

C1 Représentation des données

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

C1 Représentation des données

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 octet =

0	1	0	1	1	0	1	0
---	---	---	---	---	---	---	---

8 bits

C1 Représentation des données

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 :

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

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

C1 Représentation des données

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 :

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

- 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.

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

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 $\overline{1815}^{10}$:

C1 Représentation des données

2. ??

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

C1 Représentation des données

2. ??

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

C1 Représentation des données

2. ??

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

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

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

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

C1 Représentation des données

2. ??

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

C1 Représentation des données

2. ??

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

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

C1 Représentation des données

2. ??

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

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

C1 Représentation des données

2. ??

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 $n = \overline{a_p \dots a_1 a_0}_b$

C1 Représentation des données

2. ??

Exemples

Ecrire en base 10 les nombres ci-dessous

- $\overline{10001011}^2$

C1 Représentation des données

2. ??

Exemples

Ecrire en base 10 les nombres ci-dessous

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

C1 Représentation des données

2. ??

Exemples

Ecrire en base 10 les nombres ci-dessous

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

C1 Représentation des données

2. ??

Exemples

Ecrire en base 10 les nombres ci-dessous

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

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

Limitations mémoire et dépassement de capacité

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

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

C1 Représentation des données

2. ??

Limitations mémoire et dépassement de capacité

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

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

- 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*

C1 Représentation des données

2. ??

Limitations mémoire et dépassement de capacité

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

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

- 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` : $2^8 - 1 = 255$

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

C1 Représentation des données

2. ??

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)

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.

C1 Représentation des données

2. ??

Limitations mémoire et dépassement de capacité

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

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

- 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.

C1 Représentation des données

2. ??

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.

C1 Représentation des données

2. ??

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 }
```

C1 Représentation des données

3. ??

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.

C1 Représentation des données

3. ??

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.

C1 Représentation des données

3. ??

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 :

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

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

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

C1 Représentation des données

3. ??

Conséquences de la représentation en complément à 2

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

C1 Représentation des données

3. ??

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 $2^{n-1} - 1$

C1 Représentation des données

3. ??

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 :

C1 Représentation des données

3. ??

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*

C1 Représentation des données

3. ??

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` : $\llbracket -2^7; 2^7 - 1 \rrbracket = \llbracket -128; 127 \rrbracket$

C1 Représentation des données

3. ??

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` : $\llbracket -2^{31}; 2^{31} - 1 \rrbracket$

C1 Représentation des données

3. ??

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` : $\llbracket -2^{63}; 2^{63} - 1 \rrbracket$

C1 Représentation des données

3. ??

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**.

C1 Représentation des données

3. ??

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 $\llbracket -2^{62}; 2^{62} - 1 \rrbracket$.

C1 Représentation des données

3. ??

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**.
- 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 ramenés dans l'intervalle précédent.