# Binary Arithmetic and Complements

CSE 4205: Digital Logic Design

#### **Aashnan Rahman**

Junior Lecturer

Department of Computer Science and Engineering (CSE)

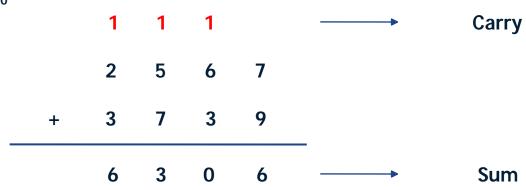
Islamic University of Technology

01

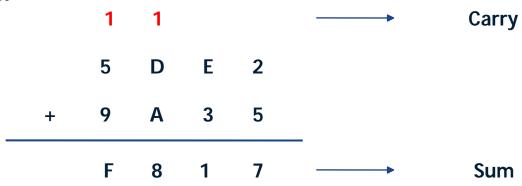
# Addition and Subtraction

How to add/subtract numbers in different bases?

## Addition

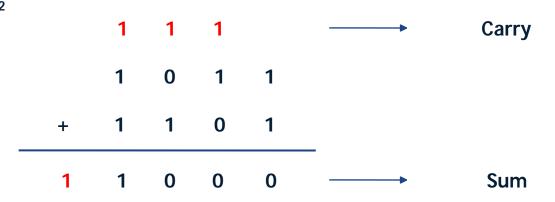


## Addition

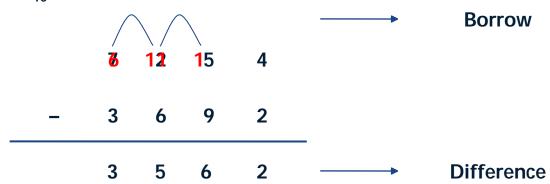


## Addition

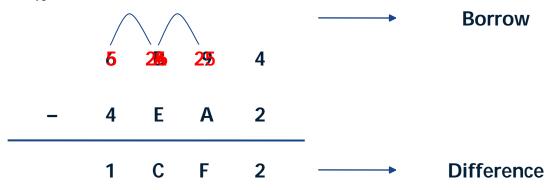
Add 
$$(1011)_2 + (1101)_2$$



### Subtraction



### Subtraction



# Try the following

- Subtract (1101)<sub>2</sub> (1011)<sub>2</sub>
- Add (ACBD)<sub>16</sub> + (1057)<sub>8</sub>
- Subtract (15) (1011) 2

# 02 Complements Negated or inverted numbers

## Complements

Complements are used to **simplify** the **subtraction operation**.

There are **two** types of complements for any base r –

- Radix Complement: r's complement
- **Diminished Radix Complement**: (r–1)'s complement

# Diminished Radix Complement

(r-1)'s complement of N =

$$r^n-r^{-m}-N$$

N= a number in base r n= no. of digits in integer part m= no. of digits in fraction part

#### Example:

- 9's complement of  $(25.639)_{10} = 10^2 10^{-3} 25.639 = (74.360)_{10}$
- 1's complement of  $(1011.101)_2 = 2^4 2^{-3} 1011.101 = (0100.010)_2$

**Hack:** Subtract every digit from (r-1)

# Radix Complement

r's complement of N =

$$r^n - N$$

N = a number in base r n = no. of digits in integer part

#### Example:

- 10's complement of  $(25.639)_{10} = 10^2 25.639 = (74.361)_{10}$
- 2's complement of  $(1011.101)_2 = 2^4 1011.101 = (0100.011)_2$

**Hack:** Leaving all least significant zeros unchanged, next first non-zero least significant digit will be subtracted from r and next all higher significant digits will be subtracted from (r-1).

or, Add r<sup>-m</sup> with (r-1)'s complement.

# 03 Use of Complements

Subtracting numbers

# Why do we need complement of a number?

- Computers are built to add, not subtract.
- Addition circuits are simple and faster.
- Using complements, we can perform subtraction using addition → simplified circuit.

Suppose you have to subtract **B** from **A** using (r-1)'s complement.

$$A - B$$
 or  $A + (-B)$ 

#### Steps -

- Find (r-1)'s complement of B.
- Add A to (r-1)'s complement of B.
- Check for end–around carry
  - If there is an end–around carry, add 1 to the result
  - If not, the result is negative, take (r-1)'s complement of the result and add a minus sign (–).

Subtract (1101)<sub>2</sub> from (10111)<sub>2</sub> using 1's complement.

Subtract (10111)<sub>2</sub> from (1101)<sub>3</sub> using 1's complement. (1101)<sub>2</sub> – (10111)<sub>2</sub> or (10111)<sub>2</sub> +(-1101)<sub>2</sub> (0000)1101) (1101)(10111) 1's complement (0000)1000) (1111 0101) No End Carry !!! 1010) **- (0000 Negative** 1's complement

# But wait

Subtract (10111)<sub>2</sub> from (10111)<sub>2</sub> using 1's complement. (10111)<sub>2</sub> – (10111)<sub>2</sub> or (10111), +(-10111) 0111) (10111)(0001)1's complement (10111) (1110 1000) (1111 1111) No End Carry !!! **-** (0000 0000) **Negative** 1's complement

#### 2 zeros?

Yes, there are 2 representations of zero in 1's complement

- Positive zero (+0): 0000...
- Negative zero (-0): 1111...

#### This is **not ideal**, because:

- It wastes one binary code
- It complicates equality checks and logic (is −0 = 0?)
- Arithmetic and comparisons need extra logic to handle both versions

How to solve then?

# 2's Complement

Subtract (1101)<sub>2</sub> from (10111)<sub>3</sub> using 2's complement. (10111)<sub>2</sub> - (1101)<sub>2</sub> or (10111)<sub>2</sub> +(-1101)<sub>2</sub> (00010111) (00010111)2's complement (00001101) (11110011) (100001010) Positive !! (00001010) **Discard Overflow** 

Subtract (10111) from (1101) using 2's complement. (1101)<sub>2</sub> – (10111)<sub>2</sub> or (1101), +(-10111), (00001101) (00001101)2's complement (00010111)(11101001) (11110110) Negative !! **- (0**0001010)<sub>2</sub> 2's complement

Subtract (10111)<sub>2</sub> from (10111)<sub>2</sub> using 2's complement. (10111)<sub>2</sub> – (10111)<sub>2</sub> or (10111), +(-10111) (10111)(00010111)2's complement (10111) (11101001) (10000000) **Discard End** Carry (00000000)

# 1s vs 2s complement

|                     | 1s Complement                         | 2s Complement                               |
|---------------------|---------------------------------------|---|
| Range               | $[-(2^{n-1}-1), -0], [0,(2^{n-1}-1)]$ | [-2 <sup>n-1</sup> , (2 <sup>n-1</sup> -1)] |
| Zero representation | -0,+0                                 | 0   |
| Arithmetic handling | Complex                               | Simpler                                     |
| Popularity          | Obsolete                              | Widely used                                 |

# How about

Subtraction using 9's/10's complement?

Why/How is it different from decimal arithmetics?

Maximum total sum at a certain position 9 + 9 + 1 = 19

In that case the sum is 9, and 1 is carried to next position

For BCD sum, sum of any BCD digit is >9 it is **INVALID**. To fix it we had 6 (decimal) with it.

Add (786)<sub>BCD</sub> with (162)<sub>BCD</sub> using BCD Addition.

#### But why do we add 6?

Because the difference between decimal carry (10) and BCD carry (1 0000) or (16) is 6.

# 05 Error Detection

How to detect errors?

#### **Error Detection**

During data transmission, electrical noise, signal degradation, or interference can cause errors.

This means bits can get flipped—a 0 might be received as a 1, or vice versa. Such bit-level changes can lead to corrupted data, which is why error detection and correction methods (like parity bits or checksums) are used.

## Parity

Parity is a simple error detection method used in data transmission.

It adds an extra bit, called the parity bit, to a group of data bits to make the number of 1s either even (even parity) or odd (odd parity).

When data is received, the system checks the parity to see if any single-bit error has occurred during transmission.

While parity can't correct errors, it can detect if a bit was flipped.

| $D_3$                                     | $\mathbf{D}_2$ | $D_1$ | $\mathbf{D}_{0}$ | Even-parity | Odd-parity |
|---|----------------|-------|------------------|-------------|------------|
| $\begin{vmatrix} D_3 & D_2 \end{vmatrix}$ | $D_2$          | 1     | ס ס              | P           | P          |
| 0   | 0              | 0     | 0                | 0           | 1          |
| 0   | 0              | 0     | 1                | 1           | 0          |
| 0   | 0              | 1     | 0                | 1           | 0          |
| 0   | 0              | 1     | 1                | 0           | 1          |
| 0   | 1              | 0     | 0                | 1           | 0          |
| 0   | 1              | 0     | 1                | 0           | 1          |
| 0   | 1              | 1     | 0                | 0           | 1          |
| 0   | 1              | 1     | 1                | 1           | 0          |
| 1   | 0              | 0     | 0                | 1           | 0          |
| 1   | 0              | 0     | 1                | 0           | 1          |
| 1   | 0              | 1     | 0                | 0           | 1          |
| 1   | 0              | 1     | 1                | 1           | 0          |
| 1   | 1              | 0     | 0                | 0           | 1          |
| 1   | 1              | 0     | 1                | 1           | 0          |
| 1   | 1              | 1     | 0                | 1           | 0          |
| 1   | 1              | 1     | 1                | 0           | 1          |

# Parity

| Data Block | Parity bit | Code word           |
|------------|------------|---------------------|
| 0000       | 0          | 00000               |
| 0001       | 1          | 00011               |
| 0010       | 1          | 00101               |
| 0011       | 0          | 00110               |
| 0100       | 1          | 01001               |
| 0101       | 0          | 01010               |
| 0110       | 0          | 01100               |
| 0111       | 1          | 0111 <mark>1</mark> |
| 1000       | 1          | 10001               |
| 1001       | 0          | 10010               |
| 1010       | 0          | 10100               |
| 1011       | 1          | 10111               |
| 1100       | 0          | 11000               |
| 1101       | 1          | 11011               |
| 1110       | 1          | 11101               |
| 1111       | 0          | 11110               |

\* Even parity is considered here

#### Checksum

A checksum is a value calculated from a block of data to help detect errors during transmission or storage.

It is sent along with the data, and the receiver recalculates the checksum to verify if the data was altered or corrupted.

If the **checksums don't match**, it means an **error** has occurred.

| $D_3$ | $\mathbf{D}_2$ | D <sub>1</sub> | $\mathbf{D_0}$ | Even-parity<br>P | Odd-parity<br>P |
|-------|----------------|----------------|----------------|------------------|-----------------|
| 0     | 0              | 0              | 0              | 0                | 1               |
| 0     | 0              | 0              | 1              | 1                | 0               |
| 0     | 0              | 1              | 0              | 1                | 0               |
| 0     | 0              | 1              | 1              | 0                | 1               |
| 0     | 1              | 0              | 0              | 1                | 0               |
| 0     | 1              | 0              | 1              | 0                | 1               |
| 0     | 1              | 1              | 0              | 0                | 1               |
| 0     | 1              | 1              | 1              | 1                | 0               |
| 1     | 0              | 0              | 0              | 1                | 0               |
| 1     | 0              | 0              | 1              | 0                | 1               |
| 1     | 0              | 1              | 0              | 0                | 1               |
| 1     | 0              | 1              | 1              | 1                | 0               |
| 1     | 1              | 0              | 0              | 0                | 1               |
| 1     | 1              | 0              | 1              | 1                | 0               |
| 1     | 1              | 1              | 0              | 1                | 0               |
| 1     | 1              | 1              | 1              | 0                | 1               |

# Thank You!!

Feel free to ask any questions