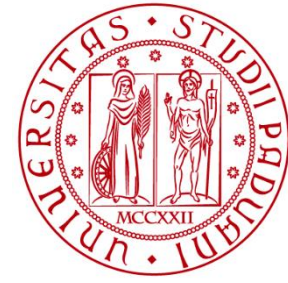




OF THE
DEPARTMENT OF
INFORMATION ENGINEERING



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Digital Systems

Number Systems, Conversions, Arithmetic, Codes

Marta Bagatin, marta.bagatin@unipd.it

Degree Course in Information Engineering
Academic Year 2023-2024

Purpose of the Lesson

- Digital systems operate on discrete elements, represented in binary form
 - The operands used in computer calculations, the letters of the alphabet, etc. are all represented through binary codes
- The scope of this lesson is to study different **number systems (binary, octal, hexadecimal)** and the associated arithmetic
- Study the mostly used binary **codes**
 - Decimal code: BCD (Binary-Coded Decimal)
 - Binary code for alphanumeric characters (ASCII)
 - Gray codes
- Parity bit

Decimal, Binary, Octal, and Hexadecimal Numbers

Decimal Numbers

- Historically, man makes use of the decimal number system (10 fingers!)
- It is a system of **positional notation** and uses 10 digits (from 0 to 9)
 - Depending on its position in the string, each digit is multiplied for the corresponding power of 10
 - **Example:** 384 => 3 hundreds, 8 tens, and 4 units

$$384_{10} = 3 \cdot 10^2 + 8 \cdot 10^1 + 4 \cdot 10^0$$

$$2719.5_{10} = 2 \cdot 10^3 + 7 \cdot 10^2 + 1 \cdot 10^1 + 9 \cdot 10^0 + 5 \cdot 10^{-1}$$

Generic Base (or Radix) r

- A number $a_{n-1} a_{n-2} \dots a_1 a_0$ in base r represents the value

$$\sum_{i=0}^{n-1} a_i r^i$$

a_{n-1} : Most significant digit
 a_0 : Least significant digit

Binary Numbers

- The binary system uses **2 digits, 0 and 1**
 - **Positional notation system:** depending on its position in the string, each digit is weighted by the corresponding power of 2
 - **MSB** (Most Significant Bit) is the bit on the left
 - **LSB** (Least Significant Bit) is the bit on the right
 - Ex: $10011_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19_{10}$
 - $110101.11_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 53.75_{10}$

Powers of Two

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096	20	1,048,576
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

$2^{10}, \sim 10^3 \rightarrow \text{K (kilo)}$

$2^{30}, \sim 10^9 \rightarrow \text{G (giga)}$

$2^{20}, \sim 10^6 \rightarrow \text{M (mega)}$

$2^{40}, \sim 10^{12} \rightarrow \text{T (tera)}$

Example: a 4 Gbyte memory contains $4 \cdot 2^{30} \cdot 8 \text{ bits} = 2^2 \cdot 2^{30} \cdot 2^3 \text{ bit} = 2^{35} \text{ bit}$

Decimal-to-Binary Conversion

- The procedure consists in subsequently subtracting powers of 2 from the decimal number N
 - 1) Find the greatest power of 2 that, subtracted from N , produces a positive difference (N_1)
 - 2) Find the greatest power of 2 that, subtracted from N_1 , produces a positive difference (N_2)
 - 3) Continue the procedure until the difference is 0

Decimal-to-Binary Conversion

– Example: convert 625_{10} to binary

$$625 - 512 = 113 = N1$$

$$512 = 2^9$$

$$113 - 64 = 49 = N2$$

$$64 = 2^6$$

$$49 - 32 = 17 = N3$$

$$32 = 2^5$$

$$17 - 16 = 1 = N4$$

$$16 = 2^4$$

$$1 - 1 = 0 = N5$$

$$1 = 2^0$$

$$\Rightarrow (625)_{10} = 2^9 + 2^6 + 2^5 + 2^4 + 2^0 = (1001110001)_2$$

Octal and Hexadecimal Numbers

- Octal numbers (**base 8, OCT**) and hexadecimal numbers (**base 16, HEX**) are widely used, as 8 and 16 are powers of 2
 - $2^3 = 8$: every octal digit corresponds to 3 binary digits
 - $2^4 = 16$: every hexadecimal digit corresponds to 4 binary digits
- These systems provide more compact representations, that are more convenient compared to binary, as they use shorter strings!

Hexadecimal Numbers

- **16 digits: 0 to 9** plus letters **A, B, C, D, E, F**, for the values 10, 11, 12, 13, 14, 15, respectively

— **Ex:** $(B65F)_{16} = 11 \cdot 16^3 + 6 \cdot 16^2 + 5 \cdot 16^1 + 15 \cdot 16^0 = (46687)_{10}$

Binary-to-Hexadecimal Conversion

- Partition the binary number into groups of 4 bits each, starting from the binary point, proceeding to the left and then to the right. Finally, assign to each group of bit the corresponding hexadecimal digit
 - **Example:** $(0010\ 1100\ 0110\ 1011.\ 1111\ 0000\ 0110)_2 = (2C6B.F06)_{16}$

Hexadecimal-to-Binary Conversion

- Reverse the procedure just seen
 - **Example:** $(3A6.C)_{16} = 0011\ 1010\ 0110\ .\ 1100 = (1110100110\ .11)_2$

Octal Numbers

- Less used than hexadecimal, octal system uses **8 digits, from 0 to 7**
- Each digit in the octal number is weighted with the corresponding power of 8
 - Example: $(127.4)_8 = 1 \cdot 8^2 + 2 \cdot 8^1 + 7 \cdot 8^0 + 4 \cdot 8^{-1} = (87.5)_{10}$
- **Conversion from binary to octal:** similar to conversion to hexadecimal. Partition the binary number into groups of **3 bits**, starting from the point, proceed to the left and then to the right. Finally, assign to each group of bits the corresponding octal digit
- **Conversion from octal to binary:** reverse procedure

Numbers in Base 10, 2, 8, and 16

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

Arithmetic with Binary Numbers

Arithmetic Operations

- Arithmetic operations in non-decimal systems follow the **same rules used in base 10**
 - The available digits and the weight in each position change depending on the number system used!

Binary Addition

- The sum of two digits can be either 0 or 1
 - $0 + 0 = 0$; $0 + 1 = 1$; $1 + 0 = 1$; $1 + 1 = 0$ with carry bit of 1
- The **carry** bit is 1 if both the digits are 1

Binary Addition

- The sum of two digits can be either 0 or 1
 - $0 + 0 = 0$; $0 + 1 = 1$; $1 + 0 = 1$; $1 + 1 = 0$ with carry bit of 1
- The **carry** bit is 1 if both the digits are 1

- Example 1

	Binary	Decimal
Position	16 8 4 2 1	
Carries	0000	
Augend	01100	12_{10}
Addend	+ 10001	+ 17_{10}
Sum	11101	29_{10}

Binary Addition

- The sum of two digits can be either 0 or 1
 - $0 + 0 = 0$; $0 + 1 = 1$; $1 + 0 = 1$; $1 + 1 = 0$ with carry bit of 1
- The **carry** bit is 1 if both the digits are 1

- Example 2

Binary

Decimal

Position 3 2 1 6 8 4 2 1

Carries 1 0 1 1 0

Augend 1 0 1 1 0

22_{10}

Addend + 1 0 1 1 1

+ 23_{10}

Sum 1 0 1 1 0 1

45_{10}

Binary Subtraction

- The difference between two digits can be either 0 or 1
- A **borrow** adds 2 to the adjacent digit on the right (as borrow in decimal adds 10)

– Example 1

	Binary	Decimal
Position	16 8 4 2 1	
Borrow	0000	
Minuend	10110	22_{10}
Subtrahend	-10010	-18_{10}
Difference	00100	4_{10}

Binary Subtraction

- The difference between two digits can be either 0 or 1
- A **borrow** adds 2 to the adjacent digit on the right (as borrow in decimal adds 10)

– Example 2

	Binary	Decimal
Position	16 8 4 2 1	
Borrow	0011	
Minuend	10110	22 ₁₀
Subtrahend	- 10011	- 19 ₁₀
Difference	00011	3 ₁₀

Binary Subtraction (Minuend < Subtrahend)

- If the minuend is smaller than the subtrahend, the order of the operand is reversed and the sign of the difference is changed

– Example 3

	Binary		Decimal
Position	16 8 4 2 1	16 8 4 2 1	
Borrow		0011	
Minuend	10011	11110	30_{10}
Subtrahend	- 11110	-10011	- 19_{10}
Difference		- 01011	- 11_{10}

Binary Multiplication

- The multiplier digits are always 0 or 1
- The **partial products** are equal to the multiplicand or equal to 0

– **Example**

	Binary	Decimal
Position	32 16 8 4 2 1	
Multiplicand	1011	11 ₁₀
Multiplier	x 101	5 ₁₀
Partial product 1	¹ 1011	
Partial product 2	0000-	
Partial product 3	1011-	
Product	110111	55₁₀

Arithmetic Operations

- Operations with numbers in base r follow similar rules as decimal and binary numbers
- In general, **operations with numbers in base r can be performed doing the operations between the respective numbers in base 10 and then converting the results in base r**

Conversion from Decimal to Other Bases

- To convert a decimal number to a number in **base r**: the number and the **subsequent quotients** are **divided by r** and the remainders are noted down, until 0 is obtained
- The conversion is given by the **remainders, considered in reverse order**

– **Example:** Convert 2471_{10} to octal number

$$2471/8 = 308 \quad \text{Remainder} = 7$$

$$308/8 = 38 \quad \text{Remainder} = 4$$

$$38/8 = 4 \quad \text{Remainder} = 6$$

$$4/8 = 0 \quad \text{Remainder} = 4$$

$$\Rightarrow (2471)_{10} = (4647)_8$$



Least significant digit

Most significant digit

Conversion from Decimal to Other Bases

- This method obviously works also for the conversion to base 2
- The conversion is given by the **remainders, considered in reverse order**
 - **Example:** Convert 41_{10} to binary

Conversion from Decimal to Other Bases

- This method obviously works also for the conversion to base 2
- The conversion is given by the **remainders, considered in reverse order**

– **Example:** Convert 41_{10} to binary

$$41/2 = 20 \quad \text{Remainder} = 1$$

$$20/2 = 10 \quad \text{Remainder} = 0$$

$$10/2 = 5 \quad \text{Remainder} = 0$$

$$5/2 = 2 \quad \text{Remainder} = 1$$

$$2/2 = 1 \quad \text{Remainder} = 0$$

$$1/2 = 0 \quad \text{Remainder} = 1$$

$$\Rightarrow (41)_{10} = (101001)_2$$

↑ Least significant digit
Most significant digit

Equivalent to convert the decimal number to a sum of powers of 2 (see slide 8):

– **Example:** $(41)_{10} = 32 + 8 + 1 = (101001)_2$

Codes

Codes

- A code **associates a set of symbols to a discrete set of elements** (for example numbers: ASCII, electrical impulses: Morse code, etc.)
- The purpose of a code is to facilitate the **storage of data or transmission of data** through specific networks
- An **n-bit binary code** is a group of n bits, which can take 2^n distinct combinations of 1 and 0
 - A set of 4 elements can be coded with a 2-bit binary code: 00, 01, 10, 11
 - A set of 8 elements can be coded with a 3-bit binary code : 000, 001, 010, 011, 100, 101, 110, 111
 - 16 elements can be coded with 4 bits, etc...

Decimal Codes

- Computers work with binary numbers, but people are used to the decimal system: one way to cope with this is to **convert decimal numbers to binary**, perform the operations in base 2, and finally convert the results back to base 10
- **BCD (Binary Coded Decimal)** is a **way of representing the decimal digits with a binary code**
 - **4 bits** are needed to represent the 10 decimal digits in base 2

Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Each decimal digit is represented with its 4-bit binary code

Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example: $(396)_{10} = (0011\ 1001\ 0110)_{\text{BCD}}$

-> 4-bit groups, representing the binary
code of each decimal digit

- The BCD representation of a number greater than 10_{10} is **different** from its binary representation!
 - Ex: $(185)_{10} = (0001\ 1000\ 0101)_{\text{BCD}} = (10111001)_2$

Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example: $(396)_{10} = (0011\ 1001\ 0110)_{\text{BCD}}$

-> 4-bit groups, representing the binary **code of each decimal digit**

- There are 6 binary combinations with no **meaning in BCD code**: 1010, 1011, 1100, 1101, 1110, 1111
- Although it leads to a **waste of bits**, BCD code is convenient in applications where inputs and outputs are decimal numbers

Alphanumeric Codes

- Not only the numbers, but also the **letters**, the **symbols**, and the **special characters** need to be represented with binary codes to be handled by a computer or digital system
- The standard code to represent alphanumeric characters is the **ASCII** (**American Standard Code for Information Interchange**) **code**
 - Historically developed for the Anglo-Saxon language and published by the American National Standards Institute in 1968

ASCII Code

		$B_7 B_6 B_5$						
$B_4 B_3 B_2 B_1$	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

- 7 bits to represent 128 characters ($B_7 B_6 B_5$: most significant bits identify the column; $B_4 B_3 B_2 B_1$: least significant bits identify the row)
- Often stored in 1 byte, with the first bit set to '0' or used for additional characters

ASCII Code

$B_4 B_3 B_2 B_1$	$B_7 B_6 B_5$							
	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

- The BCD code of digits 0-9 is **directly related with the ASCII code**: append «011» as the most significant digits to the BCD code to get the corresponding ASCII code

Unicode

- Unicode was developed in 1991, after the ASCII code, with the aim of filling the gaps (e.g. international characters were not represented by ASCII)
- Unicode is an industrial standard, providing a single **code common to all languages** (includes stressed letters, ideograms, etc.)
- Unicode assigns a unique binary code to every character, independent from the used language, platform, and program

Unicode

- **Unicode notation:** "U +" followed by a group of hexadecimal digits corresponding to the character, called **code point**
- There are different codes for the Unicode code points: UTF-8, UTF-16, UTF-32
- For example, the UTF-8 code uses a number of bytes that varies from 1 to 4 to identify each code point

Unicode: Code UTF-8

UTF-8 Encoding for Unicode Code Points

Code point range (hexadecimal)	UTF-8 encoding (binary, where bit positions with x are the bits of the code point value)
U+0000 0000 to U+0000 007F	0xxxxxxx
U+0000 0080 to U+0000 07FF	110xxxxx 10xxxxxx
U+0000 0800 to U+0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+0001 0000 to U+0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

- Example:

- T : code point **U + 0054**, which is in the range U + 0000 0000 - U + 0000 007F -> so it would be encoded with 1 byte: $(01010100)_2$
- ± : code point **U + 00B1**, which is in the range U + 0000 0080 - U + 0000 07FF -> so it would be encoded with 2 bytes: $(11000010\ 10110001)_2$

Parity Bit

- Adding a **parity bit** to a binary code is a method to detect possible errors in processing, transfer, and storage of data
- It is an additional bit, chosen such that the **total number of 1's is even** (even parity, most popular) **or** the **total number of 1's is odd** (odd parity)
 - **Examples**
 - **1000001**
 - With even parity: **0**1000001; with odd parity **1**1000001
 - **1010100**
 - with even parity: **1**1010100; with odd parity **0**1010100

Parity Bit

- The parity bit provides a check, but it does not correct the error! It can be used with binary numbers, but also with ASCII code (the parity bit is not necessarily the MSB)
- **Example:** transmission of ASCII code with parity bit
 - 8 bits are transmitted: 7 bits (ASCII code of the character) + 1 (parity bit)
 - The receiver controls the parity of the transmitted character
 - No parity (odd number of 1's) means that at least one bit was corrupted in the transmission: the receiver sends a negative acknowledge signal (NAK)
 - Parity means that no single-bit errors occurred: the receiver sends an acknowledge signal (ACK)
 - In case of error, depending on the application
 - A new transmission of the character is requested
 - If the error continues, a malfunction of the transmission is reported

Gray Code


- For some applications, it is convenient that the **codes of consecutive numbers differ by a single bit**
- The Gray code is a binary code having the property that a single bit at a time changes between consecutive codes during counting

Binary Code	Bit Changes	Gray Code	Bit Changes
000		000	
001	1	001	1
010	2	011	1
011	1	010	1
100	3	110	1
101	1	111	1
110	2	101	1
111	1	100	1
000	3	000	1

Why Using a Gray Code?

- It simplifies **the operation of electronic systems that process information organized in sequences** and reduces the probability of errors
- Consider a device indicating its position by closing and opening switches. If the position is represented with a standard binary code, positions 3 and 4 are consecutive, but all the bits in the code change going from 3 to 4

Decimal	Binary
1	000
2	001
3	011
4	100



- This can be a source of problems with real switches, as they are unlikely to change their state (open/closed) all at the same time
- Oscillations/ambiguity in the position

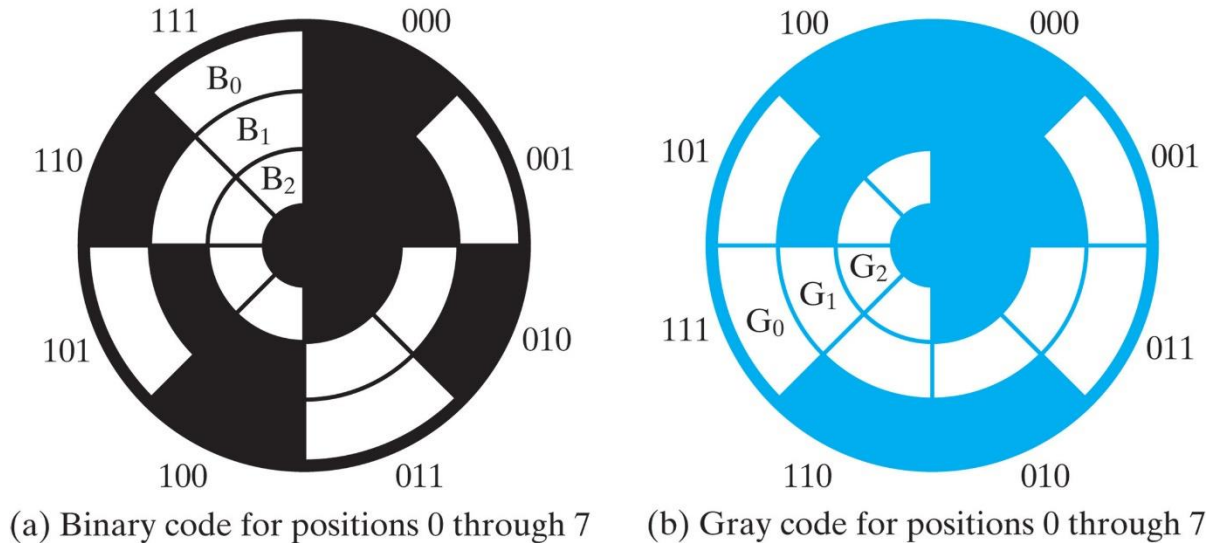
Why Using a Gray Code?

- It simplifies **the operation of electronic systems that process information organized in sequences** and reduces the probability of errors
- Consider a device indicating its position by closing and opening switches. If the position is represented with a standard binary code, positions 3 and 4 are consecutive, but all the bits in the code change going from 3 to 4

Gray Code	Bit Changes
000	
001	1
011	1
010	1
110	1
111	1
101	1
100	1
000	1

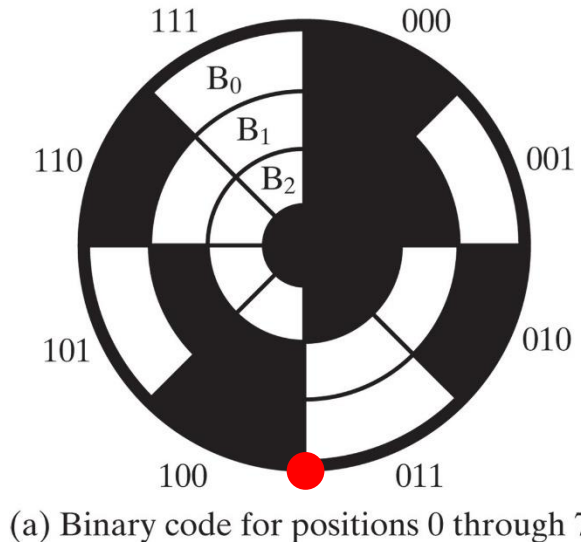
- The use of a Gray code (a single bit at a time changes) solves the problem of oscillations

Gray Code: Application Example



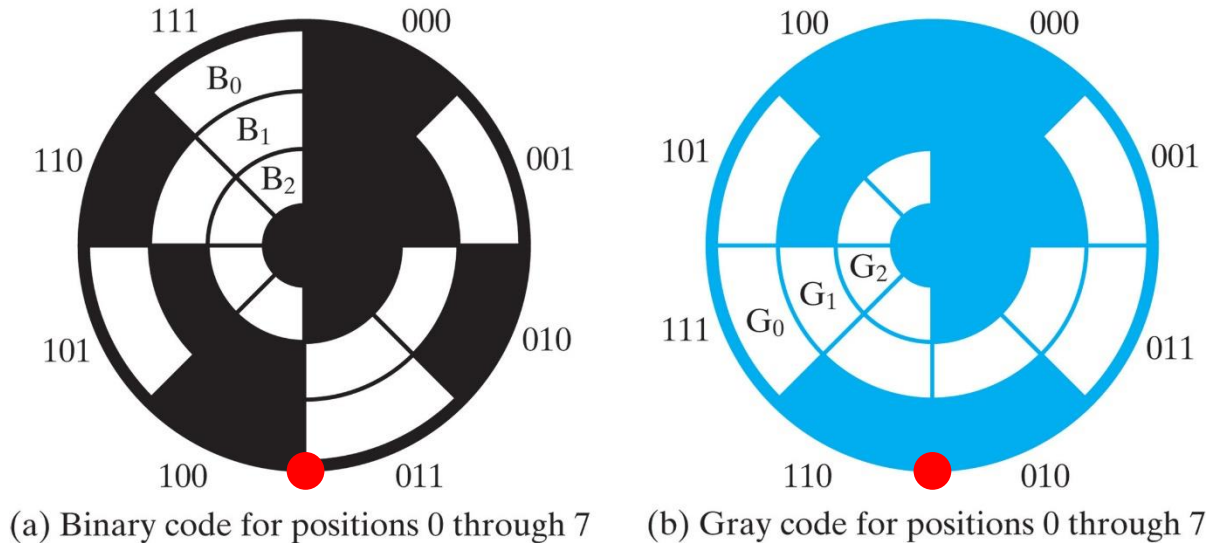
- The Gray code was patented by Frank Gray (1953) for the **encoder**, a device converting an angular position to a digital code
 - Disk (attached to a rotating shaft) containing clear and opaque areas
 - Every bit is represented by a LED (whose output is digital, 0 or 1) and an optical sensor on the opposite side of the disc: if the region is clear the sensor reads '1', if opaque it reads '0'

Gray Code: Application Example



- Suppose the sensor is on the boundary between 2 adjacent positions
 - With the standard binary code: all bits B_0 B_1 B_2 can be read as '0' or '1' (8 possibilities, 6 of which are incorrect!)

Gray Code: Application Example



- Suppose the sensor is on the boundary between 2 adjacent positions
 - With the standard binary code: all bits B₀ B₁ B₂ can be read as '0' or '1' (8 possibilities, 6 of which are incorrect!)
 - With the **Gray code**: G₂ can be read as '0' or '1', G₁ only as '1', G₀ only as '0' (with the **Gray code two positions are obtained, both correct!**)

Gray Code and Hamming Distance

- The Hamming distance is defined as the **number of bits for which the representations of two numbers differ from each other**

- The **Gray code** is a code with **Hamming distance equal to 1**, i.e. two consecutive numbers differ by only one bit (unlike the standard binary code)
- The Gray code is a **n-closed code** (the first and last numbers of the code have unitary distance)

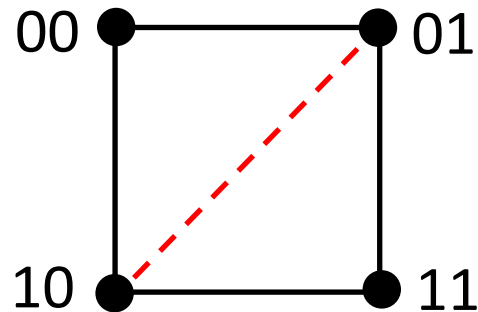
Gray Code	Bit Changes
000	
001	1
011	1
010	1
110	1
111	1
101	1
100	1
000	1

Hamming Distance

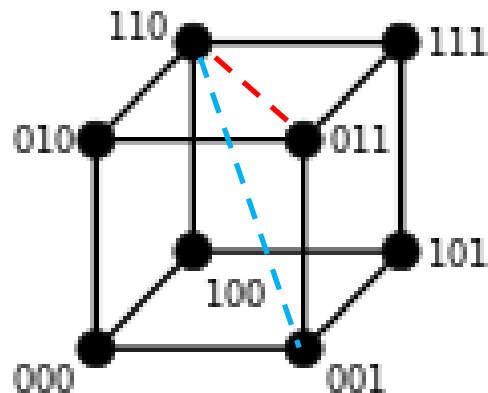
- **Geometric representation:** adjacent numbers are joined by a solid line



1 bit (one-dimensional)



2 bit (two-dimensional)



— distance 1

- - - distance 2

- - - distance 3

3 bit (three-dimensional)

Construction of n-bit Gray Code

- A Gray code can be built in a **recursive way**
- To build a Gray code from n words, the starting point is the binary sequence of n codes (with **n even**)
 - The first $n/2$ codes consist of '0' followed by the parity between every bit of the binary code and the bit on its left
 - The $n/2$ remaining codes are obtained from the sequence found at the previous step, in reversed order (mirrored) and with the MSB set at '1'
- For binary sequences of $2^n - 1$ words, the Gray code is obtained copying the MSB and replacing every remaining bit with the parity between each bit and the bit on the left

Example: Gray Code (n Even)

- Gray code (10 words)

Binary

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

The first $n/2$ codes consist of '0' followed by the parity between every bit of the binary code and the bit on its left

The $n/2$ remaining codes are constructed from the sequence found at the previous step, in reversed order (mirrored) and with the MSB set at '1'

Example: Gray Code (n Even)

- Gray code (10 words)

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	1110
0110	1010
0111	1011
1000	1001
1001	1000

The first $n/2$ codes consist of '0' followed by the parity between every bit of the binary code and the bit on its left

The $n/2$ remaining codes are constructed from the sequence found at the previous step, in reversed order (mirrored) and with the MSB set at '1'

Example: Gray Code ($2^n - 1$ words)

- To build the code Gray 7 words (3 bits)

Binary

000

001

010

011

100

101

110

Copy the most significant bit and replace each remaining bit with the parity between each bit and the bit on the its left

Example: Gray Code ($2^n - 1$ words)

- To build the code Gray 7 words (3 bits)

Binary	Gray
000	000
001	001
010	011
011	010
100	110
101	111
110	101

Copy the most significant bit and replace each remaining bit with the parity between each bit and the bit on the its left

Exercises

- a) Convert from binary to decimal: 1011001
- b) Convert from decimal to binary: 3871
- c) Convert from hex to binary: E429
- d) Convert from decimal to octal: 347
- e) Add binary numbers: $10001 + 1111$
- f) Difference between binary numbers: $10001 - 1111$
- g) Multiply binary numbers: 10001×101
- h) Represent the decimal numbers 715 and 354 in BCD
- i) Find the even parity code for the following number: 10001
- j) Find the odd parity code for the following number: 10001
- k) Build the 16-word Gray code

Disclaimer

Figures from *Logic and Computer Design Fundamentals*,
Fifth Edition, GE Mano | Kime| Martin

© 2016 Pearson Education, Ltd