

Introduction to Digital Systems

Part I (4 lectures)

2025/2026

Introduction
Number Systems and Codes
Combinational Logic Design Principles

Arnaldo Oliveira, Augusto Silva, Ioulia Skliarova, Adão Silva

Lecture 2 contents

- Addition and subtraction of unsigned nondecimal numbers
- Representation of negative numbers
- Two's-complement addition and subtraction
- Codes
 - Character codes
 - Binary-coded decimal
 - Gray code

Lecture 2 bibliography

2 NUMBER SYSTEMS AND CODES 25

2.1	Positional Number Systems	26	
2.2	Octal and Hexadecimal Numbers	27	
2.3	General Positional-Number-System Conversions	29	
2.4	Addition and Subtraction of Nondecimal Numbers	32	
2.5	Representation of Negative Numbers	34	
	<i>2.5.1 Signed-Magnitude Representation</i>	<i>2.5.2 Complement Number Systems</i>	
	<i>2.5.3 Radix-Complement Representation</i>		
	<i>2.5.4 Two's-Complement Representation</i>		
	<i>2.5.5 Diminished Radix-Complement Representation</i>		
	<i>2.5.6 Ones' Complement Representation</i>	<i>2.5.7 Excess Representations</i>	
2.6	Two's-Complement Addition and Subtraction	39	
	<i>2.6.1 Addition Rules</i>	<i>2.6.2 A Graphical View</i>	<i>2.6.3 Overflow</i>
	<i>2.6.4 Subtraction Rules</i>	<i>2.6.5 Two's-Complement and Unsigned Binary Numbers</i>	
2.7	Ones'-Complement Addition and Subtraction	44	
2.8	Binary Multiplication	45	

Addition of Binary Numbers

- Addition and subtraction of nondecimal numbers by hand uses the same technique that you know from school for decimal numbers.
- The only catch is that the addition and subtraction tables are different.
- To add two **unsigned binary numbers** X and Y , we add together the least significant bits with an initial carry (C_{in}) of 0, producing carry (C_{out}) and sum (S) bits according to the table. We continue processing bits from right to left, adding the carry out of each column into the next column's sum.

Example:

$$\begin{array}{r} 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ + \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \\ \hline 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \end{array}$$

C_{in}	x	y	C_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Subtraction of Binary Numbers

- Binary subtraction is performed similarly, using borrows (b_{in} and b_{out}) instead of carries between steps, and producing a difference bit d .

b_{in}	x	y	b_{out}	d
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Examples:

$$\begin{array}{r} 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \\ - \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ \hline 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 1 \\ 1 \ 0 \ 0 \ 0 \\ - \ 0 \ 0 \ 1 \ 1 \\ \hline 0 \ 1 \ 0 \ 1 \end{array}$$

Overflow

- With n bits it is possible to represent **unsigned integer numbers** ranging from 0 to 2^n-1 .
- If an arithmetic operation produces a result that exceeds the range of the number system, **overflow** is said to occur.
- Overflows can easily be detected by analyzing a **carry or borrow from the most significant bit**.
 - the carry bit c_{out} or the borrow bit b_{out} out of the MSB = 1

Examples:

$n=8$: [0..255]

$$173_{10} + 97_{10} = 270_{10}$$

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \\ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ + \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \\ \hline 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \end{array}$$

overflow

$n=4$: [0..15]

$$4_{10} - 11_{10} = -7_{10}$$

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 \\ 0 \ 1 \ 0 \ 0 \\ - \ 1 \ 0 \ 1 \ 1 \\ \hline 1 \ 0 \ 0 \ 1 \end{array}$$

overflow

Addition of Octal Numbers

- To add two octal numbers X and Y , we add together the least significant digits with an initial carry (C_{in}) of 0. If the *intermediate result* is less than or equal to 7, then $C_{out} = 0$ and sum (s) digit = *intermediate result*. If the *intermediate result* is greater than 7, then $C_{out} = 1$ and sum (s) digit = *intermediate result* – 8.
- We continue processing digits from right to left, adding the carry out of each column into the next column's sum.

Examples (radix 8):

$$\begin{array}{r} 0 \ 0 \ 0 \\ 3 \ 0 \ 4 \ 1 \\ + 1 \ 7 \ 3 \ 2 \\ \hline 4 \ 7 \ 7 \ 3 \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 1 \\ 3 \ 4 \ 5 \ 6 \\ + 1 \ 7 \ 3 \ 4 \\ \hline 5 \ 4 \ 1 \ 2 \end{array}$$

Addition of Hexadecimal Numbers

- To add two **hexadecimal numbers** X and Y , we add together the least significant digits with an initial carry (c_{in}) of 0. If the *intermediate result* is less than or equal to 15, then $c_{out} = 0$ and sum (s) digit = *intermediate result*. If the *intermediate result* is greater than 15, then $c_{out} = 1$ and sum (s) digit = *intermediate result* – 16.
- We continue processing digits from right to left, adding the carry out of each column into the next column's sum.

Examples (radix 16):

$$\begin{array}{r} 1 \ 0 \ 0 \\ 3 \ A \ 4 \ 1 \\ + 1 \ 7 \ 8 \ 2 \\ \hline 5 \ 1 \ C \ 3 \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 0 \\ 3 \ F \ A \ A \\ + B \ 7 \ E \ 4 \\ \hline F \ 7 \ 8 \ E \end{array}$$

Subtraction of Octal and Hexadecimal Numbers

- When subtracting **octal numbers**, a borrow brings the value 8.
- When subtracting **hexadecimal numbers**, a borrow brings the value 16.

Examples:

radix 8

$$\begin{array}{r} 1 \ 0 \ 1 \\ 3 \ 0 \ 4 \ 1 \\ - 1 \ 7 \ 3 \ 2 \\ \hline 1 \ 1 \ 0 \ 7 \end{array} \qquad \begin{array}{r} 1 \ 1 \ 1 \\ 6 \ 0 \ 0 \ 0 \\ - 1 \ 5 \ 7 \ 7 \\ \hline 4 \ 2 \ 0 \ 1 \end{array}$$

radix 16

$$\begin{array}{r} 0 \ 1 \ 1 \\ 3 \ A \ 4 \ 1 \\ - 1 \ 7 \ 8 \ 2 \\ \hline 2 \ 2 \ B \ F \end{array} \qquad \begin{array}{r} 1 \ 1 \ 1 \\ B \ 0 \ 0 \ 0 \\ - A \ 7 \ E \ 4 \\ \hline 0 \ 8 \ 1 \ C \end{array}$$

Representation of Negative Numbers

- There are many ways to represent negative numbers.
- In everyday business we use the **signed-magnitude system** (i.e. reserve a special symbol to indicate whether a number is negative).
- However, most computers use **two's-complement representation**:
 - The **most significant bit (MSB)** of a number in this system serves as the sign bit; a number is negative if and only if its MSB is 1.
 - The weight of the MSB is negative: for an n -bit number the weight is -2^{n-1} .
 - The decimal equivalent for a two's-complement binary number is computed the same way as for an unsigned number, except that the weight of the MSB is negative:
 - $D = d_{n-1}d_{n-2} \dots d_1d_0 = -2^{n-1} + \sum_{i=0}^{n-2} d_i \times 2^i$

Representation of Negative Numbers

Examples:

$$1010_2 = ???_{10}$$

$$1010_2 = -2^3 + 2^1 = -8 + 2 = -6_{10}$$

$$1111_2 = ???_{10}$$

$$1111_2 = -2^3 + 2^2 + 2^1 + 2^0 = -8 + 4 + 2 + 1 = -1_{10}$$

$$0111_2 = ???_{10}$$

$$0111_2 = 2^2 + 2^1 + 2^0 = 4 + 2 + 1 = 7_{10}$$

Two's Complement Representation

- For n bits, the range of representable numbers is $[-2^{n-1}, 2^{n-1}-1]$.
- For $n=4$, the range is [-8, 7]:

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

Conversion between Decimal and Two's Complement

- The decimal value of the number expressed in two's complement can be found by expanding the formula ($D = d_{n-1}d_{n-2} \dots d_1d_0 = -2^{n-1} + \sum_{i=0}^{n-2} d_i \times 2^i$) using radix-10 arithmetic.
- The integer number D expressed in decimal can be converted to n -bit two's complement by **successful division of D by 2** (using radix-10 arithmetic, until the result is 0) with **reverse recording** of all the obtained **remainders**.
 - If there are **empty bit positions** left, **fill them with 0s**.
 - **Do not exceed the range** of representable numbers: $[-2^{n-1}, 2^{n-1}-1]$.
 - If the number is negative, the result must be **negated**:
 - Invert all the bits individually and add 1 or
 - Copy all the bits starting from the least significant until the first 1 is copied, then invert all the remaining bits.

Conversion between Decimal and Two's Complement

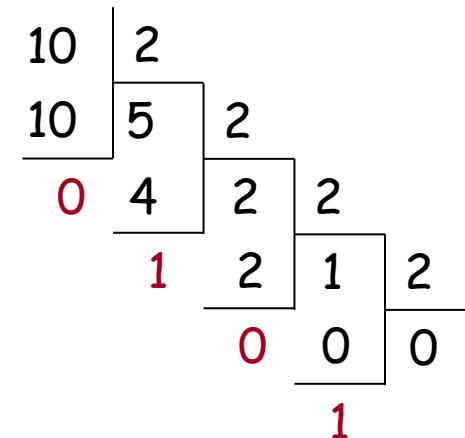
Examples (with n=8):

$$10_{10} = ???_2$$

$$10_{10} = 00001010_2$$

$$-10_{10} = ???_2$$

$$-10_{10} = 11110110_2$$



$$\begin{array}{r} 11110101 \\ + 00000001 \\ \hline 11110110 \end{array}$$

negated
add 1

The diagram shows the conversion of -10₁₀ to binary. It starts with the binary representation of 10₁₀, which is 00001010₂. A blue arrow labeled "negated" points to the negation of the number, changing it to 11110101₂. Another blue arrow labeled "add 1" points to the addition of 1, resulting in 11110110₂. The least significant bit (0) is circled in red.

Changing the Number of Bits

- We can convert an n -bit two's-complement number into an m -bit one.
- If $m > n$, perform **sign extension**:
 - append $m - n$ copies of the sign bit to the left
- If $m < n$, **discard $n - m$ leftmost bits**; however, **the result is valid only if all of the discarded bits are the same as the sign bit of the result**.

Examples:

$$n = 5$$

$$00101 = 00000101$$

$$m = 8$$

$$11110 = 11111110$$

$$n = 5$$

$$00101 = 101 \text{ - result is } \underline{\text{not}} \text{ valid}$$

$$m = 3$$

$$11110 = 110 \text{ - result is valid}$$

Two's-Complement Addition

- Addition is performed in the same way as for nonnegative numbers.
- Carries beyond the MSB are **ignored**.
- The result will always be the correct sum as long as the range of the number system is not exceeded.
- If an addition operation produces a result that exceeds the range of the number system, **overflow** is said to occur.
- Addition of two numbers with different signs can never produce overflow.
- Addition of two numbers of same sign can produce overflow if
 - the addends' signs are the same but the sum's sign is different from the addends'
 - the carry bits c_{in} into and c_{out} out of the sign position are different

Two's-Complement Addition

Examples (n=4):

$$\begin{array}{r} 1 \ 1 \ 0 \ 0 \\ 0 \ 1 \ 0 \ 0 \quad (4) \\ + 1 \ 1 \ 0 \ 1 \quad (-3) \\ \hline 0 \ 0 \ 0 \ 1 \quad (1) \end{array}$$

$$\begin{array}{r} 0 \ 1 \ 0 \\ 0 \ 0 \ 1 \ 0 \quad (2) \\ + 0 \ 0 \ 1 \ 1 \quad (3) \\ \hline 0 \ 1 \ 0 \ 1 \end{array}$$

$$\begin{array}{r} \text{Different} \\ 1 \ 0 \ 0 \ 0 \\ 1 \ 0 \ 0 \ 1 \quad (-7) \\ + 1 \ 1 \ 1 \ 0 \quad (-2) \\ \hline 0 \ 1 \ 1 \ 1 \end{array}$$

overflow

Adding two negative
numbers and the results is a
positive number



Two's-Complement Subtraction

- Two's-complement numbers may be subtracted as if they were ordinary unsigned binary numbers.
- However, **most subtraction circuits for two's-complement numbers do not perform subtraction directly.**
- Rather, they **negate the subtrahend** by taking its two's complement, and then **add** it to the minuend using the normal rules for addition ($X - Y = X + (-Y)$).
- Overflow in subtraction can be detected using the same rule as in addition.
- Negating the subtrahend and adding the minuend can be accomplished with only one addition operation:
 - Perform a bit-by-bit complement of the subtrahend and add the complemented subtrahend to the minuend with an initial carry (C_{in}) of 1 instead of 0.

Two's-Complement Subtraction

Examples (n=4):

$$\begin{array}{r} 0010 - 0011: \quad \begin{array}{r} 0001 \\ 0010 \\ + 1100 \\ \hline 1111 \end{array} \end{array}$$

$c_{in}=1$

$$\begin{array}{r} 1011 - 0110: \quad \begin{array}{r} 1011 \\ 0110 \\ + 1001 \\ \hline 0101 \end{array} \end{array}$$

Different

overflow

Information Encoding

- Digital systems are built from circuits that process binary digits
- Very few real-life problems are based on binary numbers or any numbers at all
- Some correspondence must be established between the binary digits processed by digital circuits and real-life numbers, events, and conditions
 - How to represent familiar numeric quantities? ✓
 - number systems: binary, octal, and hexadecimal
 - How to represent nonnumeric data?

Codes

- A **code** is a set of n -bit strings in which different bit strings represent different numbers or other things.
- A **code word** is a particular combination of n bit-values.
- To code m values, the code length n must respect the following equation: $n \geq \lceil \log_2 m \rceil$.



floor	encoding	encoding	encoding
basement	000	000	000001
ground floor	001	001	000010
1 st floor	010	011	000100
2 nd floor	011	010	001000
3 rd floor	100	110	010000
4 th floor	101	111	100000

Character Codes

- The most common type of nonnumeric data is text, strings of characters from some character set.
- Each character is represented in the digital system by a bit string according to an established convention.
- The most commonly used character code is **ASCII** (American Standard Code for Information Interchange).
 - ASCII represents each character with a 7-bit string, yielding a total of 128 different characters.

$b_3b_2b_1b_0$	Row (hex)	$b_6b_5b_4$ (column)									
		000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7		
0000	0	NUL	DLE	SP	0	@	P	‘	p		
0001	1	SOH	DC1	!	1	A	Q	a	q		
0010	2	STX	DC2	”	2	B	R	b	r		
0011	3	ETX	DC3	#	3	C	S	c	s		
0100	4	EOT	DC4	\$	4	D	T	d	t		
0101	5	ENQ	NAK	%	5	E	U	e	u		
0110	6	ACK	SYN	&	6	F	V	f	v		
0111	7	BEL	ETB	,	7	G	W	g	w		
1000	8	BS	CAN	(8	H	X	h	x		
1001	9	HT	EM)	9	I	Y	i	y		
1010	A	LF	SUB	*	:	J	Z	j	z		
1011	B	VT	ESC	+	;	K	[k	{		
1100	C	FF	FS	,	<	L	\	l			
1101	D	CR	GS	-	=	M]	m	}		
1110	E	SO	RS	.	>	N	^	n	~		
1111	F	SI	US	/	?	O	-	o	DEL		

Binary Codes for Decimal Numbers

- Even though binary numbers are the most appropriate for the internal computations of a digital system, most people still prefer to deal with decimal numbers.
- As a result, the external interfaces of a digital system may read or display decimal numbers, and some digital devices actually process decimal numbers directly.
- A decimal number is represented in a digital system by a string of bits, where different combinations of bit values in the string represent different decimal numbers.
- To code $m = 10$ decimal digits, at least $\lceil \log_2 10 \rceil = 4$ bits are required.
- Is the maximum number of bits limited?
- Is the number of possible codes limited?



Binary-Coded Decimal (BCD)

- Perhaps the most "natural" decimal code is **binary-coded decimal** (BCD), which encodes the digits 0 through 9 by their 4-bit unsigned binary representations, 0000 through 1001.
- The code words 1010 through 1111 are not used.
- Conversions between BCD and decimal representations are trivial, a direct substitution of four bits for each decimal digit.

Example:

$$25_{10} = 11001_2$$

$$25_{10} = 00100101_{BCD}$$

decimal digit	BCD (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Gray Code

- Sometimes, it is required to code values so that **only one bit changes** between each pair of successive code words.
- Such a code is called a **Gray code**.
- There are two convenient ways to construct a Gray code with any desired number of bits.

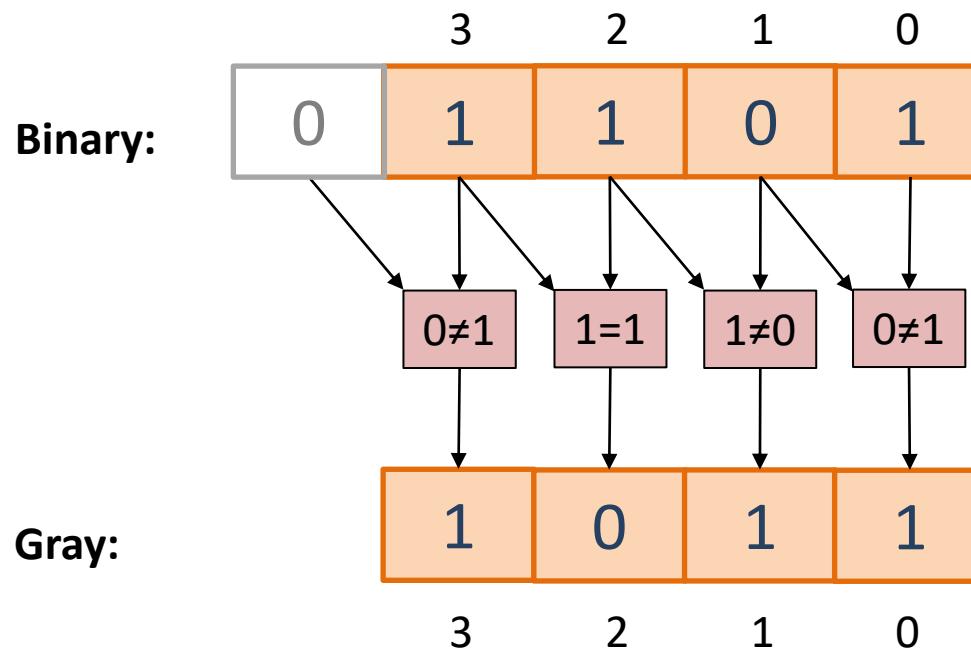
1 bit	2 bits	3 bits	4 bits
0	00	000	0000
1	01	001	0001
	11	011	0011
	10	010	0010
		110	0110
		111	0111
		101	0101
		100	0100
			1100
			1101
			1111
			1110
			1010
			1011
			1001
			1000

Constructing Gray Code

- The first method is based on the fact that Gray code is a reflected code; it can be defined (and constructed) recursively using the following rules:
 - A 1-bit Gray code has two code words, 0 and 1.
 - The first 2^n code words of an $(n + 1)$ -bit Gray code equal the code words of an n -bit Gray code, written in order with a leading 0 appended.
 - The last 2^n code words of an $(n + 1)$ -bit Gray code equal the code words of an n -bit Gray code, but written in reverse order with a leading 1 appended.

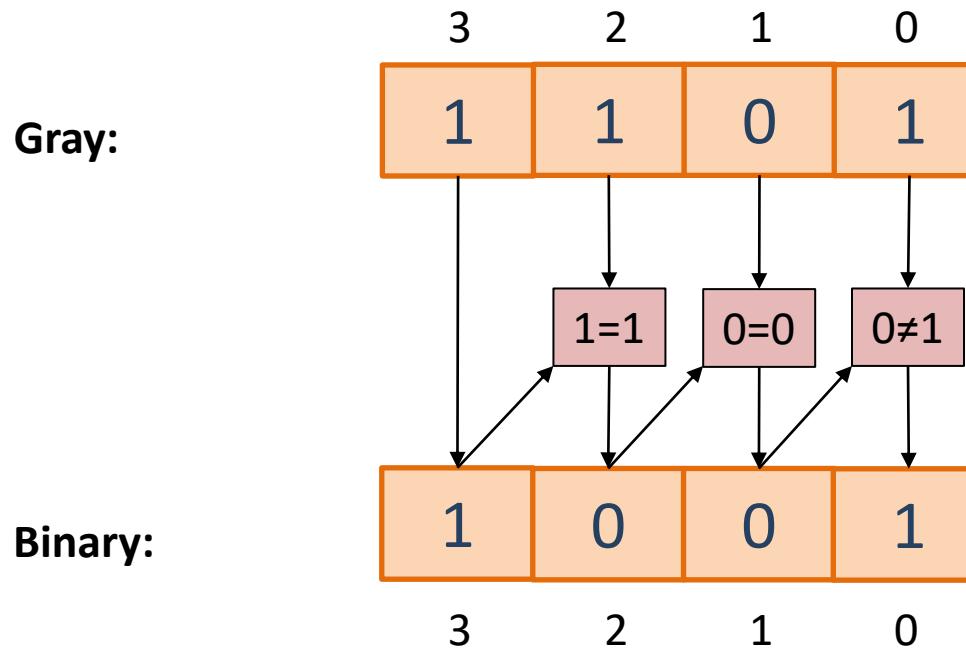
Converting Binary to Gray Code

- The second method allows us to derive an n -bit Gray-code code word directly from the corresponding n -bit binary code word:
 - The bits of an n -bit binary or Gray-code code word are numbered from right to left, from 0 to $n - 1$.
 - Add a “0” to the left
 - Bit i of a Gray-code code word is 0 if bits i and $i + 1$ of the corresponding binary code word are the same, else bit i is 1.
 - When $i + 1 = n$, bit n of the binary code word is considered to be 0



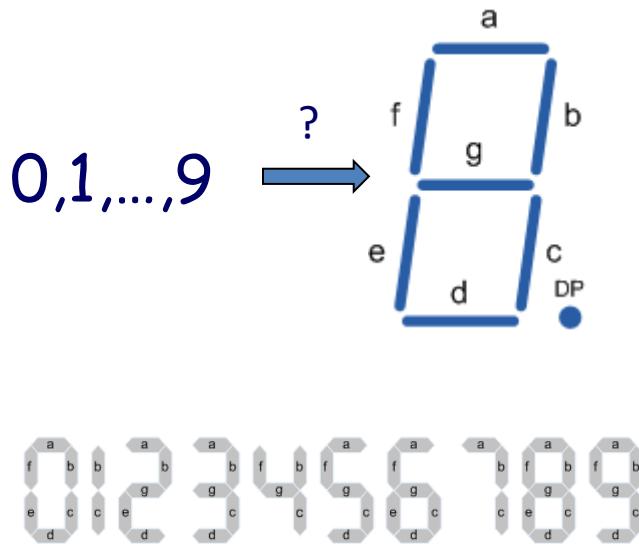
Converting Gray Code to Binary

- Similarly, an n -bit Gray-code code word can be converted to the corresponding n -bit binary code word:
 - The bits of an n -bit Gray-code code word are numbered from right to left, from 0 to $n - 1$.
 - Bit $n - 1$ of a binary code word is equal to bit $n - 1$ of a Gray-code code word.
 - Bit i ($i = n-2, n-3, \dots, 1, 0$) of a binary code word is 0 if bits i of the corresponding Gray-code code word and $i + 1$ of the corresponding binary code word are the same, else bit i is 1.



7-segment Display Codes

- 7-segment displays are used in watches, calculators, and instruments to display decimal data.
- A digit is displayed by illuminating a subset of the seven line segments.

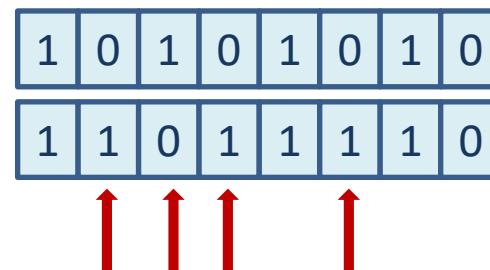


BCD	digit	individual segments						
		a	b	c	d	e	f	g
0000	0	1	1	1	1	1	1	0
0001	1	0	1	1	0	0	0	0
0010	2	1	1	0	1	1	0	1
0011	3	1	1	1	1	0	0	1
0100	4	0	1	1	0	0	1	1
0101	5	1	0	1	1	0	1	1
0110	6	1	0	1	1	1	1	1
0111	7	1	1	1	0	0	0	0
1000	8	1	1	1	1	1	1	1
1001	9	1	1	1	1	0	1	1

Hamming Distance

- The **Hamming distance** between two n -bit strings is the number of bit positions in which they differ.
- In the Gray code, the Hamming distance between each pair of successive code words is 1.

Example:



Hamming distance = 4

Bits, Bytes, Words, etc.

- The prefixes K (kilo-), M (mega-), G (giga-), and T (tera-) mean 10^3 , 10^6 , 10^9 , and 10^{12} , respectively, when referring to bps, hertz, ohms, watts, and most other engineering quantities.
- However, when referring to memory sizes, the prefixes mean 2^{10} , 2^{20} , 2^{30} , and 2^{40} .

Bit	<i>b</i>	0 or 1	1 K/k	10^3	$\approx 2^{10}$	(kilo)
Byte	<i>B</i>	8 bits	1 M	10^6	$\approx 2^{20}$	(mega)
Nibble		4 bits	1 G	10^9	$\approx 2^{30}$	(giga)
Word		8, 16, 32, 64 ... bits (depends on the context)	1 T	10^{12}	$\approx 2^{40}$	(tera)

IEEE 1541-2002:

Ki	$2^{10} = 1\ 024$	(<i>kibi</i>)
Mi	$2^{20} = 1\ 048\ 576$	(<i>mebi</i>)
Gi	$2^{30} = 1\ 073\ 741\ 824$	(<i>gibi</i>)
Ti	$2^{40} = 1\ 099\ 511\ 627\ 776$	(<i>tebi</i>)
Pi	$2^{50} = 1\ 125\ 899\ 906\ 842\ 624$	(<i>pebi</i>)
Ei	$2^{60} = 1\ 152\ 921\ 504\ 606\ 846\ 976$	(<i>exbi</i>)

Exercises

- Add/subtract the following pairs of binary numbers

$$\begin{array}{r} 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ + \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ - \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \\ \hline \end{array}$$

- Add/subtract the following pairs of octal numbers

$$\begin{array}{r} 3 \ 6 \ 5 \ 4 \\ + \ 7 \ 2 \ 5 \\ \hline \end{array}$$

$$\begin{array}{r} 7 \ 0 \ 2 \\ - \ 3 \ 7 \ 7 \\ \hline \end{array}$$

- Add/subtract the following pairs of hexadecimal numbers:

$$\begin{array}{r} B \ A \ 0 \ E \\ + \ E \ 8 \ A \ 7 \\ \hline \end{array}$$

$$\begin{array}{r} F \ 7 \ A \ B \\ - \ 1 \ C \ 6 \ 3 \\ \hline \end{array}$$

Exercises (cont.)

- Represent the following numbers in two's complement with 8 bits: 39_{10} , -22_{10} .
- Calculate the results of the following operations in two's complement with 8 bits. Detect overflows if any.

$$\begin{array}{r}
 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\
 + 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\
 \hline
 \end{array} \quad (-86_{10})$$

$$\begin{array}{r}
 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
 + 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\
 \hline
 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1
 \end{array}$$

$$\begin{array}{r}
 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
 + 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1 \\
 \hline
 \end{array} \quad (127_{10})$$

$$\begin{array}{r}
 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
 + 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1 \\
 \hline
 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0
 \end{array}$$

$$\begin{array}{r}
 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0 \\
 - 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \\
 \hline
 \end{array} \quad (-46_{10})$$

$$\begin{array}{r}
 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\
 + 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0 \\
 \hline
 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1
 \end{array}$$

Overflow
Why?

Overflow
Why?



Exercises (cont.)

Exercises (cont.)

- How many bits are required to code in BCD the number 123456_{10} ?
- Represent the following values in binary and in BCD and Gray codes.

$$\begin{array}{ll} 108_{10} & = 000100001000_{\text{BCD}} \\ & = 1101100_2 \\ & = 1011010_{\text{GRAY}} \end{array}$$

$$\begin{array}{ll} 33_8 & = 00100111_{\text{BCD}} \\ & = 011011_2 \\ & = 010110_{\text{GRAY}} \end{array}$$

- Prove that a two's-complement number can be converted to a representation with more bits by *sign extension*.
- Determine the Hamming distance between the following code words:

$$\begin{array}{ll} 011010101011 & \\ 000010101011 & = 2 \end{array}$$

Exercises (cont.)

- Airport names are encoded by sequences of three capital letters of English alphabet (having 26 letters).
- How many airports can be coded this way?
- How many bits will be required in ASCII code to binary encode the airport codes?
- And if you use the most efficient code possible to encode only uppercase letters?