

Le problème de la représentation des données

- La mémoire d'un ordinateur est composé de *bits* pouvant prendre uniquement les valeurs 0 et de 1

Le problème de la représentation des données

- La mémoire d'un ordinateur est composé de *bits* pouvant prendre uniquement les valeurs 0 et de 1
- Le regroupement de 8 bits s'appelle un octet (*byte* en anglais) c'est l'unité minimal de mémoire :

$$1 \text{ octet} = \underbrace{\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}}_{8 \text{ bits}}$$

C4 Représentation des données

1. Introduction

Le problème de la représentation des données

- La mémoire d'un ordinateur est composé de *bits* pouvant prendre uniquement les valeurs 0 et de 1
- Le regroupement de 8 bits s'appelle un octet (*byte* en anglais) c'est l'unité minimal de mémoire :

$$1 \text{ octet} = \underbrace{\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}}_{8 \text{ bits}}$$

- Toutes les données doivent donc être **représenté** en utilisant des octets.

C4 Représentation des données

1. Introduction

Le problème de la représentation des données

- La mémoire d'un ordinateur est composé de *bits* pouvant prendre uniquement les valeurs 0 et de 1
- Le regroupement de 8 bits s'appelle un octet (*byte* en anglais) c'est l'unité minimal de mémoire :

$$1 \text{ octet} = \underbrace{\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}}_{8 \text{ bits}}$$

- Toutes les données doivent donc être **représenté** en utilisant des octets.
- On s'intéresse ici à la représentation des entiers positifs et négatifs, des caractères et des flottants.

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire en utilisant 10 chiffres (0,1,2,3,4,5,6,7,8 et 9), chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire en utilisant 10 chiffres (0,1,2,3,4,5,6,7,8 et 9), chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour $\overline{1815}^{10}$:

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire en utilisant 10 chiffres (0,1,2,3,4,5,6,7,8 et 9), chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour $\overline{1815}^{10}$:

1 8 1 5

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire les entiers positifs en utilisant 10 chiffres, chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour 1815^{10} :

10^3	10^2	10^1	10^0
1	8	1	5

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire les entiers positifs en utilisant 10 chiffres, chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour 1815^{10} :

10^3	10^2	10^1	10^0
1	8	1	5

$$= 1 \times 1000 + 8 \times 100 + 1 \times 10 + 5 \times 1 = 1815$$

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire les entiers positifs en utilisant 10 chiffres, chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour 1815^{10} :

10^3	10^2	10^1	10^0
1	8	1	5

$$= 1 \times 1000 + 8 \times 100 + 1 \times 10 + 5 \times 1 = 1815$$

- De la même façon, on pourrait utiliser simplement 2 chiffres et multiplier chaque chiffre par une puissance de 2 suivant son emplacement dans le nombre.

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire les entiers positifs en utilisant 10 chiffres, chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour $\overline{1815}^{10}$:

10^3	10^2	10^1	10^0
1	8	1	5

$$= 1 \times 1000 + 8 \times 100 + 1 \times 10 + 5 \times 1 = 1815$$

- De la même façon, on pourrait utiliser simplement 2 chiffres et multiplier chaque chiffre par une puissance de 2 suivant son emplacement dans le nombre.

Par exemple, pour $\overline{11100010111}^2$:

1 1 1 0 0 0 1 0 1 1 1

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire les entiers positifs en utilisant 10 chiffres, chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour 1815 :

10^3	10^2	10^1	10^0
1	8	1	5

 $= 1 \times 1000 + 8 \times 100 + 1 \times 10 + 5 \times 1 = 1815$

- De la même façon, on pourrait utiliser simplement 2 chiffres et multiplier chaque chiffre par une puissance de 2 suivant son emplacement dans le nombre.

Par exemple, pour 11100010111² :

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

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire les entiers positifs en utilisant 10 chiffres, chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour 1815 :

10^3	10^2	10^1	10^0
1	8	1	5

$$= 1 \times 1000 + 8 \times 100 + 1 \times 10 + 5 \times 1 = 1815$$

- De la même façon, on pourrait utiliser simplement 2 chiffres et multiplier chaque chiffre par une puissance de 2 suivant son emplacement dans le nombre.

Par exemple, pour 11100010111² :

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

$$= 2^{10} + 2^9 + 2^8 + 2^4 + 2^2 + 2^1 + 2^0$$

C4 Représentation des données

2. Entiers positifs

De la base 10 à la base 2

- Nous sommes habitués à écrire les entiers positifs en utilisant 10 chiffres, chaque chiffre étant multiplié par une puissance de 10 suivant son emplacement dans le nombre.

Par exemple, pour 1815 :

10^3	10^2	10^1	10^0
1	8	1	5

$$= 1 \times 1000 + 8 \times 100 + 1 \times 10 + 5 \times 1 = 1815$$

- De la même façon, on pourrait utiliser simplement 2 chiffres et multiplier chaque chiffre par une puissance de 2 suivant son emplacement dans le nombre.

Par exemple, pour 11100010111² :

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

$$= 2^{10} + 2^9 + 2^8 + 2^4 + 2^2 + 2^1 + 2^0$$
$$= 1815$$

C4 Représentation des données

2. Entiers positifs

Ce sont des cas particuliers (avec $b = 10$ et $b = 2$), du théorème suivant :

Décomposition en base b

Tout entier $n \in \mathbb{N}$ peut s'écrire sous la forme :

$$n = \sum_{k=0}^p a_k b^k$$

avec $p \geq 0$ et $a_k \in \llbracket 0; b-1 \rrbracket$. De plus, cette écriture est unique si $a_p \neq 0$ et s'appelle *décomposition en base b de n* et on la note $n = \overline{a_p \dots a_1 a_0}_b$

C4 Représentation des données

2. Entiers positifs

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2$

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2$
- $\overline{1101001011}^2$

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2$
- $\overline{1101001011}^2$
- $\overline{421}^5$

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2$
- $\overline{1101001011}^2$
- $\overline{421}^5$
- $\overline{3EA}^{16}$

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2$
- $\overline{1101001011}^2$
- $\overline{421}^5$
- $\overline{3EA}^{16}$

On travaille ici en base 16, donc avec 16 chiffres, les lettres majuscules de A à F représentent les "chiffres" 10 à 15.

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2 = \overline{139}^{10}$
- $\overline{1101001011}^2$
- $\overline{421}^5$
- $\overline{3EA}^{16}$

On travaille ici en base 16, donc avec 16 chiffres, les lettres majuscules de A à F représentent les "chiffres" 10 à 15.

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2 = \overline{139}^{10}$
- $\overline{1101001011}^2 = \overline{843}^{10}$
- $\overline{421}^5$
- $\overline{3EA}^{16}$

On travaille ici en base 16, donc avec 16 chiffres, les lettres majuscules de A à F représentent les "chiffres" 10 à 15.

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2 = \overline{139}^{10}$
- $\overline{1101001011}^2 = \overline{843}^{10}$
- $\overline{421}^5 = \overline{111}^{10}$
- $\overline{3EA}^{16}$

On travaille ici en base 16, donc avec 16 chiffres, les lettres majuscules de A à F représentent les "chiffres" 10 à 15.

C4 Représentation des données

2. Entiers positifs

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2 = \overline{139}^{10}$
- $\overline{1101001011}^2 = \overline{843}^{10}$
- $\overline{421}^5 = \overline{111}^{10}$
- $\overline{3EA}^{16} = \overline{1002}^{10}$

On travaille ici en base 16, donc avec 16 chiffres, les lettres majuscules de A à F représentent les "chiffres" 10 à 15.

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^2 = 2^{n-1} + \dots + 1 = 2^n - 1$$

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^n = 2^{n-1} + \dots + 1 = 2^n - 1$$

- En C, les valeurs maximales représentables suivant le type d'entier positif utilisé sont donc :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^n = 2^{n-1} + \dots + 1 = 2^n - 1$$

- En C, les valeurs maximales représentables suivant le type d'entier positif utilisé sont donc :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $2^8 - 1 = 255$

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^n = 2^{n-1} + \dots + 1 = 2^n - 1$$

- En C, les valeurs maximales représentables suivant le type d'entier positif utilisé sont donc :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $2^8 - 1 = 255$
 - `uint32_t` : $2^{32} - 1 = 4\,294\,967\,295$ (≥ 4 milliards)

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^n = 2^{n-1} + \dots + 1 = 2^n - 1$$

- En C, les valeurs maximales représentables suivant le type d'entier positif utilisé sont donc :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $2^8 - 1 = 255$
 - `uint32_t` : $2^{32} - 1 = 4\,294\,967\,295$ (≥ 4 milliards)
 - `uint64_t` : $2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$ (≥ 18 milliards de milliards)

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^n = 2^{n-1} + \dots + 1 = 2^n - 1$$

- En C, les valeurs maximales représentables suivant le type d'entier positif utilisé sont donc :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $2^8 - 1 = 255$
 - `uint32_t` : $2^{32} - 1 = 4\,294\,967\,295$ (≥ 4 milliards)
 - `uint64_t` : $2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$ (≥ 18 milliards de milliards)

En cas de dépassement de capacité (*overflow* ou *underflow*), le résultat obtenu est calculé modulo la plus grande valeur maximale plus 1.

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^n = 2^{n-1} + \dots + 1 = 2^n - 1$$

- En C, les valeurs maximales représentables suivant le type d'entier positif utilisé sont donc :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $2^8 - 1 = 255$
 - `uint32_t` : $2^{32} - 1 = 4\,294\,967\,295$ (≥ 4 milliards)
 - `uint64_t` : $2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$ (≥ 18 milliards de milliards)

En cas de dépassement de capacité (*overflow* ou *underflow*), le résultat obtenu est calculé modulo la plus grande valeur maximale plus 1.

Par exemple, Les dépassement de capacité sur un `uint8_t` sont calculés modulo 256.

C4 Représentation des données

2. Entiers positifs

Limitations mémoire et dépassement de capacité

- Le nombre de bits représentant un entier est limité, le plus grand nombre représentable sur n bits est :

$$\overbrace{1 \dots 1}^n = 2^{n-1} + \dots + 1 = 2^n - 1$$

- En C, les valeurs maximales représentables suivant le type d'entier positif utilisé sont donc :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $2^8 - 1 = 255$
 - `uint32_t` : $2^{32} - 1 = 4\,294\,967\,295$ (≥ 4 milliards)
 - `uint64_t` : $2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$ (≥ 18 milliards de milliards)

En cas de dépassement de capacité (*overflow* ou *underflow*), le résultat obtenu est calculé modulo la plus grande valeur maximale plus 1.

Par exemple, Les dépassement de capacité sur un `uint8_t` sont calculés modulo 256.

- En OCaml, il n'y a pas nativement de type entier non signé.

C4 Représentation des données

2. Entiers positifs

Exemple

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main()
5  {
6      uint8_t n1 = 240;
7      uint32_t n2 = 0;
8      n1 = n1 + 20;
9      n2 = n2 - 1;
10     printf("valeur de n1 = %u\n",n1);
11     printf("valeur de n2 = %u\n",n2);
12 }
```

Quel est l'affichage produit par le programme ci-dessus ? Expliquer.

C4 Représentation des données

2. Entiers positifs

Correction

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main()
5  {
6      uint8_t n1 = 240; // 8 bits donc valeur maximale 255
7      uint32_t n2 = 0; // valeur minimale 0 (non signé)
8      n1 = n1 + 20; // overflow : 260
9      n2 = n2 - 1; // underflow : -1
10     printf("valeur de n1 = %u\n",n1); // 4 (car 260 = 4 modulo 256)
11     printf("valeur de n2 = %u\n",n2); // 4294967295 (car -1 =
    ↪ 4294967295 modulo 4294967296)
12 }
```

C4 Représentation des données

3. Représentation des entiers négatifs

Complément à deux

- La stratégie qui consiste à prendre *un bit de signe* et à représenter la valeur absolue de l'entier sur les autres présente deux difficultés : 0 est représenté deux fois et surtout l'addition binaire bit à bit ne fonctionne pas.

C4 Représentation des données

3. Représentation des entiers négatifs

Complément à deux

- La stratégie qui consiste à prendre *un bit de signe* et à représenter la valeur absolue de l'entier sur les autres présente deux difficultés : 0 est représenté deux fois et surtout l'addition binaire bit à bit ne fonctionne pas.
- La méthode utilisée est celle du complément à 2, sur n bits, on compte négativement le bit de poids 2^{n-1} et positivement les autres.

C4 Représentation des données

3. Représentation des entiers négatifs

Complément à deux

- La stratégie qui consiste à prendre *un bit de signe* et à représenter la valeur absolue de l'entier sur les autres présente deux difficultés : 0 est représenté deux fois et surtout l'addition binaire bit à bit ne fonctionne pas.
- La méthode utilisée est celle du complément à 2, sur n bits, on compte négativement le bit de poids 2^{n-1} et positivement les autres.

Par exemple, sur 8 bits :

$$\boxed{1 \mid 0 \mid 0 \mid 1 \mid 1 \mid 0 \mid 1 \mid 0} = -2^7 + 2^4 + 2^3 + 2^1 = -101$$

- De façon générale, sur n bits, la valeur en **complément à deux** de la suite bits $(b_{n-1} \dots b_0)$ est :

$$-b_{n-1} 2^{n-1} + \sum_{k=0}^{n-2} b_k 2^k$$

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :
 - `uint` : *au min 16 bits, usuellement 32 bits, dépendant du compilateur*

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :
 - `uint` : *au min 16 bits, usuellement 32 bits, dépendant du compilateur*
 - `uint8_t` : $\llbracket -2^7; 2^7 - 1 \rrbracket = \llbracket -128; 127 \rrbracket$

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $\llbracket -2^7; 2^7 - 1 \rrbracket = \llbracket -128; 127 \rrbracket$
 - `uint32_t` : $\llbracket -2^{31}; 2^{31} - 1 \rrbracket$

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $\llbracket -2^7; 2^7 - 1 \rrbracket = \llbracket -128; 127 \rrbracket$
 - `uint32_t` : $\llbracket -2^{31}; 2^{31} - 1 \rrbracket$
 - `uint64_t` : $\llbracket -2^{63}; 2^{63} - 1 \rrbracket$

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $\llbracket -2^7; 2^7 - 1 \rrbracket = \llbracket -128; 127 \rrbracket$
 - `uint32_t` : $\llbracket -2^{31}; 2^{31} - 1 \rrbracket$
 - `uint64_t` : $\llbracket -2^{63}; 2^{63} - 1 \rrbracket$



Un dépassement de capacité est un **comportement indéfini**.

C4 Représentation des données

3. Représentation des entiers négatifs


Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $\llbracket -2^7; 2^7 - 1 \rrbracket = \llbracket -128; 127 \rrbracket$
 - `uint32_t` : $\llbracket -2^{31}; 2^{31} - 1 \rrbracket$
 - `uint64_t` : $\llbracket -2^{63}; 2^{63} - 1 \rrbracket$
- ⚠ Un dépassement de capacité est un **comportement indéfini**.
- En OCaml, les entiers sont codés sur 64 bits mais un bit est réservé par le langage, l'intervalle représentable est donc $\llbracket -2^{62}; 2^{62} - 1 \rrbracket$.

C4 Représentation des données

3. Représentation des entiers négatifs

Conséquences de la représentation en complément à 2

- Les difficultés de la stratégie du *un bit de signe* sont levées.
- Le plus petit petit représentable sur n bits est alors -2^{n-1} et le plus grand $2^{n-1} - 1$
- En C, les valeurs extrêmes représentables sont :
 - `uint` : au min 16 bits, usuellement 32 bits, dépendant du compilateur
 - `uint8_t` : $\llbracket -2^7; 2^7 - 1 \rrbracket = \llbracket -128; 127 \rrbracket$
 - `uint32_t` : $\llbracket -2^{31}; 2^{31} - 1 \rrbracket$
 - `uint64_t` : $\llbracket -2^{63}; 2^{63} - 1 \rrbracket$
-  Un dépassement de capacité est un **comportement indéfini**.
- En OCaml, les entiers sont codés sur 64 bits mais un bit est réservé par le langage, l'intervalle représentable est donc $\llbracket -2^{62}; 2^{62} - 1 \rrbracket$.
Les dépassements de capacité sont calculés modulo 2^{63} puis ramené dans l'intervalle précédent.

C4 Représentation des données

3. Représentation des entiers négatifs

Méthode pratique

Pour obtenir la représentation en complément à deux sur n bits d'un entier négatif on pourra utiliser la méthode suivante :

C4 Représentation des données

3. Représentation des entiers négatifs

Méthode pratique

Pour obtenir la représentation en complément à deux sur n bits d'un entier négatif on pourra utiliser la méthode suivante :

- 1 on commence par écrire la représentation binaire de la valeur absolue de ce nombre

C4 Représentation des données

3. Représentation des entiers négatifs

Méthode pratique

Pour obtenir la représentation en complément à deux sur n bits d'un entier négatif on pourra utiliser la méthode suivante :

- 1 on commence par écrire la représentation binaire de la valeur absolue de ce nombre
- 2 on inverse tous les bits de cette représentation

C4 Représentation des données

3. Représentation des entiers négatifs

Méthode pratique

Pour obtenir la représentation en complément à deux sur n bits d'un entier négatif on pourra utiliser la méthode suivante :

- 1 on commence par écrire la représentation binaire de la valeur absolue de ce nombre
- 2 on inverse tous les bits de cette représentation
- 3 on ajoute 1, sans tenir compte de la dernière retenue éventuelle

C4 Représentation des données

3. Représentation des entiers négatifs

Méthode pratique

Pour obtenir la représentation en complément à deux sur n bits d'un entier négatif on pourra utiliser la méthode suivante :

- 1 on commence par écrire la représentation binaire de la valeur absolue de ce nombre
- 2 on inverse tous les bits de cette représentation
- 3 on ajoute 1, sans tenir compte de la dernière retenue éventuelle

La justification de cette méthode sera vue en TD.

Exemples

- 1 Quel est le nombre codé en complément à 2 sur 8 bits par $\overline{10110001}^2$?

C4 Représentation des données

3. Représentation des entiers négatifs

Méthode pratique

Pour obtenir la représentation en complément à deux sur n bits d'un entier négatif on pourra utiliser la méthode suivante :

- 1 on commence par écrire la représentation binaire de la valeur absolue de ce nombre
- 2 on inverse tous les bits de cette représentation
- 3 on ajoute 1, sans tenir compte de la dernière retenue éventuelle

La justification de cette méthode sera vue en TD.

Exemples

- 1 Quel est le nombre codé en complément à 2 sur 8 bits par $\overline{10110001}^2$?
- 2 Donner l'écriture en complément à 2 sur 8 bits de -12 .

C4 Représentation des données

3. Représentation des entiers négatifs

Méthode pratique

Pour obtenir la représentation en complément à deux sur n bits d'un entier négatif on pourra utiliser la méthode suivante :

- 1 on commence par écrire la représentation binaire de la valeur absolue de ce nombre
- 2 on inverse tous les bits de cette représentation
- 3 on ajoute 1, sans tenir compte de la dernière retenue éventuelle

La justification de cette méthode sera vue en TD.

Exemples

- 1 Quel est le nombre codé en complément à 2 sur 8 bits par $\overline{10110001}^2$?
- 2 Donner l'écriture en complément à 2 sur 8 bits de -12 .
- 3 Donner l'écriture en complément à 2 sur 8 bits de -75 .

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 - 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits :

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 - 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits :

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 :

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 : 11110100

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 - 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 - 2. On inverse tous les bits : 11110011
 - 3. On ajoute 1 : 11110100
- ③ Ecriture en complément à 2 sur 8 bits de -75 .

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 : 11110100
- ③ Ecriture en complément à 2 sur 8 bits de -75 .
 1. On écrit $75 = 64 + 8 + 2 + 1$ en binaire sur 8 bits :

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 : 11110100
- ③ Ecriture en complément à 2 sur 8 bits de -75 .
 1. On écrit $75 = 64 + 8 + 2 + 1$ en binaire sur 8 bits : 01001011

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 : 11110100
- ③ Ecriture en complément à 2 sur 8 bits de -75 .
 1. On écrit $75 = 64 + 8 + 2 + 1$ en binaire sur 8 bits : 01001011
 2. On inverse tous les bits :

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 : 11110100
- ③ Ecriture en complément à 2 sur 8 bits de -75 .
 1. On écrit $75 = 64 + 8 + 2 + 1$ en binaire sur 8 bits : 01001011
 2. On inverse tous les bits : 10110100

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 : 11110100
- ③ Ecriture en complément à 2 sur 8 bits de -75 .
 1. On écrit $75 = 64 + 8 + 2 + 1$ en binaire sur 8 bits : 01001011
 2. On inverse tous les bits : 10110100
 3. On ajoute 1 :

C4 Représentation des données

3. Représentation des entiers négatifs

Correction

- ① En complément à 2 sur 8 bits, $\overline{10110001}^2 = -2^7 + 2^5 + 2^4 + 2^0 = -78$
- ② Ecriture en complément à 2 sur 8 bits de -12 .
 1. On écrit $12 = (8 + 4)$ en binaire sur 8 bits : 00001100
 2. On inverse tous les bits : 11110011
 3. On ajoute 1 : 11110100
- ③ Ecriture en complément à 2 sur 8 bits de -75 .
 1. On écrit $75 = 64 + 8 + 2 + 1$ en binaire sur 8 bits : 01001011
 2. On inverse tous les bits : 10110100
 3. On ajoute 1 : 10110101

C4 Représentation des données

4. Nombre en virgule flottante

Ecriture dyadique

De la même façon que les chiffres après la virgule d'un nombre en écriture décimale utilisent les puissances de 10 négatives, par exemple :

C4 Représentation des données

4. Nombre en virgule flottante

Ecriture dyadique

De la même façon que les chiffres après la virgule d'un nombre en écriture décimale utilisent les puissances de 10 négatives, par exemple :

$$\overline{14,05}^{10} =$$

10^1	10^0	,	10^{-1}	10^{-2}
1	4	,	0	5

C4 Représentation des données

4. Nombre en virgule flottante

Ecriture dyadique

De la même façon que les chiffres après la virgule d'un nombre en écriture décimale utilisent les puissances de 10 négatives, par exemple :

$$\overline{14,05}^{10} =$$

10^1	10^0	,	10^{-1}	10^{-2}
1	4	,	0	5

En écriture binaire (ou dyadique) les chiffres après la virgule correspondent aux puissances négatives de 2 :

C4 Représentation des données

4. Nombre en virgule flottante

Ecriture dyadique

De la même façon que les chiffres après la virgule d'un nombre en écriture décimale utilisent les puissances de 10 négatives, par exemple :

$$\overline{14,05}^{10} = \begin{array}{|c|c|c|c|c|} \hline 10^1 & 10^0 & , & 10^{-1} & 10^{-2} \\ \hline 1 & 4 & , & 0 & 5 \\ \hline \end{array}$$

En écriture binaire (ou dyadique) les chiffres après la virgule correspondent aux puissances négatives de 2 :

$$\overline{10,01}^2 = \begin{array}{|c|c|c|c|c|} \hline 2^1 & 2^0 & , & 2^{-1} & 2^{-2} \\ \hline 1 & 0 & , & 0 & 1 \\ \hline \end{array}$$

C4 Représentation des données

4. Nombre en virgule flottante

Ecriture dyadique

De la même façon que les chiffres après la virgule d'un nombre en écriture décimale utilisent les puissances de 10 négatives, par exemple :

$$\overline{14,05}^{10} = \begin{array}{|c|c|c|c|c|} \hline 10^1 & 10^0 & , & 10^{-1} & 10^{-2} \\ \hline 1 & 4 & , & 0 & 5 \\ \hline \end{array}$$

En écriture binaire (ou dyadique) les chiffres après la virgule correspondent aux puissances négatives de 2 :

$$\overline{10,01}^2 = \begin{array}{|c|c|c|c|c|} \hline 2^1 & 2^0 & , & 2^{-1} & 2^{-2} \\ \hline 1 & 0 & , & 0 & 1 \\ \hline \end{array}$$

et donc $\overline{10,01}^2 = \overline{2,25}^{10}$

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

Exemple

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.

Exemple

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

Par exemple si on veut écrire $\overline{0,59375}^{10}$ en binaire :

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

Par exemple si on veut écrire $\overline{0,59375}^{10}$ en binaire :

- $0,59375 \times 2 = 1,1875 \geq 1$ donc on ajoute 1 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

Par exemple si on veut écrire $\overline{0,59375}^{10}$ en binaire :

- $0,59375 \times 2 = 1,1875 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- $0,1875 \times 2 = 0,375 < 1$ donc on ajoute 0 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

Par exemple si on veut écrire $\overline{0,59375}^{10}$ en binaire :

- $0,59375 \times 2 = 1,1875 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- $0,1875 \times 2 = 0,375 < 1$ donc on ajoute 0 à l'écriture dyadique
- $0,375 \times 2 = 0,75 < 1$ donc on ajoute 0 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

Par exemple si on veut écrire $0,59375^{10}$ en binaire :

- $0,59375 \times 2 = 1,1875 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- $0,1875 \times 2 = 0,375 < 1$ donc on ajoute 0 à l'écriture dyadique
- $0,375 \times 2 = 0,75 < 1$ donc on ajoute 0 à l'écriture dyadique
- $0,75 \times 2 = 1,5 \geq 1$ donc on ajoute 1 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

Par exemple si on veut écrire $\overline{0,59375}^{10}$ en binaire :

- $0,59375 \times 2 = 1,1875 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- $0,1875 \times 2 = 0,375 < 1$ donc on ajoute 0 à l'écriture dyadique
- $0,375 \times 2 = 0,75 < 1$ donc on ajoute 0 à l'écriture dyadique
- $0,75 \times 2 = 1,5 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- $0,5 \times 2 = 1,0 \geq 1$ donc on ajoute 1 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Méthode : du décimal au dyadique

Pour traduire une partie décimale en écriture dyadique :

- Multiplier la partie décimale par 2. Si ce produit est supérieur ou égal à 1, ajouter 1 à l'écriture dyadique sinon ajouter 0.
- Recommencer avec la partie décimale de ce produit tant qu'elle est non nul.

Exemple

Par exemple si on veut écrire $\overline{0,59375}^{10}$ en binaire :

- $0,59375 \times 2 = 1,1875 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- $0,1875 \times 2 = 0,375 < 1$ donc on ajoute 0 à l'écriture dyadique
- $0,375 \times 2 = 0,75 < 1$ donc on ajoute 0 à l'écriture dyadique
- $0,75 \times 2 = 1,5 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- $0,5 \times 2 = 1,0 \geq 1$ donc on ajoute 1 à l'écriture dyadique
- On s'arrête car la partie décimale du produit est 0 et $\overline{0,59375}^{10} = \overline{0,10011}^2$

Exemples

- 1 Donner l'écriture décimale de $\overline{1101,0111}^2$

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$
- $$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- 1 Donner l'écriture décimale de $\overline{1101,0111}^2$
 $\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$
- 2 Donner l'écriture dyadique $3,5$

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- 1 Donner l'écriture décimale de $\overline{1101,0111}^2$
$$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$
- 2 Donner l'écriture dyadique $3,5$
$$\overline{3,5}^{10} = \overline{11,1}^2$$

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- 1 Donner l'écriture décimale de $\overline{1101,0111}^2$
$$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$
- 2 Donner l'écriture dyadique $3,5$
$$\overline{3,5}^{10} = \overline{11,1}^2$$
- 3 Donner l'écriture dyadique $0,1$

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- 1 Donner l'écriture décimale de $\overline{1101,0111}^2$
$$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$
- 2 Donner l'écriture dyadique $3,5$
$$\overline{3,5}^{10} = \overline{11,1}^2$$
- 3 Donner l'écriture dyadique $0,1$

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$
$$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$
- ② Donner l'écriture dyadique $3,5$
$$\overline{3,5}^{10} = \overline{11,1}^2$$
- ③ Donner l'écriture dyadique $0,1$
 - ① $0,1 \times 2 = 0,2 < 1$ donc on ajoute 0 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$
$$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$
- ② Donner l'écriture dyadique $3,5$
$$\overline{3,5}^{10} = \overline{11,1}^2$$
- ③ Donner l'écriture dyadique $0,1$
 - ① $0,1 \times 2 = 0,2 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ② $0,2 \times 2 = 0,4 < 1$ donc on ajoute 0 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$
 $\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$
- ② Donner l'écriture dyadique $3,5$
 $\overline{3,5}^{10} = \overline{11,1}^2$
- ③ Donner l'écriture dyadique $0,1$
 - ① $0,1 \times 2 = 0,2 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ② $0,2 \times 2 = 0,4 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ③ $0,4 \times 2 = 0,8 < 1$ donc on ajoute 0 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$
$$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$
- ② Donner l'écriture dyadique $3,5$
$$\overline{3,5}^{10} = \overline{11,1}^2$$
- ③ Donner l'écriture dyadique $0,1$
 - ① $0,1 \times 2 = 0,2 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ② $0,2 \times 2 = 0,4 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ③ $0,4 \times 2 = 0,8 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ④ $0,8 \times 2 = 1,6 \geq 1$ donc on ajoute 1 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$
 $\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$
- ② Donner l'écriture dyadique $3,5$
 $\overline{3,5}^{10} = \overline{11,1}^2$
- ③ Donner l'écriture dyadique $0,1$
 - ① $0,1 \times 2 = 0,2 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ② $0,2 \times 2 = 0,4 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ③ $0,4 \times 2 = 0,8 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ④ $0,8 \times 2 = 1,6 \geq 1$ donc on ajoute 1 à l'écriture dyadique
 - ⑤ $0,6 \times 2 = 1,2 \geq 1$ donc on ajoute 1 à l'écriture dyadique

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$
 $\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$
- ② Donner l'écriture dyadique $3,5$
 $\overline{3,5}^{10} = \overline{11,1}^2$
- ③ Donner l'écriture dyadique $0,1$
 - ① $0,1 \times 2 = 0,2 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ② $0,2 \times 2 = 0,4 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ③ $0,4 \times 2 = 0,8 < 1$ donc on ajoute 0 à l'écriture dyadique
 - ④ $0,8 \times 2 = 1,6 \geq 1$ donc on ajoute 1 à l'écriture dyadique
 - ⑤ $0,6 \times 2 = 1,2 \geq 1$ donc on ajoute 1 à l'écriture dyadique
 - ⑥ Le processus se poursuit indéfiniment car on est revenu à l'étape 2.

C4 Représentation des données

4. Nombre en virgule flottante

Exemples

- ① Donner l'écriture décimale de $\overline{1101,0111}^2$

$$\overline{1101,0111}^2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} + 2^{-4} = \overline{13,4375}^{10}$$

- ② Donner l'écriture dyadique $3,5$

$$\overline{3,5}^{10} = \overline{11,1}^2$$

- ③ Donner l'écriture dyadique $0,1$

① $0,1 \times 2 = 0,2 < 1$ donc on ajoute 0 à l'écriture dyadique

② $0,2 \times 2 = 0,4 < 1$ donc on ajoute 0 à l'écriture dyadique

③ $0,4 \times 2 = 0,8 < 1$ donc on ajoute 0 à l'écriture dyadique

④ $0,8 \times 2 = 1,6 \geq 1$ donc on ajoute 1 à l'écriture dyadique

⑤ $0,6 \times 2 = 1,2 \geq 1$ donc on ajoute 1 à l'écriture dyadique

⑥ Le processus se poursuit indéfiniment car on est revenu à l'étape 2.

$$\overline{0,1}^{10} = \overline{0,0001100110011\dots}^2$$

C4 Représentation des données

4. Nombre en virgule flottante

🕒 Ecriture scientifique

Ecrire un nombre en **notation scientifique** c'est l'écrire sous la forme

C4 Représentation des données

4. Nombre en virgule flottante

🕒 Ecriture scientifique

Ecrire un nombre en **notation scientifique** c'est l'écrire sous la forme

$$\pm a \times 10^n$$

Exemples

C4 Représentation des données

4. Nombre en virgule flottante

🕒 Ecriture scientifique

Ecrire un nombre en **notation scientifique** c'est l'écrire sous la forme

$$\pm a \times 10^n$$

- avec $a \in [1; 10[$, appelée **mantisse** (l'écriture décimal de a n'a qu'un seul chiffre non nul à gauche de la virgule)

Exemples

C4 Représentation des données

4. Nombre en virgule flottante

🕒 Ecriture scientifique

Ecrire un nombre en **notation scientifique** c'est l'écrire sous la forme

$$\pm a \times 10^n$$

- avec $a \in [1; 10[$, appelée **mantisse** (l'écriture décimale de a n'a qu'un seul chiffre non nul à gauche de la virgule)
- et $n \in \mathbb{Z}$ appelée **exposant**.

Exemples

C4 Représentation des données

4. Nombre en virgule flottante

🕒 Ecriture scientifique

Ecrire un nombre en **notation scientifique** c'est l'écrire sous la forme

$$\pm a \times 10^n$$

- avec $a \in [1; 10[$, appelée **mantisse** (l'écriture décimale de a n'a qu'un seul chiffre non nul à gauche de la virgule)
- et $n \in \mathbb{Z}$ appelée **exposant**.

Exemples

- $7200000000000 = 7,2 \times 10^{12}$.

C4 Représentation des données

4. Nombre en virgule flottante

🕒 Ecriture scientifique

Ecrire un nombre en **notation scientifique** c'est l'écrire sous la forme

$$\pm a \times 10^n$$

- avec $a \in [1; 10[$, appelée **mantisse** (l'écriture décimale de a n'a qu'un seul chiffre non nul à gauche de la virgule)
- et $n \in \mathbb{Z}$ appelée **exposant**.

Exemples

- $7200000000000 = 7,2 \times 10^{12}$.
- $0,0000054 = 5,4 \times 10^{-6}$.

C4 Représentation des données

4. Nombre en virgule flottante

🕒 Ecriture scientifique

Ecrire un nombre en **notation scientifique** c'est l'écrire sous la forme

$$\pm a \times 10^n$$

- avec $a \in [1; 10[$, appelée **mantisse** (l'écriture décimale de a n'a qu'un seul chiffre non nul à gauche de la virgule)
- et $n \in \mathbb{Z}$ appelée **exposant**.

Exemples

- $7200000000000 = 7,2 \times 10^{12}$.
- $0,0000054 = 5,4 \times 10^{-6}$.
- 0 ne peut pas s'écrire en notation scientifique.

C4 Représentation des données

4. Nombre en virgule flottante

Virgule flottante

Les nombres non entiers en informatique, sont représentés en **virgule flottante**. Cette représentation :

- se fonde sur l'écriture scientifique et utilise la base 2, c'est à dire l'écriture dyadique en utilisant une mantisse et un exposant de taille limitée.

Virgule flottante

Les nombres non entiers en informatique, sont représentés en **virgule flottante**. Cette représentation :

- se fonde sur l'écriture scientifique et utilise la base 2, c'est à dire l'écriture dyadique en utilisant une mantisse et un exposant de taille limitée.
- La norme IEEE-754 définit deux formats codés respectivement sur 32 et 64 bits et stockés dans l'ordre signe/exposant/mantisse :

C4 Représentation des données

4. Nombre en virgule flottante

Virgule flottante

Les nombres non entiers en informatique, sont représentés en **virgule flottante**. Cette représentation :

- se fonde sur l'écriture scientifique et utilise la base 2, c'est à dire l'écriture dyadique en utilisant une mantisse et un exposant de taille limitée.
- La norme IEEE-754 définit deux formats codés respectivement sur 32 et 64 bits et stockés dans l'ordre signe/exposant/mantisse :

	Signe	Exposant	Mantisse	C	OCaml
32 bits	1 bit	8 bits	23 bits	<code>float</code>	x
64 bits	1 bit	11 bits	52 bits	<code>double</code>	<code>float</code>

C4 Représentation des données

4. Nombre en virgule flottante

Virgule flottante

Les nombres non entiers en informatique, sont représentés en **virgule flottante**. Cette représentation :

- se fonde sur l'écriture scientifique et utilise la base 2, c'est à dire l'écriture dyadique en utilisant une mantisse et un exposant de taille limitée.
- La norme IEEE-754 définit deux formats codés respectivement sur 32 et 64 bits et stockés dans l'ordre signe/exposant/mantisse :

	Signe	Exposant	Mantisse	C	OCaml
32 bits	1 bit	8 bits	23 bits	<code>float</code>	x
64 bits	1 bit	11 bits	52 bits	<code>double</code>	<code>float</code>

- L'exposant est décalé de façon à toujours être stocké sous la forme d'un entier positif. Ce décalage est de $127(=2^8 - 1)$ pour le format 32 bits et de $1023(=2^{11} - 1)$ pour le format 64 bits.

C4 Représentation des données

4. Nombre en virgule flottante

Virgule flottante

Les nombres non entiers en informatique, sont représentés en **virgule flottante**. Cette représentation :

- se fonde sur l'écriture scientifique et utilise la base 2, c'est à dire l'écriture dyadique en utilisant une mantisse et un exposant de taille limitée.
- La norme IEEE-754 définit deux formats codés respectivement sur 32 et 64 bits et stockés dans l'ordre signe/exposant/mantisse :

	Signe	Exposant	Mantisse	C	OCaml
32 bits	1 bit	8 bits	23 bits	<code>float</code>	x
64 bits	1 bit	11 bits	52 bits	<code>double</code>	<code>float</code>

- L'exposant est décalé de façon à toujours être stocké sous la forme d'un entier positif. Ce décalage est de $127(=2^8 - 1)$ pour le format 32 bits et de $1023(=2^{11} - 1)$ pour le format 64 bits.
- Certaines valeurs spéciales de l'exposant et de la mantisse servent à représenter des valeurs particulières (infinis, zéros, NaN).

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 1

Donner la représentation sur 32 bits du nombre $-168,75$

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 1

Donner la représentation sur 32 bits du nombre $-168,75$

- 1 Le nombre est négatif, donc le bit de signe est 1.

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 1

Donner la représentation sur 32 bits du nombre $-168,75$

① Le nombre est négatif, donc le bit de signe est 1.

② $168,75^{10} = 10101000,11^2$

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 1

Donner la représentation sur 32 bits du nombre $-168,75$

- 1 Le nombre est négatif, donc le bit de signe est 1.
- 2 $168,75^{10} = 10101000,11^2$
- 3 La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :
 $168,75^{10} = 1,010100011^2 \times 2^7$

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 1

Donner la représentation sur 32 bits du nombre $-168,75$

- 1 Le nombre est négatif, donc le bit de signe est **1**.
- 2 $168,75^{10} = 10101000,11^2$
- 3 La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :
 $168,75^{10} = 1,010100011^2 \times 2^7$
- 4 L'exposant est donc 7, et avec le décalage il est stocké sous la forme $7+127 = 134$. C'est à dire **10 000 110** en base 2.

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 1

Donner la représentation sur 32 bits du nombre $-168,75$

- 1 Le nombre est négatif, donc le bit de signe est **1**.
- 2 $168,75^{10} = 10101000,11^2$
- 3 La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :
 $168,75^{10} = 1,010100011^2 \times 2^7$
- 4 L'exposant est donc 7, et avec le décalage il est stocké sous la forme $7+127 = 134$. C'est à dire **10 000 110** en base 2.
- 5 On complète la mantisse par des zéros de façon à avoir 23 bits et le 1 initial n'est pas stocké afin d'économiser un bit. La mantisse est donc **01 010 001 100 000 000 000 000**

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 1

Donner la représentation sur 32 bits du nombre $-168,75$

- 1 Le nombre est négatif, donc le bit de signe est **1**.
- 2 $168,75^{10} = 10101000,11^2$
- 3 La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :
 $168,75^{10} = 1,010100011^2 \times 2^7$
- 4 L'exposant est donc 7, et avec le décalage il est stocké sous la forme $7+127 = 134$. C'est à dire **10 000 110** en base 2.
- 5 On complète la mantisse par des zéros de façon à avoir 23 bits et le 1 initial n'est pas stocké afin d'économiser un bit. La mantisse est donc

01 010 001 100 000 000 000 000

Le nombre $-168,75$ est donc stocké sous la forme :

1 **10 000 110** **01 010 001 100 000 000 000 000**

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 2

Donner la représentation sur 32 bits du nombre 0,1

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 2

Donner la représentation sur 32 bits du nombre 0,1

- 1 Le nombre est positif, donc le bit de signe est 0.

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 2

Donner la représentation sur 32 bits du nombre 0,1

① Le nombre est positif, donc le bit de signe est 0.

② $0,1^{10} = 0,000110011001100\dots^2$

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 2

Donner la représentation sur 32 bits du nombre 0,1

① Le nombre est positif, donc le bit de signe est 0.

② $\overline{0,1}^{10} = \overline{0,000110011001100\dots}^2$

③ La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :

$$\overline{0,1}^{10} = \overline{1,10011001100\dots}^2 \times 2^{-4}$$

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 2

Donner la représentation sur 32 bits du nombre 0,1

- 1 Le nombre est positif, donc le bit de signe est **0**.
- 2 $0,1^{10} = 0,000110011001100\dots^2$
- 3 La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :
 $0,1^{10} = 1,10011001100\dots^2 \times 2^{-4}$
- 4 L'exposant est donc -4 , et avec le décalage il est stocké sous la forme $-4 + 127 = 123$. C'est à dire **01 111 011** en base 2.

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 2

Donner le représentation sur 32 bits du nombre 0,1

① Le nombre est positif, donc le bit de signe est 0.

② $0,1^{10} = 0,000110011001100\dots^2$

③ La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :

$$0,1^{10} = 1,10011001100\dots^2 \times 2^{-4}$$

④ L'exposant est donc -4 , et avec le décalage il est stocké sous la forme $-4 + 127 = 123$. C'est à dire 01 111 011 en base 2.

⑤ La mantisse est infinie, on la limite au 23 premiers bits (c'est un arrondi et non une troncature) le 1 initial n'est pas stocké afin d'économiser un bit. La mantisse est donc 10 011 001 100 110 011 001 101

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 2

Donner la représentation sur 32 bits du nombre 0,1

① Le nombre est positif, donc le bit de signe est 0.

② $0,1^{10} = 0,000110011001100\dots^2$

③ La mantisse est décalée de façon à n'avoir qu'un chiffre non nul avant la virgule :

$$0,1^{10} = 1,10011001100\dots^2 \times 2^{-4}$$

④ L'exposant est donc -4 , et avec le décalage il est stocké sous la forme $-4 + 127 = 123$. C'est à dire 01 111 011 en base 2.

⑤ La mantisse est infinie, on la limite au 23 premiers bits (c'est un arrondi et non une troncature) le 1 initial n'est pas stocké afin d'économiser un bit. La mantisse est donc 10 011 001 100 110 011 001 101

Le nombre 0,1 est donc stocké sous la forme :

0 01 111 011 10 011 001 100 110 011 001 101

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 3

Quel nombre est stocké sous la forme :

0 10 000 100 01 010 101 000 000 000 000 000

- 1 Le bit de signe est 0, le nombre est positif

Exemple 3

Quel nombre est stocké sous la forme :

0 10 000 100 01 010 101 000 000 000 000 000

- 1 Le bit de signe est 0, le nombre est positif
- 2 L'exposant est $\overline{10000100}^2 = \overline{132}^{10}$, c'est à dire 5 en soustrayant le décalage de 127.

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 3

Quel nombre est stocké sous la forme :

0 10 000 100 01 010 101 000 000 000 000 000

- 1 Le bit de signe est 0, le nombre est positif
- 2 L'exposant est $\overline{10000100}^2 = \overline{132}^{10}$, c'est à dire 5 en soustrayant le décalage de 127.
- 3 Le 1 initial de la mantisse n'est pas stocké et donc elle est en réalité de $\overline{1,010101010000000000000000}^2 = \overline{1,33203125}^{10}$

C4 Représentation des données

4. Nombre en virgule flottante

Exemple 3

Quel nombre est stocké sous la forme :

0 10 000 100 01 010 101 000 000 000 000 000

- 1 Le bit de signe est 0, le nombre est positif
- 2 L'exposant est $\overline{10000100}^2 = \overline{132}^{10}$, c'est à dire 5 en soustrayant le décalage de 127.
- 3 Le 1 initial de la mantisse n'est pas stocké et donc elle est en réalité de $\overline{1,010101010000000000000000}^2 = \overline{1,33203125}^{10}$
- 4 Ce nombre est donc $1,33203125 \times 2^5 = 42,625$.

C4 Représentation des données

5. Conséquences de l'arithmétique à virgule flottante

! Attention !

Cette représentation approximative des nombres réels induit des conséquences importantes :

! Attention !

Cette représentation approximative des nombres réels induit des conséquences importantes :

- Les tests d'égalité entre flottants ne sont pas pertinents. On doit les éviter ou les effectuer à un ε près.

! Attention !

Cette représentation approximative des nombres réels induit des conséquences importantes :

- Les tests d'égalité entre flottants ne sont pas pertinents. On doit les éviter ou les effectuer à un ε près.

A titre d'exemple le test $0.1 + 0.2 == 0.3$ renvoie faux

! Attention !

Cette représentation approximative des nombres réels induit des conséquences importantes :

- Les tests d'égalité entre flottants ne sont pas pertinents. On doit les éviter ou les effectuer à un ε près.
A titre d'exemple le test $0.1 + 0.2 == 0.3$ renvoie faux
- Les valeurs calculées par un programme peuvent être très éloignées des valeurs théoriques d'un algorithme.

! Attention !

Cette représentation approximative des nombres réels induit des conséquences importantes :

- Les tests d'égalité entre flottants ne sont pas pertinents. On doit les éviter ou les effectuer à un ε près.
A titre d'exemple le test $0.1 + 0.2 == 0.3$ renvoie faux
- Les valeurs calculées par un programme peuvent être très éloignées des valeurs théoriques d'un algorithme.
Des exemples seront vus en TP.