Data Representation

Dr. Voicu Groza

SITE Hall, Room 5017
562 5800 ext. 2159
Groza@EECS.uOttawa.ca

uOttawa

L'Université canadienne
Canada's university

Université d'Ottawa | University of Ottawa

www.uOttawa.ca

Outline

- Number Systems
- Fixed Point Representation
- Arithmetic operations
- Floating Point Representation
- BCD, ASCII

Number Systems

- There are four main types of number systems: *binary*, *octal*, *hexadecimal*, and *decimal* (commonly used by humans).
- In general, a number system of **base**, or **radix**, *r* is a system that uses distinct symbols for *r* digits.
- A number *x* in base *r*, noted as $(x)r$, has the following general form:

$$(x)_r = x_n\, x_{n-1}\ \ldots\ x_1\, x_0 \,.\, x_{-1}\ x_{-2}\ \ldots\ x_{-m}\ ,$$

where $x \in \{0, 1, \ldots, r-1\}$ and $\in \{n, n-1, \ldots, -m\}$.

- To **<u>convert</u>** a number in the form above **<u>from base *r* to the decimal base</u>** (with radix 10), the following formula is used

$$(x)_r = (x_n \times r^n + x_{n-1} \times r^{n-1} + \cdots + x_0 \times r^0 + x_{-1} \times r^{-1} + \cdots + x_{-m} \times r^{-m})_{10}$$

$r \longrightarrow 10$

Binary Number System

- The **binary** system uses **2** as its radix.
- It employs two binary digits, (bits): 0 and 1
- Represents any number using the positional notation.
- The most significant bit (MSB) is the leftmost bit of a binary number.
- The least significant bit (LSB) is the rightmost bit of a binary number.

Positional Notation

- The value of a digit depends on its placement within a number.
- In base 10, the positional values are (starting from the decimal point to the left): $1(10^0)$, $10(10^1)$, $100(10^2)$, $1000(10^3)$, etc.
- In base-2, the positional values are $1(2^0)$, $2(2^1)$, $4(2^2)$, $8(2^3)$…

Number … in whatever base   *Decimal value of the given number*

Decimal: $2\,017_{10}$ $= 2\times10^3 + 0\times10^2 + 1\times10^1 + 7\times10^0$

$= 2{,}000 + 0 + 10 + 7 = \mathbf{2\,017}$

Binary:

2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

msb

lsb

1 1 1 1 1 1 0 0 0 0 1

$\mathbf{11111100001}_2 = 1\times2^{10} + 1\times2^9 + 1\times2^8 + 1\times2^7 + 1\times2^6 + 1\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 =$

$2 \to 10$ $= 1{,}024 + 512 + 256 + 128 + 64 + 32$ $+ 1 = \mathbf{2017}$

N	2^N	Comments
0	1	
1	2	
2	4	
3	8	
4	16	
5	32	
6	64	
7	128	
8	256	
9	512	
10	1,024	"Kilo"as 2^{10} is the closest power of 2 to 1,000 (decimal)
11	2,048	
…………	……………	……………………….....
15	32,768	2^{15} Hz often used as clock crystal frequency in digital watches
…………	……………	……………….....
20	1,048,576	"Mega" as 2^{20} is the closest power of 2 to 1,000,000 (decimal)
…………	……………	………………...
30	1,073,741,824	"Giga" as 2^{30} is the closest power of 2 to 1,000,000,000(decimal)
…………	……………	………………...
40	…	"Terra" as 2^{40} is the closest power of 2 to 10^{12} (decimal)

Powers of 2
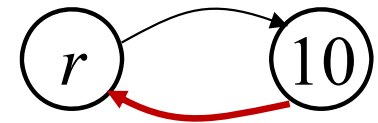
Conversion

- Converting a decimal number to its equivalent representation in radix r is carried out by separating the number into its integer and fraction parts. Then convert each part separately.

- The conversion of the decimal integer part into a base r representation is done by successive divisions by r and accumulation of the remainders.

- The conversion of the decimal fraction part to radix r representation is accomplished by successive multiplications by r and accumulating the integer digits so obtained.

uOttawa

Decimal-to-Binary Conversion

2 ⟶ 10
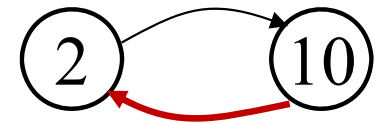
| 2008	/ 2		
1004	0	lsb	2^0
502	0		2^1
251	0		2^2
125	1		2^3
62	1		2^4
31	0		2^5
15	1		2^6
7	1		2^7
3	1		2^8
1	1		2^9
0	1	msb	2^{10}

$$1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0_2$$

$$2^{10}\ .\ .\ 2^7\ .\ .\ .\ 2^3\ .\ .\ 2^0$$

■ <u>Repeated Division by 2</u>

1. Divide the number to be converted by 2.The remainder, 0 or 1, is the LSB of the binary value.

2. Divide the quotient from Step 1 by 2. The remainder, 0 or 1, is the next most significant bit.

3. Continue to execute Step 2 until the quotient is 0. The last remainder is the MSB.

8

Negative Powers of 2

$N < 0$	2^N
-1	$2^{-1} = 0.5$
-2	$2^{-2} = 0.25$
-3	$2^{-3} = 0.125$
-4	$2^{-4} = 0.0625$
-5	$2^{-5} = 0.03125$
-6	$2^{-6} = 0.015625$
-7	$2^{-7} = 0.0078125$
-8	$2^{-8} = 0.00390625$
-9	$2^{-9} = 0.001953125$
-10	$2^{-10} = 0.0009765625$
...	

$2 \longrightarrow 10$
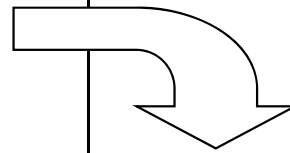
◆ Fractional Binary Numbers
Binary numbers less than 1

Binary	Decimal value
0.101101	$= 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-6} = \mathbf{0.703125_{10}}$

Fractional-Decimal -to- Fractional-Binary Conversion

1. Multiply the decimal fraction by 2. The integer part (0 or 1) is the first bit to the right of the binary point.

2. Discard the integer part from Step 1 and repeat Step 1 until the fraction repeats or terminates.

3. In real applications one might have to use a given number of bits, imposed by the number of bits of the registers that carry the result. Conversion error can be calculated

	2 x	0.703125_{10}		weight
msb	1	.40625	0.5	2^{-1}
	0	.8125	0	2^{-2}
	1	.625	0.125	2^{-3}
	1	.25	0.0625	2^{-4}
	0	.5	0	2^{-5}
lsb	1	.0	0.015625	2^{-6}
			0.703125	

$$0.101101 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6} =$$
$$= 0.5 + 0.125 + 0.0625 + 0.015625 =$$

10

Hexadecimal Number System

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

Binary:

$$11111011010_2$$

Binary -> Hex **conversion**

111 1101 1010

$7_{10} \quad 13_{10} \quad 10_{10} \Longleftarrow Decimal$

Hexadecimal:

$$7DA_{16} = 7 \times 16^2 + 13 \times 16^1 + 10 \times 16^0$$

Hex->decimal $= 2010_{10}$

11

uOttawa
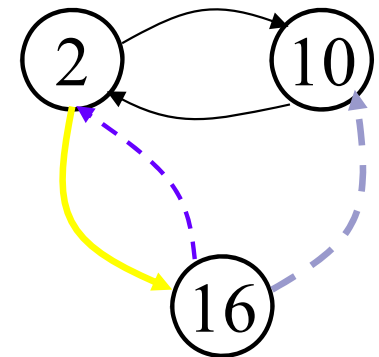
Converting from base 2 to base 8: partition the base 2 into groups of three and pad the outer to the left with 0s
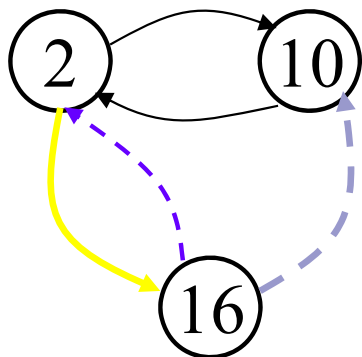
Binary (base 2)	Octal (base 8)	Decimal (base 10)	Hexadecimal (base 16)
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

12

uOttawa

Binary, Octal, and Hexadecimal Numbers

- The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three bits each.
- The corresponding octal digit is then assigned to each group of bits.
- The string of octal digits obtained gives the octal equivalent of the binary number.
- Converting from binary to hexadecimal is similar except that the bits are divided in groups of 4 bits each this time.

Example

1	2	7	5	4	3	Octal
1 0 1 0	1 1 1 1	0 1 1	0 0 0 1 1			Binary
A	F	6	3			Hexadecimal

13

Binary Number Representations

■ In computer systems, binary numbers are represented as unsigned or signed numbers.

■ **Unsigned** numbers are numbers that are always assumed to be non-negative (i.e., ≥ 0).

■ **Signed** numbers are numbers that can be either non-negative or negative.

■ The sign of a signed number is determined by the value of a pre-specified bit, known as the **sign bit**, in the number's representation.

■ The sign bit is usually taken to be the left-most bit in the number's representation.

■ The common convention in a binary system is that the sign bit is 0 for a non-negative number and 1 for a negative number. 14
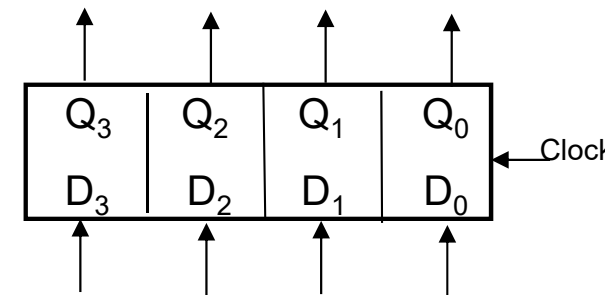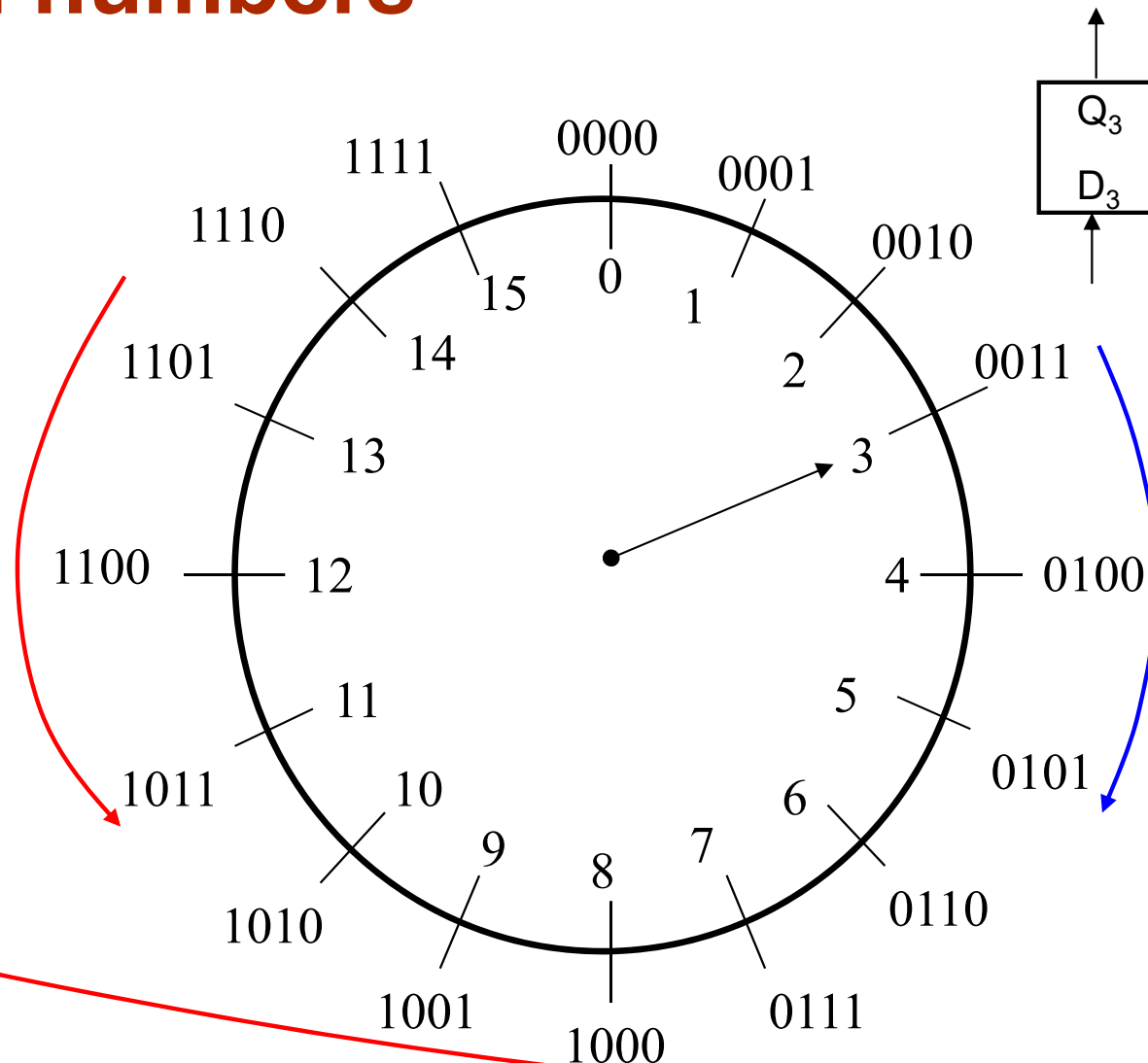
uOttawa

Unsigned numbers

- the weight of the MSB (Most Significant Bit) is 2^{n-1}
- the range of representable numbers is $[0, 2^n-1]$

Q_3	Q_2	Q_1	Q_0
D_3	D_2	D_1	D_0

Clock

Subtraction

Computer format

6	0110
- 3	- 0011
3	0011

8	1000
- 9	- 1001
-1	**1** 1111

Addition

3	0011
+ 4	+0100
7	0111

8	1000
+ 9	+1001
17	**1** 0001

Clock face labels: 0000 (0), 0001 (1), 0010 (2), 0011 (3), 0100 (4), 0101 (5), 0110 (6), 0111 (7), 1000 (8), 1001 (9), 1010 (10), 1011 (11), 1100 (12), 1101 (13), 1110 (14), 1111 (15)

Overflow: = the result of an operation is <u>out of the domain</u> $[0, 2^n-1]$

= **carry**/**borrow** (bit 2^n) is set

15

Complements

- In digital computers, complements are mainly used to represent negative numbers.

- There are two types of complements for each base r system:

- The $(r-1)$'s complement, and

- the r's complement.

- For instance, for the binary system (base 2), there are the 1's complement and the 2's complement.

$(r - 1)$'s Complement

- Given a number N in base r with n digits, the $(r - 1)$'s complement of N is defined as $[(r^n - 1) - N]_r$
- The $(r - 1)$'s complement of a number may also be derived by subtracting each digit from $(r - 1)$.

r's Complement

- The r's complement of an n-digit number N in base r is defined by

$$(r^n - N)_r \quad \text{if } N \# 0,$$
$$0 \qquad\qquad \text{if } N = 0$$

- The r's complement can also be computed by adding 1 to the $(r - 1)$'s complement.
- Another way of computing the r's complement is by leaving all least significant 0's unchanged, subtract the first non-zero least significant digit from r, and then subtracting all higher significant digits from $(r - 1)$.
- **Remark**: If the original number N contains a radix point, it should be removed temporarily to form the complement wanted. Then the radix point is restored to the complemented number in the **same** relative position.
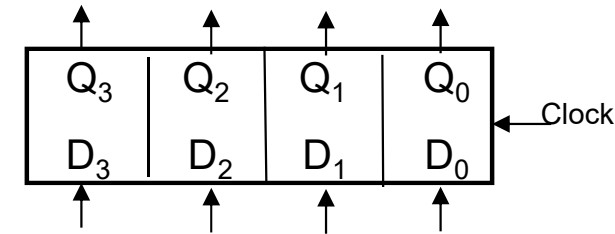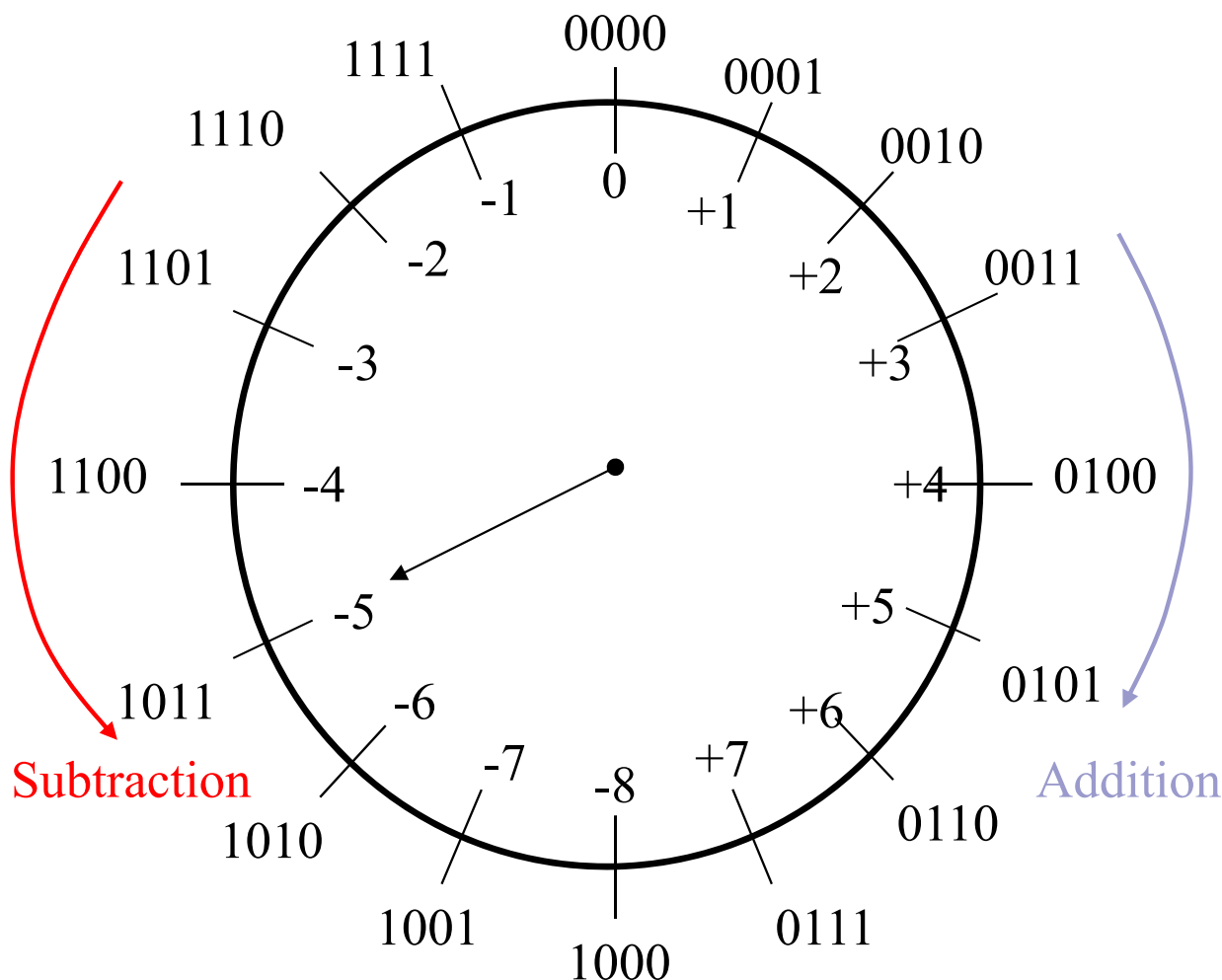
17

2's Complement of a Number

(2's complement of a number P) = $2^n - P$
Since a rotation of 2^n brings back to point 0,
doing (2's complement of a number P) = $-P$ (*negate P*)
=> One can find **negative P** by complementing P

Q_3	Q_2	Q_1	Q_0	← Clock
-------	-------	-------	-------	
D_3	D_2	D_1	D_0	

To find
(2's complement of P) =

$$= 2^n - P =$$
$$= [(2^n - 1) - P] + 1 =$$
= [1's complement of P] +1

$5_{10} = 0101_2$

1010 = *1's complement of 5*
+1
——————
1011 = -5_{10}

OR

Examine the bits of P from right to left and copy all bits that are 0 and the first bit that is 1 and then complement the rest of the bits!

0000
1111   0001
1110   0010
      -1   0   +1
1101   -2   +2   0011
      -3   +3
1100 — -4   +4 — 0100
      -5   +5
1011   -6   +6   0101
Subtraction   -7 -8 +7   Addition
1010   0110
   1001   1000   0111

18

Signed numbers

- When a signed number is negative, the sign bit assumes a value of 1 while the rest of the number may be represented in one of three possible ways:

1) Signed magnitude representation
2) Signed 1's complement representation
3) Signed 2's complement representation

- The **signed magnitude representation (#1)** of a negative number consists of the sign bit followed by the **magnitude** of the number.

- In the other two representations, the number is represented in either the 1's or 2's complement of the number's **positive signed** value.

- Example: represent -14 with 8 bits, including the sign bit:

1) In signed magnitude form, −14 is represented as: 10001110 .
2) In signed 1's complement form, −14 is represented as: 11110001.
3) In signed 2's complement form, −14 is represented as: 11110010.

19

3) 2's Complement **Representation** of Signed Numbers

- The MSB (Most Significant Bit) = the sign bit
- **Positive number** = the *sign bit is* **0** and it is written in front of the number's *magnitude* (represented by its true binary value).

IMPORTANT NOTE: Positive numbers have the same representation in *sign-magnitude*, *1's complement* or *2's complement format*!

- **Negative number** = obtained by finding the 2's complement of the corresponding positive number (represented as explained above, i.e., including **0** as sign bit) that we want to negate. The result of the complementation will have the sign bit automatically changed to **1**!

- The *decimal equivalent* of a signed binary number that is represented in *2's complement format* is computed the same as for an unsigned number, except
  - ☐ the weight of the MSB (i. e., the sign bit:) is **-2^{n-1}** instead of **$+2^{n-1}$**

    Two ways to find the decimal equivalent of a signed number represented in 2's complement:
    a) e.g., for *n* = 5 bits: 11010 = $-2^4 + 2^3 + 2^1$ = -16 + 8+ 2 = -6, **or:**
    b) looking at say *N* = **1**1010 (and knowing that it is a sign number in 2's complement **representation**) one can see that it is a negative number N = –*x*. To find its magnitude |*N*| = *x*, one should negate *N* = –*x*, by finding its 2's complement: |*N*| = *x* = -(-*x*) = 2's compl. (**-*x***) = 2's compl. (**1**1010) = 00110 = 6 => *N* = -x = -6

  - ☐ the range of representable numbers is **[-2^{n-1}, $2^{n-1}-1$]**

20

uOttawa

BINARY REPRESENTATION OF SIGNED NUMBERS

Q_3	Q_2	Q_1	Q_0
D_3	D_2	D_1	D_0

Clock
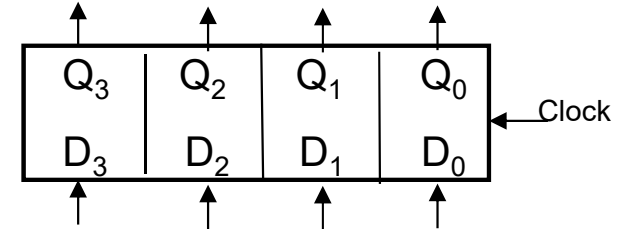
	# > 0	+5	# < 0	-5
1) sign-magnitude representation	0 followed by magnitude	**0**101	1 followed by magnitude	**1**101
2) 1's complement representation	0 followed by magnitude	**0**101	1's complement of the corresponding positive #	**1**010
3) 2's complement representation	0 followed by magnitude	**0**101	2's complement of the corresponding positive #	**1**011

Finding the complement of a number is equivalent with finding its negative, e.g.

2's complement of 5 = 2's complement of 0101 = 1011 = -5

2's complement of -5 = 2's complement of 1011 = 0101 = +5

Potential ambiguities of terminology

- One should be cautious when using the term *2's complement*, as it can mean either a **number format** or a **mathematical operator**.

  For example, 0111 represents decimal +7 in *two's-complement notation*, but the *two's complement of 7* in a 4-bit register is actually the "1001" bit string, which is the *two's complement representation* of **−7**.

- The statement "convert x to two's complement" may be ambiguous, since it could describe either:
  - ☐ the process of **representing x in two's-complement notation** without changing its value, or
  - ☐ the **calculation of the 2's complement**, which is the arithmetic negative of x if **2's complement representation** is used.

Converting from <u>two's complement representation</u>

- A 2's-complement number system encodes positive and negative numbers in a binary number representation. The weight of each bit is a power of two, except for the most significant bit, whose weight is the negative of the corresponding power of 2.

- The value w of a signed N-bit integer is given by the following formula:

$$w = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i\, 2^i$$

- The most significant bit (a_{N-1}) determines the sign of the number and is sometimes called the sign bit. Unlike in *sign-magnitude representation*, the sign bit also has the weight $-(2^{N-1})$ shown above. Using N bits, all integers from $-(2^{N-1})$ to $2^{N-1} - 1$ can be represented.

(From Wikipedia, the free encyclopedia) **22**

uOttawa

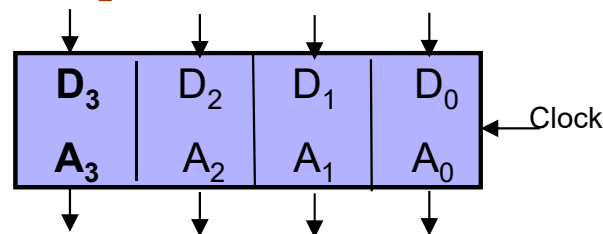# Addition and Subtraction of Signed Numbers Using 2's Complement Representation

Addition

+3	**0011**
+ (+4)	+ **0**100
+7	**0**111

+6	**0**110
+ (-3)	+ **1**101
+3	**1 0**011

-2	**1**110
+ (-6)	+ **1**010
-8	**1 1**000

+4	**0**100
+ (-7)	+ **1**001
-3	**1**101

D_3	D_2	D_1	D_0
A_3	A_2	A_1	A_0

Clock

Subtraction

add to the minuend the negate of the subtrahend (i.e., the subtrahend's 2's complement)

+ 6	110
- (+3)	- 011
+ 6	**0**110
+(-3)	+ **1**101
+ 3	**1 0**011

- 2	- 010
- (+6)	- 110
- 2	**1**110
+(-6)	+ **1**010
- 8	**1** 1000

"paper" format

Computer format

■ **Carry doesn't mean overflow in 2's compl.!**

23

Adding multi-bit numbers:

A	B	C_{in}	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

An adder is a combinational circuit made of full adders FA – see block diagram below. In this example the added numbers are $A_7 ... A_2 A_1 A_0$ and $B_7 ... B_2 B_1 B_0$, which are stored in two 8-bit registers A and B. It can be used for signed and unsigned numbers.

The sum is represented by $S_7 ... S_2 S_1 S_0$, & is to be fed into the input of another register. The input carry to the i-th bit is c_i, with i = 0, 1, 2, ...7; the **output carry** of the bit 7 is **C_8**.

108_{10} +   **0** 1 1 1 1 0 0 0 0 ← **Carry** → **C_8** | C_7 | C_6 C_5 C_4 C_3 C_2 C_1 | C_0

108_{10} + → **0 1 1 0 1 1 0 0** + A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0 +

90_{10} → **0 1 0 1 1 0 1 0** B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0

198_{10} ← **1 1 0 0 0 1 1 0** ← **Sum** → S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0

Detection of Addition Overflow

- **Overflow = the result is out of the domain**
- An overflow in an addition operation can be detected
  - For **unsigned numbers** $[0, 2^n-1]$: from the end carry-out of the most significant bit. A value of 1 means an overflow has occurred.
  - For **2's Complement Representation of Signed #'s** $[-2^{n-1}, 2^{n-1}-1]$ **an overflow is signalized :**
    1. if the addends have the same sign, but the sign of the sum is different than the addends' sign; or
    2. if the carry into the sign bit position and the carry out of the sign bit position are different,.

a_3 (sign)	a_2	a_1	a_0
1	1	0	1

b_3 (sign)	b_2	b_1	b_0
1	0	1	0

A +
B
———
S

carry: c_4

	c_3		c_2	c_1	c_0
a_3 (sign)		a_2	a_1	a_0	
b_3 (sign)		b_2	b_1	b_0	
s_3 (sign)		s_2	s_1	s_0	

$$OFL = c_n \oplus c_{n-1} = c_4 \oplus c_3$$

Carry: **1 0** 0 0

-3
-6
-9 < -8 → **OFL!**

1101+
1010
————
0111 = +7?

Carry   0 **1** 0 0

+5
+ +6
————
+11>+7

0101+
0110
————
1011 = -5?

25

Addition Overflow

- For **unsigned numbers**, an overflow in an addition operation is detected from the end carry out of the most significant bit. A value of 1 means an overflow has occurred.

- An overflow in a subtraction operation cannot possibly exist.

- For **signed numbers**, an overflow in <u>an addition or a subtraction</u> operation can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If the two carries are different, then an overflow condition is produced.

Additions / Subtractions with Signed Numbers Using Two's Complement Representation (1)

The **2's complement <u>representation</u>** of a negative number is the 2's complement of its absolute value.

■ **Add -176_{10} to -204_{10} in 2's Complement <u>representation</u>**

- How many bits are needed for operands and result to avoid overflow in 2's complement representation?

  Both -176 and -204 can be represented with at least 9 bits (1 sign bit + 8 bits for magnitude)

  The result is $-176 - 204 = -380 \subset [-512, +511] = [-2^9, 2^9-1] \rightarrow n-1 = 9 \rightarrow n = 10$

  or: $-380 = -(01\ 0111\ 1100) \rightarrow$ (in 2's Complement representation) = 10 1000 0100

  => **10 bits are needed, including the sign bit**

- Find 2's complement representation of the operands:

  Start from +176 = 00 1011 0000 $\rightarrow$ -176 (1's Complement) = 11 0100 1111
  -176 (in 2's Complement representation) = 11 0100 1111+1 = 11 0101 0000

  Then, from +204 = 00 1100 1100 $\rightarrow$ -204 (1's Complement) = 11 0011 0011
  -204 (in 2's Complement representation) = 11 0011 0011+1 = 11 0011 0100

27

uOttawa

Additions / Subtractions with Signed Numbers Using Two's Complement Representation (2)

$$C_{10}\ C_9\ C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1\ C_0$$

Carry   1 1 0 1 1 1 0 0 0 0 0

(-176)   **1** 1 0 1 0 1 0 0 0 0

+(-204)   **1** 1 0 0 1 1 0 1 0 0

_____

1 0 1 0 0 0 0 1 0 0

What is **1**0 1000 0100 (the result)? Let's convert it to signed decimal!

The msb is **1** => the result is a negative number (say –**x** = **1**0 1000 0100).

But what is the magnitude of this negative number (i.e. **x** = **|–x|**)?

To find **x**, let's do –(-x),

i.e. let's do complement **1**0 1000 0100 to get x = –(**1**0 1000 0100)

01 0111 1011 (1's Complement)

+   1

_____

01 0111 1100   = 380_{10} (decimal) = x = the magnitude of the negative number -x

Then the answer in decimal is **1**0 1000 0100$_2$ = -x = -380_{10}

Additions / Subtractions with Signed Numbers Using Two's Complement Representation (3)

Two numbers are given in 2's complement representation, employing 5 bits which include the sign bit.

$$X = 01010 \text{ and } Y = 10101$$

1. Calculate the sum (S = X+Y) and the difference (D=X-Y) of these numbers using additions and 2's complementation only; indicate if overflow occurs, and explain how a circuit can detect these situations.
2. <u>Convert in decimal</u> <u>and write each intermediate and final result</u>, to check the correctness of your assertions.

<u>SOLUTION</u>
Since both <u>X AND Y ARE IN 2'S COMPLEMENT REPRESENTATION</u>, their value in decimal is
$X = (01010)_2 = +10_{10}$ and
$Y = (10101)_2 = -|Y|$; $|Y| = -Y = $ 2's complement of $Y = (01011)_2 = +11_{10} => \underline{\mathbf{Y}} = -|Y| = \underline{\mathbf{- 11}}_{10}$
To calculate D we need $-Y$, which we just calculated above: $-Y = $ 2's complement of $Y = (01011)_2 = +11_{10}$

S=X+Y		2's complement representation					
Base 10	CY	0	0	0	0	0	
$\mathbf{10_{10}}$	X		0	1	0	1	0
$\mathbf{-11_{10}}$	+Y		1	0	1	0	1
	S		1	1	1	1	1

D=X-Y		2's complement representation					
Base 10	CY	0	1	0	1	0	
$\mathbf{10_{10}}$	X		0	1	0	1	0
$\mathbf{-(-11_{10})}$	-Y		0	1	0	1	1
	D		1	0	1	0	1

$S = (1111)_2 = -|S|$;
$|S| = -S = $ 2's complement of $S = (00001)_2 = +1_{10}$
$=> \underline{\mathbf{S}} = -|S| = \underline{\mathbf{- 1}}_{10}$

$D = (10101)_2 = -|D|$
$|D| = $ 2's complement of $D = +(01011)_2 = +11_{10}$
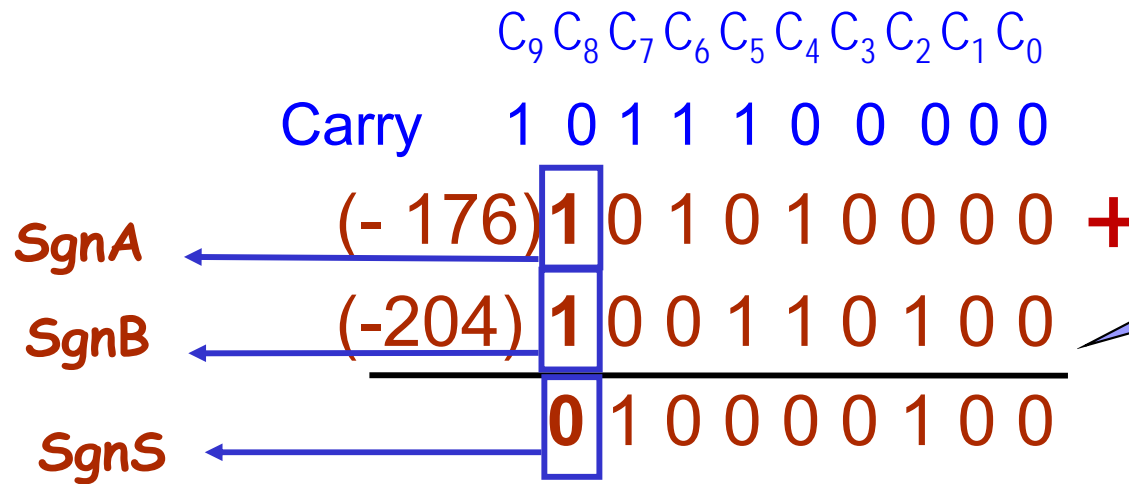$\Rightarrow D = -|D| = -11_{10}$!!!???
$\Rightarrow$ Overflow: adding 2 positive numbers (10 and 11) should give a positive number not -11_{10} !!!???

29

Addition/Subtraction Overflow Detection (1)

- **Add -176 (decimal) to -204 (decimal) using Two's Complement using only <u>9-bit representation</u>?**
- **Derive the logical expressions to detect overflow in terms of the last two carry bits or the sign bit of added numbers ?**

- The range of sign numbers in 2's complement representation with 9 bits is [-256,+255], so -380 is out of range.
- So, it is expected to get an overflow

30

Addition Overflow Detection (2)

$$C_9 \ C_8 \ C_7 \ C_6 \ C_5 \ C_4 \ C_3 \ C_2 \ C_1 \ C_0$$

Carry   1 0 1 1 1 0 0 0 0 0

(- 176) **1** 0 1 0 1 0 0 0 0 **+**

SgnA

(-204) **1** 0 0 1 1 0 1 0 0

SgnB

0 1 0 0 0 0 1 0 0

SgnS

Remember: A, B and S are stored in 9 bit registers not 10 like in the previous question

<u>Overflow Detection Expressions</u>

Overflow is detected if

1. if (SgnA=SgnB) ≠SgnS

$$(\overline{SgnA}.\overline{SgnB}.SignS \ +SgnA.SgnB.\overline{SgnS})$$

2. or if the carry bits TO (C_8) & FROM (C_9) the sign bit are different $(\overline{C_9}.C_8 + C_9.\overline{C_8}) = C_9 \oplus C_8$

Floating-Point Representation

- A floating-point is always interpreted to represent a number in the following form:
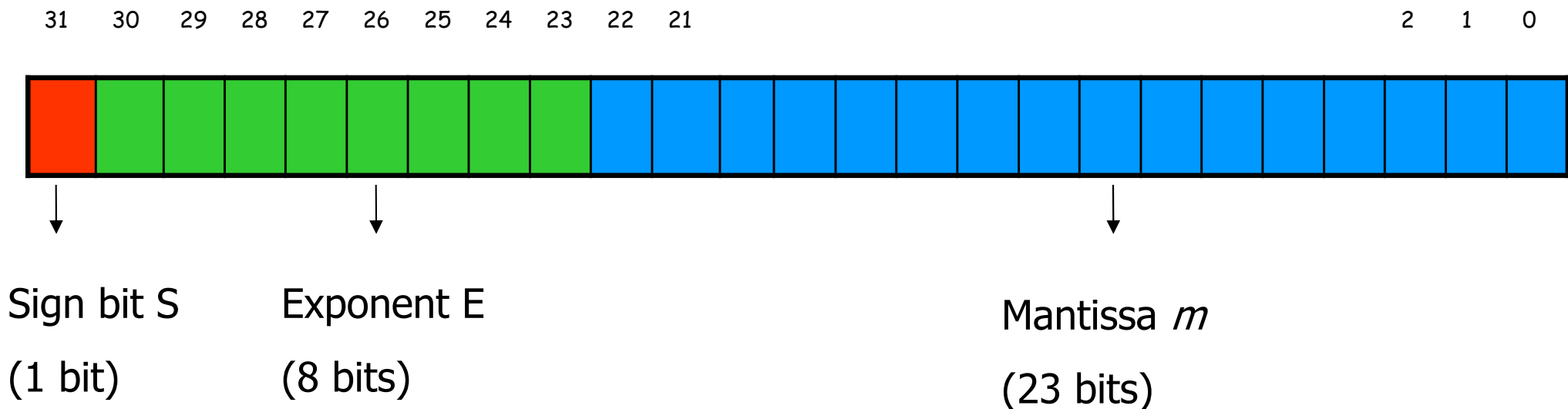
$$m \times r^e$$

- Only the **mantissa** m and the **exponent** e are physically stored in registers (including their signs).

- The radix and radix-point position in the mantissa are supposed to be known (fixed for a given hardware).

- A floating-point number is said to be **normalized** if the most significant digit (excluding the **sign bit**) is non-zero

- +1001.110 with an 8-bit mantissa and a 6-bit exponent:

Mantissa	Exponent
01001110	000100

uOttawa

IEEE 754 Standard

- Most common standard for representing floating point numbers
- Single precision: 32 bits, consisting of...
  - Sign bit (1 bit)
  - Exponent (8 bits)
  - Mantissa (23 bits)
- Double precision: 64 bits, consisting of…
  - Sign bit (1 bit)
  - Exponent (11 bits)
  - Mantissa (52 bits)

uOttawa

IEEE 754 Standard (32 bits) - Single Precision

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | | | | | | | | | | | 2 | 1 | 0 |

Sign bit S

(1 bit)

Exponent E

(8 bits)

Mantissa m

(23 bits)

$$value = (-1)^{sign}\left(1 + \sum_{i=1}^{23} b_{-i} 2^{-i}\right) \times 2^{(e-127)}$$

Range: $[-2^{128}, 2^{128}] \rightarrow [-10^{12}, 10^{12}]$

34

uOttawa

IEEE 754 Standard (32 bits) - Single Precision

Sign bit:

- positive, negative

Exponent:

- using unsigned number to represent signed number
- biased by adding 127 to the actual value (offset binary)

Mantissa:

- normalized, if 0<exponent<255, the first bit of mantissa (not shown) is 1
- de-normalized, if exponent = 0, mantissa is not =0,
- +/- 0, exponent = 0 , mantissa = 0
- +/- infinity, exponent = 255, mantissa = 0
- NaN, exponent = 255, mantissa is not 0

35

Converting <u>to</u> Floating Point

■ Express 36.5625_{10} as a 32-bit floating point number?

Express original value in binary

$$36.5625_{10} = 100100.1001_2$$

Normalize

$$100100.1001_2 = 1.001001001_2 \times 2^5$$

Put S, E, and M together to form 32-bit binary result

$$0\ 10000100\ 00100100100000000000000_2$$

S   E = 5+127   M

Converting <u>from</u> Floating Point (1)

- **What decimal value is represented by the following 32-bit floating point number?**

$$\texttt{C17B0000}_{16}$$

uOttawa

Converting <u>from</u> Floating Point (2)

- Express in binary and find S, E, and M

$$C17B0000_{16} =$$

1   10000010   11110110000000000000000$_2$

S   E   M

1 = negative
0 = positive

38

Converting <u>from</u> Floating Point (3)

- Find "real" exponent, n

n = E – 127

= 10000010_2 – 127

= 130 – 127

= 3

- Put S, M, and n together to form binary result
- (Don't forget the implied "1." on the left of the mantissa.)

`-1.1111011`$_2$ `x 2`^{n} `=`

`-1.1111011`$_2$ `x 2`^{3} `=`

`-1111.1011`$_2$

Converting <u>from</u> Floating Point (4)

– Express result in decimal

-1111.1011_2

-15

2^{-1} = 0.5
2^{-3} = 0.125
2^{-4} = 0.0625
0.6875

Answer: -15.6875

ASCII
American Standard Code for Information Interchange

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
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	{	|	}	~	

Binary-Coded Decimal (BCD)

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
ASCII$_{16}$ for Decimal	30	31	32	33	34	35	36	37	38	39

Example:

Decimal	127
BCD	0001 0010 0111
ASCII$_{16}$ for decimal	31 32 37
ASCII$_2$ for decimal	0011 0001 0011 0010 0011 0111
Binary	0111 1111
Hex	7F
ASCII$_{16}$ for Hex	37 46
ASCII$_2$ for Hex	0011 0111 0100 0110

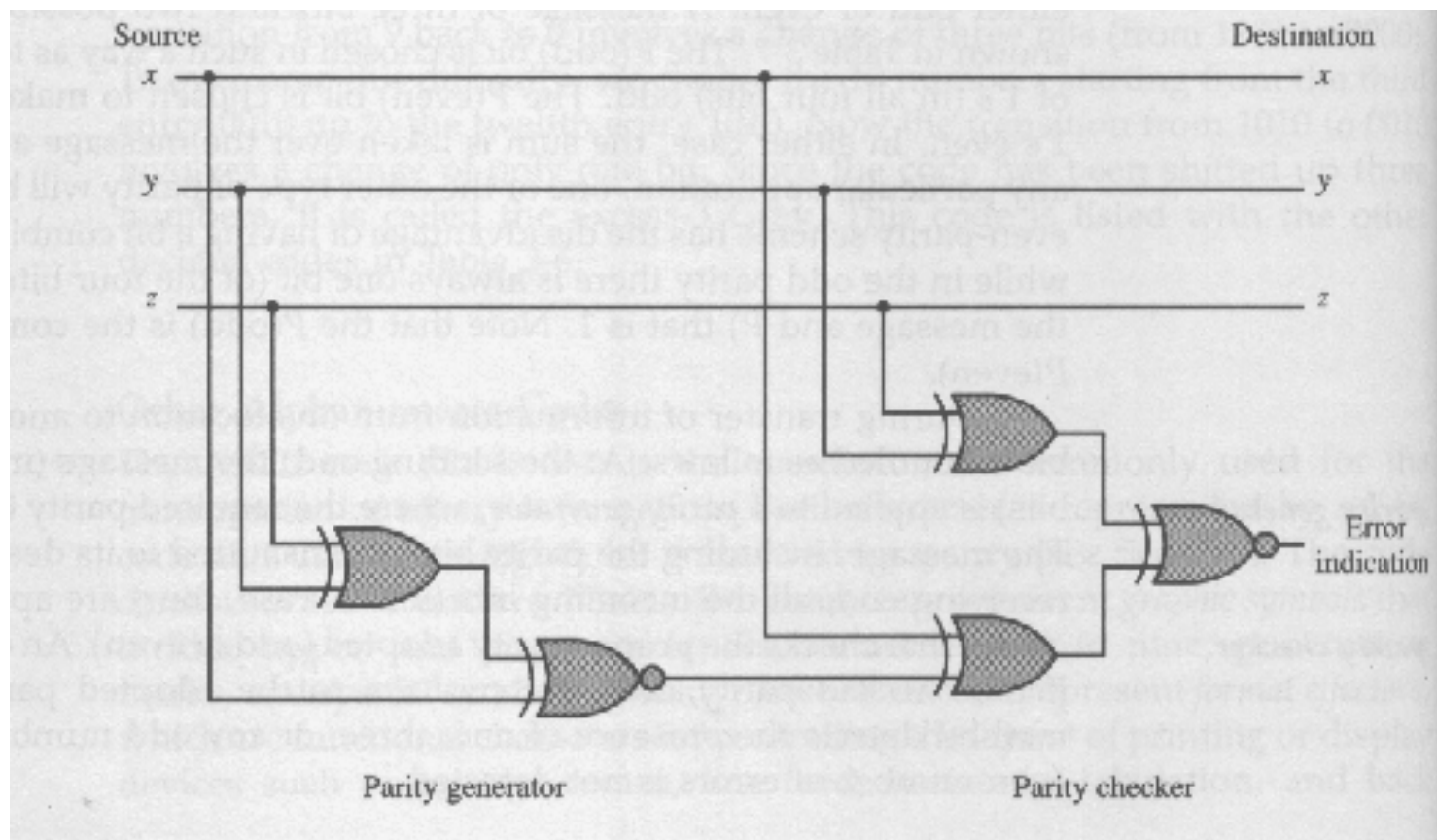| Decimal number	Binary-coded decimal (BCD) number	
0	0000	
1	0001	
2	0010	
3	0011	Code
4	0100	for one
5	0101	decimal
6	0110	digit
7	0111	
8	1000	
9	1001	
10	0001 0000	
20	0010 0000	
50	0101 0000	
99	1001 1001	
248	0010 0100 1000	

Error Detection Codes

- Binary information transmitted through a communication medium is subject to external noise that can change bits from 0 to 1 and vice versa.

- An error detection code is a binary code that detects digital errors during transmission.

- The most common error detection code is the **parity code**.

- A parity code is an extra bit added to the binary message to make the total of the number of 1's either odd (odd-parity code) or even (even-parity code).

- A message of three bits and two possible parity bits is shown below:

Message xyz	P(odd)	P(even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

43

Parity Code

- At the sending end, the message is applied to a parity generator to generate the required parity bit.

- At the receiving end, all the incoming bits (including the parity bit) are applied to a parity checker to check for the existence of an error.

44

uOttawa

ANNEX: Subtraction with pencil and paper of unsigned numbers

2^3	2^2	2^1	2^0			2^3	2^2	2^1	2^0		Step 0:
0	1	1	0	-		A_3	A_2	A_1	A_0		$D_0 = A_0 - B_0 = 0 - 1$!?
0	0	1	1			B_3	B_2	B_1	B_0		=> we need to borrow from
						D_3	D_2	D_1	D_0		$A_1 = 1 \times 2^1 = 2 = b_1$
		1+1		b (borrow)		b_4	b_3	b_2	b_1		
0	1	0	0	-		A_3	A_2	A_1	A_0		so $D_0 = b_1 + A_0 - B_0 = 2 + 0 - 1 = 1$
0	0	1	1			B_3	B_2	B_1	B_0		
		1				D_3	D_2	D_1	D_0		
	0	1+1		b (borrow)		b_4	b_3	b_2	b_1		Step 1:
0	1	0	0	-		A_3	A_2	A_1	A_0		$D_1 = A_1 - B_1 = 0 - 1$!?
0	0	1	1			B_3	B_2	B_1	B_0		=> we need to borrow from $A_2 = 2 = b_2$
			1			D_3	D_2	D_1	D_0		
	0 1+1	1+1		b (borrow)		b_4	b_3	b_2	b_1		
0	0	0	0	-		A_3	A_2	A_1	A_0		so $D_1 = b_2 + A_1 - B_1 = 2 + 0 - 1 = 1$
0	0	1	1			B_3	B_2	B_1	B_0		
		1	1			D_3	D_2	D_1	D_0		
	0 1+1	1+1		b (borrow)		b_4	b_3	b_2	b_1		Step 2:
0	0	0	0	-		A_3	A_2	A_1	A_0		$D_2 = A_2 - B_2 = 0 - 0 = 0$ no need to borrow
0	0	1	1			B_3	B_2	B_1	B_0		$b_3 = 0$
	0	1	1			D_3	D_2	D_1	D_0		
0	1	1	0	b (borrow)		b_4	b_3	b_2	b_1		Step 3:
0	0	0	0	-		A_3	A_2	A_1	A_0		$D_3 = A_3 - B_3 = 0 - 0 = 0$ no need to borrow
0	0	1	1			B_3	B_2	B_1	B_0		$b_4 = 0$
0	0	1	1			D_3	D_2	D_1	D_0		
0	1	1	0	b (borrow)		b_4	b_3	b_2	b_1		Step 4:
0	0	0	0	-		A_3	A_2	A_1	A_0		$b_4 = 0$ => no overflow
0	0	1	1			B_3	B_2	B_1	B_0		
0	0	1	1			D_3	D_2	D_1	D_0		

45