Digital Systems

COE 202

Digital Logic Design

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

Welcome to COE 202

- Course Webpage:

  http://faculty.kfupm.edu.sa/coe/mudawar/coe202/

- Lecture Slides:

  http://faculty.kfupm.edu.sa/coe/mudawar/coe202/lectures/

- Assignments:

  http://faculty.kfupm.edu.sa/coe/mudawar/coe202/assignments.htm

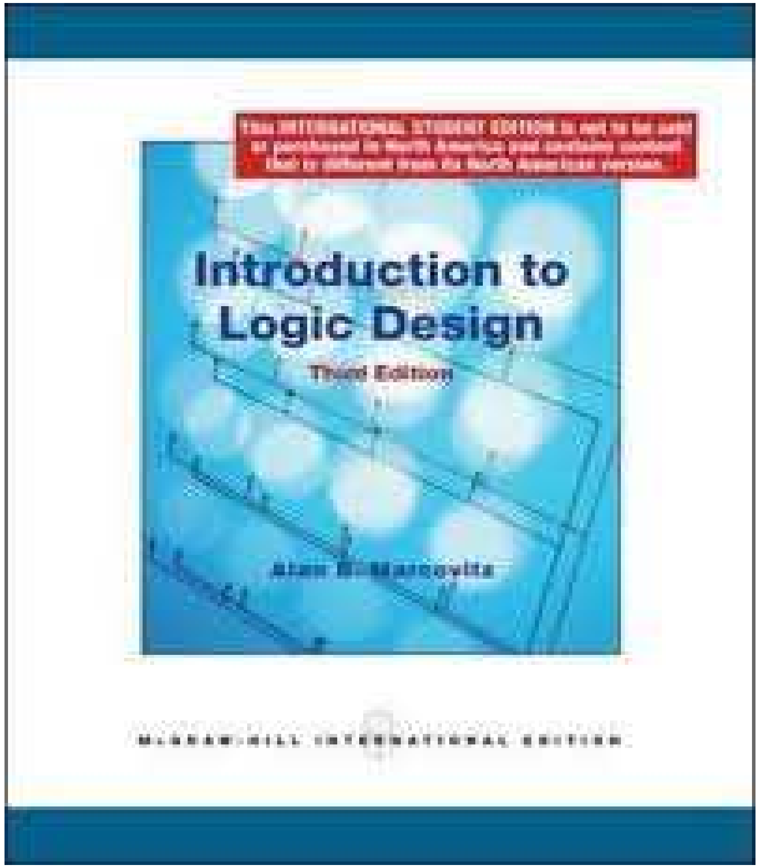
- Blackboard:

  https://blackboard.kfupm.edu.sa/

Which Book will be Used?

- Introduction to Logic Design

- Alan B. Marcovitz
  - Third Edition
  - McGraw Hill
  - 2010



What you will I Learn in this Course?

- Towards the end of this course, you should be able to:
  - Carry out arithmetic computation in various number systems
  - Apply rules of Boolean algebra to simplify Boolean expressions
  - Translate truth tables into equivalent Boolean expressions and logic gate implementations and vice versa
  - Design efficient combinational and sequential logic circuit implementations from functional description of digital systems
  - Use software tools to simulate and verify the operation of logic circuits

Is it Worth the Effort?

- Absolutely!
- Digital circuits are employed in the design of:
  - Digital computers
  - Data communication
  - Digital phones
  - Digital cameras
  - Digital TVs, etc.
- This course provides the fundamental concepts and the basic tools for the design of digital circuits and systems

Grading Policy

❖ Assignments & Quizzes	20%
❖ Midterm Exam I	20%
❖ Midterm Exam II	25%
❖ Final Exam	35%

- NO makeup exam will be given whatsoever

Presentation Outline

- Analog versus Digital Systems
- Digitization of Analog Signals
- Binary Numbers and Number Systems
- Number System Conversions
- Representing Fractions
- 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
  - Like a continuous mathematical function, No discontinuity points
  - Other examples?
- Digital means using numerical digits
- Digital parameters have fixed set of discrete values
  - Example: month number ∈ {1, 2, 3, …, 12}
  - Thus, the month number is a digital parameter (cannot be 1.5!)
  - Other examples?

Analog versus Digital System

- Are computers analog or digital systems?

  Computer are digital systems

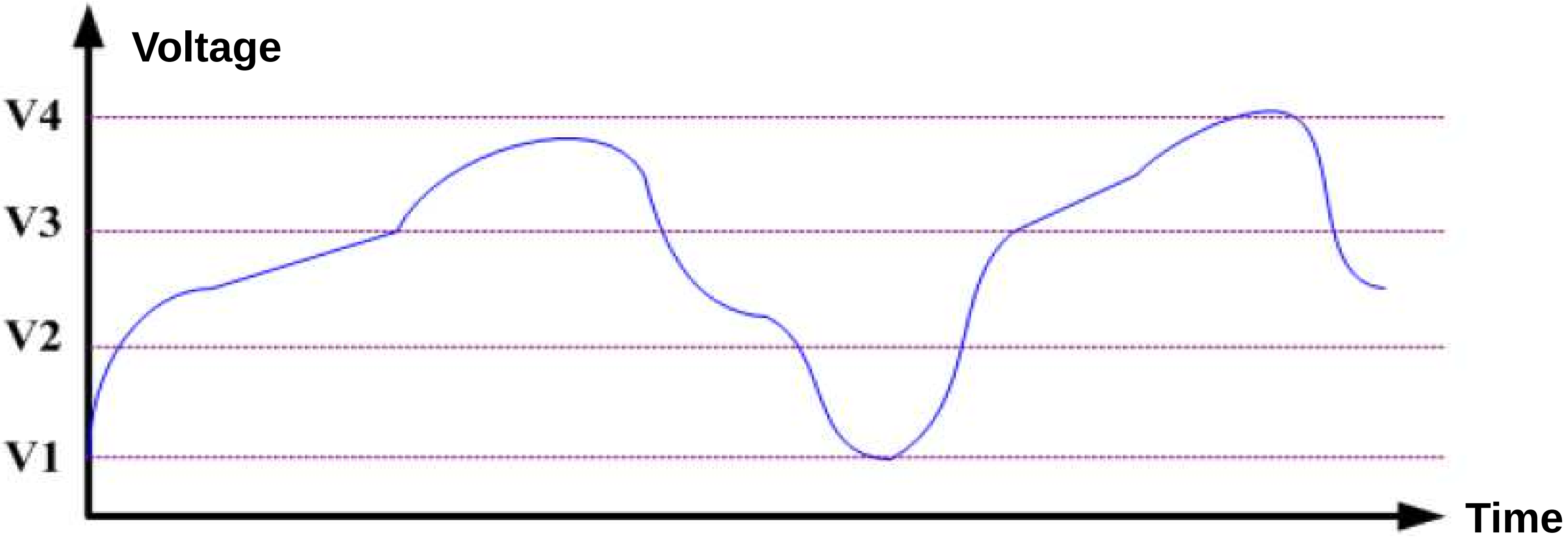
- Which is easier to design an analog or a digital system?

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

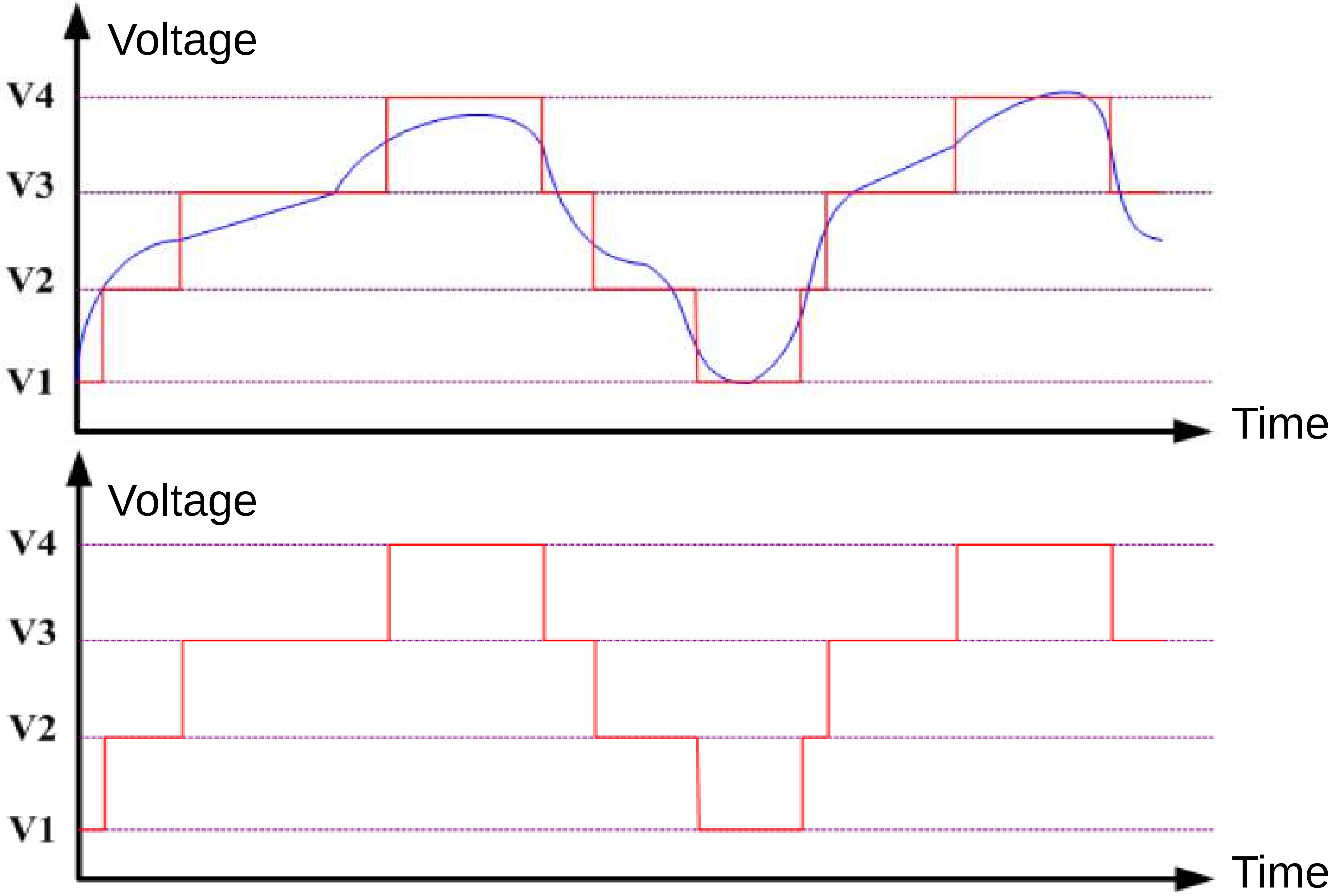
- The world around us is analog
- It is common to convert analog parameters into digital form
- This process is called **digitization**

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}



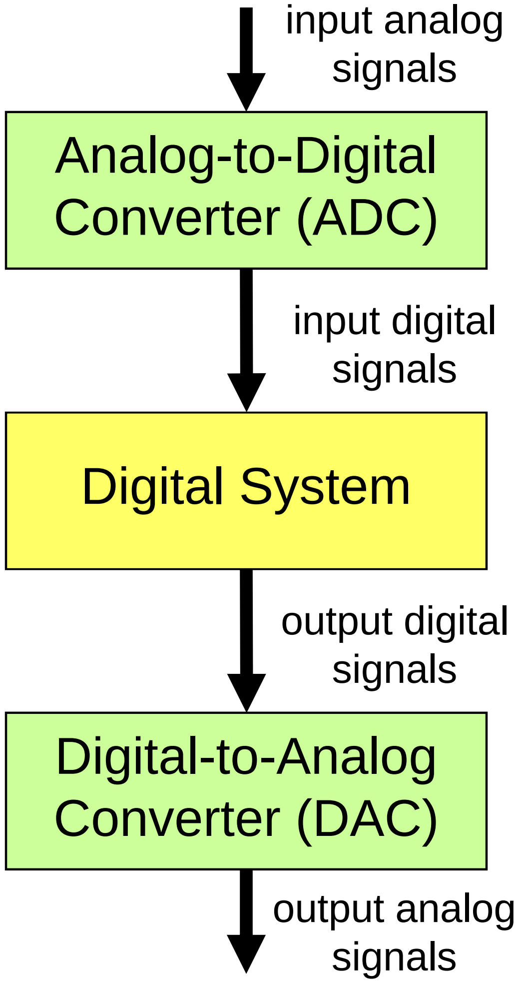
Digitization of Analog Signals – cont'd



- 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

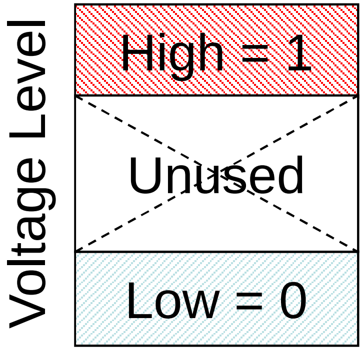


Next . . .

- Analog versus Digital Systems
- Digitization of Analog Signals
- Binary Numbers and Number Systems
- Number System Conversions
- Representing Fractions
- Binary Codes

How do Computers Represent Digits?

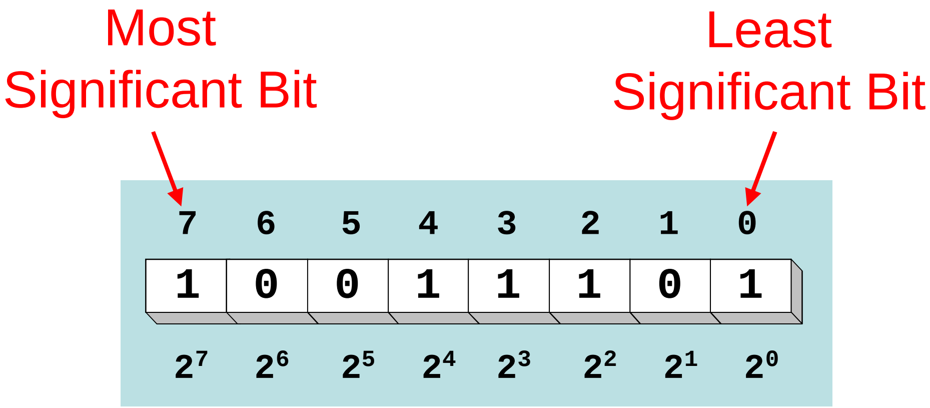
- Binary digits (0 and 1) are the simplest to represent
- Using electric voltage
  - Used in processors and digital circuits
  - High voltage = 1, Low voltage = 0



- 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

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)

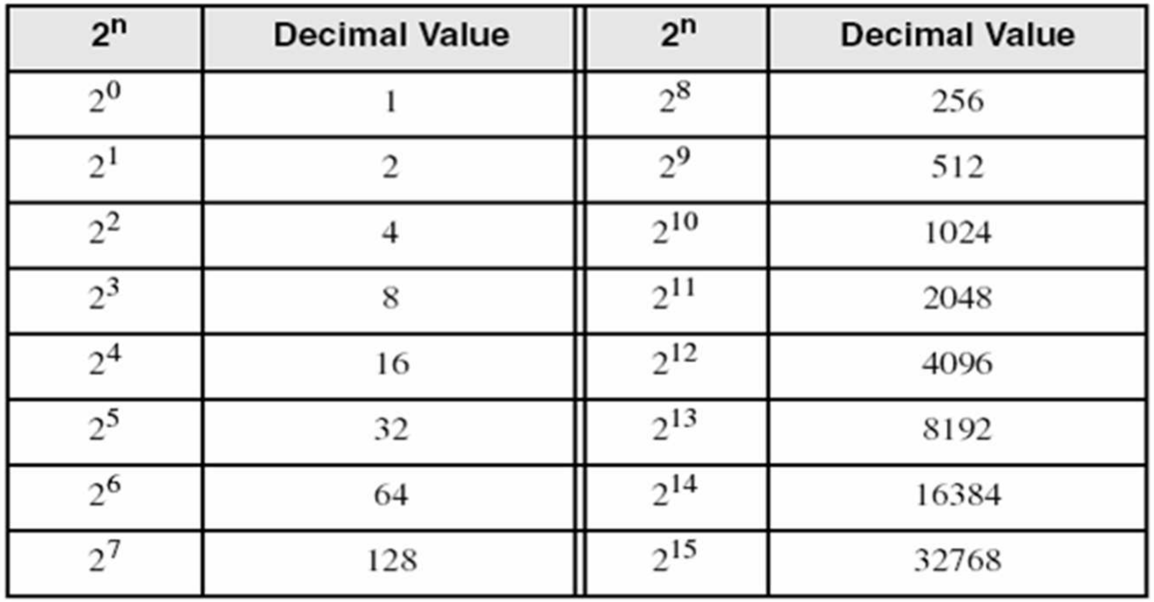


Decimal Value of Binary Numbers

- 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 $(10011101)_2$ = $2^7 + 2^4 + 2^3 + 2^2 + 1 = 157$

7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

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 $= \mathrm{d}_{n-1}\times r^{\,n-1} + \mathrm{d}_{n-2}\times r^{\,n-2} + \ldots + \mathrm{d}_1\times r + \mathrm{d}_0$

Examples: $(11011)_2 = \quad 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2 + 1 = 27$

$(2107)_8 = \quad 2\times8^3 + 1\times8^2 + 0\times8 + 7 = 1095$

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 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

<table>
<tr><td colspan="2">3</td><td colspan="3">5</td><td colspan="3">3</td><td colspan="3">0</td><td colspan="3">5</td><td colspan="3">5</td><td colspan="3">2</td><td colspan="3">3</td><td colspan="3">6</td><td colspan="3">2</td><td colspan="3">4</td><td>Octal</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>32-bit binary</td></tr>
<tr><td colspan="4">E</td><td colspan="4">B</td><td colspan="4">1</td><td colspan="4">6</td><td colspan="4">A</td><td colspan="4">7</td><td colspan="4">9</td><td colspan="4">4</td><td>Hexadecimal</td></tr>
</table>

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

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

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

- 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 – 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 Systems
- Digitization of Analog Signals
- Binary Numbers and Number Systems
- Number System Conversions
- Representing Fractions
- Binary Codes

Representing Fractions

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

$$N_r = \underbrace{d_{n-1}d_{n-2} \dots d_1d_0}_{\text{Integer Part}} \underset{\substack{\uparrow \\ \text{Radix Point}}}{.} \underbrace{d_{-1}\,d_{-2} \dots d_{-m+1}\,d_{-m}}_{\text{Fraction Part}} \qquad 0 \le d_i < r$$

- 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

- $(2409.87)_{10}$ $= 2\times10^3 + 4\times10^2 + 9 + 8\times10^{-1} + 7\times10^{-2}$
- $(1101.1001)_2$ $= 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-4} = 13.5625$
- $(703.64)_8$ $= 7\times8^2 + 3 + 6\times8^{-1} + 4\times8^{-2} = 451.8125$
- $(A1F.8)_{16}$ $= 10\times16^2 + 16 + 15 + 8\times16^{-1} = 2591.5$
- $(423.1)_5$ $= 4\times5^2 + 2\times5 + 3 + 5^{-1} = 113.2$
- $(263.5)_6$ 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
0.375 × 2 = 0.75	0.75	0
0.75 × 2 = 1.5	0.5	1
0.5 × 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

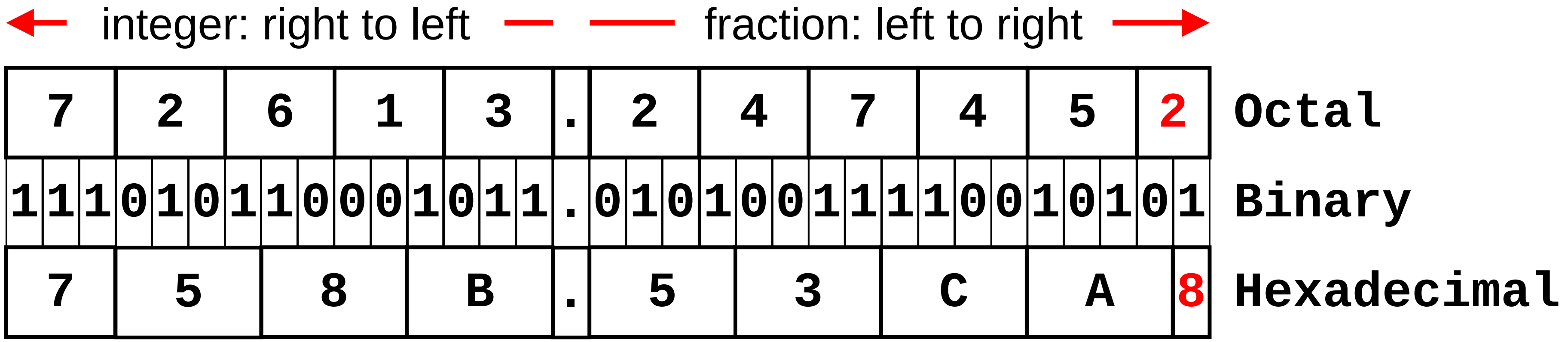
$N = 139.6875 = (213.54)_8$

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

7 2 6 1 3 . 2 4 7 4 5 2	**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 1	**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 = $r^{-1} + r^{-2} + \ldots + r^{-m} = 1 - r^{-m}$

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

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

Next . . .

- Analog versus Digital Systems
- Digitization of Analog Signals
- Binary Numbers and Number Systems
- Number System Conversions
- Representing Fractions
- 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 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*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

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 'L' = 4C (hex) = 76 (decimal)
  - ASCII code for 'a' = 61 (hex) = 97 (decimal)

Control Characters

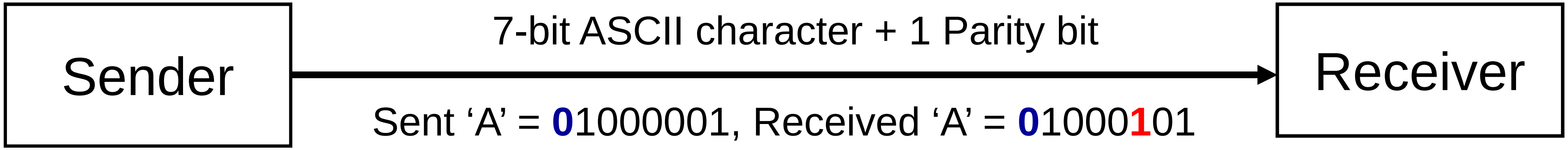
- The first 32 characters of ASCII table are used for control
- Control character codes = 00 to 1F (hexadecimal)
  - Not shown in previous slide
- Examples of Control Characters
  - Character 0 is the NULL character ⇒ used to terminate a string
  - Character 9 is the Horizontal Tab (HT) character
  - 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

Detecting Errors



Sender

7-bit ASCII character + 1 Parity bit

Receiver

Sent ‘A’ = **0**1000001, Received ‘A’ = **0**1000**1**01

- 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