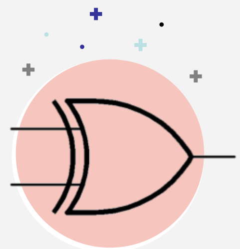
V1 (@2025)



Digital Circuits

Lecture 1:

Number Systems and Binary System

By: M.Razeghizadeh

Start now!

- A number system is a code that uses specific symbols to represent a set of values with assigned significance.

- **Decimal Number System:**

  0 1 2 3 4 5 6 7 8 9

  $4538 = 4000 + 500 + 30 + 8$
  $= 4 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 8 \times 10^0$

- The general representation of a number in base 10 (decimal) is:

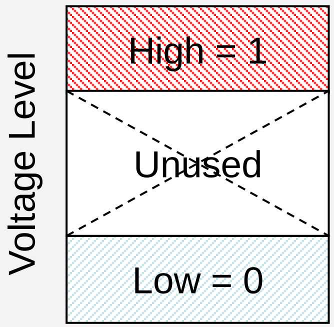
$$N = d_n \times 10^n + d_{n-1} \times 10^{n-1} + \cdots + d_1 \times 10^1 + d_0 \times 10^0 + d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + \ldots$$

Where:

- d_i represents the digits (0 to 9) at different positions.
- The exponent of 10 indicates the place value of each digit.
- Digits to the left of the decimal point ($d_n, d_{n-1}, \ldots$) represent integer values.
- Digits to the right of the decimal point ($d_{-1}, d_{-2}, \ldots$) represent fractional values.

How do Computers Represent Digits?

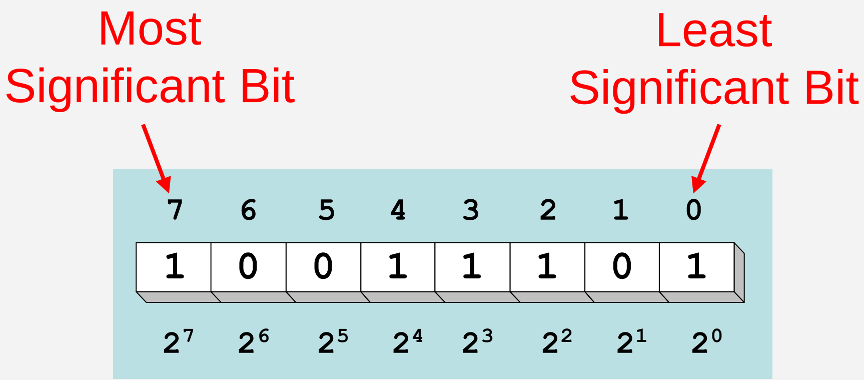
- Binary digits (0 and 1) are the simplest to represent
- Using electric voltage
  - Used in processors and digital circuits
  - High voltage = 1, Low voltage = 0



- Using electric charge
  - Used in memory cells
  - Charged memory cell = 1, discharged memory cell = 0
- Using magnetic field
  - Used in magnetic disks, magnetic polarity indicates 1 or 0
- Using light
  - Used in optical disks, optical lens can sense the light or not

Binary Numbers

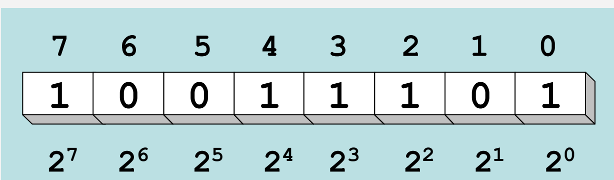
- Each binary digit (called a bit) is either 1 or 0
- Bits have no inherent meaning, they can represent …
  - Unsigned and signed integers
  - Fractions
  - Characters
  - Images, sound, etc.



- Bit Numbering
  - Least significant bit (LSB) is rightmost (bit 0)
  - Most significant bit (MSB) is leftmost (bit 7 in an 8-bit number)

Decimal Value of Binary Numbers

- Each bit represents a power of 2
- Every binary number is a sum of powers of 2
- Decimal Value = $(d_{n-1} \times 2^{n-1}) + \ldots + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Binary $(10011101)_2 = 2^7 + 2^4 + 2^3 + 2^2 + 1 = 157$



Some common powers of 2

2^n	Decimal Value	2^n	Decimal Value
2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

Different Representations of Natural Numbers

XXVII	Roman numerals (not positional)
27	Radix-10 or decimal number (positional)
11011_2	Radix-2 or binary number (positional)

Fixed-radix positional representation with *n* digits

Number N in radix $r = (d_{n-1}d_{n-2} \ldots d_1d_0)_r$

N_r (Value) $= d_{n-1} \times r^{n-1} + d_{n-2} \times r^{n-2} + \ldots + d_1 \times r + d_0$

Examples: $(11011)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 = 27$

$(2107)_8 = 2 \times 8^3 + 1 \times 8^2 + 0 \times 8 + 7 = 1095$

Convert Decimal to Binary

- Repeatedly divide the decimal integer by 2
- Each remainder is a binary digit in the translated value
- Example: Convert 37_{10} to Binary

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

least significant bit

$37 = (100101)_2$

most significant bit

stop when quotient is zero

Decimal to Binary Conversion

- $N = (d_{n-1} \times 2^{n-1}) + \ldots + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Dividing N by 2 we first obtain
  - $\text{Quotient}_1 = (d_{n-1} \times 2^{n-2}) + \ldots + (d_2 \times 2) + d_1$
  - $\text{Remainder}_1 = d_0$
  - Therefore, first remainder is least significant bit of binary number
- Dividing first quotient by 2 we first obtain
  - $\text{Quotient}_2 = (d_{n-1} \times 2^{n-3}) + \ldots + (d_3 \times 2) + d_2$
  - $\text{Remainder}_2 = d_1$
- Repeat dividing quotient by 2
  - Stop when new quotient is equal to zero
  - Remainders are the bits from least to most significant bit

Popular Number Systems

- Binary Number System: Radix = 2
  - Only two digit values: 0 and 1
  - Numbers are represented as 0s and 1s
- Octal Number System: Radix = 8
  - Eight digit values: 0, 1, 2, …, 7
- Decimal Number System: Radix = 10
  - Ten digit values: 0, 1, 2, …, 9
- Hexadecimal Number Systems: Radix = 16
  - Sixteen digit values: 0, 1, 2, …, 9, A, B, …, F
  - A = 10, B = 11, …, F = 15
- Octal and Hexadecimal numbers can be converted easily to Binary and vice versa

Octal and Hexadecimal Numbers

- Octal = Radix 8
- Only eight digits: 0 to 7
- Digits 8 and 9 not used
- Hexadecimal = Radix 16
- 16 digits: 0 to 9, A to F
- A=10, B=11, …, F=15
- First 16 decimal values (0 to 15) and their values in binary, octal and hex.

Decimal Radix 10	Binary Radix 2	Octal Radix 8	Hex Radix 16
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary, Octal, and Hexadecimal

- Binary, Octal, and Hexadecimal are related:

  Radix 16 = 2^4 and Radix 8 = 2^3

- Hexadecimal digit = 4 bits and Octal digit = 3 bits

- Starting from least-significant bit, group each 4 bits into a hex digit or each 3 bits into an octal digit

- Example: Convert 32-bit number into octal and hex

3	5	3	0	5	5	2	3	6	2	4	**Octal**

1	1	1	0	1	0	1	1	0	0	0	1	0	1	1	0	1	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	**32-bit binary**

E	B	1	6	A	7	9	4	**Hexadecimal**

Converting Octal & Hex to Decimal

- Octal to Decimal: $N_8 = (d_{n-1} \times 8^{n-1}) + \ldots + (d_1 \times 8) + d_0$
- Hex to Decimal: $N_{16} = (d_{n-1} \times 16^{n-1}) + \ldots + (d_1 \times 16) + d_0$
- Examples:

$(7204)_8 = (7 \times 8^3) + (2 \times 8^2) + (0 \times 8) + 4 = 3716$

$(3BA4)_{16} = (3 \times 16^3) + (11 \times 16^2) + (10 \times 16) + 4 = 15268$

Converting Decimal to Hexadecimal

- Repeatedly divide the decimal integer by 16
- Each remainder is a hex digit in the translated value
- Example: convert 422 to hexadecimal

Division	Quotient	Remainder
422 / 16	26	6
26 / 16	1	A
1 / 16	0	1

$422 = (1A6)_{16}$

- To convert decimal to octal divide by 8 instead of 16

Important Properties

- How many possible digits can we have in Radix r ?

  r digits: 0 to $r - 1$

- What is the result of adding 1 to the largest digit in Radix r?

  Since digit r is not represented, result is $(10)_r$ in Radix r

  Examples: $1_2 + 1 = (10)_2$ $7_8 + 1 = (10)_8$

  $9_{10} + 1 = (10)_{10}$ $F_{16} + 1 = (10)_{16}$

- What is the largest value using 3 digits in Radix r?

  In binary: $(111)_2 = 2^3 - 1$

  In octal: $(777)_8 = 8^3 - 1$

  In decimal: $(999)_{10} = 10^3 - 1$

  In Radix r:

  largest value $= r^3 - 1$

Important Properties

- How many possible values can be represented …

Using *n* binary digits?	2^n values: 0 to $2^n - 1$
Using *n* octal digits	8^n values: 0 to $8^n - 1$
Using *n* decimal digits?	10^n values: 0 to $10^n - 1$
Using *n* hexadecimal digits	16^n values: 0 to $16^n - 1$
Using *n* digits in Radix *r* ?	r^n values: 0 to $r^n - 1$

Representing Fractions

- A number N_r in *radix* r can also have a fraction part:

$$N_r = \underbrace{d_{n-1} d_{n-2} \ldots d_1 d_0}_{\text{Integer Part}} \,.\, \underbrace{d_{-1}\, d_{-2} \ldots d_{-m+1}\, d_{-m}}_{\text{Fraction Part}} \qquad 0 \le d_i < r$$

↑ Radix Point

- The number N_r represents the value:

$$N_r = d_{n-1} \times r^{n-1} + \ldots + d_1 \times r + d_0 + \qquad \textbf{(Integer Part)}$$

$$d_{-1} \times r^{-1} + d_{-2} \times r^{-2} \ldots + d_{-m} \times r^{-m} \qquad \textbf{(Fraction Part)}$$

$$N_r = \sum_{i=0}^{i=n-1} d_i \times r^i + \sum_{j=-m}^{j=-1} d_j \times r^j$$

Examples of Numbers with Fractions

- $(2409.87)_{10}$ $= 2\times10^3 + 4\times10^2 + 9 + 8\times10^{-1} + 7\times10^{-2}$
- $(1101.1001)_2$ $= 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-4} = 13.5625$
- $(703.64)_8$ $= 7\times8^2 + 3 + 6\times8^{-1} + 4\times8^{-2} = 451.8125$
- $(A1F.8)_{16}$ $= 10\times16^2 + 16 + 15 + 8\times16^{-1} = 2591.5$
- $(423.1)_5$ $= 4\times5^2 + 2\times5 + 3 + 5^{-1} = 113.2$
- $(263.5)_6$ Digit 6 is NOT allowed in radix 6

Converting Decimal Fraction to Binary

- Convert $N = 0.6875$ to Radix 2
- Solution: Multiply N by 2 repeatedly & collect integer bits

Multiplication	New Fraction	Bit	
0.6875 × 2 = 1.375	0.375	1	→ First fraction bit
0.375 × 2 = 0.75	0.75	0	
0.75 × 2 = 1.5	0.5	1	
0.5 × 2 = 1.0	0.0	1	→ Last fraction bit

- Stop when new fraction = 0.0, or when enough fraction bits are obtained
- Therefore, $N = 0.6875 = (0.1011)_2$
- Check $(0.1011)_2 = 2^{-1} + 2^{-3} + 2^{-4} = 0.6875$

Converting Fraction to any Radix r

- To convert fraction N to any radix r

  $N_r = (0.d_{-1}\, d_{-2} \ldots d_{-m})_r = d_{-1} \times r^{-1} + d_{-2} \times r^{-2} \ldots + d_{-m} \times r^{-m}$

- Multiply N by r to obtain d_{-1}

  $N_r \times r = d_{-1} + d_{-2} \times r^{-1} \ldots + d_{-m} \times r^{-m+1}$

- The integer part is the digit d_{-1} in radix r
- The new fraction is $d_{-2} \times r^{-1} \ldots + d_{-m} \times r^{-m+1}$
- Repeat multiplying the new fractions by r to obtain $d_{-2}\ d_{-3} \ldots$
- Stop when new fraction becomes 0.0 or enough fraction digits are obtained

More Conversion Examples

- Convert N = 139.6875 to Octal (Radix 8)
- Solution: N = 139 + 0.6875 (split integer from fraction)
- The integer and fraction parts are converted separately

Division	Quotient	Remainder
139 / 8	17	3
17 / 8	2	1
2 / 8	0	2

Multiplication	New Fraction	Digit
0.6875 × 8 = 5.5	0.5	5
0.5 × 8 = 4.0	0.0	4

- Therefore, 139 = $(213)_8$ and 0.6875 = $(0.54)_8$
- Now, join the integer and fraction parts with radix point

N = 139.6875 = $(213.54)_8$

Conversion Procedure to Radix r

- To convert decimal number *N* (with fraction) to radix *r*
- Convert the Integer Part
  - Repeatedly divide the integer part of number *N* by the radix *r* and save the remainders. The integer digits in radix *r* are the remainders in reverse order of their computation. If radix *r* > 10, then convert all remainders > 10 to digits A, B, … etc.
- Convert the Fractional Part
  - Repeatedly multiply the fraction of *N* by the radix *r* and save the integer digits that result. The fraction digits in radix *r* are the integer digits in order of their computation. If the radix *r* > 10, then convert all digits > 10 to A, B, … etc.
- Join the result together with the radix point

Simplified Conversions

- Converting fractions between Binary, Octal, and Hexadecimal can be simplified
- Starting at the radix pointing, the integer part is converted from right to left and the fractional part is converted from left to right
- Group 4 bits into a hex digit or 3 bits into an octal digit

← integer: right to left — — fraction: left to right →

7	2	6	1	3	.	2	4	7	4	5	2	**Octal**
111	010	110	001	011	.	010	100	111	100	101	01	**Binary**
7	5	8	B		.	5	3	C	A	8		**Hexadecimal**

- Use binary to convert between octal and hexadecimal

Important Properties of Fractions

- How many fractional values exist with m fraction bits?

  2^m fractions, because each fraction bit can be 0 or 1

- What is the largest fraction value if m bits are used?

  Largest fraction value = $2^{-1} + 2^{-2} + \ldots + 2^{-m} = 1 - 2^{-m}$

  Because if you add 2^{-m} to largest fraction you obtain 1

- In general, what is the largest fraction value if m fraction digits are used in radix r?

  Largest fraction value = $(r - 1) \times (r^{-1} + r^{-2} + \ldots + r^{-m}) = 1 - r^{-m}$

  For decimal, largest fraction value = $1 - 10^{-m}$

  For hexadecimal, largest fraction value = $1 - 16^{-m}$

Binary Arithmetic - Adding Bits

- 1 + 1 = 2, but 2 should be represented as $(10)_2$ in binary
- Adding two bits: the sum is S and the carry is C

X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- Adding three bits: the sum is S and the carry is C

0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

Binary Arithmetic - Binary Addition

- Start with the least significant bit (rightmost bit)
- Add each pair of bits
- Include the carry in the addition, if present

carry		1	1	1	1				
	0	0	1	1	0	1	1	0	(54)
+	0	0	0	1	1	1	0	1	(29)
	0	1	0	1	0	0	1	1	(83)
bit position:	7	6	5	4	3	2	1	0	

Binary Arithmetic - Subtracting Bits

- Subtracting 2 bits (X – Y): we get the difference (D) and the **borrow-out** (B) shown as 0 or -1

X	0	0	1	1
– Y	– 0	– 1	– 0	– 1
B D	0 0	-1 1	0 1	0 0

- Subtracting two bits (X – Y) with a **borrow-in = -1**: we get the difference (D) and the **borrow-out** (B)

borrow-in	-1	-1	-1	-1	-1
	X	0	0	1	1
	– Y	– 0	– 1	– 0	– 1
	B D	-1 1	-1 0	0 0	-1 1

Binary Arithmetic - Binary Subtraction

- Start with the least significant bit (rightmost bit)
- Subtract each pair of bits
- Include the borrow in the subtraction, if present

borrow			-1	-1			-1		
	0	0	1	1	0	1	1	0	(54)
–	0	0	0	1	1	1	0	1	(29)
	0	0	0	1	1	0	0	1	(25)
bit position:	7	6	5	4	3	2	1	0	

Binary Arithmetic - Binary Multiplication

- Binary Multiplication table is simple:

$0 \times 0 = 0, \quad 0 \times 1 = 0, \quad 1 \times 0 = 0, \quad 1 \times 1 = 1$

Multiplicand		$1100_2 = 12$
Multiplier	×	$1101_2 = 13$
		1100
		0000
		1100
		1100
Product		$10011100_2 = 156$

Binary multiplication is easy

0 × multiplicand = 0

1 × multiplicand = multiplicand

- n-bit multiplicand × n-bit multiplier = $2n$-bit product
- Accomplished via shifting and addition

Binary Arithmetic - Binary Multiplication

- Start with the least significant hexadecimal digits
- Let Sum = summation of two hex digits
- If Sum is greater than or equal to 16
  - Sum = Sum – 16 and Carry = 1
- Example:

carry				1	1		1	
	9	C	3	7	2	8	6	5
+	1	3	9	5	E	8	4	B
	A	F	C	D	1	0	B	0

5 + B = 5 + 11 = 16
Since Sum ≥ 16
Sum = 16 – 16 = 0
Carry = 1

Binary Arithmetic - Binary Multiplication

- Start with the least significant hexadecimal digits
- Let Difference = subtraction of two hex digits
- If Difference is negative
  - Difference = 16 + Difference and Borrow = -1
- Example:

borrow		-1		-1			-1	
	9	C	3	7	2	8	6	5
−	1	3	9	5	E	8	4	B
	8	8	A	1	4	0	1	A

Since 5 < B, Difference < 0
Difference = 16+5–11 = 10
Borrow = -1

Binary Arithmetic - Binary Multiplication

- What happens if the bits are shifted to the left by 1 bit position?

Before	0	0	0	0	0	1	0	1	= 5
After	0	0	0	0	1	0	1	0	= 10

Multiplication By 2

❖ What happens if the bits are shifted to the left by 2 bit positions?

Before	0	0	0	0	0	1	0	1	= 5
After	0	0	0	1	0	1	0	0	= 20

Multiplication By 4

❖ Shifting the Bits to the Left by n bit positions is multiplication by 2^n

❖ As long as we have sufficient space to store the bits

Binary Arithmetic - Shifting the Bits to the Right

- What happens if the bits are shifted to the right by 1 bit position?

Before	0	0	1	0	0	1	1	**0**	= 38
After	0	0	0	1	0	0	1	1	= 19, **r=0**

Division By 2

- What happens if the bits are shifted to the right by 2 bit positions?

Before	0	0	1	0	0	1	**1**	**0**	= 38
After	0	0	0	0	1	0	0	1	= 9, **r=2**

Division By 4

- Shifting the Bits to the Right by n bit positions is division by 2^n
- The **remainder r** is the value of the bits that are **shifted out**

Number Complements (Complements)

If a number N is given:

- Radix Complement (r-Complement)
- Reduced Complement (r-1 Complement)

If the number N has n digits in base r:

- Radix Complement = $r^n - N$
- Reduced Complement = $(r^n - 1) - N$

✓ Computing the reduced complement is much simpler,

Radix Complement = Reduced Complement + 1

Number Complements (Complements)

r-1 Complement

If a number N in base r has n digits, its r-1 complement is defined as $(r^n - 1) - N$.

In the decimal system (r = 10):

- $10^1 = 10$
- $10^2 = 100$
- $10^3 = 1000$
- $10^4 = 10000$
- $10^n = 100000….0$ (n 0)

→

- $10^1 - 1 = 9$
- $10^2 - 1 = 99$
- $10^3 - 1 = 999$
- $10^4 - 1 = 9999$
- $10^n - 1 = 9999….9$ (n 9)

→

To find the 9's complement of a number N:

Subtract each digit of N from 9.

Example:
N = 379
9's complement = 999 - 379 = 620

r-1 Complement

If a number N in base r has n digits, its r-1 complement is defined as $(r^n - 1) - N$.

In the decimal system (r = 2):

- $2^1 = 2 = (10)_2$
- $2^2 = 4 = (100)_2$
- $2^3 = 8 = (1000)_2$
- $2^4 = 16 = (10000)_2$
- $2^n = (100000...0)_2$ (n 0)

- $2^1 - 1 = 1$
- $2^2 - 1 = 11$
- $2^3 - 1 = 111$
- $2^4 - 1 = 1111$
- $2^n - 1 = 11111...1$ (n 1)

To find the 1's complement of a number N:

Subtract each digit of N from 1.

Example:
N = 1010
1's complement = 1111 - 1010 = 0101

Number Complements (Complements)

r-1 Complement

If a number N in base r has n digits, its r-1 complement is defined as $(r^n - 1) - N$.

- To find the 9's complement of a number N, subtract each digit of N from 9.
- The 1's complement of a binary number N is obtained by converting ones to zeros and zeros to ones.
- Result:

  The r-1 complement in base 8 and 16 is obtained by subtracting each digit from 7 and F, respectively.

r Complement (decimal)

- Since the number 10^n is represented as 1 followed by n zeros, the r complement is defined as:

- r Complement = 10^n - N
- To compute it:

	1	0	0	0	0	….	0	0
-		dn	dn-1	dn-2	dn-3		d1	d0

- Examples:
- N = 17800 → 10-Complement = 82200
- N = 5352 → 10-Complement = 4648

r Complement (Binary)

- Since the number 2^n is represented as 1 followed by n zeros, the r complement is defined as:

- r Complement = 2^n – N = $((100000 \ldots 00)_2$ (1 followed by n zeros) – N)
- To compute it:

	1	0	0	0	0	….	0	0
-		dn	dn-1	dn-2	dn-3		d1	d0

- Examples:
- 1) N = 10100 → 2-Complement = 1-Complement + 1

  1-Complement = 01011 → 2-Complement = 01100
- 2) N = 10100 → 2-Complement = 01100

r Complement

- Key Points
- First Point:

  If the number N contains a decimal point, temporarily remove it, compute its r and (r-1) complements, and then place the decimal point back in its original relative position.

- Example: The 10’s Complement of the Number 25.46?
- X.Y = 25.46 → XY=N=2546
- 10’s Complement = 10^4 - 2546 = 7454
- 7454 → 74.54

- Second Point:

  The complement of a complement of a number results in the original number:

$$N \xrightarrow{r\text{'s complement}} r^n - N \xrightarrow{r\text{'s complement}} r^n - (r^n - N) = N$$

r Complement

The 10's complement of the number **546700** is:

$$10^6 - 546700 = 1000000 - 546700 = 453300$$

The 9's complement of the number **546700** is:

$$(10^6 - 1) - 546700 = 999999 - 546700 = 453299$$

Another method to calculate the 10's complement of **546700** using the 9's complement:

$$453299 + 1 = 453300$$

Number Complements (Complements)

r Complement

- 2's Complement of 1101100:

$$2^7 - 1101100 = (10000000 - 1101100)_2 = 0010100$$

- 1's Complement of 1101100:

$$(2^7 - 1) - 1101100 = (1111111 - 1101100)_2 = 0010011$$

- Alternative Method to Compute 2's Complement Using 1's Complement:

$$0010011 + 1 = 0010100$$

Why Use Complement Method for subtraction?

- Simplifies subtraction into addition, which is easier for hardware implementation.
- No need for separate subtraction circuits in computers.
- Used in ALUs (Arithmetic Logic Units) and microprocessors.

Subtraction Using Complements

The complement method provides an efficient way to perform subtraction, especially in digital circuits and computer systems. Instead of directly subtracting one number from another, we add the complement of the subtrahend to the minuend. There are two main types of complements used in binary arithmetic:

1. **1's Complement**
2. **2's Complement**

Why Use Complement Method for subtraction?

- Simplifies subtraction into addition, which is easier for hardware implementation.
- No need for separate subtraction circuits in computers.
- Used in ALUs (Arithmetic Logic Units) and microprocessors.

Subtraction Using Complements

The subtraction of two n-digit numbers, expressed as $M - N$ in base r, can be done using the following approach:

1. Add the r's complement of N to the minuend M.
2. If $M \geq N$, the final sum will produce a carry bit, which is ignored. As a result, the remaining value is simply $M - N$.
3. If $M < N$, no carry bit will be generated in the final sum, meaning the result is $r^n - (N - M)$, which is the **r's complement** of $N - M$. To obtain the correct result, we take the usual complement and place a negative sign in front of it.

Subtraction Using Complements

Example: Perform the subtraction **8252 - 3260** using the **10's complement** method.

M – N:

```
  8252
- 3260
--------
   ?
```

10's complement of **3260 =** 6740

M + (r^n – N):

```
  8252
 +6740
--------
 14992
```

M > N:

M – N + (r^n):

r = 10 , n = 4

M – N + (10^4) = 14992

M – N = 4992

Subtraction Using Complements

Example: Perform the subtraction **2532 - 420** using the **10's complement** method.

M – N:

```
  2532
- 0420
--------
   ?
```

10's complement of **0420 =** 9580

M + (r^n – N):

```
  2532
 +9580
--------
 12112
```

M – N + (r^n):

r = 10 , n = 4

M > N:

M – N + (10^4) = 12112

M – N = 2112

Subtraction Using Complements

Example: Perform the subtraction **4550 - 7532** using the **10's complement** method.

M – N:

```
 4550
- 7532
--------
   ?
```

10's complement of **7532** = 2468

M + (r^n– N):

```
 4550
+2468
--------
 7018
```

M < N:

M – N + (r^n) → **r^n – (N – M) =** 7018

→ **r^n – [r^n – (N – M)] =** **10's complement** of 7018 **=** 2982

→ **- (M- N) =** 2982 → **M- N =** - 2982

Subtraction Using Complements

Example: Perform the subtraction 11001 **-** 10011 using the **2's complement** method.

M – N:

```
  11001
- 10011
--------
    ?
```

2's complement of 10011 **=** 01101

M + (r^n – N):

```
  11001
+ 01101
--------
 100110
```

M > N:

M – N + (r^n):

r = 2 , n = 5

M – N + (r^n) = ~~1~~00110

M – N = 00110

Subtraction Using Complements

Example: Perform the subtraction 10011 **-** 11001 using the **2's complement** method.

M – N:

```
  10011
- 11001
--------
    ?
```

→ **2's complement** of **11001** = 00111 →

M + (r^n– N):

```
  10011
+ 00111
--------
  11010
```

M < N:

M – N + (r^n) → $r^n - (N - M) =$ 11010

→ $r^n - [\, r^n - (N - M)\,] =$ **2's complement** of 11010 = 00110

→ **- (M- N) =** 00110 → **M- N =** - 00110

Subtraction Using (r-1)-Complements

Example: Perform the subtraction 8252 **-** 3260 using the **9's complement** method.

M – N:

```
 8252
- 3260
--------
   ?
```

9's complement of 3260 **=** 6739

M + ((r^n–1) - N):

```
  8252
+ 6739
--------
 14991
```

M > N:

M – N + (r^n-1):

r = 10 , n = 4

M – N + (~~r^n~~**) -1 =** ~~1~~4991

M – N - 1 = 4991

M – N = 4991 **+ 1**

M – N = 4992

Subtraction Using (r-1)-Complements

Example: Perform the subtraction 10011 **- 7532** using the **9's complement** method.

M – N:

```
 4550
- 7532
--------
   ?
```

→ **9's complement** of **7532** = 2467 →

M + (r^n– N):

```
  4550
+ 2467
--------
  7017
```

M < N:

M – N + (r^n-1) → (r^n – 1) – (N – M) = 7017

→ (r^n – 1) – [(r^n – 1) – (N – M)] = **9's complement** of 7017 = 2982

→ **- (M- N) =** 2982 → **M- N =** - 2982

Signed Numbers

- The chosen number system for representing values must be capable of displaying both signed and unsigned numbers.

- In mathematics, negative numbers are represented with a minus sign (-), while positive numbers are represented with a plus sign (+); however, in computers, everything must be represented using binary digits.

- Binary numbers can be represented in two ways: signed or unsigned.

- There are three methods for representing signed numbers:
  - ✓ Sign-Magnitude Representation
  - ✓ 1's Complement Representation
  - ✓ 2's Complement Representation

Signed Numbers

In signed numbers using this method, it is common to use **the leftmost bit** as the **sign bit**:

- **0** represents a **positive** number.
- **1** represents a **negative** number.

However, in **unsigned numbers**, the leftmost bit is simply part of the value.

- ✓ The **most significant bit (MSB)** is the leftmost bit and determines the sign in signed numbers.
- ✓ The **least significant bit (LSB)** is the rightmost bit and holds the lowest value in the binary representation.

Signed Numbers

8-bit Unsigned Number System

- **00000000 → 0**
- **11111111 → 255 (2^8 - 1)**

8-bit Signed (Sign-Magnitude) Number System

- **01111111 → +127**
- **00000000 → 0**
- **11111111 → -127** (since the leftmost bit is the sign bit)

Sign Complement Representation Method

- To represent a negative number, the **One's Complement** or **Two's Complement** method is used.
- In this approach, we take the complement (either One's or Two's Complement) of a positive number.
- Since positive numbers always start with **0** on the left, the complement of a positive number will always start with **1** on the left, indicating a negative number.

Signed Numbers

Using an 8-bit representation:

- For +9, there is only one representation: 00001001
- For -9, three different methods exist:
  - ✓ Sign-Magnitude Representation: 10001001
  - ✓ One's Complement Representation: 11110110
  - ✓ Two's Complement Representation: 11110111

Binary Codes

- How to represent characters, colors, etc?
- Define the set of all represented elements
- Assign a unique binary code to each element of the set
- Given n bits, a binary code is a mapping from the set of elements to a subset of the 2^n binary numbers
- Coding Numeric Data (example: coding decimal digits)
  - Coding must simplify common arithmetic operations
  - Tight relation to binary numbers
- Coding Non-Numeric Data (example: coding colors)
  - More flexible codes since arithmetic operations are not applied

Example of Coding Non-Numeric Data

- Suppose we want to code 7 colors of the rainbow
- As a minimum, we need 3 bits to define 7 unique values
- 3 bits define 8 possible combinations
- Only 7 combinations are needed
- Code 111 is not used
- Other assignments are also possible

Color	3-bit code
Red	000
Orange	001
Yellow	010
Green	011
Blue	100
Indigo	101
Violet	110

Minimum Number of Bits Required

- Given a set of M elements to be represented by a binary code, the minimum number of bits, n, should satisfy:

  $2^{(n-1)} < M \leq 2^n$

  $n = \lceil \log_2 M \rceil$ where $\lceil x \rceil$, called the ceiling function, is the integer greater than or equal to x

- How many bits are required to represent 10 decimal digits with a binary code?
- **Answer:** $\lceil \log_2 10 \rceil = 4$ bits can represent 10 decimal digits

Binary Coded Decimal (BCD)

- Simplest binary code for decimal digits
- Only encodes ten digits from 0 to 9
- BCD is a weighted code
- The weights are 8,4,2,1
- Same weights as a binary number
- There are six invalid code words

  1010, 1011, 1100, 1101, 1110, 1111
- Example on BCD coding:

  13 ⇔ $(0001\ 0011)_{BCD}$

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
Unused	1010 ... 1111

Warning: Conversion or Coding?

- Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a binary code
- $13_{10} = (1101)_2$ This is conversion
- $13 \Leftrightarrow (0001\ 0011)_{BCD}$ This is coding
- In general, coding requires more bits than conversion
- A number with *n* decimal digits is coded with 4*n* bits in BCD

Other Decimal Codes

- BCD, 5421, 2421, and 8 4 -2 -1 are weighted codes
- Excess-3 is not a weighted code
- 2421, 8 4 -2 -1, and Excess-3 are self complementary codes

Decimal	BCD 8421	5421 code	2421 code	8 4 -2 -1 code	Excess-3 code
0	0000	0000	0000	0000	0011
1	0001	0001	0001	0111	0100
2	0010	0010	0010	0110	0101
3	0011	0011	0011	0101	0110
4	0100	0100	0100	0100	0111
5	0101	1000	1011	1011	1000
6	0110	1001	1100	1010	1001
7	0111	1010	1101	1001	1010
8	1000	1011	1110	1000	1011
9	1001	1100	1111	1111	1100
Unused	…	…	…	…	…

Character Codes

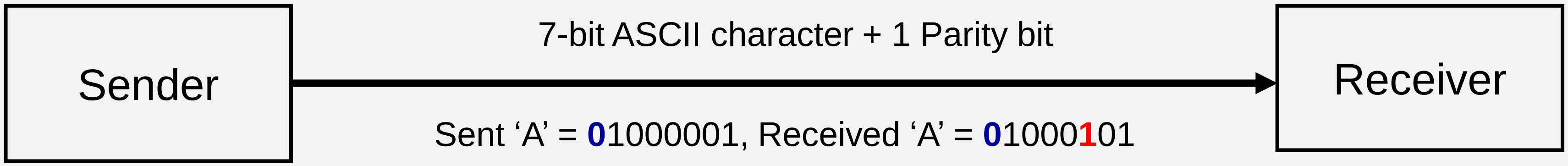
- Character sets
  - Standard ASCII: 7-bit character codes (0 – 127)
  - Extended ASCII: 8-bit character codes (0 – 255)
  - Unicode: 16-bit character codes (0 – 65,535)
  - Unicode standard represents a universal character set
    - Defines codes for characters used in all major languages
    - Each character is encoded as 16 bits
  - UTF-8: variable-length encoding used in HTML
    - Encodes all Unicode characters
    - Uses 1 byte for ASCII, but multiple bytes for other characters
- Null-terminated String
  - Array of characters followed by a NULL character

Character Codes

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th></tr>
<tr><td>2</td><td>space</td><td>!</td><td>"</td><td>#</td><td>$</td><td>%</td><td>&</td><td>'</td><td>(</td><td>)</td><td>*</td><td>+</td><td>,</td><td>-</td><td>.</td><td>/</td></tr>
<tr><td>3</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>:</td><td>;</td><td><</td><td>=</td><td>></td><td>?</td></tr>
<tr><td>4</td><td>@</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td><td>G</td><td>H</td><td>I</td><td>J</td><td>K</td><td>L</td><td>M</td><td>N</td><td>O</td></tr>
<tr><td>5</td><td>P</td><td>Q</td><td>R</td><td>S</td><td>T</td><td>U</td><td>V</td><td>W</td><td>X</td><td>Y</td><td>Z</td><td>[</td><td>\</td><td>]</td><td>^</td><td>_</td></tr>
<tr><td>6</td><td>`</td><td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td><td>g</td><td>h</td><td>i</td><td>j</td><td>k</td><td>l</td><td>m</td><td>n</td><td>o</td></tr>
<tr><td>7</td><td>p</td><td>q</td><td>r</td><td>s</td><td>t</td><td>u</td><td>v</td><td>w</td><td>x</td><td>y</td><td>z</td><td>{</td><td>|</td><td>}</td><td>~</td><td>DEL</td></tr>
</table>

- Examples:
  - ASCII code for space character = 20 (hex) = 32 (decimal)
  - ASCII code for 'L' = 4C (hex) = 76 (decimal)
  - ASCII code for 'a' = 61 (hex) = 97 (decimal)

Detecting Errors



- Suppose we are transmitting 7-bit ASCII characters
- A parity bit is added to each character to make it 8 bits
- Parity can detect all single-bit errors
  - If even parity is used and a single bit changes, it will change the parity to odd, which will be detected at the receiver end
  - The receiver end can detect the error, but cannot correct it because it does not know which bit is erroneous
- Can also detect some multiple-bit errors
  - Error in an odd number of bits

razeghizade@gmail.com

Razeghizade.pudica.ir

CREADITS: This presentation was created by M.Razeghizadeh
Please keep this slide for attribution