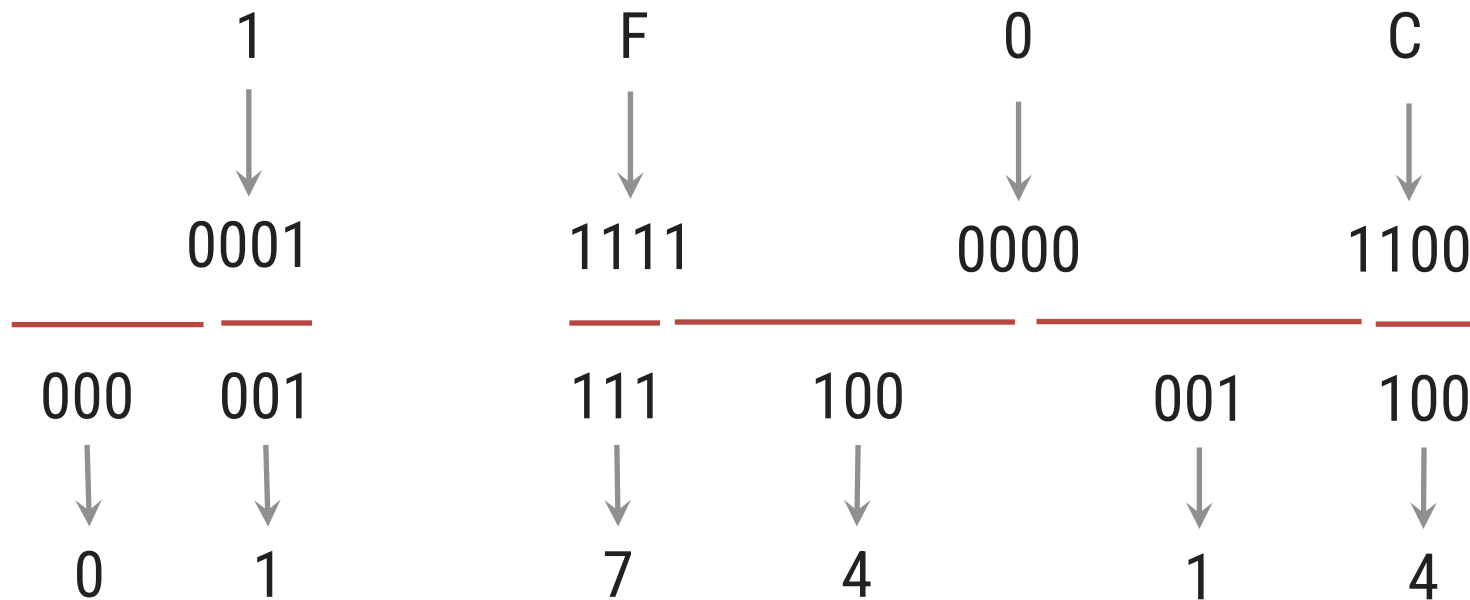
Hexa-Decimal to Octal

► Technique

- ➞ Convert Hexa-Decimal to Binary
- ➞ From given fractional **point**, group bits in **threes to right** and group bits in **threes to left**
- ➞ Convert Binary to Octal

► Example

$$1F0C_{16} = ?_8$$



$$1F0C_{16} = 17414_8$$

Binary Addition & Subtraction

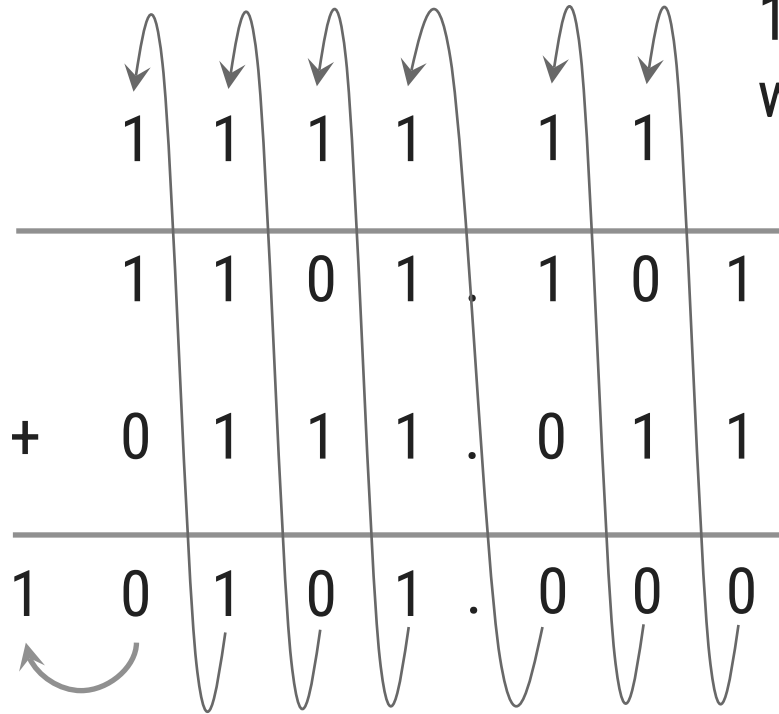
► Rules for binary addition

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10 \text{ i.e. } 0 \text{ with a carry of } 1$$



A binary addition problem: $1101.101 + 0111.011$. The result is 10101.000 . Arrows indicate the carry propagation from right to left. The first carry is from the least significant bit (1+1) to the next column. Subsequent carries occur at the second, third, fourth, fifth, and sixth columns from the right.

$$\begin{array}{r} 1101.101 \\ + 0111.011 \\ \hline 10101.000 \end{array}$$

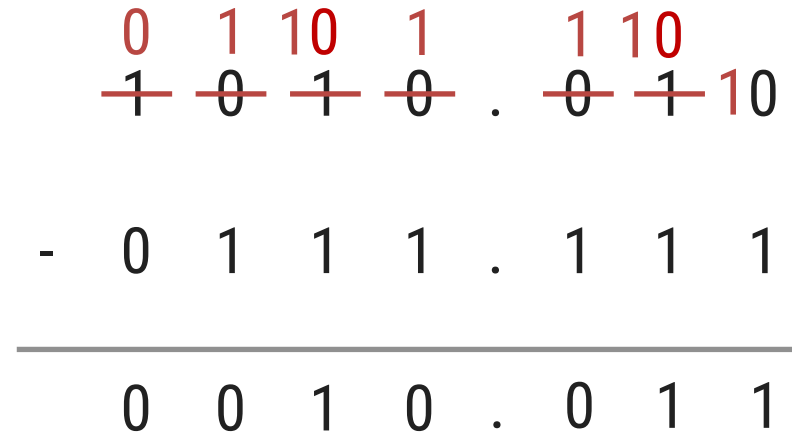
► Rules for binary subtraction

$$0 - 0 = 0$$

$$1 - 1 = 0$$

$$1 - 0 = 1$$

$$0 - 1 = 1, \text{ with a borrow } 1$$



A binary subtraction problem: $1010.100 - 0111.111$. The result is 0010.101 . Red numbers and lines indicate borrowing. Borrowing of 1 (represented as 10) is shown from the fifth column to the fourth, and from the fourth to the third. The final result is 0010.101.

$$\begin{array}{r} 0110.100 \\ - 1011.111 \\ \hline 0010.101 \end{array}$$

Binary Multiplication & Division

► Multiplication

$$\begin{array}{r} 10111 \\ x 10011 \\ \hline 10111 \\ 10111 \\ 000000 \\ 000000 \\ 10111 \\ \hline 110110101 \end{array}$$

► Division

$$\begin{array}{r|l} 110 & 101101 \\ \hline & 000 \downarrow \\ & \underline{1011} \\ & 110 \downarrow \\ & \underline{1010} \\ & 110 \downarrow \\ & \underline{1001} \\ & 110 \\ & \underline{110} \\ & 110 \\ & \underline{000} \end{array}$$

Signed Binary Numbers

- ▶ Two ways of representing signed numbers:
 - ↳ 1) Sign-magnitude form, 2) Complement form.
- ▶ Most of computers use complement form for negative number notation.
- ▶ 1's complement and 2's complement are two different methods in this type.

1's Complement

- ▶ 1's complement of a binary number is obtained by subtracting each digit of that binary number from 1.
- ▶ Example

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

(1's complement of 1101)

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

(1's complement of 101.01)

Shortcut: Invert the numbers from 0 to 1 and 1 to 0

2's Complement

- ▶ 2's complement of a binary number is obtained by adding 1 to its 1's complement.
- ▶ Example

$$\begin{array}{r} 1 \quad 1 \quad 1 \quad 1 \\ - \quad 1 \quad 1 \quad 0 \quad 0 \\ \hline 0 \quad 0 \quad 1 \quad 1 \\ + \qquad \qquad \quad 1 \\ \hline 0 \quad 1 \quad 0 \quad 0 \end{array}$$

(2's complement of 1100)

$$\begin{array}{r} 1 \quad 1 \quad 1 \quad . \quad 1 \quad 1 \\ - \quad 1 \quad 0 \quad 1 \quad . \quad 0 \quad 1 \\ \hline 0 \quad 1 \quad 0 \quad . \quad 1 \quad 0 \\ + \qquad \qquad \qquad \quad 1 \\ \hline 0 \quad 1 \quad 0 \quad . \quad 1 \quad 1 \end{array}$$


(2's complement of 101.01)

Shortcut: Starting from right side, all bits are same till first 1 occurs and then invert rest of the bits

Representation of negative number in 2's complement form

- Express -65.5 in 12 bit 2's complement form.

| | | |
|---|----|---|
| 2 | 65 | 1 |
| 2 | 32 | 0 |
| 2 | 16 | 0 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| 2 | 1 | 1 |
| | 0 | |



$$0.5 \times 2 = 1.0$$

So, result in 12-bit binary is as follows:

$$65.5_{10} = 01000001.1000_2$$

For negative number, we have to convert this into 2's complement form

$$-65.5_{10} = 10111110.1000_2$$

9's Complement

- ▶ 9's complement of a decimal number is obtained by subtracting each digit of that decimal number from 9.
- ▶ Example

$$\begin{array}{r} 9 \quad 9 \quad 9 \quad 9 \\ - \quad 3 \quad 4 \quad 6 \quad 5 \\ \hline 6 \quad 5 \quad 3 \quad 4 \end{array}$$

(9's complement of 3465)

$$\begin{array}{r} 9 \quad 9 \quad 9 \quad . \quad 9 \quad 9 \\ - \quad 7 \quad 8 \quad 2 \quad . \quad 5 \quad 4 \\ \hline 2 \quad 1 \quad 7 \quad . \quad 4 \quad 5 \end{array}$$

(9's complement of 782.54)

10's Complement

- ▶ 10's complement of a decimal number is obtained by adding 1 to its 9's complement.
- ▶ Example

$$\begin{array}{r}
 9 \quad 9 \quad 9 \quad 9 \\
 - \quad 3 \quad 4 \quad 6 \quad 5 \\
 \hline
 6 \quad 5 \quad 3 \quad 4 \\
 + \quad \quad \quad 1 \\
 \hline
 6 \quad 5 \quad 3 \quad 5
 \end{array}$$

(10's complement of 3465)

$$\begin{array}{r}
 999.99 \\
 - 782.54 \\
 \hline
 217.45 \\
 + 1 \\
 \hline
 217.46
 \end{array}$$

(10's complement of 782.54)

Subtraction using 9's complement & 10's complement

► Using 9's complement

- ➔ Obtain 9's complement of subtrahend
- ➔ Add the result to minuend and call it intermediate result
- ➔ If **carry is generated** then answer is **positive** and add the carry to Least Significant Digit (LSD)
- ➔ If there is **no carry** then answer is **negative** and take 9's complement of intermediate result and place negative sign to the result.

► Using 10's complement

- ➔ Obtain 10's complement of subtrahend
- ➔ Add the result to minuend
- ➔ If **carry is generated** then answer is **positive**, ignore carry and result itself is answer
- ➔ If there is **no carry** then answer is **negative** and take 10's complement of intermediate result and place negative sign to the result.

Subtraction using 9's complement (Examples)

► Example - 1

$$745.81 - 436.62$$

Diagram illustrating the addition of two decimal numbers using 9's complement:

First number: 745.81

Second number: 436.62

9's complement of 436.62: 563.37

Adding the numbers:

$$\begin{array}{r} 745.81 \\ + 563.37 \\ \hline 1309.18 \end{array}$$

Since the sum is greater than 999.99, we subtract 1000.00 (or add 1 to the left of the decimal point):

$$\begin{array}{r} 1309.18 \\ - 1000.00 \\ \hline 309.18 \end{array}$$

The final result is 309.18.

Subtraction using 9's complement (Examples)

► Example - 2

$$436.62 - 745.81$$

| | | | | | | | | | | | | | | | | | |
|-------|---|---|---|---|---|---|--|------------------|---|---|---|---|---|---|---|--|--|
| | 4 | 3 | 6 | . | 6 | 2 | | | | | | | | | | | |
| - | 7 | 4 | 5 | . | 8 | 1 | | 9's complement → | + | 2 | 5 | 4 | . | 1 | 8 | | |
| <hr/> | | | | | | | | | | | | | | | | | |
| - | 3 | 0 | 9 | . | 1 | 9 | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | |
|-------|---|---|---|---|---|---|--|----------------|--|--|--|--|--|--|--|--|--|
| | 4 | 3 | 6 | . | 6 | 2 | | | | | | | | | | | |
| + | 2 | 5 | 4 | . | 1 | 8 | | | | | | | | | | | |
| <hr/> | | | | | | | | | | | | | | | | | |
| | 6 | 9 | 0 | . | 8 | 0 | | | | | | | | | | | |
| | | | | | | | | 9's complement | | | | | | | | | |
| - | 3 | 0 | 9 | . | 1 | 9 | | | | | | | | | | | |

As carry is not generated, so take 9's complement of the intermediate result and add ' - ' sign to the result

Subtraction using 10's complement (Examples)

► Example - 1

$$745.81 - 436.62$$

| | | |
|--|--|---|
| $\begin{array}{r} 745.81 \\ - 436.62 \\ \hline 309.19 \end{array}$ | $\xrightarrow{\text{10's complement}}$ | $\begin{array}{r} 745.81 \\ + 563.38 \\ \hline 1309.19 \end{array}$ |
| | | \uparrow Ignore the carry |

Subtraction using 10's complement (Examples)

► Example - 2
436.62 - 745.81

| | | | | | | | | | | | | | | | |
|-------|---|---|---|---|---|---|--|-------|--|---|---|---|---|---|---|
| 4 | 3 | 6 | . | 6 | 2 | | | 4 | 3 | 6 | . | 6 | 2 | | |
| - | 7 | 4 | 5 | . | 8 | 1 | $\xrightarrow{\text{10's complement}}$ | + | 2 | 5 | 4 | . | 1 | 9 | |
| <hr/> | | | | | | | | <hr/> | | | | | | | |
| - | 3 | 0 | 9 | . | 1 | 9 | | | 6 | 9 | 0 | . | 8 | 1 | |
| | | | | | | | | | $\xrightarrow{\text{10's complement}}$ | | | | | | |
| | | | | | | | | | - | 3 | 0 | 9 | . | 1 | 9 |

As carry is not generated, so take 10's complement of the intermediate result and add ' - ' sign to the result

Subtraction using 1's complement & 2's complement

► Using 1's complement

- ➔ Obtain 1's complement of subtrahend
- ➔ Add the result to minuend and call it intermediate result
- ➔ If **carry is generated** then answer is **positive** and add the carry to Least Significant Digit (LSD)
- ➔ If there is **no carry** then answer is **negative** and take 1's complement of intermediate result and place negative sign to the result.

► Using 2's complement

- ➔ Obtain 2's complement of subtrahend
- ➔ Add the result to minuend
- ➔ If **carry is generated** then answer is **positive**, ignore carry and result itself is answer
- ➔ If there is **no carry** then answer is **negative** and take 2's complement of intermediate result and place negative sign to the result.

Subtraction using 1's complement (Examples)

► Example - 1

$$68.75 - 27.50$$

| | | |
|---------|---------------------------------------|-------------------------------------|
| 68.75 | | 01000100.1100 |
| - 27.50 | $\xrightarrow{\text{1's complement}}$ | + 11100100.0111 |
| <hr/> | | <hr/> |
| + 41.25 | | 100101001.0011 |
| | | $\xrightarrow{\quad\quad\quad} + 1$ |
| | | <hr/> |
| | | 00101001.0100 |

Subtraction using 1's complement (Examples)

► Example - 2
43.25 - 89.75

$$\begin{array}{r} 43.25 \\ - 89.75 \\ \hline -46.50 \end{array} \quad \xrightarrow{\text{1's complement}} \quad \begin{array}{r} 00101011.0100 \\ + 10100110.0011 \\ \hline 11010001.0111 \\ \xrightarrow{\text{1's complement}} \\ 00101110.1000 \end{array}$$

As carry is not generated, so take 1's complement of the intermediate result and add ' - ' sign to the result

Subtraction using 2's complement (Examples)

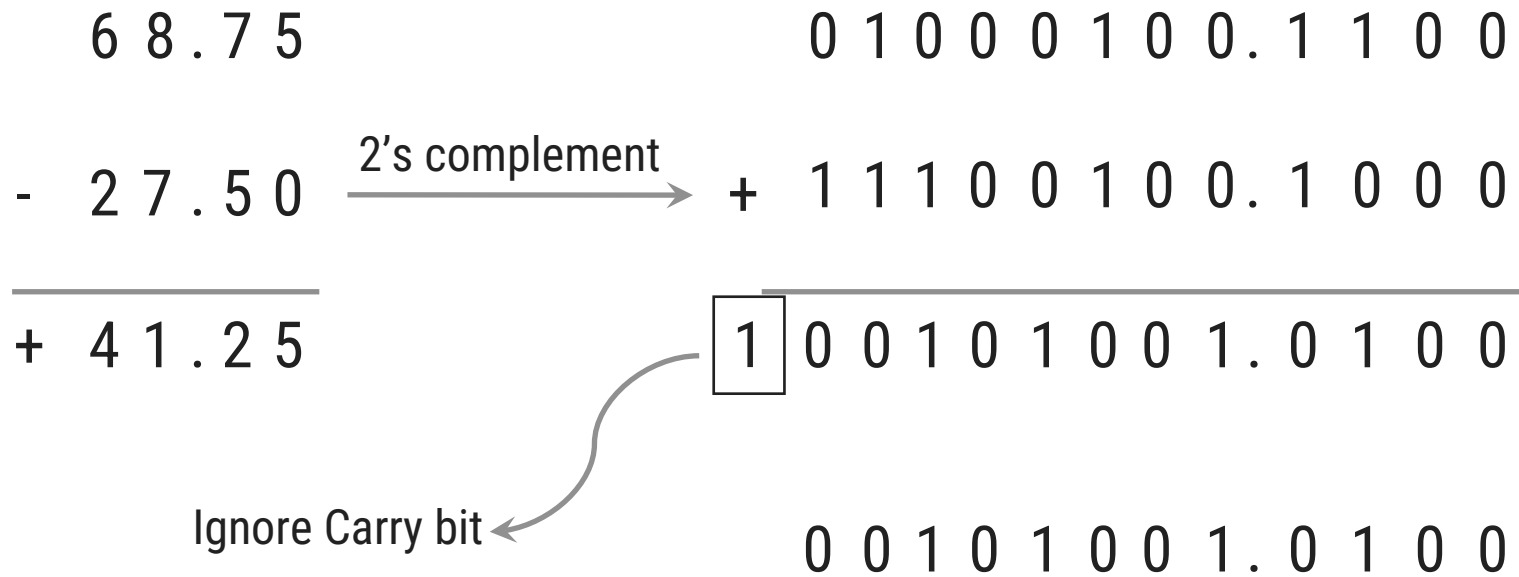
► Example - 1 68.75 - 27.50

$$\begin{array}{r} 68.75 \\ - 27.50 \\ \hline + 41.25 \end{array}$$

2's complement

$$\begin{array}{r} 01000100.1100 \\ + 11100100.1000 \\ \hline 10010100.0100 \\ 00101001.0100 \end{array}$$

Ignore Carry bit



Subtraction using 2's complement (Examples)

- Example - 2
43.25 - 89.75

$$\begin{array}{r} 43.25 \\ - 89.75 \\ \hline -46.50 \end{array} \xrightarrow{\text{2's complement}} \begin{array}{r} 00101011.0100 \\ + 10100110.0100 \\ \hline 11010001.1000 \\ \text{2's complement} \rightarrow \\ 00101110.1000 \end{array}$$

As carry is not generated, so take 2's complement of the intermediate result and add ' - ' sign to the result

Binary Codes

8421 BCD Code (Natural BCD Code)

- ▶ Each decimal digit, 0 through 9, is coded by 4-bit binary number
- ▶ 8, 4, 2 and 1 weights are attached to each bit
- ▶ BCD code is weighted code
- ▶ 1010, 1011, 1100, 1101, 1110 and 1111 are illegal codes
- ▶ Less efficient than pure binary
- ▶ Arithmetic operations are more complex than in pure binary
- ▶ Example

| | | |
|---------|------|------|
| Decimal | 1 | 4 |
| | ↓ | ↓ |
| BCD | 0001 | 0100 |
| Binary | 1110 | |

Binary Codes

| Decimal | Binary | BCD |
|---------|--------|------|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 10 | 0010 |
| 3 | 11 | 0011 |
| 4 | 100 | 0100 |
| 5 | 101 | 0101 |
| 6 | 110 | 0110 |
| 7 | 111 | 0111 |

| Decimal | Binary | BCD |
|---------|--------|-----------|
| 8 | 1000 | 1000 |
| 9 | 1001 | 1001 |
| 10 | 1010 | 0001 0000 |
| 11 | 1011 | 0001 0001 |
| 12 | 1100 | 0001 0010 |
| 13 | 1101 | 0001 0011 |
| 14 | 1110 | 0001 0100 |
| 15 | 1111 | 0001 0101 |

BCD Addition

► Example - 1

$$\begin{array}{r} 25 \\ + 13 \\ \hline 38 \end{array}$$

$$\begin{array}{r} 0010111 \\ + 00010001 \\ \hline 00111000 \end{array}$$

No carry, no illegal code. So, this is the correct sum.

Rule: If there is an illegal code or carry is generated as a result of addition, then add 0110 to particular that 4 bits of result.

BCD Addition

► Example - 2

| | | | | | | | | |
|-------|--------|-------|-------|-------|-------|-------|-------|--------|
| | | | 1 | | 111 | | | |
| | 679.6 | | 0110 | | 0111 | | 1001 | .0110 |
| + | 536.8 | + | 0101 | | 0011 | | 0110 | .1000 |
| <hr/> | | | | | | | | |
| | 1216.4 | | 1011 | | 1010 | | 1111 | .1110 |
| | | | +0110 | | +0110 | | +0110 | +.0110 |
| <hr/> | | | | | | | | |
| | | | 10001 | | 10000 | | 10101 | 1.0100 |
| | | + 1 ↙ | | + 1 ↙ | | + 1 ↙ | | + 1 ↙ |
| | | | 0001 | | 0001 | | 0110 | .0100 |
| <hr/> | | | | | | | | |
| | | | 0010 | | 0001 | | 0110 | .0100 |

All are illegal codes

Add 0110 to each

Propagate carry

Corrected sum

BCD Subtraction

► Example - 1

$$\begin{array}{r} 38 \\ - 15 \\ \hline 23 \end{array}$$

$$\begin{array}{r} 00111000 \\ - 00010101 \\ \hline 00100011 \end{array}$$

No borrow. So, this is the correct difference.

Rule: If one 4-bit group needs to take borrow from neighbor, then subtract 0110 from the group which is receiving borrow.

BCD Subtraction

► Example - 2

| | | | | | |
|------------|------------|------------|-------|--------|----------------------|
| 206.7 | 0010 | 0000 | 0110 | .0111 | |
| - 147.8 | - 0001 | 0100 | 0111 | .1000 | |
| <hr/> 58.9 | <hr/> 0000 | 1011 | 1110 | .1111 | Borrows are present |
| | | -0110 | -0110 | -.0110 | Subtract 0110 |
| | | <hr/> 0101 | 1000 | .1001 | Corrected difference |

Excess Three (XS-3) Code

- ▶ Excess Three Code = 8421 BCD + 0011(3)
- ▶ XS-3 code is non-weighted BCD code
- ▶ Also known as self complementing code
- ▶ 0000, 0001, 0010, 1101, 1110 and 1111 are illegal codes
- ▶ Example

| Decimal | 1 | 4 |
|---------|------|------|
| | ↓ | ↓ |
| BCD | 0001 | 0100 |
| XS-3 | 0100 | 0111 |

XS-3 Addition

► Example

| | | | | |
|---------|--------|-------|-------|--------|
| 247.6 | 0101 | 0111 | 1010 | .1001 |
| + 359.4 | + 0110 | 1000 | 1100 | .0111 |
| <hr/> | <hr/> | | | |
| 607.0 | 1011 | 1111 | 10110 | 1.0000 |
| | | + 1 ↙ | + 1 ↙ | |
| | <hr/> | | | |
| | 1011 | 10000 | 0111 | .0000 |
| | + 1 ↙ | | | |
| | <hr/> | | | |
| | 1100 | 0000 | 0111 | .0000 |
| | - 0011 | +0011 | +0011 | +.0011 |
| | <hr/> | | | |
| | 1001 | 0011 | 1010 | .0011 |

Carry generated

Propagate carry

Rule: Add 0011 to group which generated carry and Subtract 0011 to group which do not generated carry

Corrected Sum in XS-3

XS-3 Subtraction

► Example

| | | | | | |
|-------|------|---|-------|-------|--------|
| | 57.6 | | 1000 | 1010 | .1001 |
| - | 27.8 | - | 0101 | 1010 | .1011 |
| <hr/> | | | | | |
| | 29.8 | | 0010 | 1111 | .1110 |
| | | | +0011 | -0011 | -.0011 |
| <hr/> | | | | | |
| | | | 0101 | 1100 | .1011 |

Rule: Subtract 0011 to group which generated borrow and Add 0011 to group which do not generated borrow

Gray Code

- ▶ Only one bit changes between each pair of successive code words (**Unit distance code**).
- ▶ Gray code is a reflected code.
- ▶ Gray codes are designed recursively using following rules:
 - ➔ 1-bit Gray code has two code words, **0 and 1**.
 - ➔ The **first 2^n** code words of an $(n+1)$ -bit Gray code equal the code words of n -bit gray code, written **in order** with a leading **0 appended**.
 - ➔ The **last 2^n** code words of an $(n+1)$ -bit Gray code equal the code words of n -bit gray code, but written **in reverse order** with a leading **1 appended**.

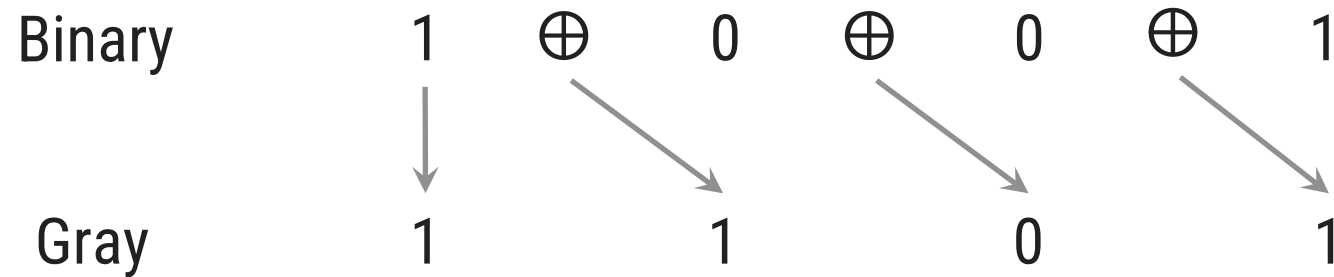
| Gray Code | | | | Decimal | 4-bit Binary |
|-----------|-------|-------|---------|---------|--------------|
| 1-bit | 2-bit | 3-bit | 4-bit | | |
| 0 | 0 0 | 0 0 0 | 0 0 0 0 | 0 | 0000 |
| 1 | 0 1 | 0 0 1 | 0 0 0 1 | 1 | 0001 |
| | 1 1 | 0 1 1 | 0 0 1 1 | 2 | 0010 |
| | 1 0 | 0 1 0 | 0 0 1 0 | 3 |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

Binary to Gray and Gray to Binary Conversion

- Conversion of n-bit Binary number (B) to Gray Code (G) is as follows:

| | | | |
|-------------|--------------------------------|--|------------------------|
| $G_n = B_n$ | $G_{n-1} = B_n \oplus B_{n-1}$ | $G_{n-2} = B_{n-1} \oplus B_{n-2} \dots$ | $G_1 = B_2 \oplus B_1$ |
|-------------|--------------------------------|--|------------------------|

- Example: Convert $(1001)_2$ to Gray Code.

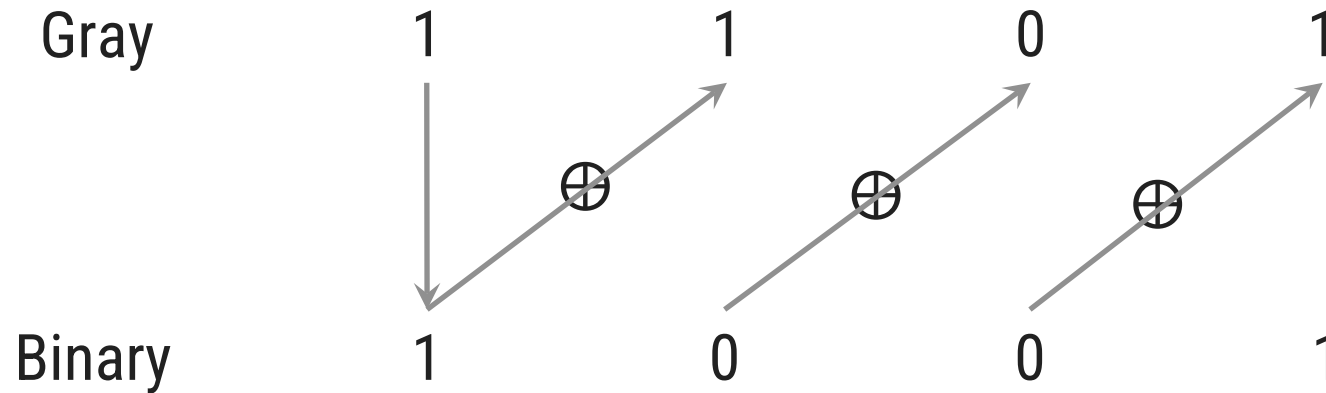


Gray to Binary Conversion

- Conversion of n-bit Gray Code (G) to Binary Number (B) is as follows:

| | | | |
|-------------|--------------------------------|--|------------------------|
| $B_n = G_n$ | $B_{n-1} = B_n \oplus G_{n-1}$ | $B_{n-2} = B_{n-1} \oplus G_{n-2} \dots$ | $B_1 = B_2 \oplus G_1$ |
|-------------|--------------------------------|--|------------------------|

- Example: Convert Gray code 1101 to Binary.



Error-Detecting Codes

- ▶ Noise can alter or distort the data in transmission.
- ▶ The 1s may get changed to 0s and 0s to 1s.
- ▶ Because digital systems must be accurate to the digit, errors can pose a serious problem.
- ▶ Single bit error should be detected & corrected by different schemes.
- ▶ **Parity**, **Check Sums** and **Block Parity** are few examples of error detecting codes.

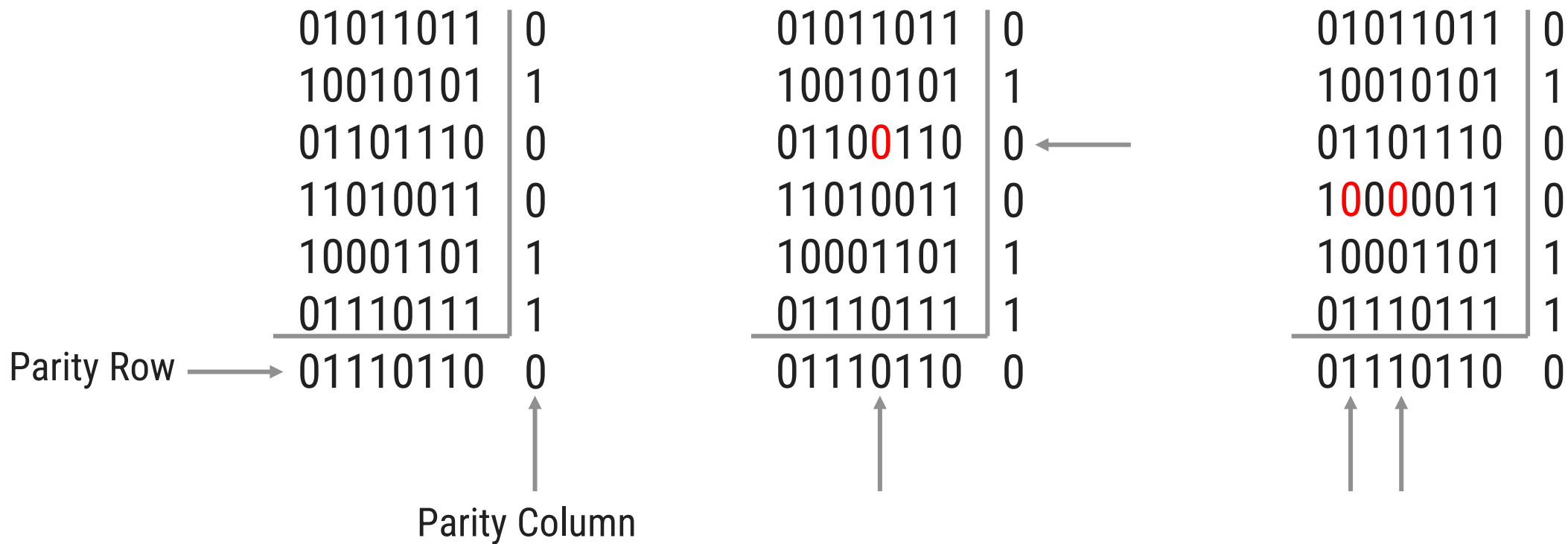
Parity

- ▶ Parity bit is the simplest technique.
- ▶ There are two types of parity – **Odd parity** and **Even parity**.
- ▶ For odd parity, the parity is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an odd number.
- ▶ For even parity, the parity is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an even number.
- ▶ For example, 0110 binary number has “1” as Odd parity and “0” as Even parity.
- ▶ Detect a single-bit error but can not detect two or more errors within the same word.
- ▶ In any practical system, there is always a finite probability of the occurrence of single error.
- ▶ E.g. In an even-parity scheme, code 10111001 is erroneous because number of 1s is odd(5), while code 11110110 is error free because number of 1s is even(6).

Check Sums

- ▶ Simple parity can not detect two errors within the same word.
- ▶ Added to the sum of the previously transmitted words
- ▶ At the transmission, the **check sum** up to that time is sent to the receiver.
- ▶ The receiver can check its sum with the transmitted sum.
- ▶ If the **two sums are the same**, then **no errors** were detected at the receiver end.
- ▶ If there is an error, the receiving location can ask for retransmission of the entire data.
- ▶ This type of transmission is used in teleprocessing system.

Block Parity



Error Correcting Code

- ▶ 7-bit Hamming Code is widely used error correcting code, containing 4 bits of data and 3 bits of even parity.

- ▶ Pattern: $P_1 P_2 D_3 P_4 D_5 D_6 D_7$

- ▶ Group - 1: $P_1 D_3 D_5 D_7$

- ▶ Group - 2: $P_2 D_3 D_6 D_7$

- ▶ Group - 3: $P_4 D_5 D_6 D_7$

- ▶ Example: Data = 1101

$$P_1 P_2 D_3 P_4 D_5 D_6 D_7 = P_1 P_2 1 P_4 1 0 1$$

$$P_1 D_3 D_5 D_7 = 1 1 1$$

$$P_2 D_3 D_6 D_7 = 1 0 1$$

$$P_4 D_5 D_6 D_7 = 1 0 1$$

- ▶ 7-bit Hamming Code is 1 0 1 0 1 0 1

- ▶ How to detect error?

- ▶ Example: Received data = 1001001

$$P_1 P_2 D_3 P_4 D_5 D_6 D_7 = 1 0 0 1 0 0 1$$

$$P_1 D_3 D_5 D_7 = 1 0 0 1 \text{ (No Error)}$$

$$P_2 D_3 D_6 D_7 = 0 0 0 1 \text{ (Error)}$$

$$P_4 D_5 D_6 D_7 = 1 0 0 1 \text{ (No Error)}$$

- ▶ The error word is $0 1 0 = 2_{10}$.

- ▶ Complement the 2nd bit (from left).

- ▶ Correct code is 1 1 0 1 0 0 1

Boolean Algebra

Boolean Algebra Laws

▶ AND laws

1. $A \cdot 0 = 0$ (*Null Law*)
2. $A \cdot 1 = A$ (*Identity Law*)
3. $A \cdot A = A$
4. $A \cdot \bar{A} = 0$

▶ Commutative laws

1. $A + B = B + A$
2. $A \cdot B = B \cdot A$

▶ OR laws

1. $A + 0 = A$ (*Null Law*)
2. $A + 1 = 1$ (*Identity Law*)
3. $A + A = A$
4. $A + \bar{A} = 1$

▶ Associative laws

1. $(A + B) + C = A + (B + C)$
2. $(A \cdot B)C = A(B \cdot C)$

Boolean Algebra Laws

▶ Distributive laws

1. $A(B + C) = AB + AC$

2. $A + BC = (A + B)(A + C)$

▶ Idempotent laws

1. $A \cdot A = A$

2. $A + A = A$

▶ Redundant Literal Rule

1. $A + \bar{A}B = A + B$

2. $A(\bar{A} + B) = AB$

▶ Absorption laws

1. $A + AB = A$

2. $A(A + B) = A$

▶ De Morgan's Theorem

1. $\overline{A + B} = \bar{A}\bar{B}$

2. $\overline{AB} = \bar{A} + \bar{B}$

Break the line change the sign

Proof of $\overline{A + B + C} = \bar{A} \bar{B} \bar{C}$

| L.H.S. | | | | | R.H.S. | | | |
|--------|---|---|-------|------------------------|-----------|-----------|-----------|---------------------------|
| A | B | C | A+B+C | $\overline{A + B + C}$ | \bar{A} | \bar{B} | \bar{C} | $\bar{A} \bar{B} \bar{C}$ |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

From truth table, it is clearly visible that L.H.S. = R.H.S. Hence, **the complement of a sum of variables is equal to the product of their individual complements.**

Proof of $\overline{ABC} = \bar{A} + \bar{B} + \bar{C}$

| L.H.S. | | | | | R.H.S. | | | |
|--------|---|---|-------|--------------------|-----------|-----------|-----------|-------------------------------|
| A | B | C | A B C | $\overline{A B C}$ | \bar{A} | \bar{B} | \bar{C} | $\bar{A} + \bar{B} + \bar{C}$ |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | | | | | | | 0 |

From truth table, it is clearly visible that L.H.S. = R.H.S. Hence, **the complement of a product of variables is equal to the sum of their individual complements.**

Reducing Boolean Expression (Example – 1)

► Reduce the expression $f = A + B[AC + (B + \bar{C})D]$

$$f = A + B[AC + (B + \bar{C})D]$$

$$f = A + B[AC + BD + \bar{C}D]$$

(Distributive law)

$$f = A + BAC + BBD + B\bar{C}D$$

(Distributive law)

$$f = A + ABC + BD + B\bar{C}D$$

(A.A = A)

$$f = A(1 + BC) + BD(1 + \bar{C})$$

$$f = A + BD$$

(1 + A = 1)

Reducing Boolean Expression (Example – 2)

► Reduce the expression $f = A[B + \bar{C}(\overline{AB + A\bar{C}})]$

$$f = A[B + \bar{C}(\overline{AB + A\bar{C}})]$$

$$f = A[B + \bar{C}(\overline{AB} \overline{A\bar{C}})]$$

(De-Morgan's law)

$$f = A[B + \bar{C}(\bar{A} + \bar{B})(\bar{A} + C)]$$

(De-Morgan's law)

$$f = A[B + \bar{C}(\bar{A}\bar{A} + \bar{A}C + \bar{B}\bar{A} + \bar{B}C)]$$

(Distributive law)

$$f = A[B + \bar{C}\bar{A} + \bar{C}\bar{A}C + \bar{C}\bar{B}\bar{A} + \bar{C}\bar{B}C]$$

(Distributive law)

$$f = A[B + \bar{C}\bar{A} + 0 + \bar{C}\bar{B}\bar{A} + 0]$$

(A.A' = 0)

$$f = AB + A\bar{C}\bar{A} + A\bar{C}\bar{B}\bar{A}$$

(Distributive law)

$$f = AB + 0 + 0$$

(A.A' = 0)

$$f = AB$$

Logic Gates

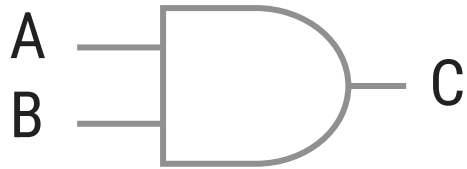
Logic Gates

- ▶ Most basic logical unit of the digital system is gate circuit.
- ▶ Types of gate circuits are as follows
 1. AND Gate
 2. OR Gate
 3. NOT Gate (Inverter)
 4. NOR Gate
 5. NAND Gate
 6. XOR Gate
 7. XNOR Gate

1. AND Gate

- ▶ AND Gate has an output which is normally at logic level “0” and only goes “HIGH” to a logic level “1” when ALL of its inputs are at logic level “1”

2-input AND Gate



Truth Table

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

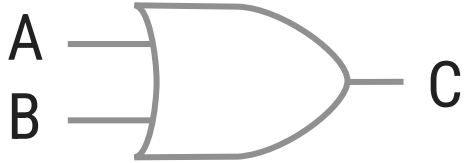
Logic Notation

$$C = A \cdot B$$

2. OR Gate

- ▶ OR Gate or Inclusive-OR gate has an output which is normally at logic level “0” and only goes “HIGH” to a logic level “1” when one or more of its inputs are at logic level “1”.

2-input OR Gate



Truth Table

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

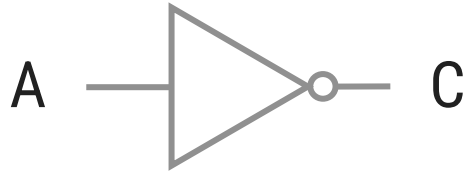
Logic Notation

$$C = A + B$$

3. NOT (Inverter) Gate

- ▶ NOT gate has an output which is always opposite to input level.

Inverter Gate



Truth Table

| A | C |
|---|---|
| 0 | 1 |
| 1 | 0 |

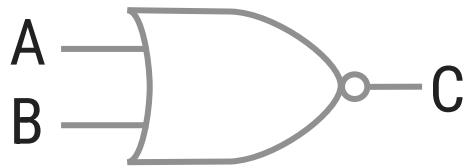
Logic Notation

$$C = \bar{A} \text{ or } C = A'$$

4. NOR Gate

- ▶ NOR Gate is an OR gate followed by an inverter.
- ▶ NOR Gate has an output which is normally at logic level “1” and only goes “LOW” to a logic level “0” when one or more of its inputs are at logic level “1”.

2-input NOR Gate



Truth Table

| A | B | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

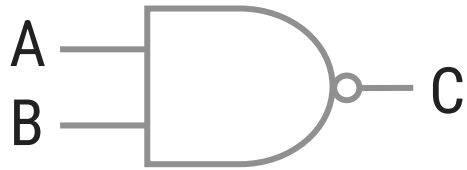
Logic Notation

$$C = (A + B)'$$

5. NAND Gate

- ▶ NAND Gate is an AND gate followed by an inverter.
- ▶ NAND Gate has an output which is normally at logic level “1” and only goes “LOW” to a logic level “0” when ALL inputs are at logic level “1”.

2-input NAND Gate



Truth Table

| A | B | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

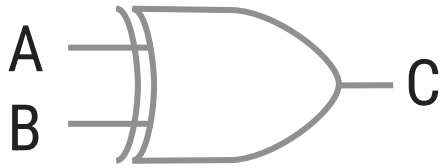
Logic Notation

$$C = (A \cdot B)'$$

6. Exclusive-OR (X-OR) Gate

- ▶ X-OR gate that has 1 state when one and only one of its two inputs assumes a logic 1 state and has 0 state when all of its input are same.
- ▶ Also known as **anti-coincidence gate** or **inequality detector**.

2-input XOR Gate



Truth Table

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

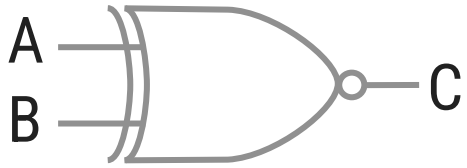
Logic Notation

$$C = A \oplus B$$

7. Exclusive-NOR (X-NOR) Gate

- ▶ X-NOR gate that has 1 state when all of its input are same and has 0 state when one of its input has 0 state and other input is 1 state.
- ▶ Also known as **coincidence gate** or **equality detector**.

2-input XNOR Gate



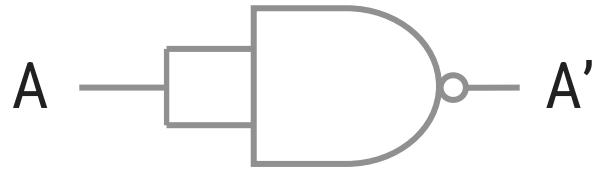
Truth Table

| A | B | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

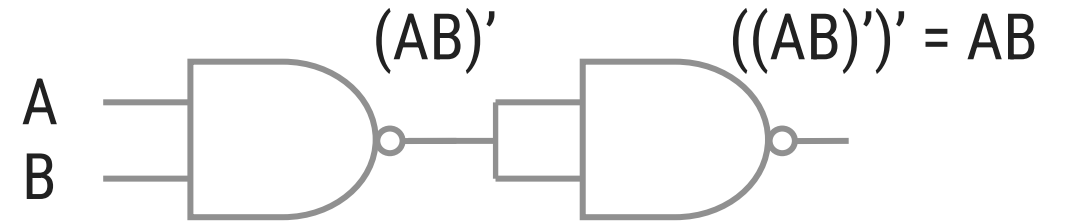
Logic Notation

$$C = A \odot B$$

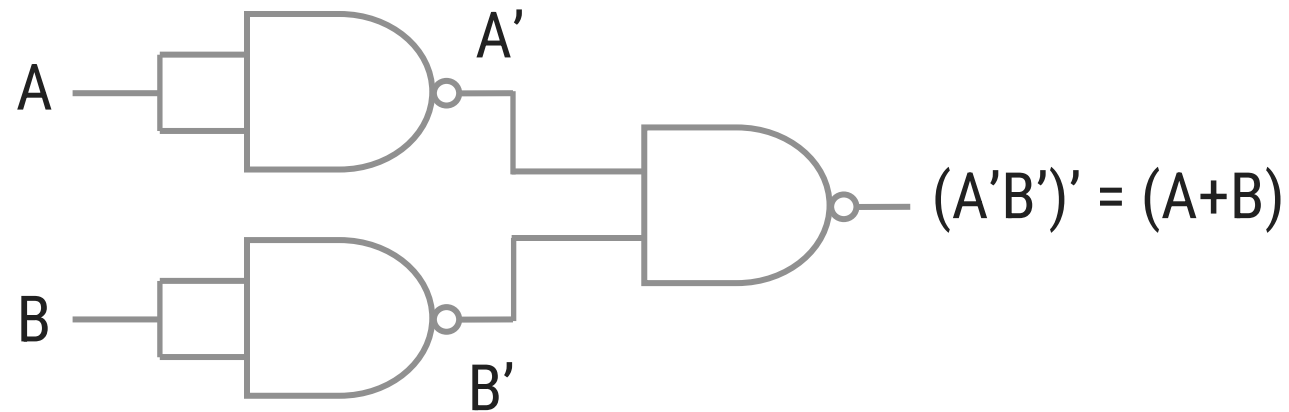
NAND as Universal Gate



NOT using NAND

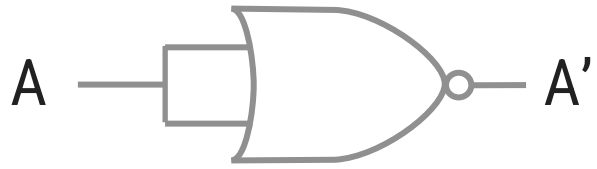


AND using NAND

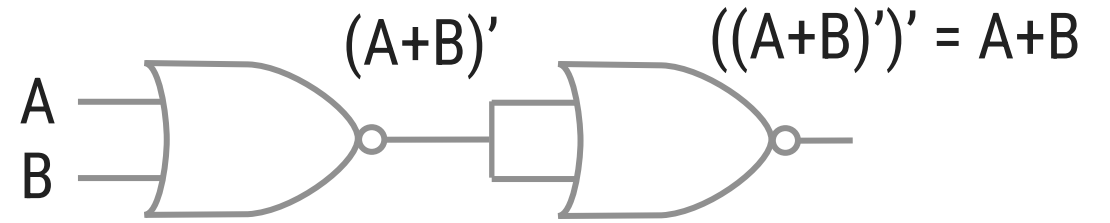


OR using NAND

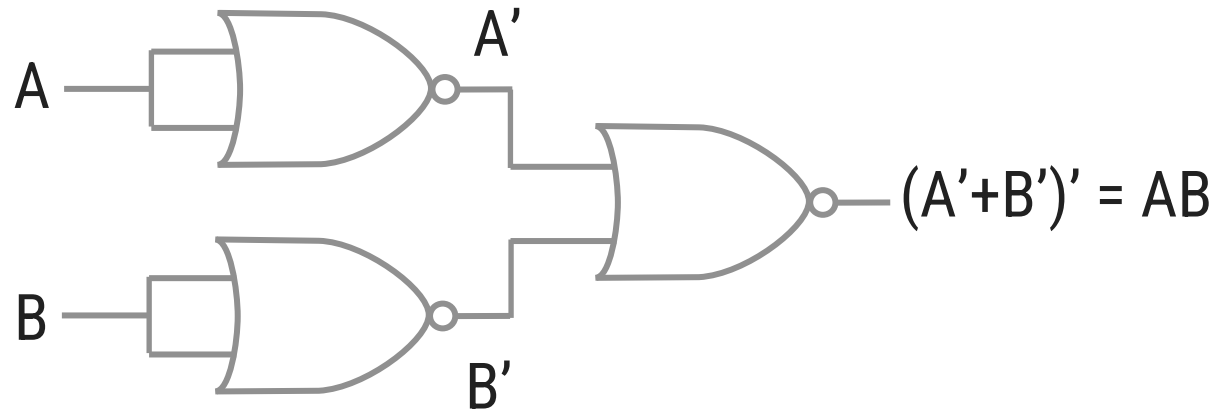
NOR as Universal Gate



NOT using NOR



OR using NOR



AND using NOR

Logic Families

Digital IC Specification

▶ Threshold voltage

- ↪ It is defined as that voltage at the input of a gate which causes a change in the state of the output from one logic level to the other.

▶ Propagation Delay

- ↪ A pulse through a gate takes a certain amount of time to propagate from input to output. This interval of time is known as the propagation delay of gate.

▶ Power dissipation

- ↪ The power dissipation, P_D , of a logic gate is the power required by the gate to operate with 50% duty cycle at a specified frequency and is expressed in milliwatts.

▶ Fan-in

- ↪ The fan-in of a logic gate is defined as the number of inputs that the gate is designed to handle.

▶ Fan-out

- ↪ The fan-out (also called the loading factor) of a logic gate is defined as the maximum number of standard loads that the output of the gate can drive without impairing its normal operation.

Digital IC Specification

▶ Voltage & Current parameters

→ $V_{IH}(\min)$, $V_{OH}(\min)$, $V_{IL}(\max)$, $V_{OL}(\max)$, I_{IH} , I_{OH} , I_{IL} , I_{OL}

▶ Noise Margin

→ The noise immunity of a logic circuit refers to the circuit's ability to tolerate noise voltages at its inputs.

→ A quantitative measure of noise immunity is called noise margin.

▶ Operating Temperatures

→ The IC gates and other circuits are temperature sensitive being semiconductor devices.

→ However, they are designed to operate satisfactorily over a specified range of temperature.

▶ Speed power products

→ A common means for measuring and comparing the overall performance of an IC family is the speed power product, which is obtained by multiplying the gate propagation delay by the gate power dissipation.

TTL v/s CMOS v/s ECL

| Characteristic | TTL | CMOS | ECL |
|--------------------------|---------------|----------------|---------------|
| Power Input | Moderate | Low | Moderate-High |
| Frequency limit | High | Moderate | Very high |
| Circuit density | Moderate-high | High-very high | Moderate |
| Circuit types per family | High | High | Moderate |

| Logic Family | Propagation delay time (ns) | Power dissipation per gate (mW) | Noise Margin (V) | Fan-in | Fan-out | Cost |
|--------------|-----------------------------|---------------------------------|------------------|--------|---------|------|
| TTL | 9 | 10 | 0.4 | 8 | 10 | Low |
| CMOS | <50 | 0.01 | 5 | 10 | 50 | Low |
| ECL | 1 | 50 | 0.25 | 5 | 10 | High |

Transistor-Transistor Logic (TTL)

- ▶ Dependence on transistors alone to perform basic logic operations.
- ▶ Most popular logic family.
- ▶ Most widely useful bipolar digital IC family.
- ▶ The TTL uses transistors operating in saturated mode.
- ▶ It is the fastest of the saturated logic families.
- ▶ Good speed, low manufacturing cost, wide range of circuits, and the availability in SSI and MSI are its merits.

Schottky TTL

- ▶ When a transistor is saturated, excess charge carriers will be stored in the base region and they must be removed before the transistor can be turned off.
- ▶ So, owing to storage time delay, the speed is reduced.
- ▶ The Schottky TTL series reduces this storage time delay by not allowing the transistor to go into full saturation.
- ▶ This is accomplished by using a Schottky barrier diode(SBD) between the base and the collector of each transistor.
- ▶ More than three times the switching speed of standard TTL, at the expense of approximately doubling the power consumption.

Tri-state TTL

- ▶ It utilizes the advantage of the high speed of operation of the totem-pole configuration and wire ANDing of the open-collector configuration.
- ▶ It is called the tri-state TTL, because it allows three possible output states: HIGH, LOW, and HIGH Impedance (Hi-Z).
- ▶ In the Hi-Z state, both the transistors in the totem-pole arrangement are turned off, so that the output terminal is a HIGH impedance to ground or V_{cc} .
- ▶ In fact, the output is an open or floating terminal, that is, neither a LOW nor a HIGH.