Introduction au binaire

Julien Baste

IUT de Lille

Séance 01

2025/2026

L'information

Une information est un message

- Elle est composée d'une suite de caractères
- Sa taille dépend de son contenu et des caractères utilisés

Dans la vie courante, pour faire passer une information nous utilisons

- Des lettres
- Des chiffres
- Quelques caractères spéciaux (" !", " ?", etc.)

Une information est alors passée comme un message écrit avec ces caractères.

Différents messages

Il existe plusieurs types d'information que l'on veut pouvoir manipuler :

- Nombres entiers (0, 1, 2, etc.)
- Réel (1.4, 3.14, etc.)
- Booléen (Vrai ou Faux)
- Caractères (a, !, #, etc.)
- Chaîne de caractères (toto, tata, etc.)
- Images
- Son
- etc.

L'informatique consiste à traiter automatiquement l'information numérique

Les caractères possibles pour stocker de l'information numérique sont binaires :

- 0 ou 1
- Troué ou non troué
- Vrai ou faux
- 0 Volt ou +5 Volts
- etc.

Une information binaire (0 ou 1) est appelée un bit (***bi**nary digi**t*** = chiffre binaire)

Représentation binaire

Un information quelconque = 1 ensemble de bits

- regroupé par paquet *(mot)* de 8 (***byte*** = octet)
- chaque bit a une position (numérotée de droite à gauche)

Représentation binaire

Un information quelconque = 1 ensemble de bits

- regroupé par paquet *(mot)* de 8 (***byte*** = octet)
- chaque bit a une position (numérotée de droite à gauche)

0	**1**	**0**	**1**	**1**	**0**	**1**	**0**
7	*6*	*5*	*4*	*3*	*2*	*1*	*0*

- dans un octet, une position correspond à un **poids**
  - bit numéro 0 . . . . bit de poids **faible** *(Least Significant Bit = lsb)*
  - bit numéro 7 . . . . bit de poids **fort** *(Most Significant Bit = msb)*
- on généralise en considérant que
  - plus le bit est à droite plus il est faible
  - plus le bit est à gauche plus il est fort

Le binaire à la main

Le binaire est compliqué à manipuler à la main :

- Le message est très long
- Beaucoup de possibilités d'erreurs de manipulation.

Pour avoir une notation plus courte, on :

- Regroupe les bits par petits blocs
- Chaque bloc est représenté par un symbole (Un chiffre ou une lettre)
  - Blocs de taille 1 .... *notation positionnelle* ***binaire***
  - Blocs de taille 3 .... *notation positionnelle* ***octale***
  - Blocs de taille 4 .... *notation positionnelle* ***hexadécimale***

Pour chaque notation :

- on a une table de conversion entre une forme binaire et un chiffre
- cette table est construite par **conversion de nombre entre bases** (2, 8 ou 16)

Notation octale

1 chiffre octal :

- est un bloc de 3 bits
- peut représenter **8** valeurs différentes
- a pour symbole
  - un chiffre *arabe* entre 0 et 7

Notation octale

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

1 chiffre octal :

- est un bloc de 3 bits
- peut représenter **8** valeurs différentes
- a pour symbole
  - un chiffre *arabe* entre 0 et 7

Notation octale

1 chiffre octal :

- est un bloc de 3 bits
- peut représenter **8** valeurs différentes
- a pour symbole
  - un chiffre *arabe* entre 0 et 7

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

Le mot binaire **001 011 010** est représenté en octal par le mot **132**

On a ici la conversion entre la base 2 (binaire) et la base 8 (octale)

ATTENTION : octet $\neq$ octale

- Un octet est un bloc de 8 bits
- La notation positionnelle octale est un regroupement de bits par bloc de taille 3

Notation hexadécimale

1 chiffre hexadécimal

- est un bloc de 4 bits
- peut représenter **16** valeurs différentes
- a pour symbole
  - un chiffre *arabe* entre 0 et 9
  - ou une lettre *latine* entre A et F

Notation hexadécimale

1 chiffre hexadécimal

- est un bloc de 4 bits
- peut représenter **16** valeurs différentes
- a pour symbole
  - un chiffre *arabe* entre 0 et 9
  - ou une lettre *latine* entre A et F

binaire	hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Notation hexadécimale

1 chiffre hexadécimal

- est un bloc de 4 bits
- peut représenter **16** valeurs différentes
- a pour symbole
  - un chiffre *arabe* entre 0 et 9
  - ou une lettre *latine* entre A et F

binaire	hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Le mot binaire **0101 1010** est représenté en hexadécimal par le mot **5A**

On a ici la conversion entre la base 2 (binaire) et la base 16 (hexadécimale)

Pour éviter les confusions des conventions d'écriture, des mots existent

	Langage C	Mathématiques
Binaire	$0\mathrm{b}c_k \cdots c_1$ Commence par un «0b» ex : 0b01011010	$(c_k \cdots c_1)_2$ Entre parenthèse indicé par un 2 ex : $(01011010)_8$
Octal	$0c_k \cdots c_1$ Commence par un «0» ex : 0132	$(c_k \cdots c_1)_8$ Entre parenthèse indicé par un 8 ex : $(132)_8$
Hexadécimal	$0\mathrm{x}c_k \cdots c_1$ Commence par «0x» ex : $0x5A$	$(c_k \cdots c_1)_{16}$ Entre parenthèses indicées par un 16 ex : $(5A)_{16}$

Ordre de stockage/transmission

Souvent, une information s'exprime sur plusieurs octets.

Deux conventions d'ordre de stockage (ou de transmission) des mots existent

Ces conventions correspondent à l'endianness

Ordre de stockage/transmission

Souvent, une information s'exprime sur plusieurs octets.

- ex : 11111111 00000000 est une information binaire sur 2 octets `0xff00`
- plus un octet est à gauche plus il est **fort** — octet de poids fort : `0xff`
- plus il est à droite plus il est **faible** — octet de poids faible : `0x00`

Deux conventions d'ordre de stockage (ou de transmission) des mots existent

Ces conventions correspondent à l'endianness

Ordre de stockage/transmission

Souvent, une information s'exprime sur plusieurs octets.

- ex : 11111111 00000000 est une information binaire sur 2 octets `0xff00`
- plus un octet est à gauche plus il est **fort** — octet de poids fort : `0xff`
- plus il est à droite plus il est **faible** — octet de poids faible : `0x00`

Deux conventions d'ordre de stockage (ou de transmission) des mots existent

- little endian (**petit boutiste**) : `0x00 0xff`
  Les octets de poids faibles sont transmis/stockés en premier
- big endian (**grand boutiste**) : `0xff 0x00`
  Les octets de poids forts sont transmis/stockés en premier

Ces conventions correspondent à l'endianness

Ordre de stockage/transmission

Souvent, une information s'exprime sur plusieurs octets.

- ex : 11111111 00000000 est une information binaire sur 2 octets `0xff00`
- plus un octet est à gauche plus il est **fort** octet de poids fort : `0xff`
- plus il est à droite plus il est **faible** octet de poids faible : `0x00`

Deux conventions d'ordre de stockage (ou de transmission) des mots existent

- little endian (**petit boutiste**) : `0x00 0xff`
  Les octets de poids faibles sont transmis/stockés en premier
- big endian (**grand boutiste**) : `0xff 0x00`
  Les octets de poids forts sont transmis/stockés en premier

Ces conventions correspondent à l'endianness

- utilisée pour le stockage en mémoire (adresse de début ou de fin)
- utilisée pour la communication d'information binaire (ordre de transmission)

Conversion : base $b \to$ base 10

- Cas général (base *b*)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

01101001 $\rightarrow$ **0** $\times 2^7 + \quad \times 2^6 + \quad \times 2^5 + \quad \times 2^4 + \quad \times 2^3 + \quad \times 2^2 + \quad \times 2^1 + \quad \times 2^0 = \quad .$

facteur	**0**							
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

01101001 $\rightarrow$ $\mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \quad \times 2^5 + \quad \times 2^4 + \quad \times 2^3 + \quad \times 2^2 + \quad \times 2^1 + \quad \times 2^0 =$.

facteur	**0**	**1**						
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

01101001 $\rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \quad \times 2^4 + \quad \times 2^3 + \quad \times 2^2 + \quad \times 2^1 + \quad \times 2^0 = \quad .$

facteur	**0**	**1**	**1**					
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

$\texttt{01101001} \rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \quad \times 2^3 + \quad \times 2^2 + \quad \times 2^1 + \quad \times 2^0 = \quad .$

facteur	**0**	**1**	**1**	**0**				
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

`01101001` $\rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \quad \times 2^2 + \quad \times 2^1 + \quad \times 2^0 = \qquad .$

facteur	**0**	**1**	**1**	**0**	**1**			
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

`01101001` $\rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \quad \times 2^1 + \quad \times 2^0 = \quad .$

facteur	**0**	**1**	**1**	**0**	**1**	**0**		
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

01101001 $\rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{0} \times 2^1 + \quad \times 2^0 = \quad .$

facteur	**0**	**1**	**1**	**0**	**1**	**0**	**0**	
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

$01101001 \rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 =$.

facteur	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion : base $b \rightarrow$ base 10

- Cas général (base b)
  - Notation : $c_\ell \cdots c_2 c_1 c_0$
  - Valeur : $n = c_\ell \times b^\ell + \cdots + c_2 \times b^2 + c_1 \times b^1 + c_0 \times b^0$
  - la position d'un bit donne son poids
  - $\forall i, c_i < b$

Example (Avec $b = 2$)

`01101001` $\rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 = \mathbf{105}.$

facteur	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

NOTE : Connaître par cœur les puissances de 2, au moins jusque 10,

est donc indispensable pour un informaticien.

Conversion

base 10 → base b

- Cas général (base b)
  - construction des chiffres de droite vers les chiffres de gauche
  - reste des divisions euclidiennes successives par b
  - arrêt quand le quotient est nul

Example (Avec $b = 2$)

35 → **1**

$$\begin{array}{r|l} 35 & 2 \\ \cline{2-2} \mathbf{1} & 17 \end{array}$$

- Cas général (base b)
  - construction des chiffres de droite vers les chiffres de gauche
  - reste des divisions euclidiennes successives par b
  - arrêt quand le quotient est nul

Example (Avec $b = 2$)

35 → **`11`**

$$\begin{array}{r|l} 17 & 2 \\ \hline \mathbf{1} & 8 \end{array} \leftarrow \begin{array}{r|l} 35 & 2 \\ \hline \mathbf{1} & 17 \end{array}$$

Conversion

base 10 → base *b*

- Cas général (base *b*)
  - construction des chiffres de droite vers les chiffres de gauche
  - reste des divisions euclidiennes successives par *b*
  - arrêt quand le quotient est nul

Example (Avec $b = 2$)

$$\mathbf{35} \rightarrow \quad \texttt{011}$$

$$\begin{array}{r|l} 8 & 2 \\ \hline \mathbf{0} & 4 \end{array} \;\leftarrow\; \begin{array}{r|l} 17 & 2 \\ \hline \mathbf{1} & 8 \end{array} \;\leftarrow\; \begin{array}{r|l} 35 & 2 \\ \hline \mathbf{1} & 17 \end{array}$$

- Cas général (base b)
  - construction des chiffres de droite vers les chiffres de gauche
  - reste des divisions euclidiennes successives par b
  - arrêt quand le quotient est nul

Example (Avec $b = 2$)

$$\mathbf{35} \rightarrow \quad \mathtt{0011}$$

$$\begin{array}{r|l} 4 & 2 \\ \cline{2-2} \mathbf{0} & 2 \end{array} \leftarrow \begin{array}{r|l} 8 & 2 \\ \cline{2-2} \mathbf{0} & 4 \end{array} \leftarrow \begin{array}{r|l} 17 & 2 \\ \cline{2-2} \mathbf{1} & 8 \end{array} \leftarrow \begin{array}{r|l} 35 & 2 \\ \cline{2-2} \mathbf{1} & 17 \end{array}$$

Conversion

base 10 → base *b*

- Cas général (base *b*)
  - construction des chiffres de droite vers les chiffres de gauche
  - reste des divisions euclidiennes successives par *b*
  - arrêt quand le quotient est nul

Example (Avec $b = 2$)

35 → **`00011`**

$$\begin{array}{r|l} 2 & 2 \\ \cline{2-2} \mathbf{0} & 1 \end{array} \leftarrow \begin{array}{r|l} 4 & 2 \\ \cline{2-2} \mathbf{0} & 2 \end{array} \leftarrow \begin{array}{r|l} 8 & 2 \\ \cline{2-2} \mathbf{0} & 4 \end{array} \leftarrow \begin{array}{r|l} 17 & 2 \\ \cline{2-2} \mathbf{1} & 8 \end{array} \leftarrow \begin{array}{r|l} 35 & 2 \\ \cline{2-2} \mathbf{1} & 17 \end{array}$$

- Cas général (base b)
  - construction des chiffres de droite vers les chiffres de gauche
  - reste des divisions euclidiennes successives par b
  - arrêt quand le quotient est nul

Example (Avec $b = 2$)

$$\mathbf{35} \rightarrow \mathtt{\mathbf{100011}}$$

$$\begin{array}{r|l} 1 & 2 \\ \cline{2-2} \mathbf{1} & \color{red}{\mathbf{0}} \end{array} \leftarrow \begin{array}{r|l} 2 & 2 \\ \cline{2-2} \mathbf{0} & 1 \end{array} \leftarrow \begin{array}{r|l} 4 & 2 \\ \cline{2-2} \mathbf{0} & 2 \end{array} \leftarrow \begin{array}{r|l} 8 & 2 \\ \cline{2-2} \mathbf{0} & 4 \end{array} \leftarrow \begin{array}{r|l} 17 & 2 \\ \cline{2-2} \mathbf{1} & 8 \end{array} \leftarrow \begin{array}{r|l} 35 & 2 \\ \cline{2-2} \mathbf{1} & 17 \end{array}$$

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0		
1		

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0		
1		1

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0	0	0
1	0	1

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0	0	0
1	0	1

Disjonction

OR	**0**	**1**
0		
1		

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0	0	0
1	0	1

Disjonction

OR	**0**	**1**
0		1
1	1	1

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0	0	0
1	0	1

Disjonction

OR	**0**	**1**
0	0	1
1	1	1

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0	0	0
1	0	1

Disjonction

OR	**0**	**1**
0	0	1
1	1	1

Disjonction exclusive

XOR	**0**	**1**
0		
1		

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0	0	0
1	0	1

Disjonction

OR	**0**	**1**
0	0	1
1	1	1

Disjonction exclusive

XOR	**0**	**1**
0		1
1	1	

Opérations logiques (sur des bits)

Les micro-processeurs manipulent des bits via des opérations *logiques*

Opération *monadique*

Négation logique

NOT	
0	1
1	0

Opérations *diadiques*

Conjonction

AND	**0**	**1**
0	0	0
1	0	1

Disjonction

OR	**0**	**1**
0	0	1
1	1	1

Disjonction exclusive

XOR	**0**	**1**
0	0	1
1	1	0

Opérations logiques (sur des bits)

x	**NOT x**
0	1
1	0

x	**y**	**x** AND **y**
0	0	0
0	1	0
1	0	0
1	1	1

x	**y**	**x** OR **y**
0	0	0
0	1	1
1	0	1
1	1	1

x	**y**	**x** XOR **y**
0	0	0
0	1	1
1	0	1
1	1	0

Opérations logiques (sur des mots)

Les opérations logiques s'appliquent sur des mots binaires complets

- application ***bit à bit*** de l'opération
- application sur tous les bits d'un mot
  - sur la représentation binaire
  - mot de taille fixe (1 octet = 8 bits)

Opérations logiques (sur des mots)

Les opérations logiques s'appliquent sur des mots binaires complets

- application ***bit à bit*** de l'opération
- application sur tous les bits d'un mot
  - sur la représentation binaire
  - mot de taille fixe (1 octet = 8 bits)

2 utilisations des opérations logiques :

- calcul des expressions booléennes . . . . Dev, Maths
- manipulation directe de valeurs binaires . . . . Système

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=								

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=								

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=								0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=							0	0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=						0	0	0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=					0	0	0	0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=				0	0	0	0	0

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
$=$			0	0	0	0	0	0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=		0	0	0	0	0	0	0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=	0	0	0	0	0	0	0	0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000
  - `0xaa` **AND** `0x55` = `0x00`

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=	0	0	0	0	0	0	0	0

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=	0	0	0	0	0	0	0	0

  - `0xaa` **AND** `0x55` = `0x00`

- 10101010 **OR** 01010101 = 11111111

	1	0	1	0	1	0	1	0
OR	0	1	0	1	0	1	0	1
=	1	1	1	1	1	1	1	1

Exemple : Opérations logiques (sur des mots)

- 10101010 **AND** 01010101 = 00000000
  - `0xaa` **AND** `0x55` = `0x00`

	1	0	1	0	1	0	1	0
AND	0	1	0	1	0	1	0	1
=	0	0	0	0	0	0	0	0

- 10101010 **OR** 01010101 = 11111111
  - `0xaa` **OR** `0x55` = `0xff`

	1	0	1	0	1	0	1	0
OR	0	1	0	1	0	1	0	1
=	1	1	1	1	1	1	1	1

Réduction/masque

Transformation opérations diadiques en opération monadique :

- une opérande est fixée à une *constante* (masque)
- l'autre opérande est *variable*

Réduction/masque

Transformation opérations diadiques en opération monadique :

- une opérande est fixée à une *constante* (masque)
- l'autre opérande est *variable*

Example

AND devient un outil de *masquage*

AND	0	1
0	0	0
1	0	1

devient

AND	x
0	0
1	x

$\forall x$:

- si l'opérande constante vaut 0 . . . . le résultat = 0
- si l'opérande constante vaut 1 . . . . le résultat = x

Processus appliqué sur chaque bit du mot

Autres opérations sur des mots (largeur définie)

Décalages

- déplacer les bits en introduisant des 0
- **DECG** : décalage à gauche $<< n$
  - on supprime les *n* bits de poids forts
  - on ajoute *n* 0 en poids faibles
- **DECD** : décalage à droite $>> n$
  - on supprime les *n* bits de poids faibles
  - on ajoute *n* 0 en poids forts

Example

- 10101010 << 1 = 01010100

Autres opérations sur des mots (largeur définie)

Décalages

- déplacer les bits en introduisant des 0
- **DECG** : décalage à gauche $<< n$
  - on supprime les n bits de poids forts
  - on ajoute n 0 en poids faibles
- **DECD** : décalage à droite $>> n$
  - on supprime les n bits de poids faibles
  - on ajoute n 0 en poids forts

Example

- 10101010 << 1 = 01010100

Autres opérations sur des mots (largeur définie)

Décalages

- déplacer les bits en introduisant des 0
- **DECG** : décalage à gauche $<< n$
  - on supprime les n bits de poids forts
  - on ajoute n 0 en poids faibles
- **DECD** : décalage à droite $>> n$
  - on supprime les n bits de poids faibles
  - on ajoute n 0 en poids forts

Example

- 10101010 << 1 = 01010100

Autres opérations sur des mots (largeur définie)

Décalages

- déplacer les bits en introduisant des 0
- **DECG** : décalage à gauche $<< n$
  - on supprime les n bits de poids forts
  - on ajoute n 0 en poids faibles
- **DECD** : décalage à droite $>> n$
  - on supprime les n bits de poids faibles
  - on ajoute n 0 en poids forts

Example

- 10101010 << 1 = 01010100
- 10101010 >> 3 = 00010101

Autres opérations sur des mots (largeur définie)

Décalages

- déplacer les bits en introduisant des 0
- **DECG** : décalage à gauche $<< n$
  - on supprime les n bits de poids forts
  - on ajoute n 0 en poids faibles
- **DECD** : décalage à droite $>> n$
  - on supprime les n bits de poids faibles
  - on ajoute n 0 en poids forts

Example

- 10101010 << 1 = 01010100
- 10101010 >> 3 = 00010101

Autres opérations sur des mots (largeur définie)

Décalages

- déplacer les bits en introduisant des 0
- **DECG** : décalage à gauche $<< n$
  - on supprime les n bits de poids forts
  - on ajoute n 0 en poids faibles
- **DECD** : décalage à droite $>> n$
  - on supprime les n bits de poids faibles
  - on ajoute n 0 en poids forts

Example

- 10101010 << 1 = 01010100
- 10101010 >> 3 = 00010101

C'est tout pour aujourd'hui