DUT-IDIA Semestre 1

Module Architecture des Ordinateurs

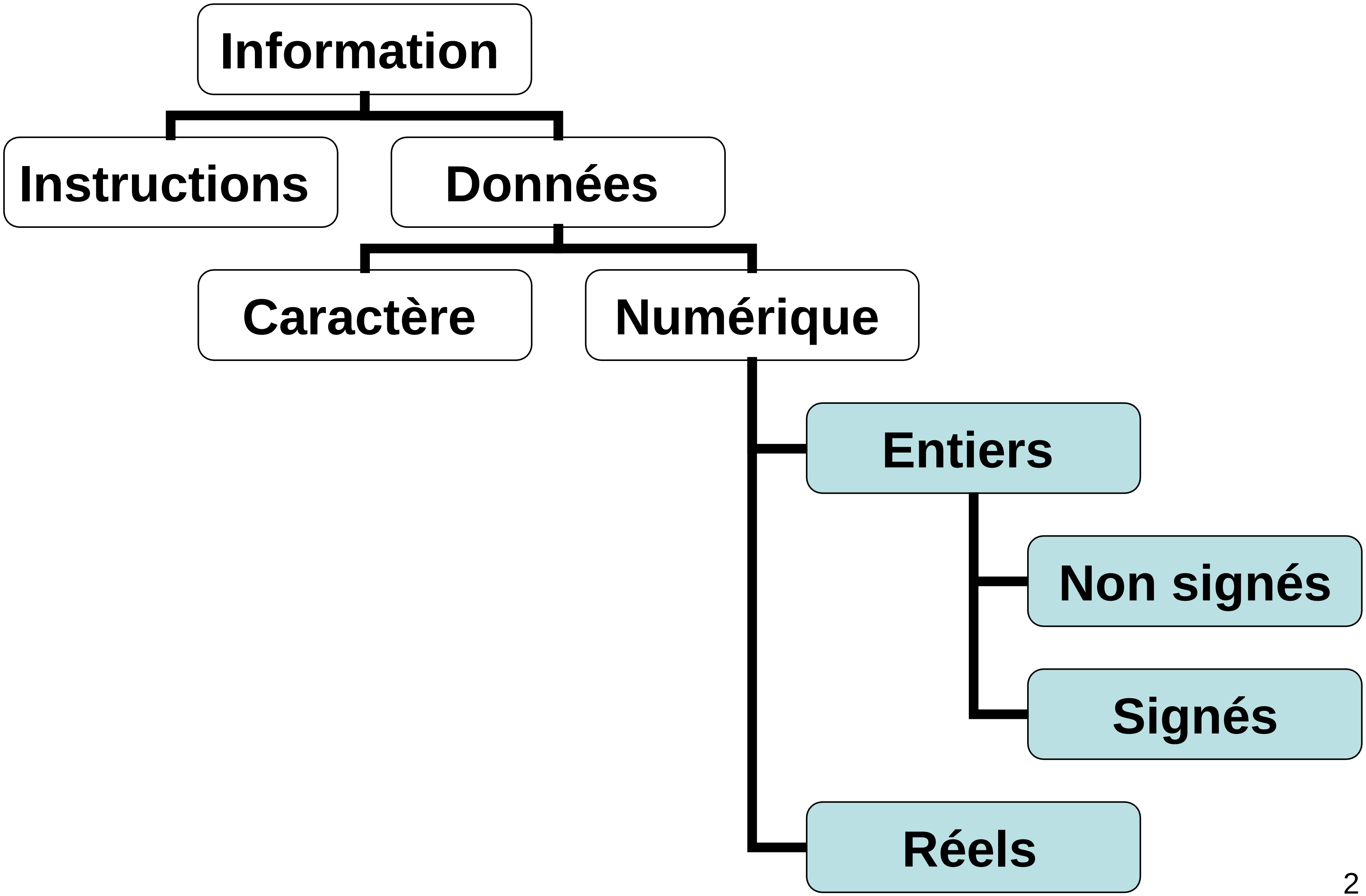
Chapitre I

SYSTÈMES DE NUMÉRATION

Pr : Mustapha Johri

Année Universitaire: 2024 - 2025

1. Quels types d'informations en informatique ?



1. Unités de mesure d'informations

- En informatique, la **base 2 (binaire)** correspond au fonctionnement d'**électronique utilisé dans l'informatique (*chargé/déchargé* ou *ouvert fermé*…)**.
- Le **Bit (BInary digIT)** est la plus petite unité de valeur, il ne peut avoir que **deux valeurs** (**1** ou **0**) .

    - **Quartet** est un groupe de **4 bits.**
    - **Octet (Byte)** est un groupe de **8 bits**, il peut avoir **256 valeurs**.

- Un **fichier** est un ensemble de **bits** qui forme ce qu'on appelle un **mot binaire**.

1. Unités de mesure d'informations

- En informatique, la **base** est en **2**, il faut que les **multiples** soient des **multiples de 2,** soit **1024 (2^{10})** et **non 1000.**

  - **1 Octet =8 bit**
  - **1 kilo-octet (Ko) = 1 Ko équivaut à 1024 octets.**
  - **1 Méga-octet (Mo) = 1 Mo = 1024 Ko.**
  - **1 Giga-octet (Go) = 1 Go = 1024 Mo.**
  - **1 Téra-octet (To) = 1 To = 1024 Go**
  - **(Peta, Exa, Zetta, Yotta…),**

1. Autres unités de mesure d’informations

- Pour mesurer les **débits** (**Réseaux et connexions internet**), il est très courant d'utiliser le **Bit/s** (**bits par seconde** ou **bps**).

- En **comparant des imprimantes**, on tombe sur le nombre des **pages par minute** (**ppm**) que l'imprimante est capable d'imprimer.

- Un **pixel** est le plus **petit carré affichable sur un écran**. On rajoute souvent l'unité **dpi** (**dot per inch** ou **ppp** pour **points par pouce** ou **pixels par pouce**).

- Plus le **nombre de pixels** par pouce est **élevé**, meilleure est la **qualité**.

2. Opérations binaires

- Les opérations sur les nombres binaires s’effectuent de la même façon que sur les nombres décimaux, mais il **ne faut pas oublier** que les seules symboles utilisés sont: **1** et **0**.

❖ Addition

- **0 + 0 = 0**
- **0 + 1 = 1**
- **1 + 1 = 0 et Retenue = 1**

❖ Soustraction

- **0 – 0 = 0**
- **1 – 0 = 1**
- **1 – 1 = 0**
- **0 – 1 = 1 et Retenue = 1**

❖ Multiplication

- **0 * 0 = 0**
- **0 * 1 = 0**
- **1 * 1 = 1**

❖ Division

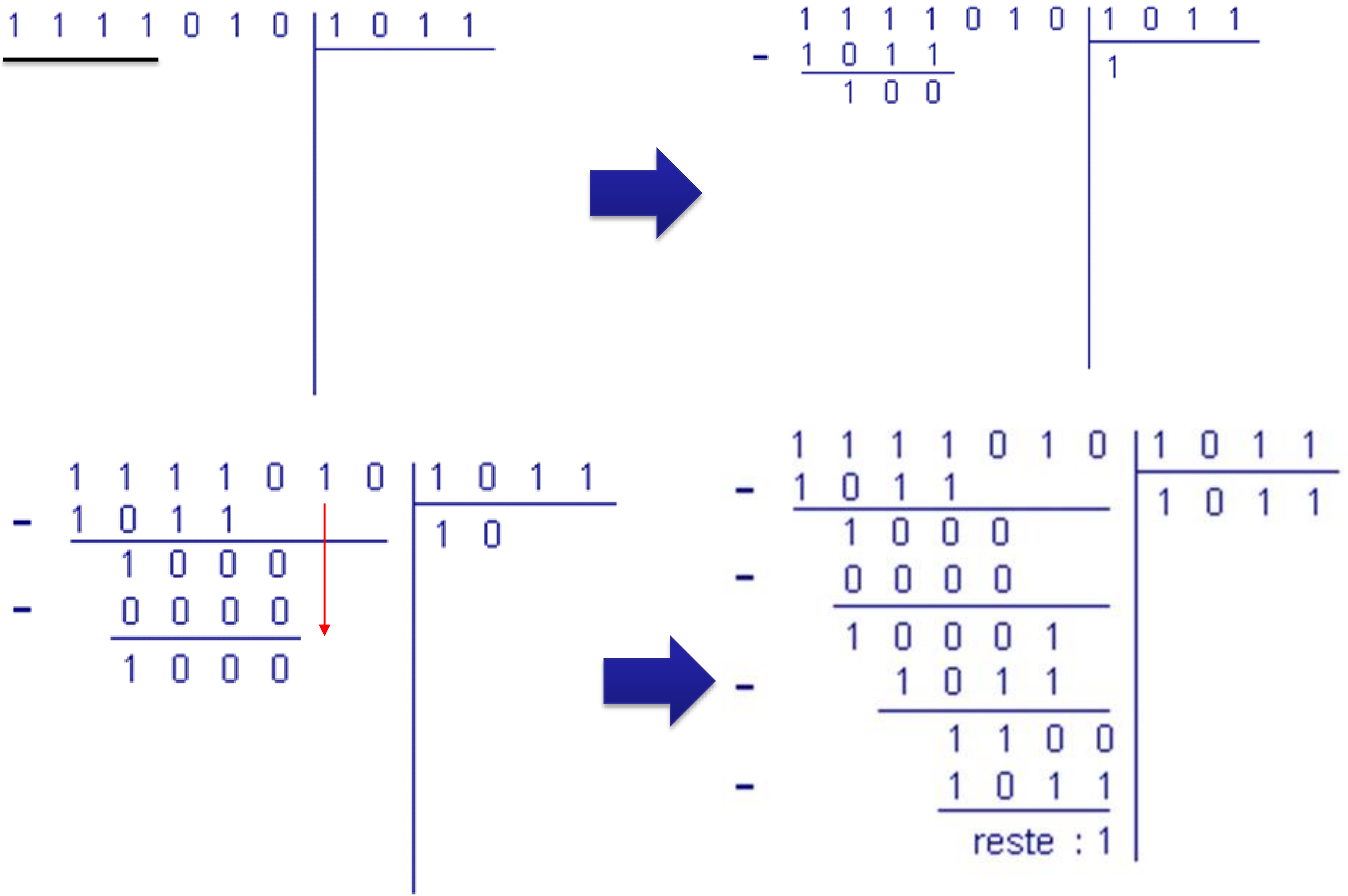
- **0 / 0 = Erreur !!**
- **0 / 1 = 0**
- **1 / 0 = Erreur !!**
- **1 / 1= 1**

2. Opérations binaires

Exercices : **Effectuez les opérations suivantes**

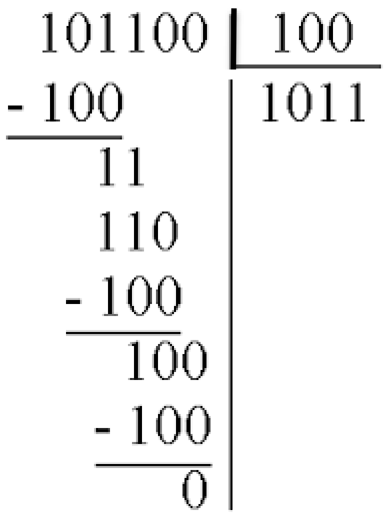
$$\begin{array}{r} 1100 \\ +\ 1000 \\ \hline 10100 \end{array} \qquad \begin{array}{r} 1100 \\ -\ 1000 \\ \hline 0100 \end{array} \qquad \begin{array}{r} 1011 \\ \times\quad 11 \\ \hline 1011 \\ 1011 \\ \hline 100001 \end{array}$$

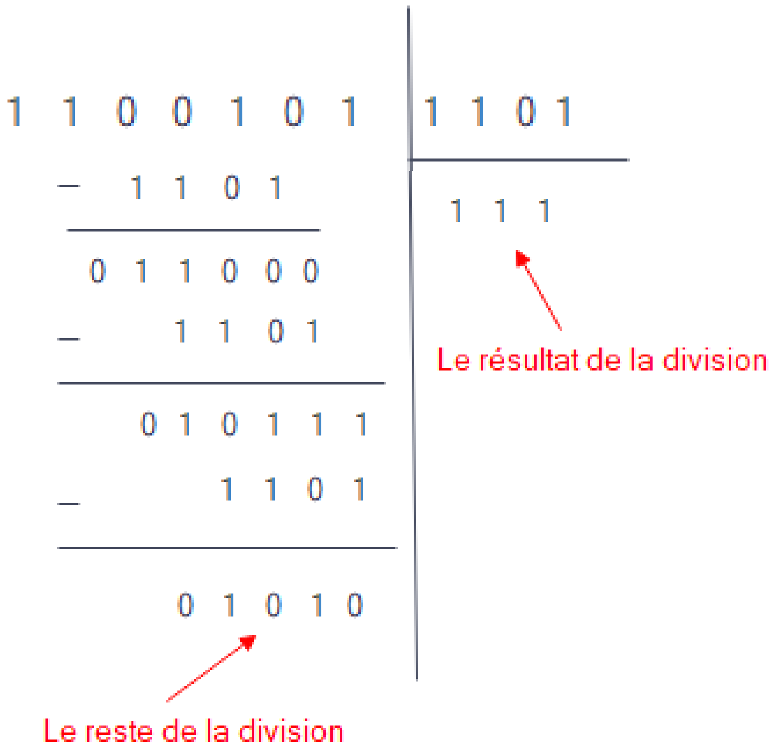
2. Opérations binaires



2. Opérations binaires

Exercices : Effectuez les divisions binaires suivantes :





3. Bases et conversions

❑ Base 10: Décimal

- On utilise **dix** symboles différents:

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

- N'importe quelle **combinaison** des symboles **{ 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 }** donne un nombre.

2 3 3 4 5 6 7

Poids fort

Poids faible

345 , 567

Partie entière

Partie fractionnaire

3. Bases et conversions

Base 10 : Décimal

- Soit le nombre **1978**, il peut être écrit sous la forme :

$$1978 = 1000 + 900 + 70 + 8$$

$$1978 = 1*1000 + 9*100 + 7*10 + 8*1$$

$$1978 = 1*10^3 + 9*10^2 + 7*10^1 + 8*10^0$$

Cette forme est appelée: **une forme polynomiale.**

- Un nombre **réel** peut être écrit aussi sous **la forme polynomiale.**

$$1978{,}265 = 1*10^3 + 9*10^2 + 7*10^1 + 8*10^0 + 2*10^{-1} + 6*10^{-2} + 5*10^{-3}$$

3. Bases et conversions

❑ Base 2: Binaire

- Dans un **système binaire**, pour exprimer n'importe quelle valeur on utilise uniquement **2 symboles : { 0 , 1}**

bit $\quad (\boxed{1}101)_2 \quad$ **La base**

$(\boxed{1}\ 1\ 0\ \boxed{1})_2$

Le bit du poids fort $\qquad$ **Le bit du poids faible**

- Un nombre binaire peut être écrit aussi sous la **forme polynomiale:**

$$(1110)_2 = 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = (14)_{10}$$

$$(1110{,}101)_2 = 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = (14{,}625)_{10}$$

3. Bases et conversions

❑ Base 8: Octal

- **8 symboles** sont utilisés dans ce système:

{ 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 }

$$(127)_8 = 1*8^2 + 2*8^1 + 7*8^0$$

$$(127{,}65)_8 = 1*8^2 + 2*8^1 + 7*8^0 + 6*8^{-1} + 5*8^{-2}$$

- Le nombre **1289** **n'existe pas** dans la base 8 puisque les symboles **8 et 9 n'appartiennent** pas à la base.

3. Bases et conversions

❑ Base 16 : Hexadécimal

- On utilise **seize (16) symboles:**

Décimal	Hexadécimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

3. Bases et conversions

❑ Conversion d'une Base X au Décimal

- Cette conversion est assez simple puisqu'il suffit de faire le développement en **polynôme** de ce nombre dans la **base X** , et de **faire la somme par la suite**.

$$(1101)_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = (13)_{10}$$

$$(1A7)_{16} = 1*16^2 + A*16^1 + 7*16^0 = 1*16^2 + 10*16^1 + 7*16^0 = 256 + 160 + 7 = (423)_{10}$$

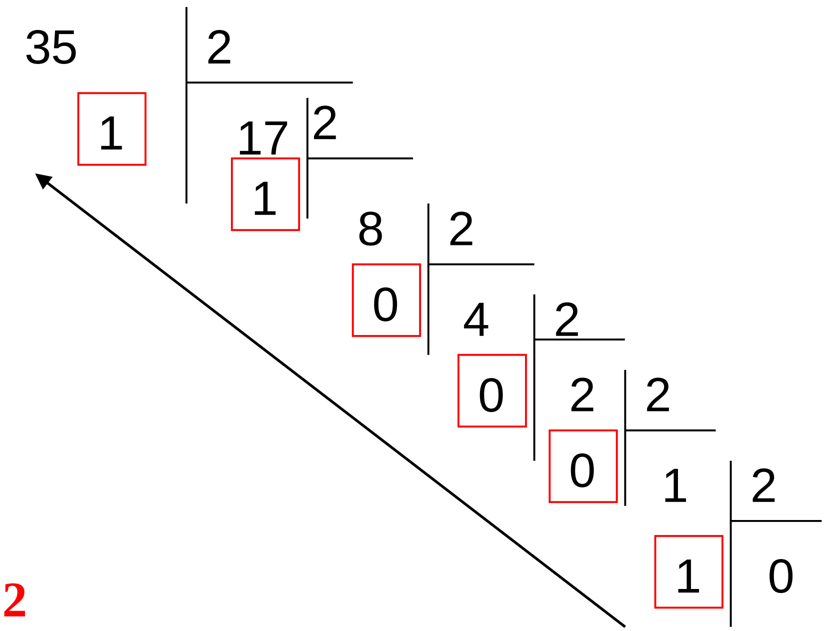
$$(1101{,}101)_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = (13{,}625)_{10}$$

$$(43{,}2)_5 = 4*5^1 + 3*5^0 + 2*5^{-1} = 20 + 3 + 0{,}4 = (23{,}4)_{10}$$

3. Bases et conversions

❑ Conversion du Décimal au Binaire

- Le principe consiste à faire des **divisions successives** du nombre **sur 2** et prendre le **reste des divisions dans l'ordre inverse.**



Après division :
on obtient : $\mathbf{(35)_{10}=(100011)_2}$

3. Bases et conversions

❑ Conversion du Décimal au Binaire (cas d'un nombre réel)

- La **partie entière** est transformée en effectuant des **divisions successives.**
- La **partie fractionnaire** est transformée en effectuant des **multiplications successives par 2** .

$35,625=(?)_2$

P.E= 35 = $(100011)_2$

PF= 0,625 = $(?)_2$

$0,625 * 2 = \boxed{1},25$

$0,25 * 2 = \boxed{0},5$

$0,5 * 2 = \boxed{1},0$

$(0,625)=(0,101)_2$

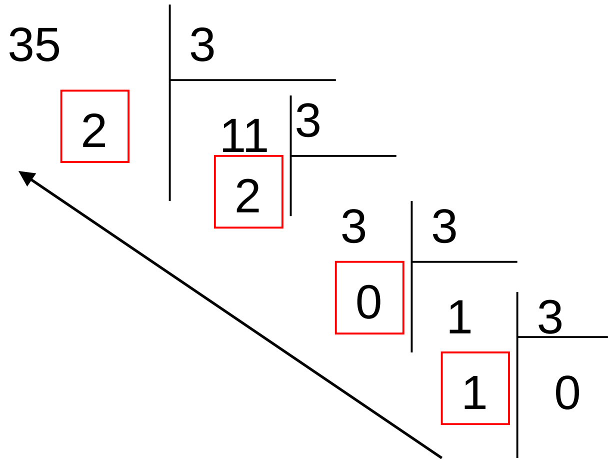
Donc **$35,625=(100011,101)_2$**

3. Bases et conversions

❑ Conversion du Décimal à une Base X

- La conversion se fait **en prenant les restes des divisions successives** sur la **base X** dans le **sens inverse**.

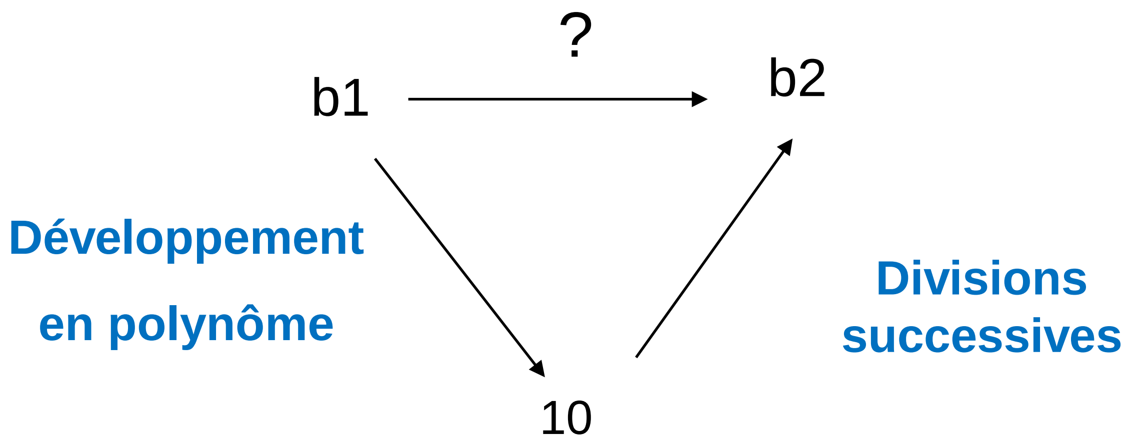
$(35)_{10}=(1022)_3$



3. Bases et conversions

❑ Conversion de la Base b1 à une Base b2

- Il n’existe pas de méthode pour passer d’une base **b1** à une autre base **b2 directement**.
- L’idée est de convertir le nombre de la **base b1** à **la base 10** et de convertir le résultat de **la base 10** à la **base b2** .



3. Bases et conversions

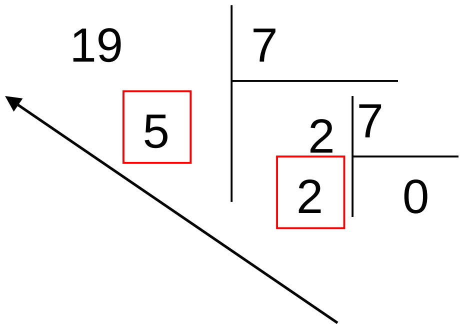
❑ Conversion de la Base b1 à une Base b2

- **Exemple** : **(34)$_5$=(?)$_7$**

$$(34)_5 = 3*5^1 + 4*5^0 = 15 + 4 = (19)_{10} = (?)_7$$

$(19)_{10}=(25)_7$

$(34)_5=(25)_7$



3. Bases et conversions

❑ Conversion du Binaire à Octal

- En **Octal,** chaque symbole de la base s'écrit sur **3 bits en binaire**.
- L'idée de base est de replacer chaque symbole dans la base octal par sa valeur en binaire sur 3 bits.

Octal	Binaire
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

❖ $(345)_8=(\underline{011}\ \underline{100}\ \underline{101})_2$

❖ $(65,76)_8=(\underline{110}\ \underline{101},\ \underline{111}\ \underline{110})_2$

❖ $(35,34)_8=(\underline{011}\ \underline{101}\ ,\ \underline{011}\ \underline{100})_2$

- Le remplacement se fait de **droit à gauche** pour la **partie entière** et **de gauche à droite** pour **la partie fractionnaire**.

3. Bases et conversions

❑ Conversion du Octal au Binaire

- L'idée de base est de faire des **regroupements de 3 bits** à partir du **poids faible**.
- Par la suite **remplacer** chaque regroupement par **la valeur octale** correspondante.

❖ $(11001010010110)_2 = (\underline{011}\ \underline{001}\ \underline{010}\ \underline{010}\ \underline{110})_2 = (31226)_8$

❖ $(110010100{,}10101)_2 = (\underline{110}\ \underline{010}\ \underline{100}\ ,\ \underline{101}\ \underline{010})_2 = (624{,}51)_8$

- Le regroupement se fait de **droit à gauche** pour la **partie entière** et de **gauche à droite** pour la **partie fractionnelle**.

3. Bases et conversions

❑ Conversion du Hexadécimal au Binaire

- En **Hexadécimal**, chaque symbole de la base s'écrit **sur 4 bits**.
- L'idée de base est de replacer chaque symbole par sa **valeur en binaire sur 4 bits**.

$(345B)_{16} = (\underline{0011}\ \underline{0100}\ \underline{0101}\ \underline{1011})_2$

$(AB3,4F6)_{16} = (\ \underline{1010}\ \underline{1011}\ \underline{0011}\ ,\ \underline{0100}\ \underline{1111}\ \underline{0110}\ )_2$

Binaire	Hexadécimal
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

3. Bases et conversions

❑ Conversion du Binaire à Hexadécimal

- L'idée de base est de faire des **regroupements de 4 bits** à partir du **poids faible**.
- Par la suite remplacer **chaque regroupement** par la valeur **Héxadécimale** correspondante .

❖ $(11001010100110)_2=(0011\ 0010\ 1010\ 0110)_2=(32A6)_{16}$

❖ $(110010100,10101)_2= (0001\ 1001\ 0100,1010\ 1000)_2=(194,A8)_{16}$

3. Bases et conversions

❑ Exercice

- Effectuer les opérations suivantes et transformer le résultat au décimal à chaque fois:

❖ $(1101,111)_2 + (11,1)_2 = (\,?\,)_2$

❖ $(43)_6 = (\,?\,)_5 = (\,?\,)_8$

❖ $(43)_{10} = (\,?\,)_2 = (\,?\,)_{16}$

❖ $(63)_8 + (35)_8 = (\,?\,)_8$

❖ $(AB3)_{16} + (23E)_{16} = (\,?\,)_{16}$

❖ $(AB3)_{16} - (23E)_{16} = (\,?\,)_{16}$

3. Bases et conversions

❑ Solution

- ❖ $(1101,111)_2 + (11,1)_2 = (10001,011)_2$
- ❖ $(43)_6 = (102)_5 = (33)_8$
- ❖ $(43)_{10} = (101011)_2 = (AB)_{16}$
- ❖ $(63)_8 + (35)_8 = (120)_8$
- ❖ $(AB3)_{16} + (23E)_{16} = (CF1)_{16}$
- ❖ $(AB3)_{16} - (23E)_{16} = (875)_{16}$

REPRÉSENTATION DES NOMBRES NÉGATIFS

1. Signe / Valeur absolue

- Il existe **deux** types d'entiers :
  - les entiers **non signés** (**positifs**)
  - les entiers **signés** (**positifs ou négatifs**)

Problème : Comment indiquer à la machine **qu'un nombre est négatif ou positif** ?

- Il existe **3 méthodes** pour représenter les nombres négatifs :
  - **Signe/ valeur absolue**
  - **Complément à 1 (Complément restreint)**
  - **Complément à 2 (Complément à vrai)**

1. Signe / Valeur absolue

- **Si on travaille sur n bits** ,

alors le bit du **poids fort** est utilisé pour indiquer le **signe** :

- **1 : signe négatif**
- **0 : signe positif**

- Les autres bits (**n -1**) désignent la **valeur absolue** du nombre.
- **Exemple : Coder -1 et +1 sur 4 bits**:

$1\ \boxed{001}$

Signe **Valeur absolue**

$0\ \boxed{001}$

Signe **Valeur absolue**

1001 est la représentation de - 1

0001 est la représentation de + 1

1. Signe / Valeur absolue

- **Sur 3 bits on obtient :**

signe	VA	valeur
0	00	+ 0
0	01	+ 1
0	10	+ 2
0	11	+ 3
1	00	- 0
1	01	- 1
1	10	- 2
1	11	- 3

- Les valeurs sont comprises entre -3 et +3

$$-3 \leq N \leq +3$$

Si on travaille sur n bits , l'intervalle des valeurs qu'on peut représenter en S/VA :

$$-(2^{(n-1)} - 1) \leq N \leq +(2^{(n-1)} - 1)$$

1. Signe / Valeur absolue

- Le **zéro** possède **deux représentations** **+0 = 000…0** et **-0 = 100…0** ce qui conduit parfois à **des difficultés au niveau des opérations arithmétiques**.

- Pour **les opérations arithmétiques**, il nous faut **deux circuits**: l'un pour **l'addition** et le deuxième pour **la soustraction**.

- L'idéal est d'utiliser **un seul circuit** pour faire les deux opérations, puisque **a- b = a + (-b).**

2. Complément à 1 (Restreint) (CA1)

- **<u>On travaille sur n bits,</u>** pour trouver le **complément à un** d'un nombre **négatif**, il suffit **d'inverser tous les bits de sa valeur absolue codée sur n bits** : si le bit est un 0 mettre à sa place un 1 et si c'est un 1 mettre à sa place un 0 .

Exemple : Coder -10 sur 4 bits et 5 bits

Sur 4 Bits	**Sur 5 Bits**
Impossible	0 1 0 1 0
	↓ ↓ ↓ ↓ ↓
	1 0 1 0 1

3. Complément à 2 (Vrai) : Sur n bits

- **Méthode 1 : CA2(N)= CA1(N)+ 1**

Trouver le **complément à vrai** de **: -69 sur 8 bits** ?

❖**CA2(-69)= CA1(-69)+ 1**

Or CA1(-69)= (10111010) donc

❖**CA2(-69)=(10111010)+ 1 = (10111011)**

- **Méthode 2 :** Pour trouver le compétemment à 2 d'un nombre négatif : il faut **parcourir les bits de la valeur absolue de ce nombre à partir du poids faible et garder tous les bits avant le premier 1 et inverser les autres bits qui viennent après**.

0	**1**	**0**	**0**	**0**	**1**	**0**	**1**
↓	↓	↓	↓	↓	↓	↓	↓
1	**0**	**1**	**1**	**1**	**0**	**1**	**1**

0	**1**	**0**	**1**	**0**	**1**	**0**	**0**
↓	↓	↓	↓	↓	↓	↓	↓
1	**0**	**1**	**0**	**1**	**1**	**0**	**0**

❑ Exercice

- Effectuer les opérations suivantes sur 8 bits:

❖ $(-23)_{10} = (?)_{va}$

❖ $(4\,3)_{10} = (?)_{ca1} = (?)_{ca2}$

❖ $(-13)_{10} = (?)_{ca1}$

❖ $(-63)_{10} = (?)_{ca1} = (?)_{ca2}$

REPRÉSENTATION DES NOMBRES RÉELS

1. Représentation en virgule fixe

- Un **nombre réel** est constitué de deux parties : la partie entière et la partie fractionnaire séparées par une virgule.

  - ❖ **Problème:** comment indiquer à la machine **la position de la virgule** ?

- Il existe **deux méthodes** pour représenter les nombre réel:

  - ❖ **Virgule fixe**: la position de la virgule est fixe.
  - ❖ **Virgule flottante**: la position de la virgule change.

1. Représentation en virgule fixe

- Dans cette représentation:
  - ❖ La **partie entière** est représentée sur **n bits.**
  - ❖ La **partie fractionnaire** sur **p bits.**
  - ❖ En plus **un bit** est utilisé pour le **signe**.

- Si **n=3** et **p=2** on va avoir les valeurs suivantes:

Signe	P.E	P.F	valeur
0	000	.00	+ 0,0
0	000	.01	+ 0,25
0	000	.10	+ 0,5
0	000	.11	+ 0,75
0	001	.00	+ 1,0
…	…	….	…..

- Dans cette représentation, les valeurs sont **limitées** et **nous n'avons pas une grande précision.**

2. Représentation en virgule flottante

❖ **Norme IEEE 754 de simple précision (32 bits)**

Signe	Exposant biaisé	Mantisse

1 bit **8 bits** **23 bits**

$$N = (-1)^s \times 1{,}M \times 2^E$$

La **mantisse M** est **normalisée** sous la forme **1,M** et **l'exposant** est ajusté en conséquence.

- La **partie M** est codée sur **23 bits**.
- On ajoute le biais ($2^{8-1} - 1 = 127$) à **E** et le total est **codé** sur **8 bits**.
- **s** est le signe.

2. Représentation en virgule flottante

- **Exemple :**

$(-15{,}625)_{10} = (\ ?\ )_{IEEE\text{-}754\text{-}32}$

$$(15{,}625)_{10} = (1111{,}101)_2 = 1{,}111101\ x2^3$$

Donc

$$M=111101 \text{ et } E= 3$$

Par suite

$$Eb = 127+3 = 130 = (10000010)_2$$

Alors

$$(-15{,}625)_{10} = (\ 1\ 10000010\ 11110100000000000000000\ )_{IEEE\text{-}754\text{-}32}$$

2. Représentation en virgule flottante

❖ **Norme IEEE 754 de double précision (64 bits)**

Signe	Exposant biaisé	Mantisse

1 bit **11 bits** **52 bits**

- La **mantisse M** est **normalisée** sous la forme **1,M** et **l'exposant** est ajusté en conséquence.
- La **partie M** est codée sur **52 bits**.
- On ajoute le biais (**$2^{11-1} - 1$** = **1023**) à **E** et le total est **codé** sur **11 bits**.
- **s** est le signe.

$$N = (-1)^s \times 1,M \times 2^E$$

AUTRES CODES

1. Codage BCD (Binary Coded Decimal)

- Pour passer du décimal au binaire , il faut effectuer des divisions successives. Il existe une autre méthode simplifiée pour le passage du décimal au binaire.

- Le principe consiste à faire des éclatement sur 4 bits et de remplacer chaque chiffre décimal par sa valeur binaire correspondante.

- **Les combinaisons supérieures à 9 sont interdites.**

Décimal	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

1. Codage BCD (Binary Coded Decimal)

❑ Exemple

1 2 9

0001 0010 1001

5 6 2

0101 0110 0010

129 = (0001 0010 1001)$_2$

562 = (0101 0110 0010)$_2$

2. Codage EXCESS3

EXCESS3 = BCD+3

Décimal	BCD+3	Binaire
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

1 2 9

0100 0101 1100

REPRÉSENTATION DES CARACTÈRES

3. Codage des caractères

- Les caractères englobent : les **lettres alphabétiques** (**A, a, B , B,..**) , **les chiffres** , et les **autres symboles** (**> , ; / : ….**).
- Le codage le plus utilisé est le **ASCII** (**American Standard Code for Information Interchange**).
- Dans ce codage chaque caractère est représenté sur **8 bits**.
- Avec **8 bits** on peut avoir $\mathbf{2^8 = 256}$ combinaisons.
- Chaque **combinaison représente un caractère.**

❖ **Exemple:**

- Le code $(\mathbf{65})_{10} = (\mathbf{01000001})_2$ correspond au caractère **A**
- Le code $(\mathbf{97})_{10} = (\mathbf{01100001})_2$ correspond au caractère **a**
- Le code $(\mathbf{58})_{10} = (\mathbf{00111010})_2$ correspond au caractère **:**

- Actuellement il existe un autre code sur 16 bits: **UNICODE**.

3. Table ASCII

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

3. Table ASCII ETENDU

Gauche \ Droite		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL	DLE	SP	0	@	P	`	p	Ç	É	á	░	└	ð	Ó	-
1	0001	SOH	DC1	!	1	A	Q	a	q	ü	æ	í	▒	┴	Ð	ß	±
2	0010	STX	DC2	"	2	B	R	b	r	é	Æ	ó	▓	┬	Ê	Ô	‗
3	0011	ETX	DC3	#	3	C	S	c	s	â	ô	ú	│	├	Ë	Ò	¾
4	0100	EOT	DC4	$	4	D	T	d	t	ä	ö	ñ	┤	─	È	õ	¶
5	0101	ENQ	NAK	%	5	E	U	e	u	à	ò	Ñ	Á	┼	ı	Õ	§
6	0110	ACK	SYN	&	6	F	V	f	v	å	û	ª	Â	ã	Í	µ	÷
7	0111	BEL	ETB	'	7	G	W	g	w	ç	ù	º	À	Ã	Î	þ	¸
8	1000	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿	©	╚	Ï	Þ	°
9	1001	HT	EM	)	9	I	Y	i	y	ë	Ö	®	╣	╔	┘	Ú	¨
A	1010	LF	SUB	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Û	·
B	1011	VT	ESC	+	;	K	[	k	}	ï	ø	½	╗	╦	█	Ù	¹
C	1100	FF	FS	,	<	L	\	l	|	î	£	¼	╝	╠	▄	ý	³
D	1101	CR	GS	-	=	M	]	m	{	ì	Ø	¡	¢	═	¦	Ý	²
E	1110	SO	RS	.	>	N	^	n	~	Ä	×	«	¥	╬	Ì	¯	■
F	1111	SI	US	/	?	O	_	o	DEL	Å	ƒ	»	┐	¤	▀	´	

FIN