

Représentation interne des entiers PARTIE 1

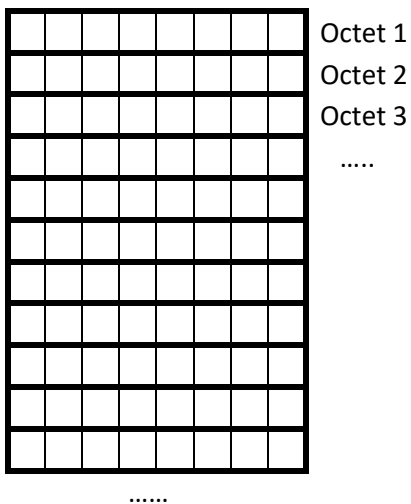
1. Notion de RAM

Le rôle principal d'un ordinateur est d'exécuter des programmes. Pour ce faire, pendant leur exécution, les programmes sont emmagasinés dans la mémoire principale de l'ordinateur, la RAM (Random Access Memory).

La RAM est composée d'un ensemble d'octets.

Un octet est un groupe de 8 bits.

Un bit est la plus petite unité de représentation de l'information. Il peut prendre 2 valeurs : 0 ou 1 (physiquement : 2 voltages différents).



La RAM qui précède contient 11 octets (Ce n'est vraiment pas beaucoup....)

Unités de capacité de la RAM : (utilisées pour la RAM ainsi que pour d'autres mémoires)

Bit 

Octet (8 bits)

Kilo-octet (1024 octets)

Méga-octet (1024 Ko)

Giga-octet (1024 Mo)

Téra-octet (1024 Go)

Exemples : RAM de 4 Mo = $4 * (1024 * 1024)$ octets = 4 194 304 octets

RAM de 8 Go = $8 * (1024 * 1024 * 1024)$ octets = 8 589 934 592 octets

Les micro-ordinateurs que l'on achète aujourd'hui ont 8,12,16, 32... Go.

Exercices :

- a) 12 Mo = combien de bits ?
- b) 512 bits = combien d'octets ?

2. Notion de programme

Programme : Suite d'instructions qui permet de produire un résultat

Exemple 1 : écrire un programme qui fait la somme et la différence de 5 et 7

Programme : Afficher 5+7

Afficher 5-7

Exemple 2 : écrire un programme qui fait la somme et la différence de 12 et 8

Programme : Afficher $12 + 8$

 Afficher $12 - 8$

Exemple 3 : écrire un programme qui fait la somme et la différence de 20 et 5

Programme : Afficher $20 + 5$

 Afficher $20 - 5$

Cette façon de faire n'est évidemment pas optimale: il faut ré écrire le programme à chaque fois qu'on change de valeurs à additionner et soustraire.

C'est pour cette raison qu'on utilise des variables dans les programmes. Pour notre exemple, avec les variables, on n'a à écrire qu'une seule fois le programme et celui-ci pourra être utilisé pour toutes les paires de nombres qu'on voudra additionner et soustraire.

Une variable est donc un contenant.

Nouvelle définition de programme :

Programme : suite d'instructions qui lit des données, les traite et produit un résultat.

Programme :

 Aller chercher des données (lecture) et les mettre dans des variables

 Traitement

 Rendre disponible les résultats (affichage ou impression) à partir de variables

Exemple simple : programme qui lit 2 nombres (les données) et qui calcule leur somme et leur différence. La somme et la différence (les résultats) sont ensuite affichés.

Lire A,B

SOMME = A + B

DIFF = A – B

Afficher SOMME, DIFF

Note : ce programme est écrit en pseudo-code. Pour le faire exécuter, il faudrait le traduire dans un langage de programmation.

En RAM :

Lire A,B
SOMME = A + B
DIFF = A – B
Afficher SOMME, DIFF
.....

Supposons que les valeurs lues sont 2 et 3 :

	Lire A,B	
	SOMME = A + B	
	DIFF = A – B	
	Afficher SOMME, DIFF	
A	2	
B	3	
SOMME	5	
DIFF	-1	

Programme

Données

Résultats

2, 3 → Programme → 5, -1

Dans l'exemple de la page précédente, on voit qu'une variable = 1 case mémoire (1 octet). On verra plus tard qu'une variable peut prendre plus d'une case mémoire si nécessaire.

Note : toujours dans l'exemple précédent, on parle de variables seulement pour les données et les résultats. Il est souvent nécessaire d'utiliser aussi des variables dans la partie traitement.

3. Les représentations internes

Dans l'exemple précédent, le programme est écrit avec des lettres et des opérateurs arithmétiques et les données et les résultats sont écrits en décimal. En réalité, en mémoire, tout est en binaire (un octet est un ensemble de bits et un bit ne peut que prendre la valeur 0 ou la valeur 1).

Nous verrons plus tard comment est représenté le programme en RAM.

Voyons d'abord comment sont représentées les valeurs dans les variables. On commence par les entiers.

Représentation interne des données et des résultats POUR LES ENTIERS

Entiers positifs :

1. Transformer l'entier en binaire
2. Compléter avec des zéros non significatifs si nécessaire


Ex : Donner la représentation interne de 25 sur 8 bits.

1. $25_{10} = 11001_2$
2. Sur 8 bits : 00011001 (les zéros non significatifs doivent être présents dans la réponse)

Entiers négatifs :

1. Transformer la valeur absolue de l'entier en binaire
2. Compléter avec des zéros non significatifs si nécessaire
3. Complémenter (à partir de la droite, recopier les chiffres jusqu'au premier 1 inclusivement et inverser les autres)

Ex : Donner la représentation interne de -34 sur 8 bits.

1. $34_{10} = 100010_2$
2. Sur 8 bits : 00100010
3. Complément : 11011110 

Notion de bit de signe : le bit le plus à gauche est appelé le bit de signe. Il prend la valeur 0 dans le cas d'un positif et 1 dans le cas d'un négatif.

Exercices : Donner la représentation interne 8 bits de :

- a) -16
- b) 42
- c) 150
- d) -2
- e) 120
- f) -2834

Solution : (remarquez qu'il y a obligatoirement 8 bits dans les réponses)

- a) 11110000
- b) 00101010
- c) 10010110 (???)
- d) 11111110
- e) 01111000
- f) $2834_{10} = 101100010010_2$ (???)

Si on revient à notre exemple de programme, en RAM, on aura :

En binaire :

A		0	0	0	0	0	0	1	0	
B		0	0	0	0	0	0	1	1	
SOMME		0	0	0	0	0	1	0	1	
DIFF		1	1	1	1	1	1	1	1	

Programme

Données

Résultats

En hexadécimal :

	Lire A,B	Programme
	SOMME = A + B	
	DIFF = A – B	
	Afficher SOMME, DIFF	
A	02	Données
B	03	
SOMME	05	Résultats
DIFF	FF	

À partir de maintenant, les contenus des données et résultats en RAM seront en hexa (sinon, je l'indiquerai).

Exercice: en utilisant le programme précédent, si on donne à A la valeur -4_{10} et à B la valeur -5_{10} , quelles seront les valeurs en mémoire de A, B et SOMME en binaire et en hexadécimal ?

Voyons maintenant comment faire le chemin inverse, c'est-à-dire comment, à partir d'une valeur en RAM, retrouver l'entier en base 10 (ce que l'ordinateur doit faire pour imprimer ou afficher une valeur, par exemple, si on demande à l'ordinateur d'afficher le contenu de la variable SOMME)

On doit d'abord déterminer si le nombre codé est positif ou négatif en regardant le bit de signe.

Si le nombre est positif (bit de signe = 0), on traduit directement en base 10.

Si le nombre est négatif (bit de signe = 1), on complémente et on traduit ensuite en base 10.

Exemple1 : on demande à l'ordinateur d'afficher à l'écran le contenu de la variable SOMME.

- Le bit de signe est 0, donc c'est une valeur positive
- L'ordinateur n'a qu'à traduire en décimal la valeur binaire : $(00000101)_2 = 5_{10}$
- La valeur affichée sera 5

Exemple2 : on demande à l'ordinateur d'afficher à l'écran le contenu de la variable DIFF.

- Le bit de signe est 1, donc c'est une valeur négative
- Il faut complémenter : le complément de 11111111 est 00000001
- L'ordinateur va traduire en décimal la valeur binaire complémentée :
 $(00000001)_2 = 1_{10}$
- La valeur affichée sera -1

Exercice : les valeurs suivantes représentent des entiers contenus dans des variables en RAM. Si on demande à l'ordinateur d'afficher ces contenus, que sera-t-il affiché ?

- a) $1\ 1\ 0\ 1\ 1\ 1\ 1\ 1_2$
- b) $0\ 1\ 0\ 1\ 0\ 1\ 0\ 1_2$
- c) 42_{16}
- d) $E8_{16}$

Solution :

- a) -33
- b) 85
- c) 66
- d) -24

Limite des représentations internes des entiers

Quel est le plus grand nombre positif que je peux représenter sur 8 bits ? Le plus grand négatif ?

Voyons un exemple sur 4 bits pour trouver une formule générale :

Du côté positif (avec « 0 » dans le bit de signe) :

0000 (0)

0001 (1)

0010 (2)

0011 (3)

0100 (4)

0101 (5)

0110 (6)

0111 (7)

Du côté négatif (avec « 1 » dans le bit de signe) :

1000 (-8)

1001 (-7)

1010 (-6)

1011 (-5)

1100 (-4)

1101 (-3)

1110 (-2)

1111 (-1)

Sur 4 bits, les valeurs possibles sont $[-8, 7] = [-2^3, 2^3-1]$ avec n =nombre de bits.

Formule générale : sur n bits, on peut représenter les chiffres $[-2^{n-1}, 2^{n-1}-1]$

Sur 8 bits (comme les exemples et exercices précédents), si on applique la formule $[-2^{n-1}, 2^{n-1}-1]$, on a :

$$[-2^{n-1}, 2^{n-1}-1] \text{ avec } n = 8 \rightarrow [-2^{8-1}, 2^{8-1}-1] = [-2^7, 2^7-1] = [-128, 127]$$

Comment peut-on être certains que le bit de signe va toujours fonctionner, peu importe le nombre de bits ?

Reprenons l'exemple sur 8 bits :

Du côté positif (bit de signe = 0), on a :

00000000 (0_{10})

00000001 (1_{10})

00000010 (2_{10})

00000011 (3_{10})

....

01111110 (126_{10})

01111111 (127_{10})

→ De 0 à 127, tous les nombres sont représentés (et la limite du côté positif sur 8 bits = 127)

Du côté néгатif (bit de signe = 1), on a :

10000000 (-128_{10})

10000001 (-127_{10})

10000010 (-126_{10})

10000011 (-125_{10})

....

01111110 (-2_{10})

11111111 (-1_{10})

→ De -1 à -128, tous les nombres sont représentés (et la limite du côté négatif sur 8 bits = 127)

Dans la réalité, on n'est pas toujours sur 8 bits. On a quelquefois besoin de 16, 32 ou 64 bits.
(Par exemple, si on donne à A la valeur 100_{10} et à B la valeur 150_{10})

Si on a à travailler avec un nombre hors de l'intervalle -128, 127, on pourra utiliser 2, 4 ou 8 octets. Il faudra alors prendre une variable plus grande. On y reviendra dans la section suivante.

Exemple : Donner la représentation interne 16 bits de 72. Donner la réponse en hexadécimal.

1. $72_{10} = 1001000_2$
2. Sur 16 bits : 000000001001000
3. En hexadécimal : 0048 (les 2 zéros à gauche doivent apparaître dans la réponse)

Exercice1 : Donner la représentation interne 16 bits de : (donnez vos réponses en hexa)

- a) -16
- b) 42
- c) 150
- d) -2
- e) 120
- f) -2834

Solution :

- a) $1111111111110000_2 = \text{FFF0}_{16}$
- b) $000000000101010_2 = \text{002A}_{16}$
- c) $0000000010010110_2 = \text{0096}_{16}$
- d) $111111111111110_2 = \text{FFFE}_{16}$
- e) $000000001111000_2 = \text{0078}_{16}$
- f) $1111010011101110_2 = \text{F4EE}_{16}$

Exercice2 : Donner la représentation interne 32 bits de : (donnez vos réponses en hexa)

- a) 56
b) -12

Solution :

- a) $000000000000000000000000111000_2 = 00000038_{16}$
 b) $11111111111111111111111110100_2 = \text{FFFFFF}4_{16}$