Chapter 03

Representation of numbers.

Chapter 03

- Encoding information → establishing a correspondence between its external representation and its internal representation in the machine, which is a sequence of bits.
- The representation (encoding) of numbers is necessary in order to store and manipulate them by a computer.
- The main problem is the limitation of the coding size: a mathematical number can take arbitrarily large values, while encoding in the machine must be done with a fixed number of bits.

Coding of natural numbers

- Natural numbers (non-negative integers) are encoded using a fixed number of bytes (1 byte = 8 bits). Commonly encountered encodings include 1, 2, 4 bytes, and more rarely, 8 bytes (64 bits).
- An n-bit encoding allows the representation of all natural numbers within the range: ***0 and 2^n-1***
- For example, with one byte (8 bits), you can represent (encode) numbers belonging to the interval:

$$[0, 2^8-1] = [0,255]$$

Coding of negative integers

- The plus **(+)** and minus **(-)** signs are not recognized by a computer, which only understands two states: 0 and 1.
- Therefore, they are represented by a bit that occupies the **leftmost position** of the considered number. This bit is called the sign bit. So, by convention, we represent the plus sign (+) as 0 and the minus sign (-) as 1.
- Negative numbers are represented in a computer using one of three methods: Sign and Absolute Value, One's Complement, or Two's Complement.

Coding of signed integers

- The representation of signed integers presents problems, especially in terms of sign representation. There are several ways to encode signed numbers:
  - ❑ Sign and absolute value coding
  - ❑ Restricted complement coding (1's complement)
  - ❑ True complement (2's complement)

Convention: Regardless of the encoding used, the most significant bit is reserved for sign representation: a negative number has a sign bit of 1, and a positive number has a sign bit of 0.

Sign and absolute value coding

- With n bits, the nth bit is reserved for sign, and the remaining n-1 bits are used for representing the absolute value of the number to be encoded.
- An n-bit encoding allows for the coding of all integers within the range: $[-(2^{n-1}-1), +(2^{n-1}-1)]$

1111	1110	1101	1100	1011	1010	1001	1000
- 7	- 6	- 5	- 4	- 3	- 2	- 1	- 0

0000	0001	0010	0011	0100	0101	0110	0111
+ 0	+ 1	+ 2	+ 3	+ 4	+ 5	+ 6	+ 7

Sign and absolute value coding

	n = 4 bits	n = 8 bits
range of converted values		
Status of the value zero		
Exemples		

Sign and absolute value coding

	n = 4 bits	n = 8 bits
range of converted values	[- 7, + 7]	[- 127 , + 127]
Status of the value zero		
Exemples		

Sign and absolute value coding

	n = 4 bits	n = 8 bits
range of converted values	[- 7, + 7]	[- 127 , + 127]
Status of the value zero	**The value zero is** 0000 = 1000	**The value zero is** 00000000 = 1000000
Exemples		

Sign and absolute value coding

	n = 4 bits	n = 8 bits
range of converted values	[- 7, + 7]	[- 127 , + 127]
Status of the value zero	**The value zero is** 0000 = 1000	**The value zero is** 00000000 = 1000000
Exemples	$+(3)_{10} = (\mathbf{0}\,011)_{SVA}$	$+(3)_{10} = (\mathbf{0}\,0000011)_{SVA}$

Sign and absolute value coding

	n = 4 bits	n = 8 bits
range of converted values	[- 7, + 7]	[- 127 , + 127]
Status of the value zero	**The value zero is** 0000 = 1000	**The value zero is** 00000000 = 1000000
Exemples	$+(3)_{10} = (\mathbf{0}\ 011)_{SVA}$	$+(3)_{10} = (\mathbf{0}\ 0000011)_{SVA}$
	$-(3)_{10} = (\mathbf{1}\ 011)_{SVA}$	$-(3)_{10} = (\mathbf{1}\ 0000011)_{SVA}$

Sign and absolute value coding

	n = 4 bits	n = 8 bits
range of converted values	[- 7, + 7]	[- 127 , + 127]
Status of the value zero	**The value zero is** 0000 = 1000	**The value zero is** 00000000 = 1000000
Exemples	$+(3)_{10} = (\mathbf{0}\ 011)_{SVA}$	$+(3)_{10} = (\mathbf{0}\ 0000011)_{SVA}$
	$-(3)_{10} = (\mathbf{1}\ 011)_{SVA}$	$-(3)_{10} = (\mathbf{1}\ 0000011)_{SVA}$
	$+(15)_{10}$ et $-(15)_{10}$	

Sign + absolute value coding

	n = 4 bits	n = 8 bits
range of converted values	[- 7, + 7]	[- 127 , + 127]
Status of the value zero	**The value zero is** 0000 = 1000	**The value zero is** 00000000 = 1000000
Exemples	$+(3)_{10} = (\mathbf{0}\ 011)_{SVA}$	$+(3)_{10} = (\mathbf{0}\ 0000011)_{SVA}$
	$-(3)_{10} = (\mathbf{1}\ 011)_{SVA}$	$-(3)_{10} = (\mathbf{1}\ 0000011)_{SVA}$
	$+(15)_{10}$ et $-(15)_{10}$ **impossible to represent**	$+(15)_{10} = (\mathbf{0}\ 0001111)_{SVA}$
		$-(15)_{10} = (\mathbf{1}\ \underbrace{0001111})_{SVA}$

Sign + Absolute Value Encoding

- Advantages:

Easy to interpret

- Disadvantages:

2 representations for zero (+0 and -0) and Problem adding two numbers of opposite signs

n = 16 bits
[- 32767 , + 32767]
La valeur zéro est = 0000 0000 0000 0000 = 1000 0000 0000 0000
$+(3)_{10} = (\mathbf{0}\ 000\ 0000\ 0000\ 0011)_{SVA}$
$-(3)_{10} = (\mathbf{1}\ 000\ 0000\ 0000\ 0011)_{SVA}$
$+(15)_{10} = (\mathbf{0}\ 000\ 0000\ 0000\ 1111)_{SVA}$
$-(15)_{10} = (\mathbf{1}\ 000\ 0000\ 0000\ 1111)_{SVA}$

Sign + Absolute Value Encoding

- Question:

Can the number -8 be represented using 4 bits?

- Answer:

It is impossible to represent the number -8 with 4 bits because its absolute value $|-8_{(10)}|$, which is equal to $1000_{(2)}$, already requires 4 bits. Therefore, we would need a minimum of 5 bits to represent it, including the sign bit.

Coding in restricted complement (CR) or 1's complement (C to 1)

One obtains the one's complement of a binary number by flipping (**changing 1 to 0 and 0 to 1**) each of its bits.

Positive numbers are encoded as in Sign and Absolute Value (SAV) encoding.

Negative numbers are derived from positive numbers through bitwise complementation, meaning:

(-N) = One's Complement (N) (assuming N is a positive number). An n-bit encoding allows for the coding of any integer within the range: $[-(2^{n-1}-1), +(2^{n-1}-1)]$

Coding in restricted complement (CR) or 1's complement (C to 1)

- **Exemple :**
- To encode +15 and -15 using one's complement encoding on 8 bits:

$(+15)_{10} =$

$(-15)_{10} =$

- Inconvenience: Two representations for zero

Coding in restricted complement (CR) or 1's complement (C to 1)

- **Exemple :**
- To encode +15 and -15 using one's complement encoding on 8 bits:

$$(+15)_{10} = (00001111)_2$$

$$(-15)_{10} = CR(00001111) =$$

- Inconvenience: Two representations for zero

Coding in restricted complement (CR) or 1's complement (C to 1)

- **Exemple :**
- To encode +15 and -15 using one's complement encoding on 8 bits:

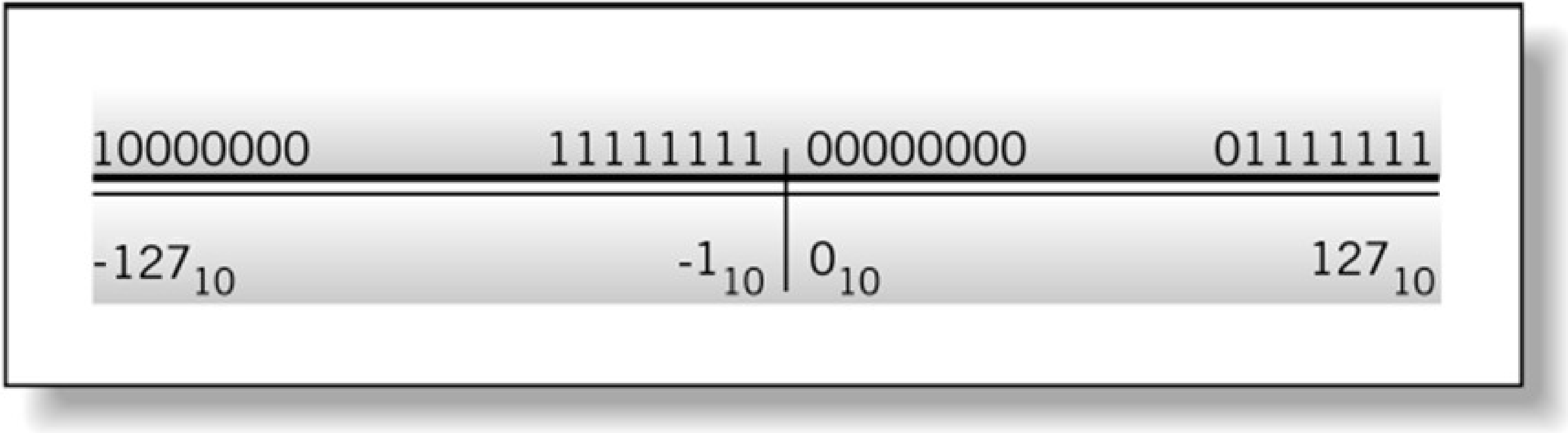
$$(+15)_{10} = (00001111)_2$$

$$(-15)_{10} = CR(00001111) = (11110000)_{CR}$$

- Inconvenience: Two representations for zero

Coding in restricted complement (CR) or 1's complement (C to 1)

- One's Complement Convention
- Range of representable numbers in one's complement on 8 bits
- This method is now obsolete



Two's Complement Encoding (CV) or Complement to 2 (C to 2)

To obtain the two's complement of an integer, simply add 1 to its one's complement:

CV(N) = CR(N) + 1, where N is any integer.

In two's complement encoding:

A positive number is represented in the same way as in the sign and absolute value encoding.

A negative number is represented by the two's complement of its opposite (which is, of course, positive).

An n-bit encoding allows for the coding of any integer within the range: $[-2^{n-1}, +(2^{n-1}-1)]$

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

- **Exemple :**

Encode +15 and -15 on 8 bits using CV coding,

$(+15)_{10} =$

$(-15)_{10} =$

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

- **Exemple :**

Encode +15 and -15 on 8 bits using CV coding,

$$(+15)_{10} = (00001111)$$

$$(-15)_{10} =$$

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

- **Exemple :**

Encode +15 and -15 on 8 bits using CV coding,

$$(+15)_{10} = (00001111)$$

$$(-15)_{10} = CR(00001111) + 1 =$$

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

- **Exemple :**

Encode +15 and -15 on 8 bits using CV coding,

$$(+15)_{10} = (00001111)$$

$$(-15)_{10} = CR(00001111) + 1 = (11110001)_{CV}$$

- Advantage: A single encoding for 0 and no issues with performing the addition operation.
- Disadvantage: Difficult to interpret.

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

- **Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' ,while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1   1 1 0 1 0 1 0 0

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

• **Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' , while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1

1 1 0 1 0 1 0 0

0

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

•**Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' ,while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1

1 1 0 1 0 1 0 0

0 0

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

•**Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' ,while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1

1 1 0 1 0 1 0 0

1 0 0

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

- **Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' ,while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1

1 1 0 1 0 1 0 0

1 1 0 0

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

•**Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' ,while inverting the remaining bits that follow.

Exemple:

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

•**Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' ,while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1

1 1 0 1 0 1 0 0

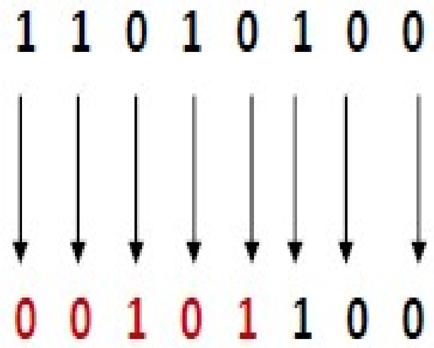
1 0 1 1 0 0

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

•**Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' ,while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1



Two's Complement Encoding (CV) or Complement to 2 (C to 2)

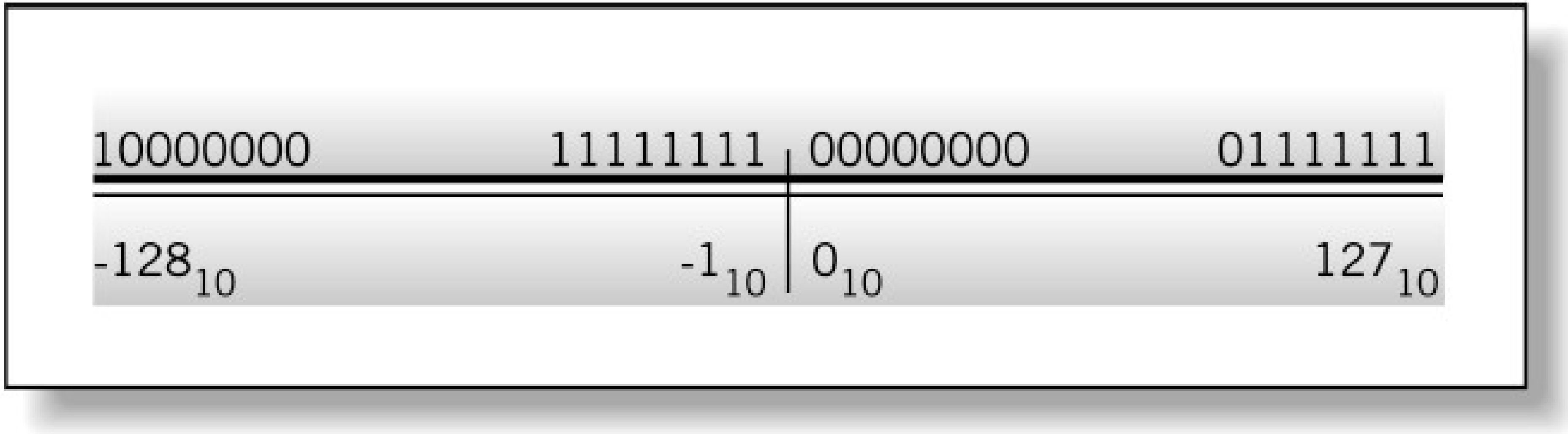
- **Note:** Alternatively, to find the two's complement of a number, you need to iterate through the bits of that number starting from the least significant bit and preserve all the bits before the first '1' , while inverting the remaining bits that follow.

Exemple:

0 1 0 0 0 1 0 1	1 1 0 1 0 1 0 0
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
1 0 1 1 1 0 1 1	0 0 1 0 1 1 0 0

Two's Complement Encoding (CV) or Complement to 2 (C to 2)

- Two's Complement Convention Range of representable numbers in two's complement on 8 bits



Subtraction using the complement method

A/ Restricted Complement RC or (C-to-1)

For a machine operating with restricted complement, subtraction is achieved by adding the restricted complement of the number to be subtracted to the number it needs to be subtracted from, along with the carry propagation (i.e., addition of the carry).

If there is no carry, it signifies that the number is negative. It is in the complemented form (restricted complement).

To obtain the desired value, one simply needs to find the RC (restricted complement) of this result.

Subtraction using the complement method

- **Exemple 1** :

Perform the following operation using the RC (Restricted Complement) technique on 8 bits : $(63)_{10}$- $(28)_{10}$.

- $(63)_{10} = (00111111)_2$
- $(28)_{10} = (00011100)_2$.
- $CR(28) = CR(00011100) = (11100011)_{CR}$.

Then, perform the addition of : $(00111111)_2 + (11100011)_{CR}$

In this example, is there a carry?

Well :

Subtraction using the complement method

- **Exemple 1** :

$$
\begin{array}{rll}
 & 0\,0111111 & \leftarrow (63)_{10} \\
+ & 1\,1100011 & \leftarrow CR(28)
\end{array}
$$

Subtraction using the complement method

- **Exemple 1** :

```
     0 0111111   ← (63)10
+    1 1100011   ← CR(28)
     ---------
  1  0 0100010
```

Subtraction using the complement method

- **Exemple 1** :

$$
\begin{array}{rll}
 & 0\,0111111 & \leftarrow (63)_{10} \\
+ & 1\,1100011 & \leftarrow CR(28) \\
\hline
= {}^{1} & 0\,0100010 &
\end{array}
$$

Carry

Subtraction using the complement method

- **Exemple 1** :

```
            0 0111111   ← (63)10
   +        1 1100011   ← CR(28)
            ---------
Carry  = 1  0 0100010
   +                1
            ---------
   =        0 0100011   ⟵ final result (+35)10
```

Subtraction using the complement method

- **Exemple 2** :

Perform the following operation using the RC (Restricted Complement) technique on 8 bits:

- $(28)_{10} - (63)_{10}$
- $(63)_{10} = (00111111)_2$ et $(28)_{10} = (00011100)_2$.
- $CR(63) = CR(00111111) = (11000000)_{CR}$.

Subtraction using the complement method

- **Exemple 2**

$$\begin{array}{ll} 0\,0011100 & \leftarrow (28)_{10} \\ 1\,1000000 & \leftarrow CR(63) \\ \hline \end{array}$$

Subtraction using the complement method

- **Exemple 2**

$$
\begin{array}{rll}
 & 0\,0011100 & \leftarrow (28)_{10} \\
 & 1\,1000000 & \leftarrow CR(63) \\
\hline
= & \mathbf{1}\,1011100 & \longleftarrow \text{Final result } (-35)_{10}.
\end{array}
$$

Subtraction using the complement method

- **Exemple 2**

$$
\begin{array}{rl}
 & 0\,0011100 \quad \leftarrow (28)_{10} \\
 & \underline{1\,1000000} \quad \leftarrow CR(63) \\
= & 1\,1011100 \quad \longleftarrow \text{Final result } (-35)_{10}.
\end{array}
$$

There is no carry

Subtraction using the complement method

- **Exemple 2**

$$
\begin{array}{rl}
& 0\,0011100 \quad \leftarrow (28)_{10} \\
& 1\,1000000 \quad \leftarrow CR(63) \\
\hline
= & 1\,1011100 \quad \longleftarrow \text{Final result } (-35)_{10}.
\end{array}
$$

- There is no carry, the result is negative, so we calculate its RC (Restricted Complement):

  $CR(11011100) = (00100011)_2 = (35)_{10}$

  confirming the equality: (28)10 - (63)10=(-35)10.

Subtraction using the complement method

B/ True Complement: The principle is the same as for the RC, except this time we ignore the carry. Instead of working with RC, we determine True Complements.

Example 1: Perform the following operation using the TC (True Complement) technique on 8 bits:

- $(63)_{10}-(28)_{10} = (63)_{10} + CV(28)$
- $(63)_{10} = (00111111)_2$
- $(28)_{10} = (00011100)_2$.
- $CV(28) = CR(00011100) + 1 = (11100100)_{\mathbf{cv}}$.

Subtraction using the complement method

- In this example, we obtain a result:

00111111	$\leftarrow (63)_{10}$
11100100	$\leftarrow$ CV(28)

Subtraction using the complement method

- In this example, we obtain a result:
- $(63)_{10}-(28)_{10} = (+35)10$.

$$
\begin{array}{llll}
 & & 00111111 & \leftarrow (63)_{10} \\
 & & 11100100 & \leftarrow CV(28) \\
 & & \text{----------} & \\
\text{Carry} \rightarrow & 1 & = 00100011 & \leftarrow \text{Final result} \quad (+35)_{10}
\end{array}
$$

Subtraction using the complement method

- In this example, there is a carry, so it needs to be ignored. We obtain a positive result: (63)10-(28)10 = (+35)10.

```
                    00111111      ← (63)10
                    11100100      ← CV(28)
                    ---------
Carry →  1      =   00100011      ← Final result  (+35)10
```

There is a carry

Subtraction using the complement method

- **Exemple 2** :

Perform the following operation using the TC (True Complement) technique on 8 bits. : $(28)_{10} - (63)_{10}$

- $(28)_{10} - (63)_{10} = (28)_{10} + CV(63)$
- $(63)_{10} = (00111111)_2$
- $(28)_{10} = (00011100)_2$
- $CV(63) = CR(0\ 0111111) + 1$

  $= (1\ 1000001)_{CV}$.

$$\begin{array}{r} 0\ 0011100 \\ +\ 1\ 1000001 \\ \hline \end{array}$$

Subtraction using the complement method

$$
\begin{array}{rrl}
 & 0\,0011100 & \leftarrow (28)_{10} \\
+ & 1\,1000001 & \leftarrow CV(63) \\
\hline
= & 1\,1011101 &
\end{array}
$$

- In this example, there is no carry; the result is negative, so we calculate its TC (True Complement):
- CV(11011101) = CR(11011101) +1 = 00100010 + 1 = 00100011. On obtient au final : $(28)_{10}-(63)_{10} = (-35)_{10}$.

Subtraction using the complement method

$$\begin{array}{rll} & 0\,0011100 & \leftarrow (28)_{10} \\ + & 1\,1000001 & \leftarrow CV(63) \\ \hline = & 1\,1011101 & \longleftarrow \text{Final result } (-35)_{10}. \end{array}$$

- In this example, there is no carry; the result is negative, so we calculate its TC (True Complement):
- CV(11011101) = CR(11011101) +1 = 00100010 + 1 = 00100011. On obtient au final : $(28)_{10}-(63)_{10} = (-35)_{10}$.

Problems related to the length of numbers

- Reminder:
- In two's complement (true complement) on n bits, the numbers are between -2^{n-1} et $+(2^{n-1}-1)$
- Addition of two positive numbers:
- When adding two positive numbers, it is possible to obtain a negative result (the sign bit of the result is 1). This is because the result does not fall within the authorized range with the given number of bits.

Problems related to the length of numbers

- Example:
- Perform the following operation using the Overflow technique on 8 bits: $(+49)_{10} + (88)_{10}$ In this example, we added two positive numbers, both fitting into 8 bits.
- Unfortunately, we obtained a result that is outside the range of values allowed for coding in 8 bits.

$$[-2^{n-1}, +(2^{n-1} - 1)] = [-128, +127]$$

- with n = 8. Indeed, the result of 137 (49+88=137) is outside this interval.

Problems related to the length of numbers

	0 0110001		$\leftarrow (+49)_{10}$
+	0 1011000		$\leftarrow (88)_{10}$

=	1 0001001		

Problems related to the length of numbers

$$
\begin{array}{lll}
 & \mathbf{0}\,0110001 & \leftarrow (+49)_{10} \\
+ & \underline{\mathbf{0}\,1011000} & \leftarrow (88)_{10} \\
= & \mathbf{1}\,0001001 &
\end{array}
$$

Overflow

Problems related to the length of numbers

- Addition of two negative numbers:
- When adding two negative numbers represented by their Two's Complement (sign bit as 1), it is possible to obtain a positive result (the sign bit of the result is 0).
- Indeed, there is always a carry because the most significant bits of the numbers being added are 1. Example 1: Perform the following operation using the Overflow technique on 8 bits: $(-32)_{10} + (-31)_{10}$

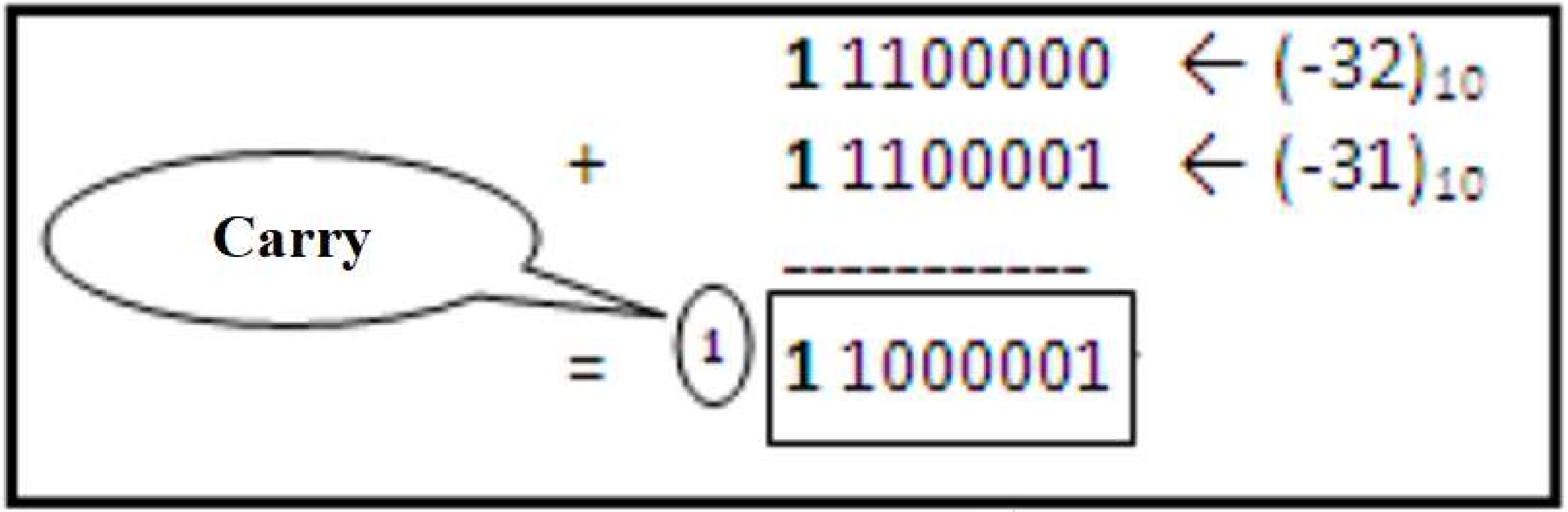
Problems related to the length of numbers

- $(-32)_{10} + (-31)_{10}$

$$\begin{array}{rrrl} & & 1\,1100000 & \leftarrow (-32)_{10} \\ + & & 1\,1100001 & \leftarrow (-31)_{10} \\ & & \text{-----------} & \\ = & ^{1} & 1\,1000001 & \end{array}$$

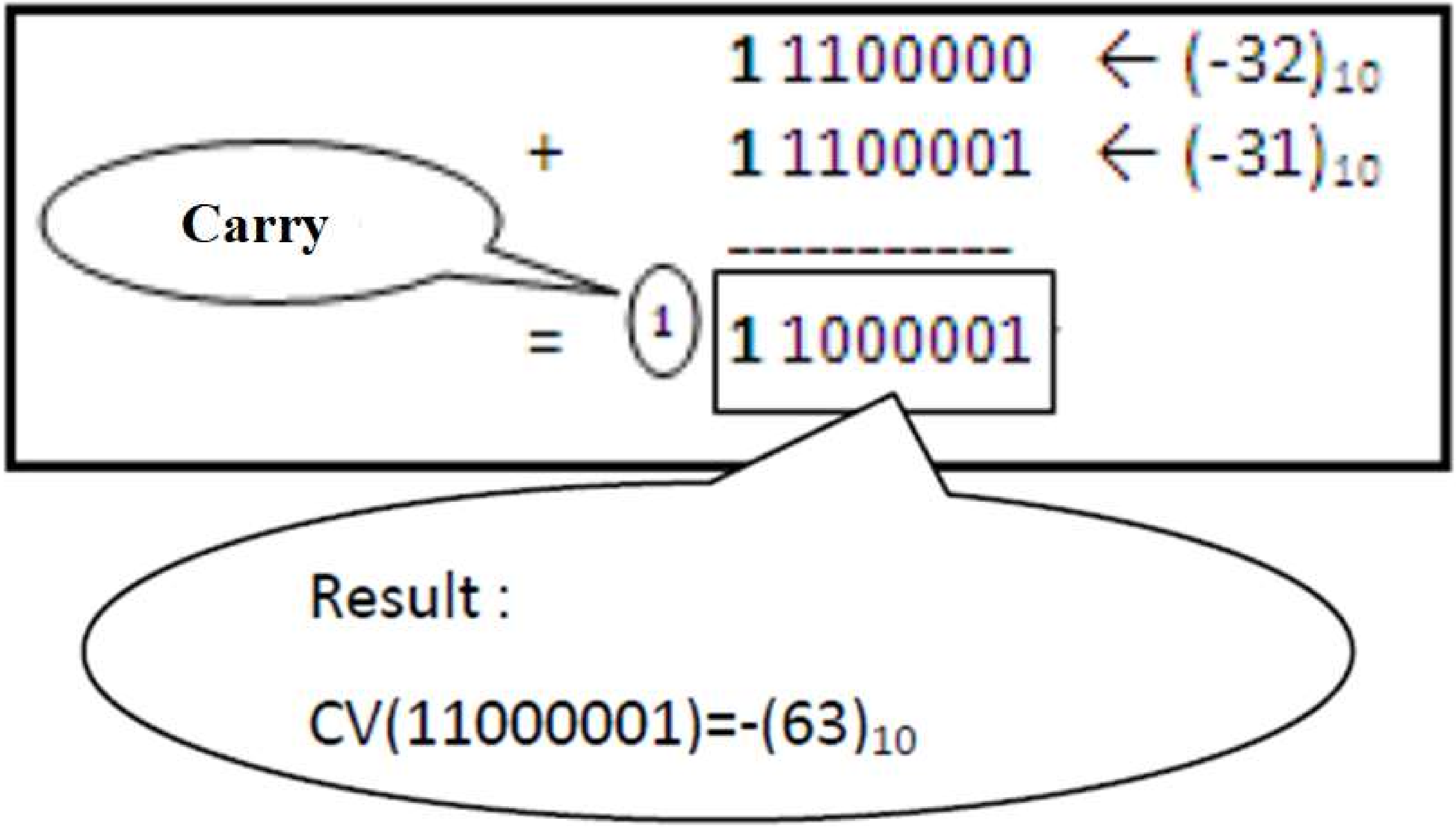
Problems related to the length of numbers

- $(-32)_{10} + (-31)_{10}$



Problems related to the length of numbers

- $(-32)_{10} + (-31)_{10}$



Problems related to the length of numbers

- In this example, we added two negative numbers (notice the sign bit is at 1) and obtained a negative number (observe the sign bit at 1).

- Although we obtained a carry, our result is correct; we simply need to ignore this carry (as we are using two's complement or true complement coding here).

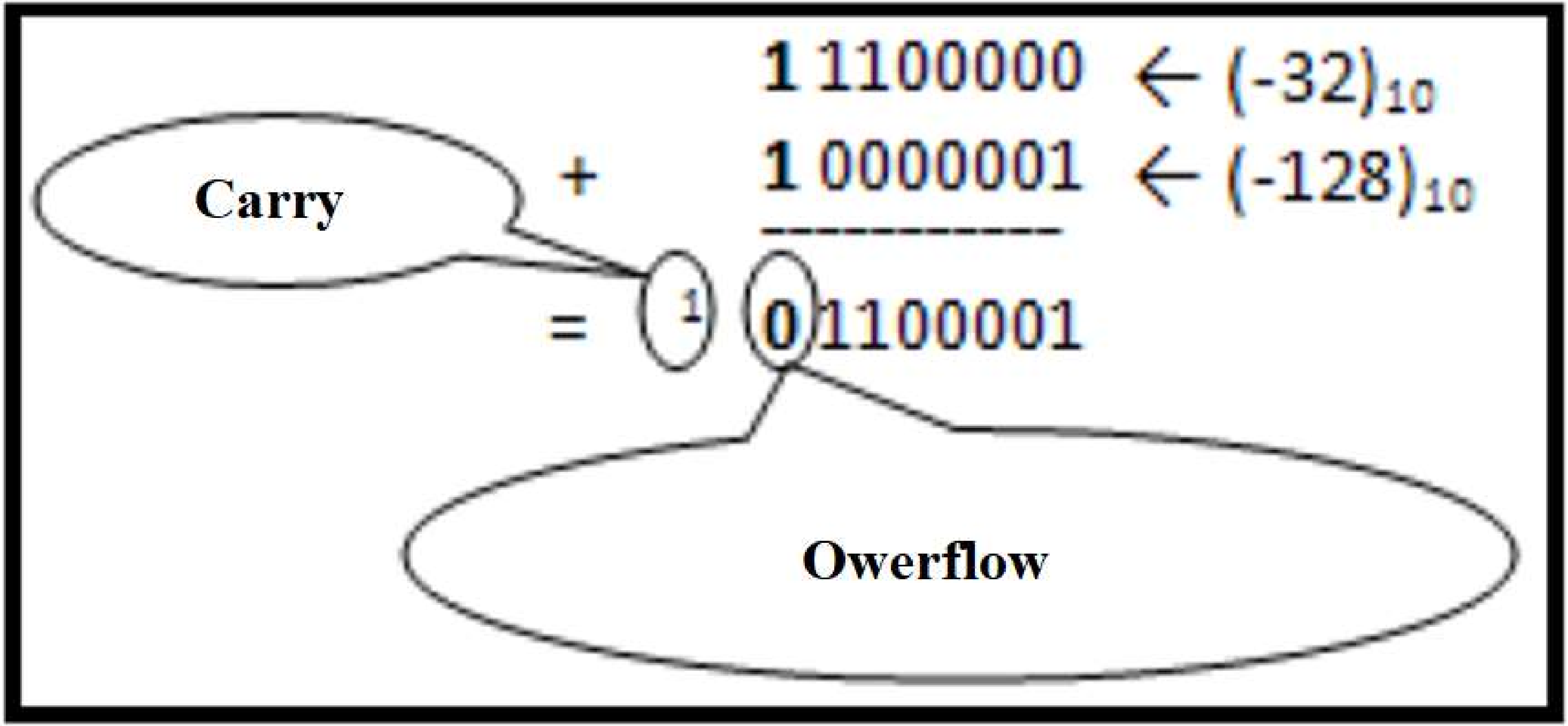
Problems related to the length of numbers

- Example 2: Perform the following operation using Overflow technique on 8 bits: $(-32)_{10} + (-128)_{10}$

- By ignoring the carry, we obtain a positive result (sign bit is 0); therefore, we deduce there is overflow or capacity exceeding.

- In decimal: $(-32)_{10} + (-127)_{10} = (-159)_{10}$. -159 is not within the range [-128 and +127].

Problems related to the length of numbers



Problems related to the length of numbers

- Overflow Indicator:
- Computers use an overflow indicator, which is set to 1 if the sign bit of the result is 0 while the two numbers being added are negative, or when the sign bit of the result is 1 while the two numbers being added are positive.