Le codage des entiers

Un codage connu de tous les processeurs

Quelle est la question

Quels sont les codages de nombres intéressants à utiliser dans l'ordinateur ?

- On pourrait considérer un codage unique pour tous les nombres (qui serait celui des nombres à virgule).
- Ce serait correct, mais
  - pas efficace (les calculs seraient trop lents)
  - Le programmeur aurait plus de travail

Deux exemples de calculs

- On répartit en deux tas égaux les 27 tonnes de sables de la cargaison

  Réponse attendue : 13,5 tonnes par tas

- On répartit en deux demi-groupes les 27 élèves du groupe 7

  Réponse attendue : 13 et 14 élèves dans les demi-groupes

Les types de nombre

- Un type, c'est le choix d'un codage
- Presque tous les langages de programmation prévoient au moins un type nombre à virgule (*flottant*) et un type entier.
- On crée souvent plus de types pour permettre de consommer moins de temps et d'espace, à condition que le programmeur choisisse le bon type.

Exemple du C

- Entiers courts
- Entiers longs
- Entiers courts non signés
- Entiers longs non signés
- (et aussi :
  - Caractères
  - Nombres à virgule simple précision
  - Nombre à virgule double précision)

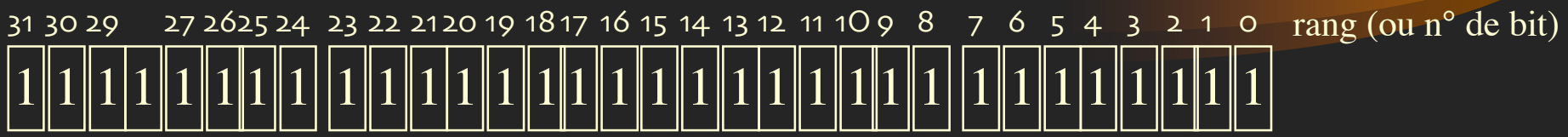
Les entiers non signés

- C'est ce qui se rapproche le plus de ce qu'on a vu sur la description d'une zone mémoire (binaire, hexadécimal ou décimal).
- Seule différence : on travaille en taille fixe :
  - entiers courts : (en général) sur 2 octets
  - entiers longs : (en général) sur 4 octets
- Il y a un plus grand entier 1111111…11. Quand on calcule 111…1 + 1, le processeur renvoie 0

Entiers non signés courts

- Le plus petit entier non signé court : $(0000000000000000)_b = 0$
- Le plus grand entier non signé court : $(1111111111111111)_b = 1+2+4+\ldots+2^{15} = 2^{16} - 1 = 65535$
- Conviennent pour de petits calculs

Entiers non signés longs



- Le plus petit entier non signé long :
  $(0000000000000000\ 0000000000000000)_b = 0$
- Le plus grand entier non signé long :
  $(1111111111111111\ 1111111111111111)_b$
  $= 1+2+4+\ldots+2^{31} = 2^{32} - 1 = 4\ 294\ 967\ 295$
- Calculs plus importants, avec 10 chiffres exacts

Les "grands" nombres : s'y retrouver

- Une remarque essentielle :

$$2^{10} = 1024 \approx 1000 = 10^3$$

$1024 = 1$ kilo $= (10000000000)_b$ *10 zéros*

Même approximation pour les puissances de 1024 :

- $2^{20} = 1024 * 1024 \approx 1000 * 1000 = 10^6$

$2^{20} = 1$ méga $= (100\ldots\ldots\ldots000)_b$ *20 zéros*

- $2^{30} = 1024 * 1024 * 1024 \approx 1000 * 1000 * 1000 = 10^9$
  2^{30} = 1 Giga = $(100\ldots\ldots\ldots\ldots000)_b$ *30 zéros*

- 2^{40} = 1 Téra = $(100\ldots\ldots\ldots\ldots000)_b$ *40 zéros*

- Avec cette règle, on calcule pour le plus grand octet non signé sur 4 octets :
  $2^{32} = 2^2 * 2^{30}$ = 4 Giga ≈ 4 000 000 000
  (valeur exacte 4 294 967 296)

Se rappeler :

$2^a * 2^b = 2^{a+b}$

Utiliser l'hexadécimal

- Remarque 1 : $100_h = 256_d$, donc

  1 kilo = $1024_d = 400_h$ (4 fois 256)

  1 mega = 1 kilo * 1 kilo = $10\ 00\ 00_h$ (car $4*4=10_h$)
  $= 100\ 000_h$

  1 giga = 1 kilo * 1 mega = $400\ 00\ 000_h = 40\ 000\ 000_h$

- Remarque 2 : on peut écrire le code d'un entier en hexadécimal. Par exemple, pour 32 bits il faut 8 chiffres hexa.

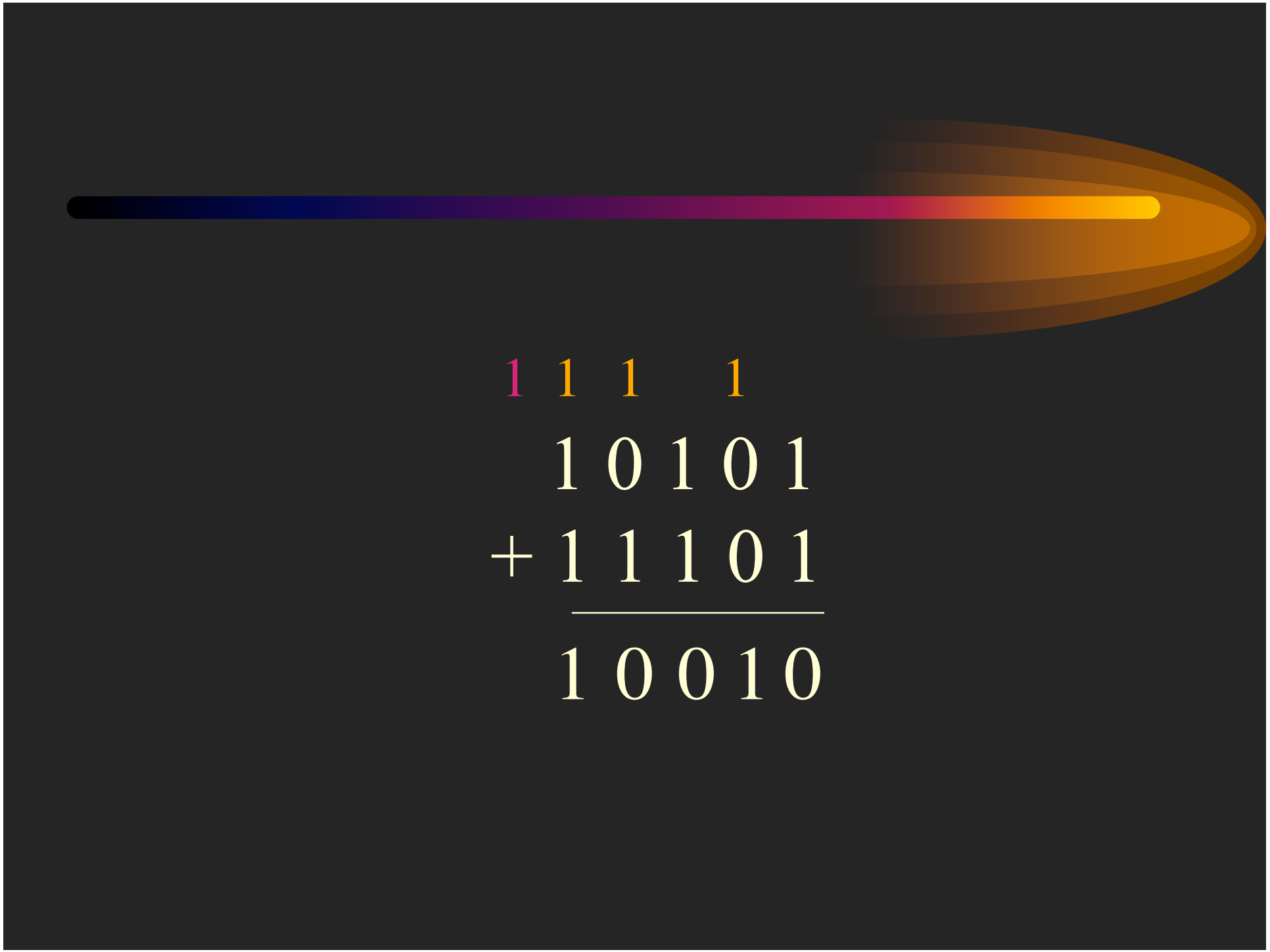
- Comme F_h représente 1111_b, le plus grand nombre sur 32 bits s'écrit FFFF $FFFF_h$
- 1 méga s'écrit $0010\ 0000_h$

Opérations en binaire

Comment fait-on une addition ?

- En décimal, on écrit 1 + 1 = 2
- En binaire : $1_b + 1_b = 10_b$
- L'algorithme d'addition en colonne avec report de la retenue reste valide :

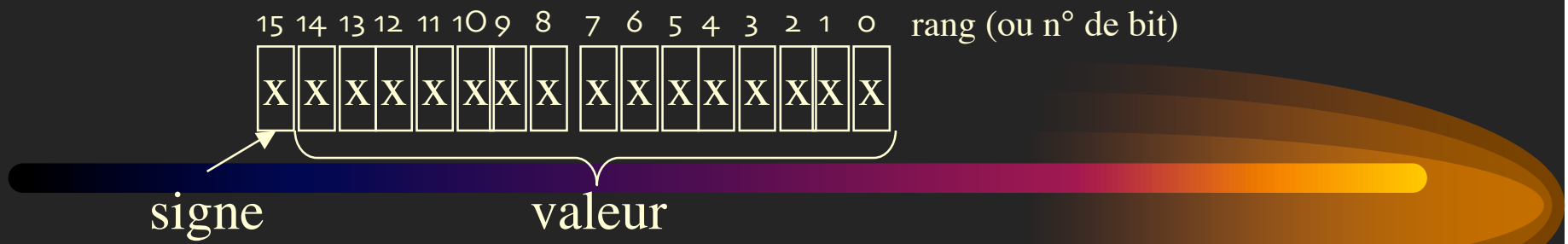
10101
+11101


1 1 1 1
1 0 1 0 1
+ 1 1 1 0 1
1 0 0 1 0

Entiers signés courts

On utilise un code sur deux octets, mais on veut représenter des nombres positifs ou négatifs

- Puisqu'on a 16 bits, on a $2^{16} = 65536$ codes (= représentations) différents.
- On choisit le bit de rang 15 pour coder le signe $\Rightarrow$ une moitié des codes pour les nombres ≥ 0, le reste pour les <0



- Positifs : bit 15 à 0, on lit la valeur comme d'habitude ⟹ même résultat que si c'était un non signé
- Négatifs : bit 15 à 1, on décide que le même algorithme d'addition que pour les non signés doit rester utilisable

  (parce que le processeur est plus simple à faire)

- On a vu que sur 16 bits, le processeur calcule 1111 1111 1111 1111 + 1 = 0

$\Rightarrow$ $1111\ 1111\ 1111\ 1111_b$ est le code de -1

- De même

$$\begin{array}{r} 1111\ 1111\ 0101\ 1010 \\ \underline{+\ 0000\ 0000\ 1010\ 0110} \\ = 0000\ 0000\ 0000\ 0000 \end{array}$$

$\Rightarrow$ puisque $0000\ 0000\ 1010\ 0110_b$ code 166, $1111\ 1111\ 0101\ 1010_b$ code -166

Décoder un entier signé négatif (1)

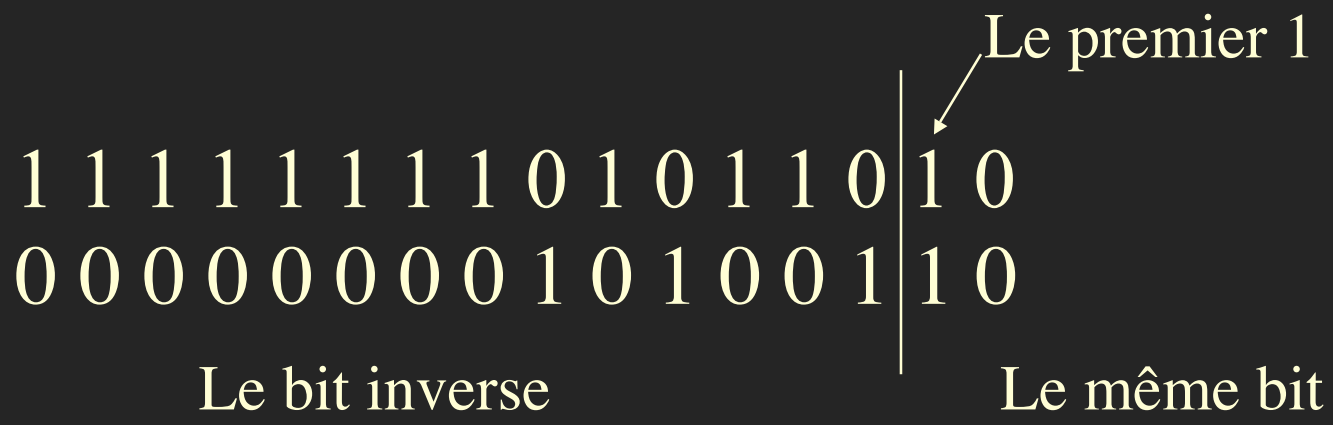
Deux techniques possibles (qui donnent le même résultat…)

- Décoder avec le code des non signés, puis soustraire 65536 (2^{16}, parce qu'on est sur 16 bits)

En non signé, $1111\ 1111\ 0101\ 1010_b$ = $2^{15}+2^{14}+2^{13}+2^{12}+2^{11}+2^{10}+2^9+2^8+2^6+2^4+2^3+2^1=65370$ puis 65370 - 65536 = - 166

Décoder un entier signé négatif (2)

- Calculer l'entier signé opposé par la technique du complément à deux, puis décoder l'opposé
- Complément à deux :



- Remarque 1 :
  si a est le complément à 2 de b, alors b est le complément à deux de a.
- Remarque 2 : le complément à deux dépend de la taille d'entiers avec laquelle on travaille.

  Complément à deux de 1101 0100 :

  sur 1 octet : 0010 1100 (positif)

  sur 2 octets : 1111 1111 0010 1100 (négatif)

Entiers signés longs

- On a 2^{32} = 4 Giga = 4 294 967 296 codes (= représentations) différents
- On code le signe en utilisant le bit (de rang) 31
- On représente les nombres de
  -2 147 483 648 à 2 147 483 647
- Les algorithmes de décodage sont identiques à ceux vus pour les signés courts.

Nombres signés en hexadécimal

- En signé, un nombre est négatif si le chiffre binaire de gauche est 1, donc si le chiffre hexadécimal de gauche est ≥ 8
  - 4FFF AG32$_h$ est positif ($4_h = 0100_b$)
  - A00B 56F4$_h$ est négatif ($A_h = 1010_b$)

Complément à deux

- Pour le complément à deux, il faut que le total des deux nombres soit 0. Ex