Introduction aux systèmes informatiques

Codage et Systèmes d'exploitation

Pôle ASR – Module M1101 – Semestre 1

Bruno BEAUFILS

⟨bruno.beaufils@univ-lille.fr⟩

https://beaufils.u-lille.fr

Université de Lille, IUT « A », Département informatique

Année 2020/2021

http://m1101.iutinfo.fr

Codage

Contrainte physique

Informatique = **traitement automatique de l'information**

1. **information**

   Diversité de nature de l'information

   - simple ..... *nombre entier, réel, booléen*
   - complexe ..... *caractères, suite (chaînes) de caractères, images, sons, etc.*

2. **traitement automatique**

   Utilisation d'ordinateur = machine **électrique**

   - constituée de composants électroniques

     objets physiques qui dirigent le courant électrique (porte)

   - électricité ⇒ 2 états observables
     - 0 pas de courant
     - 1 circulation de courant

⇒ **représentation binaire**

Contrainte physique

Informatique = **traitement automatique de l'information**

1. **information**
   Diversité de nature de l'information
   - simple ..... *nombre entier, réel, booléen*
   - complexe ..... *caractères, suite (chaînes) de caractères, images, sons, etc.*

2. **traitement automatique**
   Utilisation d'ordinateur = machine **électrique**
   - constituée de composants électroniques
     objets physiques qui dirigent le courant électrique (porte)
   - électricité ⇒ 2 états observables
     0 pas de courant
     1 circulation de courant

⇒ **représentation binaire**

Contrainte physique

Informatique = **traitement automatique de l'information**

1. **information**
   Diversité de nature de l'information
   - simple ..... *nombre entier, réel, booléen*
   - complexe ..... *caractères, suite (chaînes) de caractères, images, sons, etc.*

2. **traitement automatique**
   Utilisation d'ordinateur = machine **électrique**
   - constituée de composants électroniques
     objets physiques qui dirigent le courant électrique (porte)
   - électricité ⇒ 2 états observables
     0 pas de courant
     1 circulation de courant

⇒ **représentation binaire**

Contrainte physique

Informatique = **traitement automatique de l'information**

1. **information**
   Diversité de nature de l'information
   - simple ..... *nombre entier, réel, booléen*
   - complexe ..... *caractères, suite (chaînes) de caractères, images, sons, etc.*

2. **traitement automatique**
   Utilisation d'ordinateur = machine **électrique**
   - constituée de composants électroniques
     objets physiques qui dirigent le courant électrique (porte)
   - électricité ⇒ 2 états observables
     0 pas de courant
     1 circulation de courant

⇒ représentation binaire

Représentation binaire

Une information binaire = 1 **bit** (***b**inary dig**it*** = chiffre binaire)

- 0 Volt ou +5 Volts
- 0 ou 1
- Faux ou Vrai
- etc.

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)

- 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

Représentation binaire

Une information binaire = 1 **bit** (***bi**nary dig**it*** = chiffre binaire)

- 0 Volt ou +5 Volts
- 0 ou 1
- Faux ou Vrai
- etc.

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

Représentation binaire

Une information binaire = 1 **bit** (***b**inary dig**it*** = chiffre binaire)

- 0 Volt ou +5 Volts
- 0 ou 1
- Faux ou Vrai
- etc.

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

Notation

La représentation binaire est verbeuse

- difficile à lire
- difficile à manipuler
- source d'erreurs de manipulation

Pour noter de manière plus **concise**, on :

1. regroupe les bits par paquets
2. représente chaque paquet par un symbole (*un chiffre*)
   - paquet de 1 ..... *notation positionnelle* ***binaire***
   - paquet de 3 ..... *notation positionnelle* ***octale***
   - paquet de 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

La représentation binaire est verbeuse

- difficile à lire
- difficile à manipuler
- source d'erreurs de manipulation

Pour noter de manière plus **concise**, on :

1. regroupe les bits par paquets
2. représente chaque paquet par un symbole (*un chiffre*)
   - paquet de 1 ..... *notation positionnelle* ***binaire***
   - paquet de 3 ..... *notation positionnelle* ***octale***
   - paquet de 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

La représentation binaire est verbeuse

- difficile à lire
- difficile à manipuler
- source d'erreurs de manipulation

Pour noter de manière plus **concise**, on :

1. regroupe les bits par paquets
2. représente chaque paquet par un symbole (*un chiffre*)
   - paquet de 1 ..... *notation positionnelle* ***binaire***
   - paquet de 3 ..... *notation positionnelle* ***octale***
   - paquet de 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

La représentation binaire est verbeuse

- difficile à lire
- difficile à manipuler
- source d'erreurs de manipulation

Pour noter de manière plus **concise**, on :

1. regroupe les bits par paquets
2. représente chaque paquet par un symbole (*un chiffre*)
   - paquet de 1 ..... *notation positionnelle* ***binaire***
   - paquet de 3 ..... *notation positionnelle* ***octale***
   - paquet de 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 paquet 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 **01 011 010** est représenté en octal par le mot **132**

Notation octale

1 chiffre octal

- est un paquet 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 **01 011 010** est représenté en octal par le mot **132**

Notation octale

1 chiffre octal

- est un paquet 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 **01 011 010** est représenté en octal par le mot **132**

Notation hexadécimale

1 chiffre hexadécimal

- est un paquet 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**

Notation hexadécimale

1 chiffre hexadécimal

- est un paquet 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**

Notation hexadécimale

1 chiffre hexadécimal

- est un paquet 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**

Écriture

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

	octal	**hexadécimal**
Assembleur standard	$c_k \cdots c_1$O	\$$c_k \cdots c_1$
	finir par un « O »	*commencer par un « $ »*
	1320	$5A
Assembleur Intel	$c_k \cdots c_1$Q	$c_k \cdots c_1$H
	finir par un « Q »	*finir par un « H »*
	132Q	5AH
Langage C	0$c_k \cdots c_1$	0x$c_k \cdots c_1$
	commencer par un « 0 »	*commencer par « 0x »*
	0132	0x5A
Mathématiques	$(c_k \cdots c_1)_8$	$(c_k \cdots c_1)_{16}$
	entourer de parenthèses et spécifier la base d'expression des chiffres en indice	
	$(132)_8$	$(5A)_{16}$
Pour les octets	$k = 3$	$k = 2$

Écriture

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

- Assembleur standard
- Assembleur Intel
- Langage C
- Mathématiques

	octal	**hexadécimal**
	$c_k \cdots c_1$O	$\$c_k \cdots c_1$
	finir par un « O »	*commencer par un « $ »*
	`1320`	`$5A`
	$c_k \cdots c_1$Q	$c_k \cdots c_1$H
	finir par un « Q »	*finir par un « H »*
	`132Q`	`5AH`
	0$c_k \cdots c_1$	0x$c_k \cdots c_1$
	commencer par un « 0 »	*commencer par « 0x »*
	`0132`	`0x5A`
	$(c_k \cdots c_1)_8$	$(c_k \cdots c_1)_{16}$
	entourer de parenthèses et spécifier la base d'expression des chiffres en indice	
	$(132)_8$	$(5A)_{16}$
Pour les octets	$k = 3$	$k = 2$

Écriture

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

	octal	**hexadécimal**
• Assembleur standard	$c_k \cdots c_1$O *finir par un « O »* 1320	$\$c_k \cdots c_1$ *commencer par un « $ »* $5A
• Assembleur Intel	$c_k \cdots c_1$Q *finir par un « Q »* 132Q	$c_k \cdots c_1$H *finir par un « H »* 5AH
• Langage C	0$c_k \cdots c_1$ *commencer par un « 0 »* 0132	0x$c_k \cdots c_1$ *commencer par « 0x »* 0x5A
• Mathématiques	$(c_k \cdots c_1)_8$ *entourer de parenthèses et spécifier la base d'expression des chiffres en indice* $(132)_8$	$(c_k \cdots c_1)_{16}$ $(5A)_{16}$
Pour les octets	$k = 3$	$k = 2$

Écriture

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

	octal	**hexadécimal**
• Assembleur standard	$c_k \cdots c_1$O *finir par un « O »* 1320	$\$c_k \cdots c_1$ *commencer par un « $ »* $5A
• Assembleur Intel	$c_k \cdots c_1$Q *finir par un « Q »* 132Q	$c_k \cdots c_1$H *finir par un « H »* 5AH
• Langage C	0$c_k \cdots c_1$ *commencer par un « 0 »* 0132	0x$c_k \cdots c_1$ *commencer par « 0x »* 0x5A
• Mathématiques	$(c_k \cdots c_1)_8$ *entourer de parenthèses et spécifier la base d'expression des chiffres en indice* $(132)_8$	$(c_k \cdots c_1)_{16}$ $(5A)_{16}$
Pour les octets	$k = 3$	$k = 2$

Écriture

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

	octal	**hexadécimal**
• Assembleur standard	$c_k \cdots c_1$O	$\$c_k \cdots c_1$
	finir par un « O »	*commencer par un « $ »*
	1320	$5A
• Assembleur Intel	$c_k \cdots c_1$Q	$c_k \cdots c_1$H
	finir par un « Q »	*finir par un « H »*
	132Q	5AH
• Langage C	0$c_k \cdots c_1$	0x$c_k \cdots c_1$
	commencer par un « 0 »	*commencer par « 0x »*
	0132	0x5A
• Mathématiques	$(c_k \cdots c_1)_8$	$(c_k \cdots c_1)_{16}$
	entourer de parenthèses et spécifier la base d'expression des chiffres en indice	
	$(132)_8$	$(5A)_{16}$
Pour les octets	$k = 3$	$k = 2$

Ordre de transmission/stockage

Une information peut avoir besoin de ***plusieurs octets*** pour être exprimée

- si 11111111 00000000 est une information binaire
  - s'exprime $(11111111\ 00000000)_2$ ou `0xff00`
- 2 octets sont nécessaires à sa représentation
- plus un octet est à gauche plus il est **fort**, plus il est à droite plus il est **faible**
  - `0xff` est l'octet de poids fort, `0x00` est l'octet de poids faible

Historiquement on transmet (ou stocke) ***uniquement*** des ***octets*** (un par un)
Deux conventions d'***ordre de transmission*** (ou de stockage) des mots existent

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

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)

Ordre de transmission/stockage

Une information peut avoir besoin de ***plusieurs octets*** pour être exprimée

- si 11111111 00000000 est une information binaire
  - s'exprime $(11111111\ 00000000)_2$ ou `0xff00`
- 2 octets sont nécessaires à sa représentation
- plus un octet est à gauche plus il est **fort**, plus il est à droite plus il est **faible**
  - `0xff` est l'octet de poids fort, `0x00` est l'octet de poids faible

Historiquement on transmet (ou stocke) ***uniquement*** des ***octets*** (un par un)
Deux conventions d'***ordre de transmission*** (ou de stockage) des mots existent

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

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)

Ordre de transmission/stockage

Une information peut avoir besoin de ***plusieurs octets*** pour être exprimée

- si 11111111 00000000 est une information binaire
  - s'exprime $(11111111\ 00000000)_2$ ou `0xff00`
- 2 octets sont nécessaires à sa représentation
- plus un octet est à gauche plus il est **fort**, plus il est à droite plus il est **faible**
  - `0xff` est l'octet de poids fort, `0x00` est l'octet de poids faible

Historiquement on transmet (ou stocke) ***uniquement*** des ***octets*** (un par un)
Deux conventions d'***ordre de transmission*** (ou de stockage) des mots existent

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

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)

Ordre de transmission/stockage

Une information peut avoir besoin de ***plusieurs octets*** pour être exprimée

- si 11111111 00000000 est une information binaire
  - s'exprime $(11111111\ 00000000)_2$ ou `0xff00`
- 2 octets sont nécessaires à sa représentation
- plus un octet est à gauche plus il est **fort**, plus il est à droite plus il est **faible**
  - `0xff` est l'octet de poids fort, `0x00` est l'octet de poids faible

Historiquement on transmet (ou stocke) ***uniquement*** des ***octets*** (un par un)
Deux conventions d'***ordre de transmission*** (ou de stockage) des mots existent

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

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 → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

$011\,01001 \rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 105.$

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

$0110\,1001 \rightarrow \mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 105.$

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

Connaître par cœur les puissances de 2, au moins jusque 10, est donc indispensable pour un informaticien.

Conversion

base b → 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
  - $\forall i, c_i < b$
- Dans notre cas $b = 2$

Exemple

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

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

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
- Dans notre cas $b = 2$

Exemple

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

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

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
- Dans notre cas $b = 2$

Exemple

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

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

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
- Dans notre cas $b = 2$

Exemple

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

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

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
- Dans notre cas $b = 2$

Exemple

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

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

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
- Dans notre cas $b = 2$

Exemple

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

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

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
- Dans notre cas $b = 2$

Exemple

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

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

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 logique

AND	0	1
0	0	0
1	0	1

- Disjonction logique

OR	0	1
0	0	1
1	1	1

- Disjonction exclusive logique

XOR	0	1
0	0	1
1	1	0

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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

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 logique

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

- Disjonction logique

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

- Disjonction exclusive logique

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)
  - indépendance de chaque bit

Exemple

- `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

  - `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

- `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

- `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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`

	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

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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`

	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

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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`

	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

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

- `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

- `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

  - `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

  - `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

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)
  - indépendance de chaque bit

Exemple

- 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

  - `0xaa OR 0x55 = 0xff`

2 utilisations des opérations logiques :

- calcul des expressions booléennes ..... AP, Maths
- manipulation directe de valeurs binaires ..... ASR

Réduction/masque

Transformation opérations diadiques en opération monadique :

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

Exemple

`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

Réduction/masque

Transformation opérations diadiques en opération monadique :

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

Exemple

`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

Exemple

- $10101010 \ll 1 = 01010100$
- $10101010 \gg 3 = 00010101$

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

Exemple

- $10101010 \ll 1 = 01010100$
- $10101010 \gg 3 = 00010101$

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

Exemple

- $10101010 \ll 1 = 01010100$
- $10101010 \gg 3 = 00010101$

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

Exemple

- $10101010 \ll 1 = 01010100$
- $10101010 \gg 3 = 00010101$

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

Exemple

- $10101010 \ll 1 = 01010100$
- $10101010 \gg 3 = 00010101$

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

Exemple

- $10101010 \ll 1 = 01010100$
- $10101010 \gg 3 = 00010101$