Digital Systems and Binary Numbers

ENCS2340 - Digital Systems

Dr. Ahmed I. A. Shawahna
Electrical and Computer Engineering Department
Birzeit University

Presentation Outline

- ❖ **Analog versus Digital Circuits**
- ❖ **Digitization of Analog Signals**
- ❖ Binary Numbers and Number Systems
- ❖ Number System Conversions
- ❖ Representing Fractions
- ❖ Arithmetic Operations
- ❖ Complements of Numbers
- ❖ Signed Binary Numbers
- ❖ Binary Codes

Analog versus Digital

- Analog means continuous
- Analog parameters have continuous range of values
  - Example: temperature is an analog parameter
  - Temperature increases/decreases continuously
  - Other analog parameters?
  - Sound, speed, voltage, current, time
- Digital means discrete using numerical digits
- Digital parameters have fixed set of discrete values
  - Example: month number ∈ {1, 2, 3, …, 12}, month cannot be 1.5!
  - Other digital parameters?
  - Alphabet letters, ten decimal digits, twenty-four hours, sixty minutes

Analog versus Digital System

❖ Are computers analog or digital systems?

Computer are digital systems

❖ Which is easier to design an analog or a digital system?

Digital systems are easier to design, because they deal with a limited set of values rather than an infinitely large range of continuous values

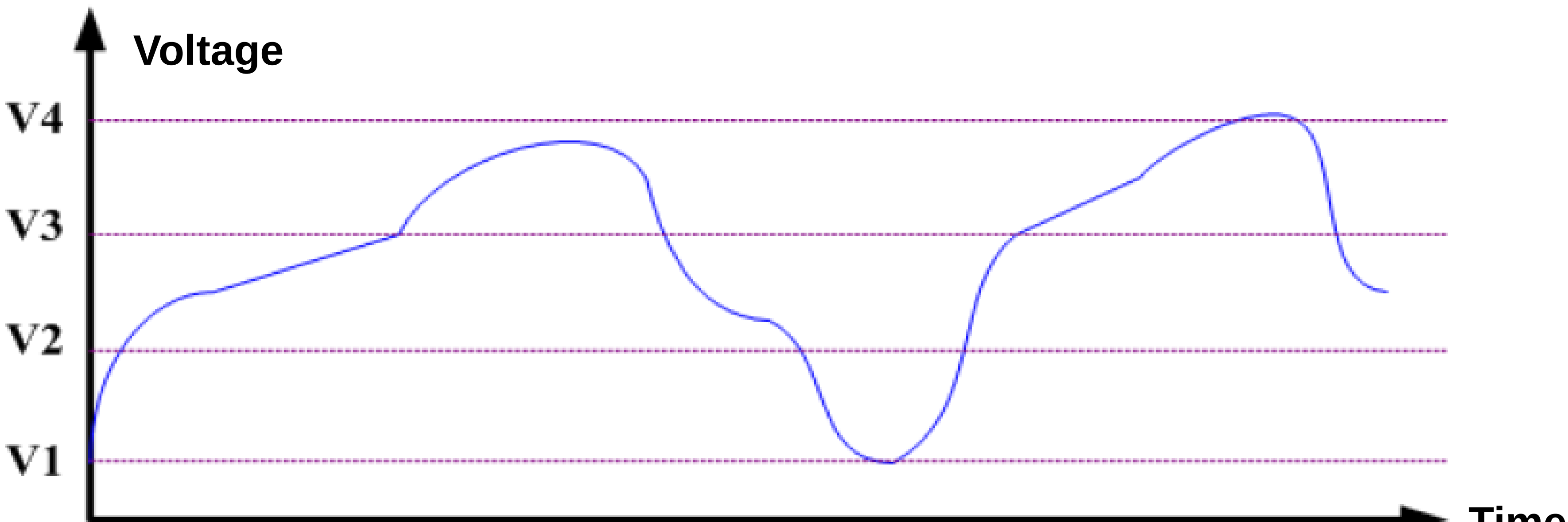
❖ The world around us is analog

❖ It is common to convert analog parameters into digital form

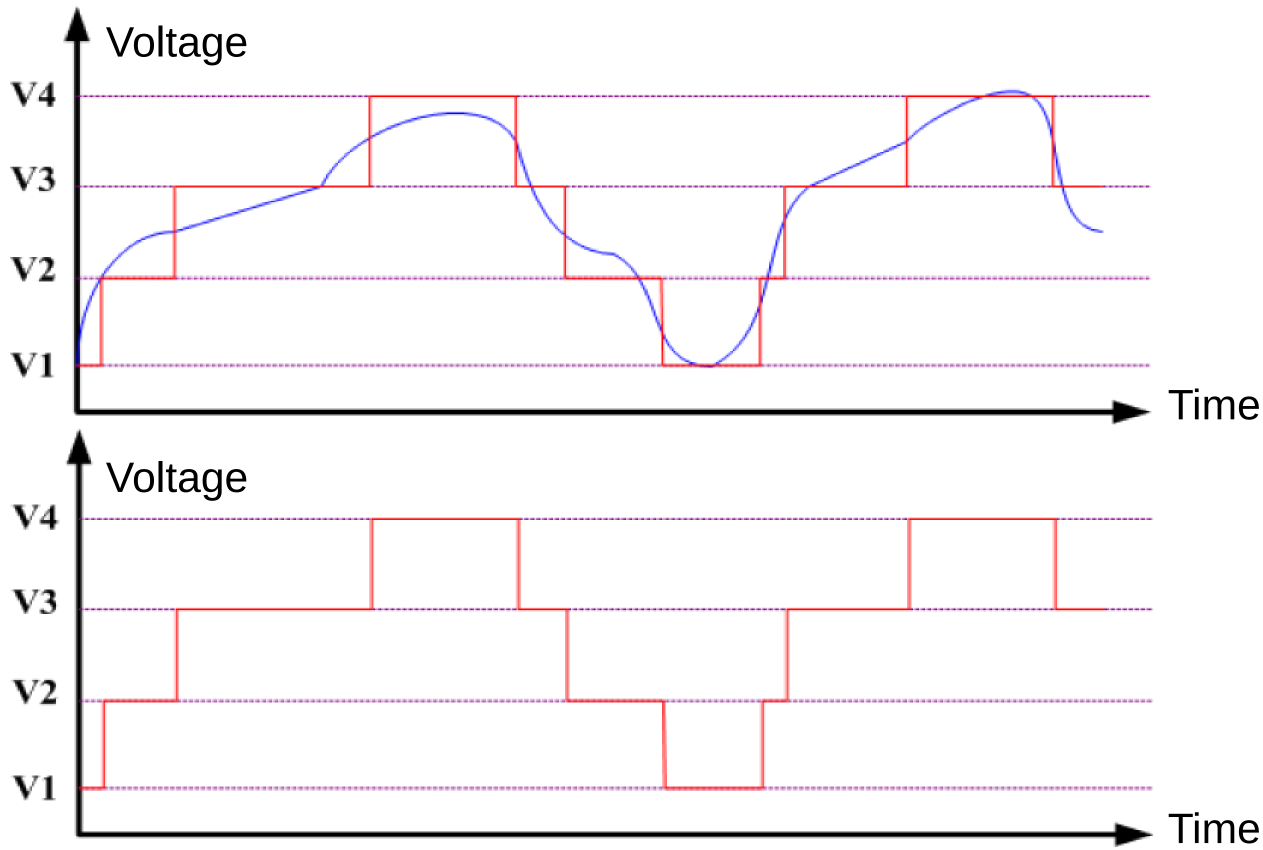
❖ This process is called **digitization**

Digitization of Analog Signals

- Digitization is converting an analog signal into digital form
- Example: consider digitizing an analog voltage signal
- Digitized output is limited to four values = {V1,V2,V3,V4}



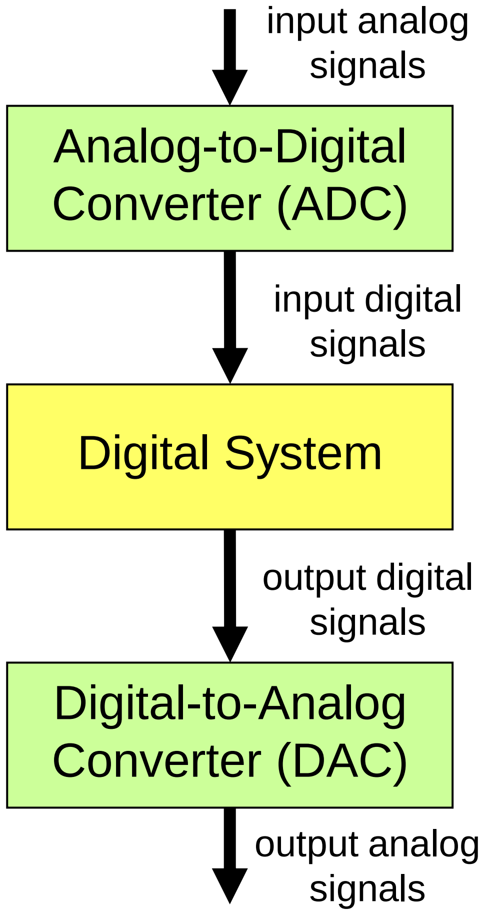
Digitization of Analog Signals – cont'd



- Some loss of accuracy, why?
- How to improve accuracy? Add more voltage values

ADC and DAC Converters

- Analog-to-Digital Converter (ADC)
  - Produces digitized version of analog signals
  - Analog input => Digital output
- Digital-to-Analog Converter (DAC)
  - Regenerate analog signal from digital form
  - Digital input => Analog output
- Our focus is on digital systems only
  - Both input and output to a digital system are digital signals

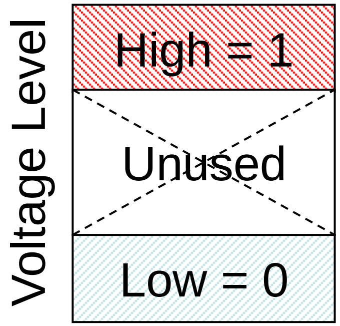


Next . . .

- ❖ Analog versus Digital Circuits
- ❖ Digitization of Analog Signals
- ❖ **Binary Numbers and Number Systems**
- ❖ **Number System Conversions**
- ❖ Representing Fractions
- ❖ Arithmetic Operations
- ❖ Complements of Numbers
- ❖ Signed Binary Numbers
- ❖ Binary Codes

How do Computers Represent Digits?

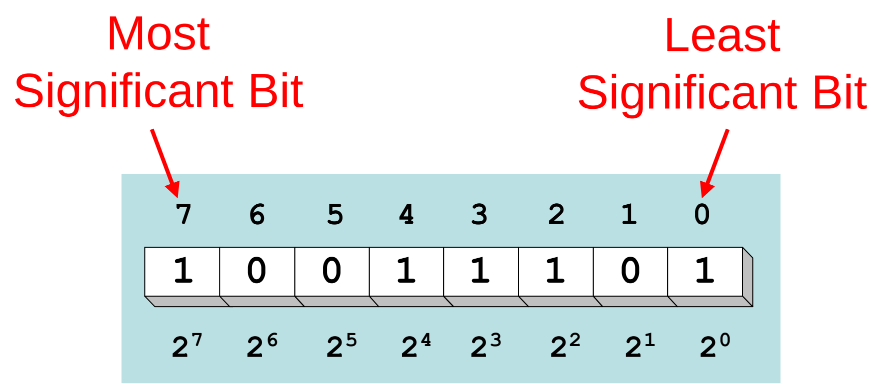
- Binary digits (0 and 1) are the simplest to represent
- Using electric voltage
  - Used in processors and digital circuits
  - High voltage = 1, Low voltage = 0



- 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
  - Encodes binary data in the form of *pits* (0, no light reflection when read) and *lands* (1, due to light reflection when read)

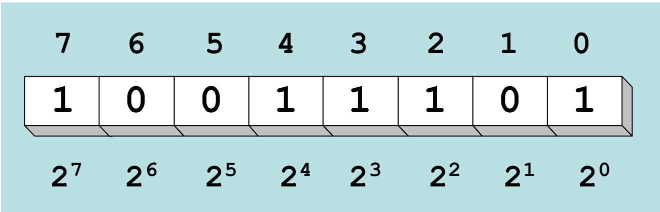
Binary Numbers

- 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

Positional Number Systems

Different Representations of Natural Numbers

XXVII Roman numerals (not positional)

27 Radix-10 or decimal number (positional)

11011_2 Radix-2 or binary number (also positional)

Fixed-radix positional representation with *n* digits

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

N_r Value $= d_{n-1}{\times}r^{\,n-1} + d_{n-2}{\times}r^{\,n-2} + \ldots + d_1{\times}r^{\,1} + d_0{\times}r^{\,0}$

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

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

Positional Number Systems

- **Example:** Show how the value of the decimal number **9375** is estimated.

First Position Index

First Position Index (0)

Position	3	2	1	0
Number	9	3	7	5
Weight	1000	100	10	1
Value	9 x 1000	3x100	7x10	5x1
Value	9000 +	300 +	70 +	5

- **Example:** Convert $(2051)_4$ to decimal.
  - Invalid number in radix-4, only digits values {0, 1, 2, 3} are allowed.

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

- $N = (d_{n-1} \times 2^{n-1}) + \ldots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Dividing N by 2 we first obtain
  - $\text{Quotient}_1 = (d_{n-1} \times 2^{n-2}) + \ldots + (d_2 \times 2^1) + (d_1 \times 2^0)$
  - $\text{Remainder}_1 = d_0$
  - Therefore, first remainder is least significant bit of binary number
- Dividing first quotient by 2 we first obtain
  - $\text{Quotient}_2 = (d_{n-1} \times 2^{n-3}) + \ldots + (d_2 \times 2^0)$
  - $\text{Remainder}_2 = d_1$
- 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 to15) and their values in binary, octal and hex. Memorize table

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: $N_8 = (d_{n-1} \times 8^{n-1}) + ... + (d_1 \times 8) + d_0$
- Hex to Decimal: $N_{16} = (d_{n-1} \times 16^{n-1}) + ... + (d_1 \times 16) + d_0$
- Example: Convert $(4207)_8$ to Decimal.

  $(4207)_8 = (4 \times 8^3) + (2 \times 8^2) + (0 \times 8) + 7 = 2183$

- Example: Convert $(3BA4)_{16}$ to Decimal.

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

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

least significant digit

most significant digit

stop when quotient is zero

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

- 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: $1_2 + 1 = (10)_2$ $\quad$ $7_8 + 1 = (10)_8$

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

- 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 – cont'd

❖ 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 $r^n - 1$

Next . . .

- ❖ Analog versus Digital Circuits
- ❖ Digitization of Analog Signals
- ❖ Binary Numbers and Number Systems
- ❖ Number System Conversions
- ❖ **Representing Fractions**
- ❖ Arithmetic Operations
- ❖ Complements of Numbers
- ❖ Signed Binary Numbers
- ❖ Binary Codes

Representing Fractions

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

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

Radix Point

- The number N_r represents the value:

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

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

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

Examples of Numbers with Fractions

- $(2409.87)_{10}$ $= 2\times10^3 + 4\times10^2 + 9\times10^0 + 8\times10^{-1} + 7\times10^{-2}$
- $(1101.1001)_2$ $= 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-4} = 13.5625$
- $(703.64)_8$ $= 7\times8^2 + 3\times8^0 + 6\times8^{-1} + 4\times8^{-2} = 451.8125$
- $(A1F.8)_{16}$ $= 10\times16^2 + 1\times16^1 + 15\times16^0 + 8\times16^{-1} = 2591.5$
- $(423.1)_5$ $= 4\times5^2 + 2\times5^1 + 3\times5^0 + 1\times5^{-1} = 113.2$
- $(263.5)_6$ 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
$0.6875 \times 2 = 1.375$	0.375	1
$0.375 \times 2 = 0.75$	0.75	0
$0.75 \times 2 = 1.5$	0.5	1
$0.5 \times 2 = 1.0$	0.0	1

First fraction bit

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 $d_{-2}\ d_{-3}$ ...
- 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 →

Octal	7 2 6 1 3 . 2 4 7 4 5 2
Binary	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 1
Hexadecimal	7 5 8 B . 5 3 C A 8

- ❖ 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 = $1 - 16^{-m}$

More Conversion Examples

- Convert $(299.8195)_{10}$ to $()_{12}$ using **at most** *two fractional digits*, if necessary.
- Solution: $N = 299 + 0.8195$ (split integer from fraction)
- The integer and fraction parts are converted separately

Division	Quotient	Remainder
299 / 12	24	B
24 / 12	2	0
2 / 12	0	2

Multiplication	New Fraction	Digit
0.8195 × 12 = 9.834	0.834	9
0.834 × 12 = 10.008	0.008	A

- Therefore, $299 = (20B)_{12}$ and $0.8195 = (0.9A)_{12}$
- Now, join the integer and fraction parts with radix point

  $N = 299.8195 = (20B.9A)_{12}$

Next . . .

- ❖ Analog versus Digital Circuits
- ❖ Digitization of Analog Signals
- ❖ Binary Numbers and Number Systems
- ❖ Number System Conversions
- ❖ Representing Fractions
- ❖ **Arithmetic Operations**
- ❖ Complements of Numbers
- ❖ Signed Binary Numbers
- ❖ Binary Codes

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 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	

Subtracting Bits

❖ Subtracting two 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 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 Multiplication

❖ Binary Multiplication table is simple:

0×0=0, 0×1=0, 1×0=0, 1×1=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 × *m*-bit multiplier = (*n*+*m*)-bit product

❖ Accomplished via shifting and addition

Shifting the Bits to the Left

❖ 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

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**

Hexadecimal Addition

- 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

Hexadecimal Subtraction

- 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

Next . . .

- ❖ Analog versus Digital Circuits
- ❖ Digitization of Analog Signals
- ❖ Binary Numbers and Number Systems
- ❖ Number System Conversions
- ❖ Representing Fractions
- ❖ Arithmetic Operations
- ❖ **Complements of Numbers**
- ❖ Signed Binary Numbers
- ❖ Binary Codes

Complements of Numbers

- Complements are used to **simplify the subtraction operation** and for easy manipulation of certain logical rules and events
- Two types of complements for each *base-r* system:
  - Diminished radix complements ((r – 1)'s complements)
  - Radix complements (r's complements)
- Diminished radix complement
  - Given a number ***N*** in base ***r*** having ***n*** digits, the **(r – 1)'s** complement of ***N*** is defined as **$(r^n – 1) – N$**
- Radix complement
  - Given a number ***N*** in base ***r*** having ***n*** digits, the **r's** complement of ***N*** is defined as **$r^n – N$** ➔ **(r – 1)'s** complement **plus 1**

Diminished Radix Complements

- The (r – 1)'s complement of an *n*-digit number *N* is defined as **$(r^n - 1) - N$**
- For **decimal** (*r* = 10) number *N*, *n* = 6, 9's complement
  - 9's complement of 546700 = 999999 – 546700 = 453299
  - 9's complement of 012398 = 999999 – 012398 = 987601
- For **binary** (*r* = 2) number *N*, *n* = 7, 1's complement
  - 1's complement of 1011000 = 1111111 – 1011000 = 0100111
  - 1's complement of 0101101 = 1111111 – 0101101 = 1010010
  - **Formed by changing 1's to 0's and 0's to 1's**
- For **octal** (*r* = 8) number *N*, *n* = 5, 7's complement
  - 7's complement of 15372 = 77777 – 15372 = 62405
  - 7's complement of 01746 = 77777 – 01746 = 76031

Radix Complements

- The r's complement of an *n*-digit number *N* is defined as **(r^n – N, for $N \neq 0$ and 0 for $N = 0$)**
- For **decimal** (r = 10) number *N*, n = 6, 10's complement
  - 10's complement of 546700 = 1000000 – 546700 = 453300
  - 10's complement of 012398 = 1000000 – 012398 = 987602
- For **binary** (r = 2) number *N*, n = 7, 2's complement
  - 2's complement of 1011000 = 10000000 – 1011000 = 0101000
  - 2's complement of 0101101 = 10000000 – 0101101 = 1010011
  - **The 2's complement of *N* = (1's complement of *N*) + 1**
- For **octal** (r = 8) number *N*, n = 5, 8's complement
  - 8's complement of 15372 = 100000 – 15372 = 62406
  - 8's complement of 01746 = 100000 – 01746 = 76032

Computing the 2's Complement - Binary Value

starting value	00100100_2
step1: Invert the bits (1's complement)	11011011_2
step 2: Add 1 to the value from step 1	$+ \quad 1_2$
sum = 2's complement representation	11011100_2

2's complement of $11011100_2 = 00100011_2 + 1 = 00100100_2$

The 2's complement of the 2's complement of *A* is equal to *A*

Another way to obtain the 2's complement:

Start at the least significant 1

Leave all the 0s to its right unchanged

Complement all the bits to its left

Binary Value

= $00100\boxed{1}00$ least significant 1

2's Complement

= $11011\boxed{1}00$

Remarks - Complements of Numbers

- The **complement of the complement** restores the number to its original value
  - The (r – 1)'s complement of *N* is $(r^n - 1) - N$, so that the complement of the complement is $(r^n - 1) - [(r^n - 1) - N] = N$
  - The r's complement of *N* is $r^n - N$, so that the complement of the complement is $r^n - (r^n - N) = N$
- If there is a radix point, the radix point is temporarily removed during the complement process. Then, it is restored to the complemented number in the same relative position
  - 1's complement of 10110.00 is 01001.11
  - 2's complement of 0101.101 is 1010.011

Subtraction with r's Complements

- Replace subtraction with addition
- The subtraction of two *n*-digit unsigned numbers (*M* – *N*) in base *r* can be done as follows
  - Compute the r's complement of N, i.e., $r^n - N$
  - Add M to the r's complement of N, i.e., $M - N = M + (r^n - N)$
- If $M \geq N$, the sum ($M - N + r^n$) will produce an end carry r^n, which can be discarded; what is left is the result $M - N$
- If $M < N$, the sum ($r^n - (N - M)$) does not produce an end carry, and it is equal to the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Examples - Subtraction with r's Complements

❖ Using 10's complement do 72532 – 3250

```
        72532
+       96750 → 10's comp of 3250
      1 69282
Answer = 69282
```

❖ Using 10's complement do 3250 – 72532

```
        03250
+       27468 → 10's comp of 72532
        30718 → no end carry
Answer = –(10's comp of 30718) = – 69282
```

Examples - Subtraction with r's Complements

- Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction **(a)** X – Y and **(b)** Y – X by using the 2's complements.

(a)

$$\begin{array}{rr} X = & 1010100 \\ \text{2's complement of } Y = + & \underline{0111101} \\ \text{Sum} = & 10010001 \\ \text{Discard end carry } 2^7 = - & \underline{10000000} \\ \textit{Answer: } X - Y = & 0010001 \end{array}$$

(b)

$$\begin{array}{rr} Y = & 1000011 \\ \text{2's complement of } X = + & \underline{0101100} \\ \text{Sum} = & 1101111 \end{array}$$

There is no end carry. Therefore, the answer is – (2's complement of 1101111) = – 0010001

Subtraction with (r – 1)'s Complements

- The subtraction of two *n*-digit unsigned numbers (*M* – *N*) in base *r* can be done as follows
  - Compute the (r – 1)'s complement of N, i.e., $(r^n – 1) - N$
  - Add M to the (r – 1)'s complement of N,
    i.e., $M – N = M + ((r^n – 1) – N)$
- If M ≥ N, the sum ($M – N + r^n – 1$) will produce an end carry r^n, remove the end carry and adding 1 to the sum (end-around carry); what is left is the result M – N
- If M < N, the sum ($(r^n – 1) – (N – M)$) does not produce an end carry, and it is equal to the (r – 1)'s complement of (N – M). To obtain the answer in a familiar form, take the (r – 1)'s complement of the sum and place a negative sign in front.

Examples - Subtraction with (r – 1)'s Complements

❖ Using 9's complement do 72532 – 3250

```
      72532
  +   96749 → 9's comp of 3250
    1 69281
  +       1 → end around carry
      69282
```

❖ Using 9's complement do 3250 – 72532

```
     03250
 +   27467 →     9's comp of 72532
     30717 → –(9's comp of 30717) = –69282
```

Examples - Subtraction with (r – 1)'s Complements

❖ Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction **(a)** X – Y and **(b)** Y – X by using the 1's complements.

(a)

$$
\begin{aligned}
X &= \quad 1010100 \\
\text{1's complement of } Y &= +\ \underline{0111100} \\
\text{Sum} &= \quad 10010000 \\
\text{End-around carry} &= +\ \underline{\qquad\quad 1} \\
\textit{Answer}: X - Y &= \quad 0010001
\end{aligned}
$$

(b)

$$
\begin{aligned}
Y &= \quad 1000011 \\
\text{1's complement of } X &= +\ \underline{0101011} \\
\text{Sum} &= \quad 1101110
\end{aligned}
$$

There is no end carry. Therefore, the answer is – (1's complement of 1101110) = – 0010001

Next . . .

- ❖ Analog versus Digital Circuits
- ❖ Digitization of Analog Signals
- ❖ Binary Numbers and Number Systems
- ❖ Number System Conversions
- ❖ Representing Fractions
- ❖ Arithmetic Operations
- ❖ Complements of Numbers
- ❖ **Signed Binary Numbers**
- ❖ Binary Codes

Signed Binary Numbers

- Several ways to represent a signed number
  - Sign-Magnitude
  - 1's complement
  - 2's complement
- Divide the range of values into two parts
  - First part corresponds to the positive numbers (≥ 0)
  - Second part correspond to the negative numbers (< 0)
- The 2's complement representation is widely used
  - Has many advantages over other representations
  - Standard way to represent signed integers in computers

Sign-Magnitude Representation

n-bit number

Sign Bit	bit *n*-2	. . .	bit 2	bit 1	bit 0

Magnitude = *n* – 1 bits

- Independent representation of the sign and magnitude
- Leftmost bit is the sign bit: 0 is positive and 1 is negative
- Using *n* bits, largest represented magnitude = $2^{n-1} - 1$

Sign-magnitude
8-bit representation of +45

0	0	1	0	1	1	0	1

Sign-magnitude
8-bit representation of -45

1	0	1	0	1	1	0	1

Properties of Sign-Magnitude

❖ Symmetric range of represented values:

For *n*-bit register, range is from $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$

For example, if n = 8 bits then range is -127 to +127

❖ Two representations for zero: +0 and -0 **NOT Good!**

❖ Two circuits are needed for addition & subtraction **NOT Good!**

- ✧ In addition to an adder, a second circuit is needed for subtraction
- ✧ Sign and magnitude parts should be processed independently
- ✧ Sign bit should be examined to determine addition or subtraction
- ✧ Addition of numbers of different signs is converted into subtraction
- ✧ Increases the cost of the add/subtract circuit

Sign-Magnitude Addition / Subtraction

Eight cases for Sign-Magnitude Addition / Subtraction

Operation	ADD Magnitudes	Subtract Magnitudes	
		A >= B	A < B
(+A) + (+B)	+(A+B)		
(+A) + (-B)		+(A–B)	-(B–A)
(-A) + (+B)		-(A–B)	+(B–A)
(-A) + (-B)	-(A+B)		
(+A) – (+B)		+(A–B)	-(B–A)
(+A) – (-B)	+(A+B)		
(-A) – (+B)	-(A+B)		
(-A) – (-B)		-(A–B)	+(B–A)

Signed 1's Complement Representation

- The leftmost bit indicates the sign. Positive numbers starts with 0 (sign-bit = 0), while negative number starts with 1 (sign-bit = 1)
- The signed-1's-complement system negates a number by taking its 1's complement (1's complement of *A* is the **negative of *A***)
- Example: Consider the number **9**, represented in binary with 8 bits
  signed-1's-complement representation (+9): $(00001001)_2$
  signed-1's-complement representation (-9): $(11110110)_2$
- Range of values is $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$
  For example, if n = 8 bits, range is -127 to +127
- Two representations for zero: +0 and -0 **NOT Good!**
  1's complement of $(0\ldots000)_2 = (1\ldots111)_2$
  $-0 = (1\ldots111)_2$ **NOT Good!**

Signed 2's Complement Representation

- The leftmost bit indicates the sign. Positive numbers starts with 0 (sign-bit = 0), while negative number starts with 1 (sign-bit = 1)
- The signed-2's-complement system negates a number by taking its 2's complement (2's complement of *A* is the **negative of *A***)
- Example: Consider the 8-bit number A = $(00101100)_2$ = +44
  2's complement of *A* = $(11010100)_2$ = -44
- Range of represented values: -2^{n-1} to $+(2^{n-1} - 1)$
  For example, if *n* = 8 bits then range is -128 to +127
- There is only **one zero** = $(0…000)_2$ (all bits are zeros)
- The 2's complement representation assigns a negative weight to the sign bit (most-significant bit)

1	0	1	1	0	1	0	0
-128	64	32	16	8	4	2	1

-128 + 32 + 16 + 4 = -76

Values of Different Representations

8-bit Binary Representation	Unsigned Value	Sign Magnitude Value	1's Complement Value	2's Complement Value
00000000	0	+0	+0	0
00000001	1	+1	+1	+1
00000010	2	+2	+2	+2
. . .	. . .	. . .	. . .	. . .
01111101	125	+125	+125	+125
01111110	126	+126	+126	+126
01111111	127	+127	+127	+127
10000000	128	-0	-127	-128
10000001	129	-1	-126	-127
10000010	130	-2	-125	-126
. . .	. . .	. . .	. . .	. . .
11111101	253	-125	-2	-3
11111110	254	-126	-1	-2
11111111	255	-127	-0	-1

Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Signed 2's Complement Addition

- The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits. A carry out of the sign-bit position is discarded
- In order to obtain a correct answer, we must ensure that the result has a sufficient number of bits to accommodate the sum
- If we start with two n-bit numbers and the sum occupies n + 1 bits, we say that an overflow occurs
  - **Solution**: add another 0 to a positive number or another 1 to a negative number in the *most significant position* to extend the number to n + 1 bits (**sign extension**) and then perform the addition

Converting Subtraction into Addition

- When computing **A – B**, convert **B** to its 2's complement

  A – B = A + (2's complement of B)

- **Same adder** is used for **both addition and subtraction**

  This is the biggest advantage of 2's complement

```
borrow:    -1 -1     -1            carry:  1  1     1  1
         0 1 0 0 1 1 0 1                 0 1 0 0 1 1 0 1
       – 0 0 1 1 1 0 1 0       ⇨       + 1 1 0 0 0 1 1 0    (2's complement)
         ---------------                 ---------------
         0 0 0 1 0 0 1 1                 0 0 0 1 0 0 1 1    (same result)
```

- Final carry is **ignored**, because

  A + (2's complement of B) = A + (2^n – B) = (A – B) + 2^n

  Final carry = 2^n, for n-bit numbers

Carry versus Overflow

- Carry is important when …
  - Adding (or subtracting) **unsigned integers**
  - Indicates that the **unsigned sum** is out of range
  - Sum > maximum (or Sum < maximum) unsigned *n*-bit value
- Overflow is important when …
  - Adding or subtracting **signed integers**
  - Indicates that the **signed sum** is out of range
- Overflow occurs when …
  - Adding two positive numbers and the sum is negative
  - Adding two negative numbers and the sum is positive
- Simplest way to detect Overflow: $\boldsymbol{V = C_{n-1} \oplus C_n}$
  - $\boldsymbol{C_{n-1}}$ and $\boldsymbol{C_n}$ are the carry-in and carry-out of the most-significant bit

Carry and Overflow Examples

- We can have carry without overflow and vice-versa
- Four cases are possible (Examples on 8-bit numbers)

				1					
	0	0	0	0	1	1	1	1	15
+	0	0	0	0	1	0	0	0	8
	0	0	0	1	0	1	1	1	23

Carry = 0 Overflow = 0

1	1	1	1	1					
	0	0	0	0	1	1	1	1	15
+	1	1	1	1	1	0	0	0	248 (-8)
	0	0	0	0	0	1	1	1	7

Carry = 1 Overflow = 0

	1								
	0	1	0	0	1	1	1	1	79
+	0	1	0	0	0	0	0	0	64
	1	0	0	0	1	1	1	1	143 (-113)

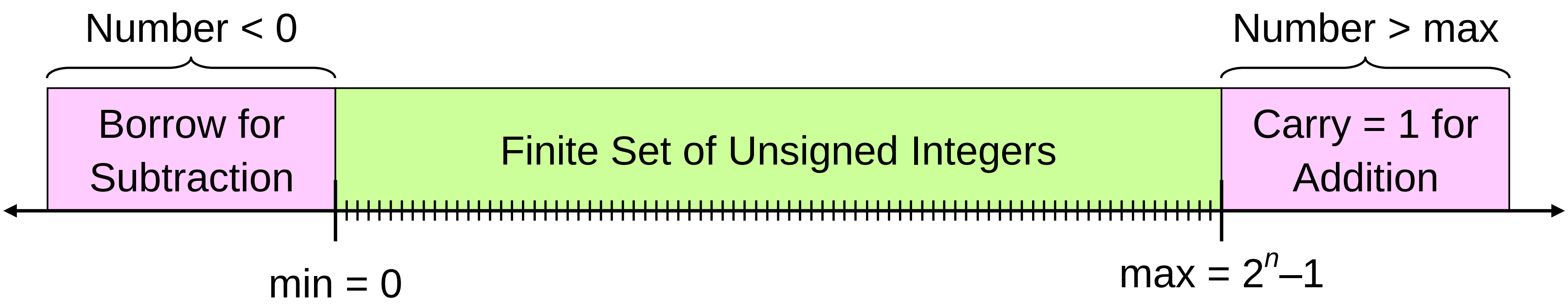
Carry = 0 Overflow = 1

1			1	1					
	1	1	0	1	1	0	1	0	218 (-38)
+	1	0	0	1	1	1	0	1	157 (-99)
	0	1	1	1	0	1	1	1	119

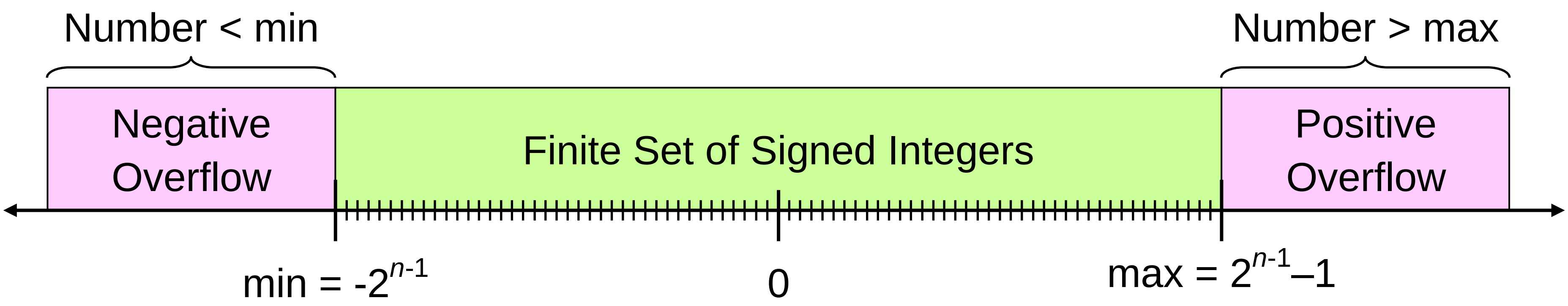
Carry = 1 Overflow = 1

Range, Carry, Borrow, and Overflow

❖ Unsigned Integers: *n*-bit representation



❖ Signed Integers: 2's complement representation



Exercise - Signed Binary Numbers

1. The decimal value for the signed 2's complement binary value (10110) is -9?

(Single Choice) *

- True
- False

Next . . .

- ❖ Analog versus Digital Circuits
- ❖ Digitization of Analog Signals
- ❖ Binary Numbers and Number Systems
- ❖ Number System Conversions
- ❖ Representing Fractions
- ❖ Arithmetic Operations
- ❖ Complements of Numbers
- ❖ Signed Binary Numbers
- ❖ **Binary Codes**

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 least 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

Decimal Codes

- ❖ Binary number system is most natural for computers
- ❖ But people are used to the decimal number system
- ❖ Must convert decimal numbers to binary, do arithmetic on binary numbers, then convert back to decimal
- ❖ To simplify conversions, decimal codes can be used
- ❖ Define a binary code for each decimal digit
- ❖ Since 10 decimal digits exit, a 4-bit code is used
- ❖ But a 4-bit code gives 16 unique combinations
- ❖ 10 combinations are used and 6 will be unused

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 $\Leftrightarrow$ $(0001\ 0011)_{BCD}$

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 ($4 \times n$) bits in BCD

Exercise - Binary Coded Decimal (BCD)

1. **Counting the number of days in a month in binary requires (how many?)** Answer 1 **bits, whereas counting the same in BCD requires (how many?)** Answer 2 **bits.** (Fill in the blank)

Answer 1

Type your answer

Answer 2

Type your answer

BCD Addition

- We use binary arithmetic to add the BCD digits

```
  1000          8
+ 0101        + 5
------        -------
  1101         13 (>9)
```

- If the result is more than 9, it must be corrected to use 2 digits
- To correct the digit, add 6 to the digit sum

```
  1000                          8
+ 0101                        + 5
------                        ---------------
  1101                         13 (>9)
+ 0110                        + 6 (add 6)
------                        ---------------
1 0011  ←  Final answer        19 (carry + 3)
           in BCD
```

Exercise - BCD Addition

1. Decode the decimal variables A = 23 and B = 17 to binary using the BCD code. Then, find the summation of their BCD codes (i.e., A (BCD) + B (BCD) = ?? (BCD))

(Short Answer)

Short answer (9 characters)

Multiple Digit BCD Addition

Add: 2905 + 1897 in BCD

Showing carries and digit corrections

carry	+1	+1	+1	
	0010	1001	0000	0101
+	0001	1000	1001	0111
	0100	10010	1010	1100
digit correction		0110	0110	0110
	0100	1000	0000	0010

Final answer: 2905 + 1897 = 4802

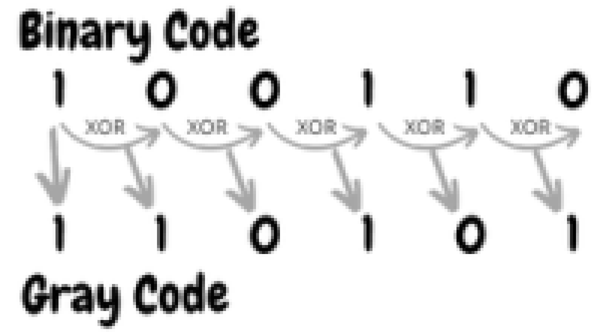
Gray Code (Reflected Binary Code)

- Two successive values differ in only one bit (binary digit)
- For example, in going from 7 to 8,
  - The Gray code changes from <u>0</u>100 to **<u>1</u>**100
  - By contrast, with binary numbers the change from 7 to 8 will be from <u>0111</u> to **<u>1000</u>**
- Very useful in the normal sequence of binary numbers generated by the hardware that may cause an error or ambiguity during the transition from one number to the next

Decimal	Gray	Binary
00	0000	0000
01	0001	0001
02	0011	0010
03	0010	0011
04	0110	0100
05	0111	0101
06	0101	0110
07	0100	0111
08	1100	1000
09	1101	1001
10	1111	1010
11	1110	1011
12	1010	1100
13	1011	1101
14	1001	1110
15	1000	1111

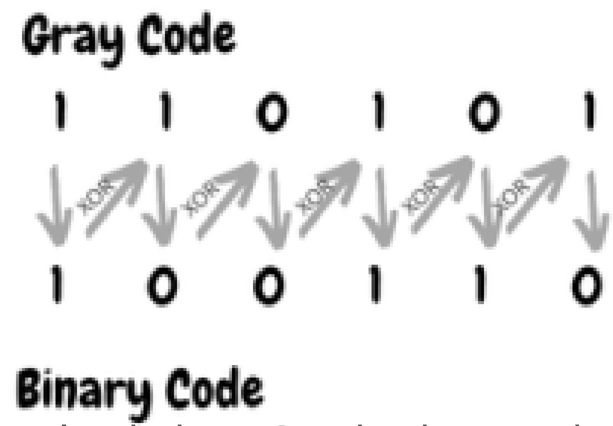
Conversion between Gray Code and Binary

- **From Binary to Gray:**
  - The most significant bit (MSB) of the Gray code is always equal to the MSB of the given Binary code
  - Other bits of the output Gray code can be obtained by **XORing** binary code bit at the index and previous index



A	0	0	1	1
B	0	1	0	1
A* XOR *B	0	1	1	0

- **From Gray to Binary:**
  - The MSB of the binary code is always equal to the MSB of the given binary
  - Other bits of the output binary code can be obtained by checking gray code bit at that index. If current gray code bit is 0, then copy previous binary code bit, else copy invert of previous binary code bit



Exercise - Gray Code

1. The Gray code for the binary value (100110) is Answer 1 , and this code and the Gray code (110111) Answer 2 (are/are not) successive codes. (Fill in the blank)

Answer 1

Type your answer

Answer 2

Type your answer

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	…	…	…	…	…

Exercise - Excess-3 Code

1. **The decimal number 17 can be represented in binary as** Answer 1 **and in Excess-3 as** Answer 2 **.** (Fill in the blank)

Answer 1

Type your answer

Answer 2

Type your answer

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)
    - Defines codes for characters used in all major languages
    - Used in Windows-XP, each character is encoded as 16 bits
    - Arabic codes: from $(0600)_{16}$ to $(06FF)_{16}$
  - 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

Printable ASCII Codes

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

❖ Examples:

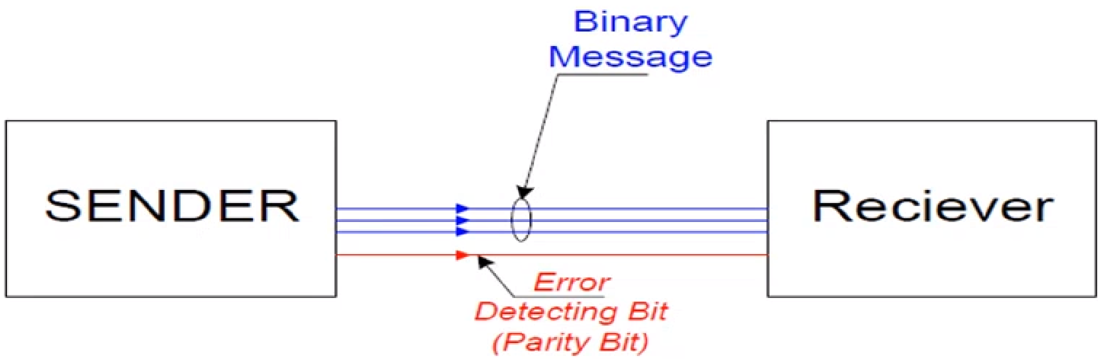
- ASCII code for space character = 20 (hex) = 32 (decimal)
- ASCII code for ‘A' = 41 (hex) = 65 (decimal)
- ASCII code for 'a' = 61 (hex) = 97 (decimal)

Control Characters

- The first 32 characters of ASCII table are used for control
- Control character codes = 00 to 1F (hexadecimal)
  - Not shown in previous slide
- Examples of Control Characters
  - Character 00 is the NULL character ⇒ used to terminate a string
  - Character 09 is the Horizontal Tab (HT) character
  - Character 0A (hex) = 10 (decimal) is the Line Feed (LF)
  - Character 0D (hex) = 13 (decimal) is the Carriage Return (CR)
  - The LF and CR characters are used together
    - They advance the cursor to the beginning of next line
- One control character appears at end of ASCII table
  - Character 7F (hex) is the Delete (DEL) character

Parity Bit & Error Detection Codes

- Binary data are typically transmitted between computers
  - Because of noise, a corrupted bit will change value
  - To detect errors, extra bits are added to each data value
- Parity bit: is used to make the number of 1's odd or even
  - Even parity: number of 1's in the transmitted data is even
  - Odd parity: number of 1's in the transmitted data is odd



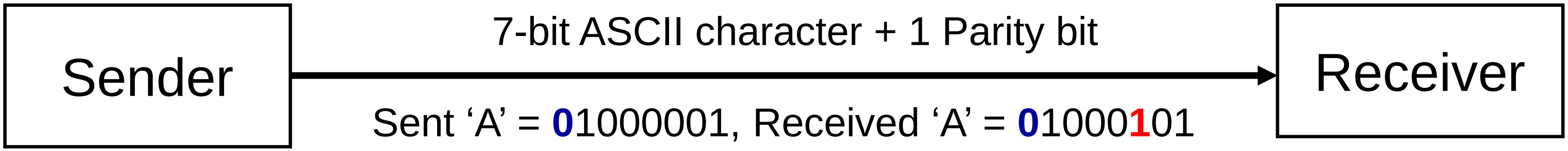
7-bit ASCII Character	With Even Parity	With Odd Parity
'A' = 1000001	**0** 1000001	**1** 1000001
'T' = 1010100	**1** 1010100	**0** 1010100

Exercise - Parity Bit & Error Detection

1. A communication system uses a 1-bit parity scheme for error detection. The receiver receives a byte represented in hexadecimal as "D3" without error. The parity scheme used is __________ (even/odd) parity. (Single Choice) *

- even
- odd

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

Exercise - Detecting Errors

1. If you type the word "BC" on your keyboard, what is the binary sequence sent to the computer using 8-bit ASCII with the 8th most-significant bit being an odd parity bit. Note that the 7-bit ASCII code of "A" in hexadecimal is 41. (Short Answer)

Short answer (20 characters)