R1.03 : Introduction à l'architecture des ordinateurs

Olivier ROUSSEL
olivier.roussel@univ-artois.fr

26/09/2022, version 4297

Le Programme Nationnal

Quel objectif pour cette ressource ?

- L'objectif de cette ressource est de découvrir la structure et les composants d'un ordinateur.

Quels savoirs de référence à étudier ?

- Architecture générale d'un ordinateur, histoire et évolution de l'informatique
- Codage (codage des informations de base : nombres, caractères)
- Arithmétique des traitements associés
- Etude d'un ordinateur personnel (composants...)
- Evolution des technologies et des systèmes

Comment cette ressource fait-elle monter en compétence ?

- Cette ressource permettra de découvrir les différents composants matériels et logiciels internes qui constituent un ordinateur, de manière à appréhender le fonctionnement, mais aussi les limites de leur utilisation.

Mots-clefs : Architecture, Codage, Binaire

Contenu

- codage des informations de base (int, float, char)
- étude des composants d'un ordinateur
- lien avec la SAE 1.03 : installation d'un poste de travail

Modalités

Des chiffres :

- 8 semaines de cours,
- 1 séance de TD par semaine,
- 1 séance de TD en plus 1 semaine sur 2
- 3 séances de cours,
- 1 DS en CM, de multiples contrôles en TD qui donneront une note de DS.

Les règles en CM, TD, TP :

- Aucun appareil électronique (ordinateur, calculatrice, téléphone, ...)

Les règles pour les DS :

- Aucun appareil électronique (ordinateur, calculatrice, téléphone, ...)
- **seul document autorisé : une feuille manuscrite, format A4, recto-verso, avec votre nom dessus**

Avertissement !

- Certains transparents de ce cours contiennent un certain nombre de simplifications...
- ...donc un certain nombre de mensonges

Codage des informations de base

Ordinateur/calculateur

- Fondamentalement, un ordinateur ne manipule que des nombres. Toute information (texte, son, image,...) est codée par des nombres.
- Le terme calculateur est plus juste que le terme ordinateur

Codage des entiers naturels (positifs ou nul)

Chiffres/Nombres

- Un chiffre est un symbole (un caractère) qui apparaît dans l'écriture des nombres.
- Un nombre s'écrit avec des chiffres (tout comme un mot s'écrit avec des lettres).
- Autrement dit, un nombre est l'analogue d'un mot, un chiffre est l'analogue d'une lettre.

Représentation en base 10

- Habituellement, on utilise 10 chifres (de 0 à 9) : on dit qu'on travaille en base 10. On parle aussi de système décimal (écriture décimale).
- Dans un nombre, les chiffres n'ont pas le même poids (n'ont pas la même valeur)
  - le chiffre des unités doit être multiplié par 1 (soit 10^0),
  - le chiffre des dizaines doit être multiplié par 10(soit 10^1),
  - le chiffre des centaines doit être multiplié par 100 (soit 10^2),
  - le chiffre des milliers doit être multiplié par 1000 (soit 10^3),
  - ...
- Exemple le nombre 54321 est égal à $5 * 10^4 + 4 * 10^3 + 3 * 10^2 + 2 * 10^1 + 1 * 10^0$.

Représentation en base b

- On généralise la base 10 à n'importe quelle base b en remplaçant simplement 10 par b
- En base b, il y a exactement b chiffres, de 0 à $b-1$.
- Un nombre de n chiffres en base b noté $(c_{n-1}\dots c_3c_2c_1c_0)_b$ représente le nombre $\sum_{i=0}^{n-1} c_i.b^i$.
- Autre présentation : $(c_{n-1}\dots c_3c_2c_1c_0)_b = c_{n-1}*b^{n-1}+\dots+c_3*b^3+c_2*b^2+c_1*b^1+c_0*b^0$
- Le poids d'un chiffre c_i est la puissance b^i qui lui correspond.
- c_0 est le chiffre de plus faible poids (moins significatif/least significant)
- c_{n-1} est le chiffre de plus fort poids (plus significatif/most significant)

ga - bu - zo - meu (base 4)

Et voilà les Shadoks : S02 Ep 44 | Archive INA (sur youtube)

Binaire

- Un nombre binaire est un nombre écrit en base 2.
- En binaire, il n'y a donc que 2 chiffres : 0 et 1
- Un bit (contraction de Binary Digit) est un chiffre binaire (0 ou 1).
- Dans l'écriture $(c_{n-1}\dots c_3c_2c_1c_0)_2$, le poids du bit c_i d'indice i est égal à 2^i
- Donc, $(c_{n-1}\dots c_3c_2c_1c_0)_2 = c_{n-1}.2^{n-1}+\dots+c_3.2^3+c_2.2^2+c_1.2^1+c_0.2^0$
- Exemple : $(1010\,0011)_2 = 2^7+2^5+2^1+2^0 = 128+32+2+1 = 163$
- Les chiffres d'un nombre binaire seront souvent désignés par b_i (pour bit). Un nombre sur n bits sera noté $(b_{n-1}\dots b_3b_2b_1b_0)_2$

Puissances de 2 à connaître

- $2^0 = 1$,
- $2^1 = 2$,
- $2^2 = 4$,
- $2^3 = 8$,
- $2^4 = 16$,
- $2^5 = 32$,
- $2^6 = 64$,
- $2^7 = 128$,
- $2^8 = 256$,
- $2^9 = 512$,
- $2^{10} = 1024$,
- $2^{11} = 2048$,
- $2^{12} = 4096$,
- $2^{13} = 8192$,
- $2^{14} = 16384$,
- $2^{15} = 32768$,
- $2^{16} = 65536$

Bases courantes en informatique

- base 2 (binaire) :
  - chiffres : 0, 1
  - préfixe en C++ : **0b** (exemple 0b1010'0011)
- base 8 (octal) :
  - chiffres : 0, 1, 2, 3, 4, 5, 6, 7
  - préfixe en C++ : **0** (exemple 0644)
  - correspond directement à un groupe de 3 bits
  - surtout utilisé pour coder les droits unix (rwx)
- base 10 (décimal)
  - chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - préfixe en C++ : aucun (exemple 1234)
  - utilisé pour les nombres usuels, les adresses IPv4
- base 16 (hexadécimal)
  - chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - préfixe en C++ : **0x** (exemple 0xFE62)
  - correspond directement à un groupe de 4 bits
  - utilisé pour écrire de manière compacte du binaire, les adresses IPv6

Invariance

- Changer de base ne change pas la valeur d'un nombre ! 34 = 0b10'0010 = 0x22 = 042 = $(114)_5$
- Tout comme changer de langue ne change pas la valeur du nombre ! quatre-vingt deux = octante deux = eighty two = zwei und achtzig = ottantadue = ochenta y dos

Conversion en base b : divisions successives

- n mod b (le reste de la division de n par b) donne le chiffre des "unités" du nombre n écrit en base b.
- n div b (le quotien de la division entière de n par b) donne les autres chiffres du nombre n (écrit en base b).
- En répétant ce processus, on produit les chiffres du nombre en base b, en commençant par le chiffre le moins significatif et en terminant par le plus significatif.
- Algorithme :

```
repeat
    stocker le chiffre n mod b;
    n ← n div b;
until n = 0;
afficher les chiffres dans l'ordre inverse de leur production;
```


Conversion en base b : divisions successives

- L'algorithme par divisions successives produit les chiffres dans l'ordre inverse de leur affichage.
- Il faut donc, soit les stocker dans une pile, soit utiliser la récursivité (ce qui revient au même)

Conversion en base b : les chiffres dans le bon ordre

- Si on divise n par $p = (100\dots000)_b$ (le nombre en base b qui commence par 1 suivi d'autant de zéro que possible sans dépasser n, autrement dit le plus grand b^k qui soit $\leq n$), on obtient le chiffre le plus significatif en base b.
- n mod p donne alors le reste des chiffres du nombre.
- En répétant le processus, on produit les chiffres dans le bon ordre.
- Algorithme :

```
p ← 1;
while p * b ≤ n do
    p ← p * b;
while p ≠ 0 do
    afficher n div p;
    n ← n mod p;
    p ← p div b;
```


Conversion en binaire : soustractions successives

- Quand le nombre à convertir est petit (i.e. tient sur un octet donc ≤ 255), il est souvent plus simple et plus rapide de procéder par soustraction successive
- Algorithme :

```
foreach p ∈ {128, 64, 32, 16, 8, 4, 2, 1} par ordre décroissant do
    if n ≥ p then
        n ← n − p ;
        afficher 1;
    else
        afficher 0;
```

- Cet algorithme est un cas particulier de l'algorithme qui produit les chiffres dans le bon ordre.

Lecture d'un nombre en base b

- On commence avec $n = 0$.
- Chaque fois qu'on lit un chiffre c, $n \leftarrow n * b + c$
- Algorithme :

while *un chiffre c est lu* **do**
  $n \leftarrow n * b + c$;

- N.B. dans cet algorithme, le chiffre c doit être compris entre 0 et $b - 1$

Quartet, Octet, Seizet,...

- Un quartet est un nombre qui s'écrit sur 4 bits.
- Un octet est un nombre qui s'écrit sur 8 bits.
- Un seizet (terme peu usité) est un nombre qui s'écrit sur 16 bits.
- De manière générale, on parle de mot de n bits pour un nombre qui s'écrit sur n bits.

Conversion binaire/octal

- Un chiffre en octal correspond à exactement 3 bits.
- Le tableau ci-desous donne la conversion entre groupes de 3 bits et chiffre en octal :

octal	binaire
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

- **Il faut toujours grouper les bits en partant du bit de poids le plus faible (donc en partant de la droite) !**

Conversion binaire/hexadécimal

- Un chiffre en hexadécimal correspond à exactement 4 bits.
- Le tableau ci-desous donne la conversion entre groupes de 4 bits et chiffre en hexadécimal :

hexadécimal	binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

hexadécimal	binaire
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

- **Il faut toujours grouper les bits en partant du bit de poids le plus faible (donc en partant de la droite) !**

Calculs en base b

- En base 10, vous connaissez les algorithmes pour additionner, soustraire, multiplier, diviser deux nombres et aussi calculer le successeur ou le prédecesseur d'un nombre.
- En base b, ces algorithmes sont les mêmes mais il faut remplacer 10 par b.
- En binaire, la multiplication et la division sont plus simples qu'en base 10, parce qu'on multiplie uniquement par 0 ou par 1.

Codage des entiers relatifs (positifs ou négatifs)

Calculs sur n bits

- Les processeurs travaillent avec un nombre fixe de bits (par exemple 64 bits).
- Les générations précédentes de processeurs travaillaient sur 32 bits, 16 bits ou 8 bits.
- Quand on effectue les calculs sur n bits, si le résultat d'un calcul requiert plus de n bits, les bits de poids 2^i avec $i \geq n$ sont perdus.
- Concrètement, si on ne prend pas de précaution particulière, on calcule modulo 2^n.
- Exemple sur 8 bits : 250+10=0b1111'1010 + 0b0000'1010=0b0000'0100 (le 9ème bit est perdu) =4=260 mod 256.

Codage signe et module

- Si on travaille sur n bits, on peut utiliser le bit de plus fort poids (2^{n-1}) pour coder le signe (0 pour +, 1 pour -) et les $n - 1$ bits restants pour coder la valeur absolue du nombre.
- Ce codage correspond exactement à notre notation usuelle des nombres (+12, -8).
- Exemples sur 8 bits : +12 est codé par 0b0'000'1100, -8 est codé par 0b1'000'1000.

Codage signe et module : exemple sur 4 bits

codage	valeur
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

Codage signe et module

Avantages/inconvénients

(+) codage très simple

(-) deux codages pour zéro

(-) addition/soustraction plus compliquées

(-) les négatifs sont codés après les positifs

Codage en complément à 2

- Le complément à 1 d'un nombre x s'obtient en codant x en binaire et en inversant chaque bit (0 devient 1 et 1 devient 0).
- On le notera $compl_1(x)$.
- On pourra noter que sur n bits, $x + compl_1(x) = 2^n - 1$ (que des 1 en binaire).
- Le complément à 2 d'un nombre x s'obtient en ajoutant 1 au complément à 1.
- On le notera $compl_2(x)$.
- Donc, $compl_2(x) = compl_1(x) + 1$.
- Le calcul $compl_2(x)$ permet d'obtenir le codage de $-x$ (l'opposé de x).
- En calculant sur n bits, on peut vérifier que $compl_2(compl_2(x)) = x$ (donc $-(-x) = x$) et $x + compl_2(x) = 0$ (donc $x + (-x) = 0$).

Complément à 2

- Il résulte de la définition $compl_2(x) = compl_1(x) + 1$ qu'on peut obtenir le complément à 2 d'un nombre en repérant le 1 le plus à droite et en inversant tous les bits qui se trouvent à sa gauche.
- Cas particulier : $compl_2(0) = 0$.

Codage en complément à 2

- Un nombre positif se code en binaire comme les entiers naturels.
- Pour un nombre négatif, il faut coder sa valeur absolue et calculer le complément à 2 pour obtenir l'opposé.
- Quand on a un nombre négatif, pour le décoder, il faut calculer son complément à 2 pour obtenir la valeur absolue du nombre.
- Un nombre positif a toujours son bit de poids fort à 0.
- Un nombre négatif a toujours son bit de poids fort à 1.
- Exemples sur 8 bits : +12 est codé par 0b0000'1100, -8 est codé par 0b1111'1000 (on code 8, ce qui donne 0b0000'1000, on inverse chacun des bits 0b1111'0111, et on ajoute 1 pour obtenir 0b1111'1000

Codage en complément à 2 : exemple sur 4 bits

codage	valeur
0000	0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Codage en complément à 2

Avantages/inconvénients

(+) codage simple

(+) un seul codage pour zéro

(+) addition/soustraction très simple (addition binaire usuelle)

(-) les négatifs sont codés après les positifs

(-) il y a un nombre positif de moins que chez les négatifs

Codage avec biais (offset)

- On choisit de coder 0 par une valeur o (offset) et on code un nombre x par $o + x$.
- Si on travaille sur n bits, choisir $o = 2^{n-1}$ facilite les calculs car $(o + x) + (o + y) = x + y + 2o = x + y + 2^n = x + y$ (modulo 2^n).
- Exemples sur 8 bits avec $o = 128$: +12 est codé 128+12 soit 0b1000'1100, -8 est codé par 128-8 soit 0b0111'1000.

Codage avec biais : exemple sur 4 bits, biais=8

codage	valeur
0000	-8
0001	-7
0010	-6
0011	-5
0100	-4
0101	-3
0110	-2
0111	-1
1000	0
1001	+1
1010	+2
1011	+3
1100	+4
1101	+5
1110	+6
1111	+7

Codage avec biais (offset)

Avantages/inconvénients

(+) codage simple

(+) un seul codage pour zéro

(+) addition/soustraction très simple

(+) les négatifs sont codés avant les positifs

(-) il y a un nombre positif de moins que chez les négatifs

(-) le zéro ne correspond plus à un nombre avec tous les bits à 0.

(-) les entiers positifs ne sont plus codés comme les entiers naturels

Codage des nombres réels

Nombre "réel" représenté en base b

- En base 10, la notation 123,456 signifie $1 * 10^2 + 2 * 10^1 + 3 * 10^0 + 4 * 10^{-1} + 5 * 10^{-2} + 6.10^{-3}$
- En base b, le principe est exactement le même. Il suffit de remplacer 10 par b.
- Donc, la notation $c_n \dots c_2 c_1 c_0 , c_{-1} c_{-2} c_{-3} c_{-4} c_{-5} \dots c_{-m}$ correspond au nombre

$$\sum_{i=-m}^{i=n} c_i . b^i$$

- Rappel : $b^{-i} = 1/b^i$

Puissances de 2 négatives

- $2^0 = 1$
- $2^{-1} = 0,5$
- $2^{-2} = 0,25$
- $2^{-3} = 0,125$
- $2^{-4} = 0,062\,5$
- $2^{-5} = 0,031\,25$
- $2^{-6} = 0,015\,625$
- $2^{-7} = 0,007\,812\,5$
- $2^{-8} = 0,003\,906\,25$
- $2^{-9} = 0,001\,953\,125$
- $2^{-10} = 0,000\,976\,562\,5$

Conversion en binaire

- Pour la partie entière, on procède comme vu précédemment sur les entiers.
- En base 10, si je multiplie 0,789 par 10, j'obtiens 7,89. La partie entière me donne le premier chiffre après la virgule, la partie fractionnaire me donne les chiffres qui suivent.
- Pour convertir la partie fractionnaire, on procède par multiplication par deux. Soit f la partie fractionnaire ($f < 1$)

```
while f ≠ 0 do
    f← f*2;
    if f≥1 then
        afficher(1);
        f← f-1;
    else
        afficher(0);
```

- les chiffres sont produits dans le "bon" ordre (du plus fort poids au plus faible poids, de gauche à droite).

Conversion en binaire

- Attention : un nombre en base 10 avec un nombre fini de décimales peut se traduire en un nombre en base 2 avec un nombre infini de bits après la virgule. Ex : $1/5 = 0,2 = (0,0011\underline{0011}\ldots)_2$
- 0,1 ne peut être représenté avec un nombre fini de bits après la virgule !
  Supposons qu'il suffise de m bits après la virgule pour représenter 0,1. Dans ce cas, il suffit de multiplier par 2^m pour retomber sur un nombre entier n. On aurait donc $0,1 * 2^m = n$ et donc $2^m = 10 * n$. En considérant la décomposition en facteurs premiers des deux côtés de l'équation, on s'aperçoit que c'est impossible : il n'y a que des 2 à gauche, et il y a au moins un 5 à droite.

Représentation en virgule fixe

- On représente les nombres sur $n+m$ bits au total :
  - n bits avant la virgule (partie entière)
  - m bits après la virgule (partie fractionnaire)

  $b_{n-1}\ldots b_2 b_1 b_0 , b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} \ldots b_{-m}$
- Revient à travailler avec une fraction de dénominateur constant 2^m
  $b_{n-1}\ldots b_2 b_1 b_0 , b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} \ldots b_{-m} =$

$$\frac{b_{n-1}\ldots b_2 b_1 b_0 b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} \ldots b_{-m}}{2^m}$$

- Donc les additions, soustractions et divisions se font comme sur les entiers. Pour la multiplication, il y a un décalage à droite de m qui intervient.
- Avantage : simple.
- Inconvénient : ne sait pas représenter les très petits ou les très grands nombres.

Représentation en virgule flottante : généralités

- Dérive directement de la notation scientifique $-1,6.10^{37}$ (noté -1.6E37 dans les langages informatiques).
- Généralisation à la base b

$$S.M.b^E$$

- S est le signe ($S \in \{+1, -1\}$)
- M est la mantisse (les chiffres significatifs)
- E est l'exposant
- En jouant sur l'exposant, on peut toujours se ramener au cas où $M \in [1, b[$. On dit que le nombre est normalisé.
- Quand on déplace la virgule d'un chiffre vers la gauche, on doit ajouter 1 à l'exposant : $12,34.10^4 = 1,234.10^5$
- Quand on déplace la virgule d'un chiffre vers la droite, on doit retirer 1 à l'exposant : $12,34.10^4 = 123,4.10^3$

Normalisation/Dénormalisation

- Normalisation : chaque fois que possible, on impose $1 \leq M < b$ (il suffit de jouer sur l'exposant pour y arriver). Si l'on est en base 2, la mantisse normalisée est nécessairement de la forme $(\mathbf{1}, XXX\ldots)_2$. De ce fait, on ne stockera pas explicitement le premier bit de la mantisse (mais il ne faut pas l'oublier) .
- Dénormalisation : quand on dépasse le plus petit exposant qu'on sait représenter, on doit abandonner la normalisation. On parle alors de nombre dénormalisé. Il est de la forme $(\mathbf{0}, XXX\ldots)_2$. De ce fait, on ne stockera pas explicitement le premier bit de la mantisse (mais il ne faut pas l'oublier) .
- Concrètement, en binaire, on ne stockera en mémoire que la partie fractionnaire de la mantisse, mais il ne faut pas oublier de replacer le bit implicite avant la virgule quand on décode le nombre.

Format IEEE 754 binaire

Pour représenter $S.M.b^E$ (S=signe effectif, M=mantisse effective, E=exposant effectif)

- On travaille en base 2
- Un nombre réel est codé par trois champs de bits. Dans l'ordre, du plus fort poids vers le plus faible poids :
  - le signe codé (S') sur 1 bit (0 pour $S = +1$, 1 pour $S = -1$),
  - l'exposant codé (E') sur e bits,
  - la mantisse codée (M') sur m bits.

S'	E'	M'
1 bit	e bits	m bits

- Pour simplifier les comparaisons entre nombres, l'exposant (qui peut être positif ou négatif) n'est pas codé en complément à deux.
- On stocke en fait l'exposant E auquel on ajoute une constante (exposant biaisé) afin d'obtenir une valeur positive : $E' = E + B$ avec $B = 2^{e-1} - 1$.

IEEE 754 : simple/double précision

- Il existe deux variantes courantes du format :

Précision	Type C/Java	Taille	e	m	B	plage de valeurs approximative
Simple	float	32 bits	8	23	127	10^{-38} à 10^{38}
Double	double	64 bits	11	52	1023	10^{-308} à 10^{308}

- En simple précision, il y a environ 7 chiffres significatifs (base 10).
- En double précision, il y a environ 15 chiffres significatifs (base 10).

IEEE 754 : cas possibles

Bits de E'	Valeur de M'	Type de nombre	Valeur
tous à 1	non nul	NaN	aucune
tous à 1	0	Infini	$+\infty$ ou $-\infty$
des 0 et des 1	quelconque	normalisé	$(-1)^{S'}.(1, M')_2.2^{E'-B}$
tous à 0	non nul	dénormalisé	$(-1)^{S'}.(0, M')_2.2^{1-B}$
tous à 0	0	Zéro	+0 ou -0

- Pour les nombres normalisés, E' peut aller de 1 jusque $2^e - 2$, ce qui signifie que E peut aller de $2 - 2^{e-1}$ à $2^{e-1} - 1$.
- NaN (Not A Number) est utilisé pour indiquer le résultat d'opérations impossibles (0/0, $\infty - \infty$, $\sqrt{-1}$,...).
- Deux représentations différentes de 0 (+0, -0).

IEEE 754 simple précision : récapitulatif

Un nombre de la forme $S.M.2^E$ est codé sur 32 bits, répartis en 3 champs S', E', M' :

champ	S'	E'	M'
nombre de bits	1	8	23

E'	M'	Type de nombre	Valeur
0xFF	$\neq 0$	NaN	aucune
0xFF	0	Infini	$+\infty$ ou $-\infty$
0<E'<0xFF	quelconque	normalisé	$(-1)^{S'}.(1, M')_2.2^{E'-127}$
0x00	$\neq 0$	dénormalisé	$(-1)^{S'}.(0, M')_2.2^{-126}$
0x00	0	Zéro	+0 ou -0

- Pour les nombres normalisés, E' peut aller de 1 jusque 254, ce qui signifie que E peut aller de -126 à 127.
- NaN (Not A Number) est utilisé pour indiquer le résultat d'opérations impossibles (0/0, $\infty - \infty$, $\sqrt{-1}$,...).
- Deux représentations différentes de 0 (+0, -0).

Exemple : codage de 2200,3125

On veut coder 2200,3125 au format IEEE 754 simple précision

1. codage de la partie entière : 2200 = 0b1000'1001'1000
2. codage de la partie fractionnaire : 0,3125
   - 0,3125*2=**0**,625 ⇒ le 1er chiffre après la virgule est 0
   - 0,625*2=**1**,25 ⇒ le 2ème chiffre après la virgule est 1
   - 0,25*2=**0**,5 ⇒ le 3ème chiffre après la virgule est 0
   - 0,5*2=**1**,0 ⇒ le 4ème chiffre après la virgule est 1
   - quand on obtient 0, on arrête la conversion
3. Donc, 2200,3125 = 0b1000'1001'1000,0101
4. On normalise pour obtenir un nombre de la forme 1,...*2^x
   - 0b1000'1001'1000,0101
   - = 0b1000'1001'100,0010'1*2^1
   - = 0b1000'1001'10,0001'01*2^2
   - ...
   - = 0b10,0010'0110'0001'01*2^{10}
   - = 0b1,0001'0011'0000'101*2^{11}

Exemple : codage de 2200,3125 (suite)

On a obtenu 0b1,0001'0011'0000'101*2^{11}

- Le signe est positif, donc le bit de signe sera égal à 0
- L'exposant effectif est 11, on doit lui ajouter le biais égal à 127. on obtient 138 qu'on code en binaire 0b1000'1010.
- De la mantisse, on ne retient que les chiffres après la virgule (il y a toujours un 1 avant la virgule, donc on gagne de la place en ne le stockant pas) : on obtient 0b0001'0011'0000'101
- On recolle les morceaux : signe, puis exposant codé, puis mantisse codée sur 23 bits (on complète avec des 0) : 0b0'1000'1010'0001'0011'0000'1010'0000'000
- On traduit en hexadécimal pour plus de clarté : 0b0100'0101'0000'1001'1000'0101'0000'0000 = 0x4509'8500

Exemple : décodage de 0xC049'0000

- On traduit en binaire, et on découpe en Signe (1 bit), Exposant codé (8 bits et Mantisse codée (23 bits)
- 0xC049'0000 = 0b1100'0000'0100'1001'0000'0000'0000'0000
- = 0b[1]'[1000'0000]'[1001'0010'0000'0000'0000'000]
- l'exposant codé n'est ni 0x00, ni 0xFF. Donc, le nombre est normalisé, il faut remettre 1,... devant la mantisse. On obtient M=1,1001'0010'0000'0000'0000'000
- l'exposant codé est 0b1000'0000, donc 128. Il faut soustraire le biais égal à 127. On obtient 1 qui est l'exposant effectif.
- le bit de signe est 1, donc le nombre est négatif
- On obtient -0b1,1001'0010'0000'0000'0000'000*2^1. Ici, le plus simple est de se débarasser de l'exposant en décalant la virgule. On obtient -0b11,0010'01.
- On fait la somme des puissances de 2 : -0b11,0010'01 = -($2^1 + 2^0 + 2^{-3} + 2^{-6}$) = -(2+1+0,125+0,015 625) = -3,140625

Exemple : décodage de 0x0068'0000

- On traduit en binaire, et on découpe en Signe (1 bit), Exposant codé (8 bits) et Mantisse codée (23 bits)
- 0x0068'0000 = 0b[0]'[0000'0000]'[1101'0000'0000'0000'0000'000]
- l'exposant codé est 0x00. Donc, le nombre est dénormalisé, il faut remettre 0,... devant la mantisse et l'exposant effectif est 2^{-126}. On obtient M=0,1101
- le bit de signe est 0, donc le nombre est positif
- On obtient +0b0,1101*2^{-126}, soit $+(2^{-1} + 2^{-2} + 2^{-4}) * 2^{-126}$ ou encore 0,8125*2^{-126}.
- On ne peut guère aller plus loin sans calculatrice... ($2^{-126} = 1,175.10^{-38}$)

Exemple : codage de 1000.2^{-140}

- Traduction en binaire : 1000.2^{-140} = 0b11'1110'1000.2^{-140}
- Normalisation 0b11'1110'1000.2^{-140} = 0b1,1111'0100'0.2^{-131}
- L'exposant est trop petit, on doit dénormaliser, c'est à dire écrire le nombre sous la forme $S.M.2^{-126}$
- Dénormalisation 0b1,1'1110'1000.2^{-131} = 0b0,0000'1111'1010'00.2^{-126}
- Le nombre est positif ⇒ $S' = 0$
- Le nombre est dénormalisé ⇒ $E' = 0$
- On ne conserve que les chiffres après la virgule (le 0,... est implicite) et on complète avec des 0 pour avoir 23 bits M' = 0b0000'1111'1010'0000'0000'0000'000
- On obtient 0b0'0000'0000'0000'1111'1010'0000'0000'0000'000 soit 0x0007'D000

IEEE 754 : remarques générales

- Le codage ne permet qu'une approximation des nombres réels. Les calculs sont donc faux ($3 * (1/3) \neq 1$) !
- Le résultat d'un calcul peut dépasser la plus grande valeur représentable (overflow) ou la plus petite valeur (underflow).
- Les réels représentés ne sont pas régulièrement espacés sur l'axe réel (plus grande concentration vers 0).
- Différents arrondis possibles (vers 0, $+\infty$, $-\infty$ ou au plus proche).
- Pas de norme pour représenter les flottants en mémoire ou les échanger sur le réseau (endianness).

IEEE 754 : opérations

- addition, soustraction
  On se ramène au même exposant, on additionne/soustrait les mantisses et on renormalise.
- multiplication
  multiplication des signes, multiplication des mantisses, addition des exposants, normalisation
- division
  multiplication des signes, division des mantisses, soustraction des exposants, normalisation

Codage des caractères

Codage des caractères

- Rappel : un ordinateur ne manipule que des nombres.
- Un caractère doit donc être représenté par un nombre.
- Dans certains langages (C/C++), il n'y a aucune différence entre le caractère (par exemple 'A') et le nombre qui code ce caractère (65). Les deux sont interchangeables. 'A' n'est rien d'autre qu'une notation pratique pour 65. On peut écrire 'A'+1 qui représente à la fois le nombre 66 et le caractère 'B'.
- Un codage est un tableau qui associe à chaque caractère un nombre
- Il existe de très nombreux codages des caractères.

EBCDIC

- Extended Binary Coded Decimal Interchange Code : un vieux codage (du temps des cartes perforées) basé sur le BCD et dans lequel les lettres ont des codes qui ne sont pas consécutifs.
- Code sur 8 bits.
- De multiples variantes nationales.

EBCDIC : Exemple d'une variante

(sous réserve d'erreur de transcription)

quartet haut	quartet bas	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0	0x00	NUL	SOH	STX	ETX	SEL	HT	RNL	DEL	GE	SPS	RPT	VT	FF	CR	SO	SI
16	0x10	DLE	DC1	DC2	DC3	RES ENP	NL	BS	POC	CAN	EM	UBS	CU1	IFS	IGS	IRS	IUS
32	0x20	DS	SOS	FS	WUS	BYP INP	LF	ETB	ESC	SA	SFE	SM SW	CSP	MFA	ENQ	ACK	BEL
48	0x30			SYN	IR	PP	TRN	NBS	EOT	SBS	IT	RFF	CU3	DC4	NAK		SUB
64	0x40	SPC	RSP										.	<	(	+	|
80	0x50	&										!	$	*	)	;	¬
96	0x60	-	/									|	,	%	_	>	?
112	0x70										`	:	#	@	'	=	"
128	0x80		a	b	c	d	e	f	g	h	i						±
134	0x90		j	k	l	m	n	o	p	q	r						
160	0xA0		~	s	t	u	v	w	x	y	z						
176	0xB0	^										[	]				
192	0xC0	{	A	B	C	D	E	F	G	H	I	SHY					
208	0xD0	}	J	K	L	M	N	O	P	Q	R						
224	0xE0	\		S	T	U	V	W	X	Y	Z						
240	0xF0	0	1	2	3	4	5	6	7	8	9						EO

Code ASCII

- ASCII=American Standard Code for Information Interchange
- code sur 7 bits
- pas de caractères accentués
- Les 32 premiers caractères sont des caractères de contrôle et ne sont pas imprimables

Table ASCII

quartet haut	quartet bas	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0	0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
16	0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
32	0x20	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	0x70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Pour une lettre donnée, il faut additionner la valeur du quartet de poids fort trouvé sur la ligne au quartet de poids faible trouvé dans la colonne
- Exemples :
  - 'A' a pour code 64+1=65, ou 0x40+0x01=0x41
  - 'h' a pour code 96+8=104, ou 0x60+0x08=0x68

Détail des caractères de contrôle (1)

Code ASCII	Abbréviation	Touche	Nom
0	NUL	Ctrl-@	Null (nul)
1	SOH	Ctrl-A	Start of Header (début d'en-tête)
2	STX	Ctrl-B	Start of Text (début du texte)
3	ETX	**Ctrl-C**	End of Text (fin du texte)
4	EOT	**Ctrl-D**	End of Transmission (fin de transmission)
5	ENQ	Ctrl-E	Enquiry (demande)
6	ACK	Ctrl-F	Acknowledge (accusé de réception)
7	**BEL**	Ctrl-G	Bell (caractère d'appel)
8	**BS**	Ctrl-H	Backspace (espacement arrière)
9	**HT**	Ctrl-I	Horizontal Tab (tabulation horizontale)
10	**LF**	Ctrl-J	Line Feed (saut de ligne)
11	VT	Ctrl-K	Vertical Tab (tabulation verticale)
12	**FF**	Ctrl-L	Form Feed (saut de page)
13	**CR**	Ctrl-M	Carriage Return (retour chariot)
14	SO	Ctrl-N	Shift Out (fin d'extension)
15	SI	Ctrl-O	Shift In (démarrage d'extension)

Détail des caractères de contrôle (1)

Code ASCII	Abbréviation	Touche	Nom
16	DLE	Ctrl-P	Data Link Escape
17	DC1	**Ctrl-Q**	Device Control 1
18	DC2	Ctrl-R	Device Control 2
19	DC3	**Ctrl-S**	Device Control 3
20	DC4	Ctrl-T	Device Control 4
21	NAK	Ctrl-U	Negative Acknowledge (accusé de réception négatif)
22	SYN	Ctrl-V	Synchronous Idle
23	ETB	Ctrl-W	End of Transmission Block (fin du bloc de transmission)
24	CAN	Ctrl-X	Cancel (annulation)
25	EM	Ctrl-Y	End of Medium (fin de support)
26	SUB	**Ctrl-Z**	Substitute (substitution)
27	**ESC**	Ctrl-[	Escape (échappement)
28	FS	Ctrl-\	File Separator (séparateur de fichier)
29	GS	Ctrl-]	Group Separator (séparateur de groupe)
30	RS	Ctrl-^	Record Separator (séparateur d'enregistrement)
31	US	Ctrl-_	Unit Separator (séparateur d'unité)
127	DEL	Ctrl-?	Delete

Bon à savoir

- Ctrl-C : arrêter définitivement (tuer) un programme
- Ctrl-Z : suspendre l'exécution d'un programme (dans le but de reprendre son exécution plus tard)
- Ctrl-S : suspendre le défilement de l'affichage
- Ctrl-Q : reprendre le défilement de l'affichage
- Ctrl-D : terminer la communication
- Ctrl-G, BEL : bip
- Tab, HT : avancer le curseur jusqu'au prochain repère d'alignement sur la ligne
- CR : revenir en début de ligne, sans changer de ligne
- LF : passer à la ligne suivante, sans nécessairement changer de colonne
- Ctrl-L, FF : éjecter la page courante

Représentation de la fin de ligne

- CR sur les très vieux Macintosh (avant Mac OS X)
- LF sous Unix (et donc les nouveaux Macintosh)
- CR LF sous Windows

iso-8859-X (latin-X)

- Extension de l'ASCII où l'on utilise les codes de 128 à 255 pour représenter des caractères supplémentaires (caractères accentués, etc...).
- Comme 128 codes ne suffisent pas pour représenter tous les caractères européens, on a un codage spécifique pour chaque groupe de pays utilisant les mêmes caractères accentués.
- iso-8859-1 (latin-1) pour la France (iso-8859-15 avec le symbole €)
- Il est indispensable de préciser le codage utilisé (en particulier dans les pages web) sinon le contenu affiché sera différent dans un autre pays.

Unicode

- Passage à un code unique. La norme prévoit de coder jusqu'à un peu plus de 1 million de caractères. Actuellement, environ 144 000 caractères sont définis.
- Chaque caractère se voit attribué un code, c'est à dire un nombre qu'on représente en hexadécimal (sur au moins 4 chiffres) avec le préfixe 'U+'. Par exemple, 'A' aura le code U+0041.
- Les 128 premiers codes correspondent au code ASCII (compatibilité ascendante)

Unicode

- Le code d'un caractère peut être représenté en mémoire de différentes manières dont les principales sont UTF-32, UTF-16 et UTF-8 (Unicode Transformation Format).
- Les codes de U+D800 à U+DFFF sont réservés pour le codage UTF-16 (surrogate pair, "paire de seizet d'indirection") et ne peuvent pas représenter un caractère.
- Les codes au delà de U+10FFFF sont également interdits.
- Un même caractère peut avoir plusieurs représentations. Par exemple 'é' peut se représenter comme U+00E9 (le caractère 'é') ou bien U+0065 suivi de U+0301 (le caractère 'e' combiné avec l'accent aigu).

UTF-32

- Un caractère est codé sur 32 bits (4 octets).
- Représentation très simple, mais prend beaucoup de place (taille multipliée par 4 pour un texte en latin-1)
- UTF-32LE signifie qu'on utilise l'ordre Little Endian (octet de poids faible en premier, octet de poids fort en dernier).
- UTF-32BE signifie qu'on utilise l'ordre Big Endian (octet de poids fort en premier, octet de poids faible en dernier).

UTF-16

- Un caractères est codé sur un ou deux mots de 16 bits, comme suit :

Code du caractère en binaire	UTF-16
xxxx'xxxx'xxxx'xxxx	xxxx'xxxx'xxxx'xxxx
000u'uuuu'xxxx'xxxx'xxxx'xxxx	1101'10ww'wwxx'xxxx 1101'11xx'xxxx'xxxx

Avec wwww = uuuuu - 1

- Comme les codes de U+D800 à U+DFFF ne sont pas assignés à un caractère, il ne peut y avoir de confusion entre le codage sur 1 mot et celui sur 2 mots.
- Les caractères les plus usités sont représentés sur 1 seul mot (2 octets).
- uuuuu est compris entre 1 et 10. On lui retire 1 dans le codage (wwww) pour ne pas avoir une valeur inutilisée.

UTF-16 (suite)

- **Attention : l'ordre lexicographique sur les seizets diffère de l'ordre lexicographique des codes de caractères !** De ce fait, il faut éviter d'utiliser l'encodage UTF-16.
- Utilisé par Java
- UTF-16LE signifie qu'on utilise l'ordre Little Endian (octet de poids faible en premier, octet de poids fort en second).
- UTF-16BE signifie qu'on utilise l'ordre Big Endian (octet de poids fort en premier, octet de poids faible en second).

UTF-8

- Un caractère est codé sur 1 à 4 octets. Les caractères ASCII utilisent 1 seul octet (compatibilité ascendante).

Code du caractère en binaire	Nb bits	octet 1	octet 2	octet 3	octet 4
0xxx.xxxx	7	0xxx.xxxx			
0000.0yyy.yyxx.xxxx	11	110y.yyyy	10xx.xxxx		
zzzz.yyyy.yyxx.xxxx	16	1110.zzzz	10yy.yyyy	10xx.xxxx	
000u.uuzz.zzzz.yyyy.yyxx.xxxx	21	1111.0uuu	10zz.zzzz	10yy.yyyy	10xx.xxxx

- Le nombre de bits à 1 au début du premier octet donne le nombre total d'octets utilisés (exception : si le premier octet commence par un 0, il y a exactement un octet)
- Les octets qui suivent le premier octet d'un caractère commencent tous par les bits 10, et contiennent 6 bits du code.
- Les séquences en dehors de cette liste et les séquences qui correspondraient aux codes U+D800 à U+DFFF sont interdites, ainsi que toute séquence qui pourrait être codée de manière plus courte.

UTF-8 : autre présentation

- On écrit le code du caractère en binaire, et on repère dans le tableau le plus petit nombre de bits nécessaires

Nb bits	octet 1	octet 2	octet 3	octet 4
7	0b0'[7 bits]			
11	0b110'[5 bits]	0b10'[6 bits]		
16	0b1110'[4 bits]	0b10'[6 bits]	0b10'[6 bits]	
21	0b1111.0'[3 bits]	0b10'[6 bits]	0b10'[6 bits]	0b10'[6 bits]

- On groupe les bits du code par paquets de 6, en partant des bits de poids faible (à droite)
- Si nécessaire, on ajoute des 0 à gauche (non significatifs) pour obtenir le nombre de bits indiqué.
- On copie les bits au bon endroit dans le codage UTF-8. Le premier octet contient toujours les bits de poids fort, le dernier octet les bits de poids faible.

UTF-8 (suite)

- On notera que l'ordre lexicographique sur les chaînes UTF-8 coïncide avec l'ordre lexicographique sur les octets.
- Attention, avec UTF-8, le nombre d'octets d'une chaîne de caractères n'est plus nécessairement égal au nombre de caractères dans cette chaîne (comme c'était le cas en ASCII ou latin-1) !
- Un caractère en UTF-8 occupe au plus 4 octets, tandis qu'en UTF-32, il occupe toujours 4 octets.

UTF et endianness

- Codages explicitement big-endian : UTF-32BE et UTF-16BE
- Codages explicitement little-endian : UTF-32LE et UTF-16LE
- En l'absence de précision, un premier caractère U+FFFE (BOM, Byte Order Mark) permet de distinguer entre une représentation big/little endian. Ce caractère BOM ne fait pas partie du texte. En l'absence de caractère BOM, on présume que la représentation est big-endian.

Exemple d'encodage UTF-8

- On veut encoder les caractères U+0041, U+03A9, U+8A9E et U+10384
- U+0041 nécessite 7 bits, il est codé tel quel par 0x41
- U+03A9 (0b011'1010'1001) nécessite 11 bits, il est codé sur 2 octets. On regroupe les bits par groupe de 6 : (0b01110'101001)
  - le premier octet commence par 0b110, suivi des 5 bits de poids fort du code soit 0b01110. On obtient 0b110'01110, soit 0xCE.
  - le second octet commence par 0b10, suivi des 6 bits de poids faible du code soit 0b101001. On obtient 0x10'101001, soit 0xA9
  - Donc, U+03A9 est codé en UTF-8 par 0xCE, 0xA9

Exemple d'encodage UTF-8 (suite)

- U+8A9E nécessite 16 bits, il est codé sur 3 octets. On regroupe les bits par 6 : 0b1000'101010'011110
  - le premier octet commence par 0b1110, suivi des 4 bits de poids fort du code soit 0b1000. On obtient 0b1110'1000, soit 0xE8.
  - le deuxième octet commence par 0b10, suivi des 6 bits suivants du code soit 0b101010. On obtient 0x10'101010, soit 0xAA
  - le dernier octet commence par 0b10, suivi des 6 bits de poids faible du code soit 0b011110. On obtient 0x10'011110, soit 0x9E
  - Donc, U+8A9E est codé en UTF-8 par 0xE8, 0xAA, 0x9E

Exemple d'encodage UTF-8 (suite)

- U+10384 nécessite 21 bits, il est codé sur 4 octets. On regroupe les bits par 6 : 0b000'010000'001110'000100
  - le premier octet commence par 0b11110, suivi des 3 bits de poids fort du code soit 0b000. On obtient 0b11110'000, soit 0xF0.
  - le deuxième octet commence par 0b10, suivi des 6 bits suivants du code soit 0b'010000. On obtient 0x10'010000, soit 0x90
  - le troisième octet commence par 0b10, suivi des 6 bits suivants du code soit 0b'001110. On obtient 0x10'001110, soit 0x8E
  - le dernier octet commence par 0b10, suivi des 6 bits de poids faible du code soit 0b000100. On obtient 0x10'000100, soit 0x84
  - Donc, U+10384 est codé en UTF-8 par 0xF0, 0x90, 0x8E, 0x84

Exemple de décodage UTF-8

- Un texte est composé des octets 0x78, 0xCE, 0xBC.
- 0x78 = 0b**0**111'1000. Le bit de poids fort est à 0, il n'y a donc qu'un octet, le code unicode est donc directement U+0078
- 0xCE = 0b**110**'01110. Il y a a 2 bits à 1 avant le premier bit à 0 (en poids fort). Il y a donc 2 octets au total. Le second octet est 0xBC = 0b**10**'111100. On efface les bits imposés par le codage (en gras) et on concatène les bits qui restent. On obtient 0b01110'111100, on traduit en hexadécimal pour obtenir U+03BC

Exemple de décodage UTF-8 (suite)

- Un texte est composé des octets 0xE0,0xA4, 0x95, 0xF0, 0x90, 0x82, 0xB6.
- 0xE0=0b**1110**'0000. Il y a a 3 bits à 1 avant le premier bit à 0 (en poids fort). Il y a donc 3 octets au total. Le deuxième octet est 0xA4 = 0b**10**'100100. Le troisième est 0x95 = 0b**10**'010101. On efface les bits imposés par le codage (en gras) et on concatène les bits qui restent. On obtient 0b0000'100100'010101, on traduit en hexadécimal pour obtenir U+0915
- 0xF0=0b**11110**'000. Il y a a 4 bits à 1 avant le premier bit à 0 (en poids fort). Il y a donc 4 octets au total : 0x90 = 0b**10**'010000, 0x82 = 0b**10**'000010, 0xB6 = 0b**10**'110110. On efface les bits imposés par le codage (en gras) et on concatène les bits qui restent. On obtient 0b000'010000'000010'110110, on traduit en hexadécimal pour obtenir U+100B6.

Les composants d'un ordinateur

Rappel sur les unités

Les préfixes kilo, méga, giga, téra, etc., correspondent aux mêmes multiplicateurs que dans tous les autres domaines : des puissances de 10.

- 1 kilooctet (ko) = 10^3 octets = 1 000 octets
- 1 mégaoctet (Mo) = 10^6 octets = 1 000 ko
- 1 gigaoctet (Go) = 10^9 octets = 1 000 Mo
- 1 téraoctet (To) = 10^{12} octets = 1 000 Go
- 1 pétaoctet (Po) = 10^{15} octets = 1 000 To
- 1 exaoctet (Eo) = 10^{18} octets = 1 000 Po
- 1 zettaoctet (Zo) = 10^{21} octets = 1 000 Eo
- 1 yottaoctet (Yo) = 10^{24} octets = 1 000 Zo

Rappel sur les unités (2)

Depuis 1998, il convient d'utiliser les préfixes kibi pour "kilo binaire", mébi pour "méga binaire", gibi pour "giga binaire", tébi pour "téra binaire", etc.

- 1 kibioctet (Kio) = 2^{10} octets = 1 024 octets
- 1 mébioctet (Mio) = 2^{20} octets = 1 024 Kio
- 1 gibioctet (Gio) = 2^{30} octets = 1 024 Mio
- 1 tébioctet (Tio) = 2^{40} octets = 1 024 Gio
- 1 pébioctet (Pio) = 2^{50} octets = 1 024 Tio
- 1 exbioctet (Eio) = 2^{60} octets = 1 024 Pio
- 1 zebioctet (Zio) = 2^{70} octets = 1 024 Eio
- 1 yobioctet (Yio) = 2^{80} octets = 1 024 Zio

Dans un ordinateur, il y a

- Un boîtier
- Une alimentation
- Une carte mère
- Un processeur
- De la mémoire vive
- Une carte graphique
- Des disques durs
- Des lecteurs de disque
- ...

Mise en garde

- Les évolutions technologiques sont très rapides. Les informations données dans ces transparents deviendront très vite obsolètes !
- Cette présentation n'est évidemment pas exhaustive !
- De nombreux sites permettent d'obtenir des détails

ATTENTION

- Avant toute manipulation dans un ordinateur, il faut s'assurer qu'il est **complètement** éteint. Quand on éteint un PC à partir du système ou par le bouton de la face avant, une partie de la carte mère reste alimentée. Il est indispensable d'éteindre le PC au niveau de l'alimentation, ou en l'absence de bouton arrière de débrancher le PC. Attendre que les condensateurs se déchargent !
- L'électronique d'un ordinateur est sensible aux décharges électrostatiques (ESD). Avant de toucher à un quelconque composant, il faut se décharger en touchant la terre (fiche de terre ou carcasse du boîtier s'il est toujours branché). Il faut utiliser un bracelet antistatique pour rester connecté à la terre pendant toute manipulation.

Le boîtier

Le boîtier

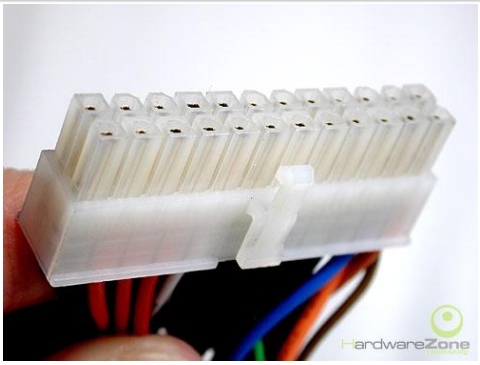
- Il protège mécaniquement les composants internes
- Il doit assurer le refroidissement des composants internes (vérifier la qualité de la ventilation). Un processeur de milieu de gamme consomme environ 100 W. Une carte graphique consomme entre 100 et 400 W (20 W pour les cartes basiques). Cette chaleur doit impérativement être évacuée !
- emballement thermique : plus un transistor est chaud, plus il dissipe d'énergie. Donc il chauffe de plus en plus (rétroaction positive), jusqu'à son éventuelle destruction.
- Il existe différentes tailles et format : tour/desktop, grande/moyenne tour, ...
- Le boîtier doit correspondre à la taille de la carte mère.

L'alimentation

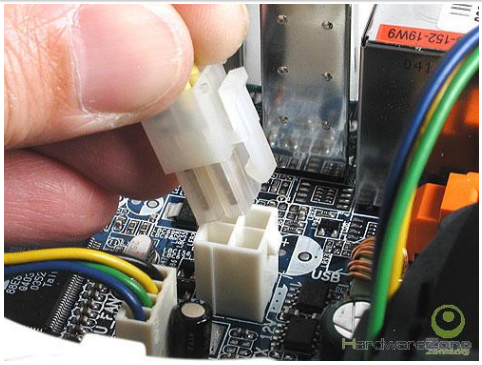
L'alimentation

- Elle fournit le courant continu en 5V, 12V et 3.3V (plusieurs dizaines d'ampères).
- Elle a généralement une puissance de 500W à 800W. Attention aux différences entre puissance crête et puissance stabilisée.
- Son rendement exprimé en pourcentage donne le rapport Puissance fournie à la carte mère/Puissance consommée. Rendements courants : 70%, 80%. On peut avoir des rendements >90%.
- Un PC qui consomme 500W et qui reste allumé 24H/24H consomme environ 4400 kWh par an (soit environ 400€)

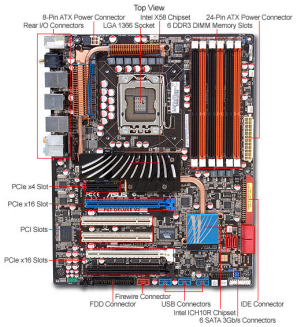
Alimentation carte-mère



Alimentation processeur



La carte mère



La carte mère

Elle abrite

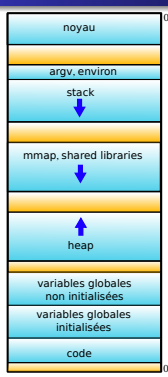
- le processeur
- la mémoire vive (RAM)
- une mémoire morte (ROM) stockant l'UEFI/BIOS (en lecture seule, sauf pour les mises à jour du firmware)
- des connecteurs externes (USB, réseau, son,...)
- des connecteurs internes pour cartes d'extension (carte vidéo, ...)
- des connecteurs internes pour relier les disques durs, DVD, ...
- un chipset qui assure la communication entre ces différents éléments par l'intermédiaire d'un bus.

La mémoire vive (RAM)

La mémoire vive (RAM)

- Différentes capacités par barette (1Go à 8Go par exemple)
- Différentes spécifications incompatibles (tensions, fréquences,...)
  - DDR
  - DDR2
  - DDR3
  - DDR4
- Différentes manière d'associer les barettes (dual ou triple channel)
- Le contenu de la mémoire vive s'efface dès qu'on éteint l'ordinateur !
- À un instant donné, on peut soit écrire en RAM, soit lire en RAM, mais pas les deux en même temps.

La mémoire vive



- La mémoire est un tableau d'octets (un gros tableau puisque qu'avec 8Go, il y a plus de 8 milliards d'octets)
- Chaque case du tableau est identifiée par un indice qu'on appelle adresse.
- La mémoire contient aussi bien les instructions du programme que les variables manipulées.

Swap (mémoire virtuelle)

- Si l'on n'a pas assez de RAM pour tous les programmes que l'on veut faire tourner, on peut utiliser une partie du disque dur pour simuler de la mémoire vive. C'est l'espace de swap (fichier ou partition).
- Attention, un disque dur a un temps d'accès d'environ 10ms, contre environ 10ns pour la RAM.
- Donc, dès qu'on utilise le swap de manière importante la machine est très fortement ralentie !

Le processeur

- C'est lui qui exécute les instructions.
- Le processeur contient une Unité Arithmétique et Logique qui effectue les calculs (circuit combinatoire).
- Des registres (variables internes au processeur)
- Une interface avec la mémoire et les circuits périphériques
- Un décodeur d'instructions qui charge depuis la mémoire les instructions à exécuter, les décode, et pilote l'UAL, les registres et l'interface avec les autres circuits de la carte mère.

Le processeur (dessus, dessous)

Le processeur de profil

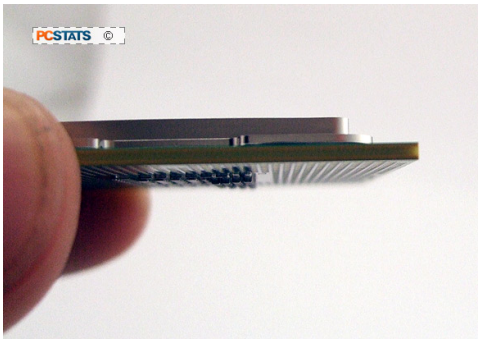
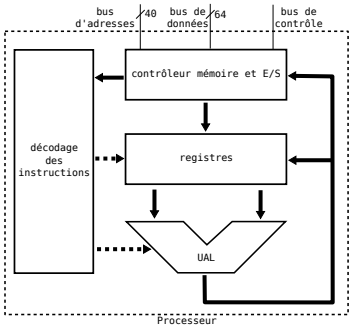


Schéma simplifié d'un processeur



Modèle von Neumann

- Le processeur a un registre spécial IP qui contient l'adresse de la prochaine instruction à exécuter.
- Il charge depuis la RAM la prochaine instruction à l'adresse IP
- Il décode cette instruction et configure ses circuits pour exécuter ces instructions (sélection des registres à utiliser, calcul à effectuer).
- Si l'instruction le précise, il récupère des données depuis la RAM ou il y écrit des données (pas les deux en même temps).
- Il exécute l'instruction
- Il passe à l'instruction suivante.

Le processeur

Il se caractérise par

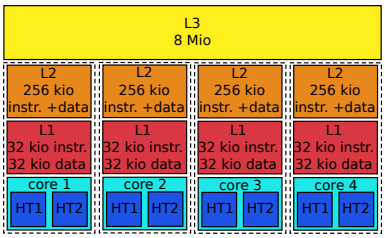
- la taille de ses registres (8, 16, 32, 64 bits)
- l'espace mémoire adressable (taille RAM maximale)
- sa fréquence (2 à 5 GHZ actuellement)
- le nombre de cœurs. Un cœur est un processeur à part entière (UAL, registre, décodage,...). Un processeur avec *n* cœurs peut exécuter *n* instructions en parallèle. Attention cependant, les différents cœurs se font concurrence pour l'accès à la mémoire.
- hyperthreading : présenter au système plusieurs processeurs en ajoutant un minimum de circuits (registres,...)
- sa mémoire cache

La puissance du processeur dépend de très nombreux facteurs.

La mémoire cache

- Le processeur est beaucoup plus rapide que la mémoire vive (RAM).
- Il existe des mémoires rapides (statiques, utilisant des bascules) mais qui coûtent plus cher que la RAM habituelle (mémoire dynamique, 1 transistor/condensateur par bit).
- Pour optimiser les performances en limitant le coût, on utilise une mémoire cache (rapide) qui contient une copie des données présentes en RAM (lente). Tant que les données sont présentes dans le cache, le processeur va vite. S'il manque une donnée, le processeur doit attendre que la RAM la lui fournisse.
- Il existe différent niveaux de cache. Ex : L1, 32Ko à 90Go/s, L2 256Ko à 35Go/s, L3 8Mo à 25Go/s (pour comparaison RAM : quelques Go à 12Go/s).

Exemple de niveaux de caches

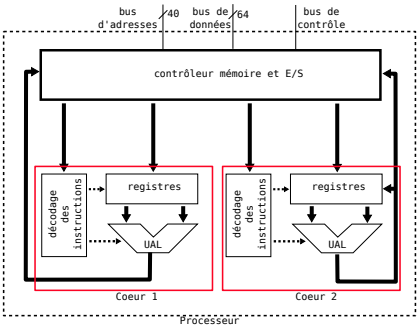


Parallèlisme

Une machine est parallèle si elle sait effectuer plusieurs tâches en même temps. Même le PC de bureau est devenu parallèle.
Il existe différents types de parallélisme :

- SIMD (Single Instruction, Multiple Data)
  On applique la même opération à plusieurs données (ex : instructions MMX)
- MIMD (Multiple Instructions, Multiple Data)
  Des opérations s'exécutent en parallèle sur des données différentes (ex : multi-processeurs (SMP), multi-coeurs)
- ...

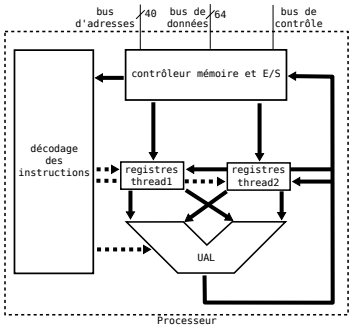
Processeur multi-cœurs



Processeur multi-cœurs

- Un processeur multi-cœurs contient en fait sur le même circuit intégré plusieurs processeurs appelés cœurs (cores) qui sont complètement indépendants.
- Les différents cœurs exécutent simultanément des instructions différentes (parallélisme).
- Les différents cœurs partagent le même accès à la mémoire (goulet d'étranglement).

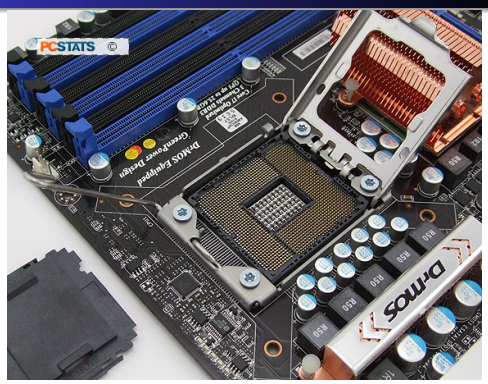
Hyperthreading



Hyperthreading

- Dans un processeur avec hyperthreading, on simule deux processeurs logiques en dupliquant les registres, mais en partageant le reste des circuits, en particulier la partie exécution des instructions.
- Chaque processeur logique a ses propres registres et donc, pour le programmeur, est parfaitement indépendant.
- Quand un des processeurs logiques est bloqué (par exemple parce qu'il attend des données en provenance de la mémoire), on en profite pour exécuter des instructions de l'autre processeur logique.
- De manière générale, on intercale les instructions des processeurs logiques dans le pipeline d'exécution (sorte de chaîne d'assemblage), de manière à maximiser le nombre d'instructions exécutées.
- On peut combiner hyperthreading et multi-cœurs. Chaque cœur gère deux processeurs logiques avec l'hyperthreading.

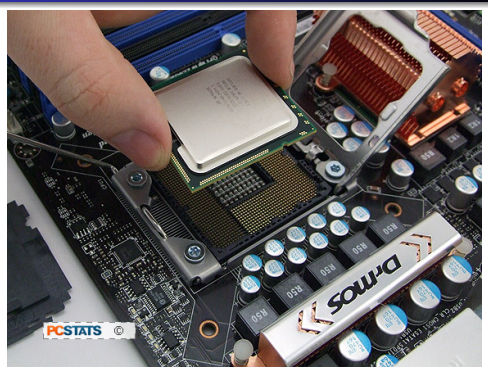
Le socket



Le socket

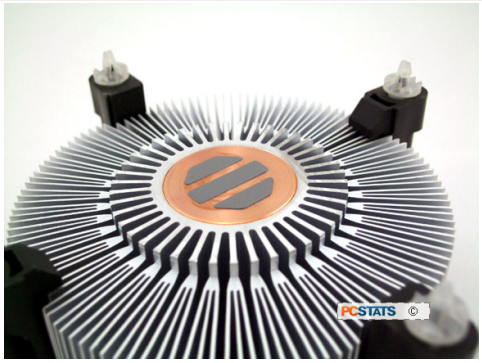
- C'est le support du processeur qui contient les contacts qui permettent au processeur de dialoguer avec la carte mère.
- Différents types de sockets, absolument incompatibles !
- Le processeur doit correspondre au socket de la carte mère.
- Plus de 1000 contacts sur les processeurs récents.

La pose du processeur

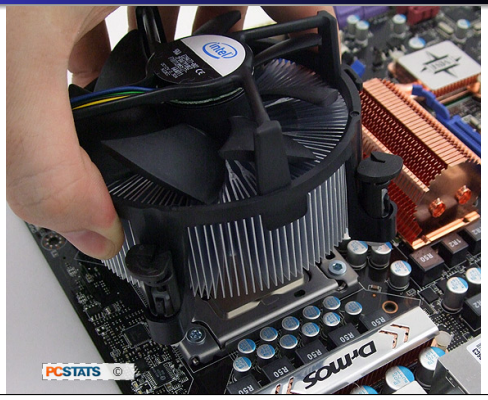


Le verrouillage

Le ventilateur (vu de dessous, avec la pâte thermique)



La pose du ventilateur



Le ventilateur

- La pâte thermique permet de transférer la chaleur du processeur vers le ventilateur
- Le ventilateur est INDISPENSABLE !
- Attention à la poussière !

Bus

- Un bus est un ensemble de fils qui permet de transmettre des informations. Un bus respecte une norme qui précise combien de fils, quelles tensions, quelle fréquences, ... sont utilisés. La norme précise aussi le type de connecteur.
- Un bus parallèle utilise $n+1$ fils pour transmettre n bits simultanément sur de courtes distances (sur de longues distances, les bits n'arrivent plus en même temps).
- Un bus série utilise au moins 2 fils pour transmettre les bits séquentiellement, les uns à la suite des autres.

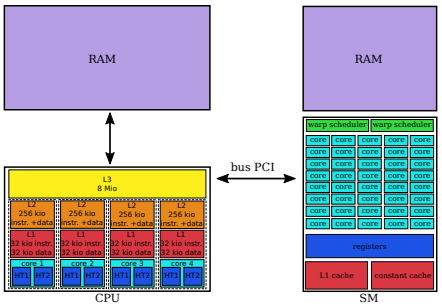
Connecteurs pour cartes d'extension

- PCI Express (PCIe) : bus série inspiré des réseaux.
- Un port PCIe $\times n$ (avec $n \in \{1, 2, 4, 8, 16\}$) contient n lignes de communications bidirectionnelles.
- Le débit d'une ligne dépend de la version de la norme :
  - PCIe 1.x : 0,25 Go/s
  - PCIe 2.x : 0,5 Go/s
  - PCIe 3.x : 0,985 Go/s
  - PCIe 4.x : 1,969 Go/s
  - PCIe 5.x : 3,938 Go/s
- Le débit est multiplié par le nombre de lignes. Donc, un port PCIe à la norme PCIe 4.x aura un débit de 31,5 Go/s.
- Un port PCIe $\times 16$ peut fournir 75 W à la carte d'extension.

Carte graphique

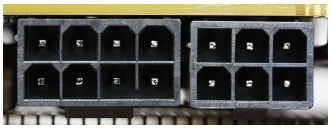
- Elle mémorise le tableau (2D) des pixels à afficher à l'écran.
- Elle contient des milliers de processeurs spécialisés (GPU : Graphical Processing Unit) et assez simples qui permettent d'accélérer les calculs 3D ou les opérations 2D.
- En simplifiant, ces processeurs effectuent la même opération, chacun sur des pixels différents (calculs de type vectoriel).
- Elle possède sa propre mémoire (quelques Go)
- Il est possible de coupler des cartes graphiques.
- Elle consomme beaucoup d'énergie (quelques centaines de watts) et nécessite des connecteurs électriques additionnels (connecteur 6 broches pour 75 W, 8 broches pour 150 W).
- Attention au ventilateur de la carte (bruit, poussière) !

CPU vs GPU



La carte graphique

La carte graphique (alimentation additionnelle



Les sorties vidéos

- VGA

- DVI

- Display Port

- HDMI

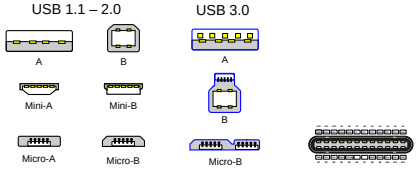
Le panneau arrière de la carte mère

Les connecteurs arrières

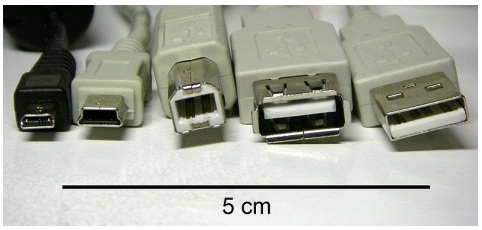
- USB (Universal Serial Bus)
- RJ45 réseau (10,100,1000 Mbit/s, 2,5 Gb/s, 10 Gb/s)
- vidéo
- son
- port série RS232, port parallèle DB25 (obsolètes)

USB

- USB (Universal Serial Bus)
  - branchement à chaud
  - USB1 : 1,5 Mbit/s à 12 Mbit/s
  - USB2 : jusque 480 Mbit/s
  - USB3.0 : jusque 5 Gbit/s
  - USB3.2 : jusque 20 Gbit/s (connecteur USB-C)
- de multiples connecteurs, un côté machine (A), un côté périphérique (B)



Prises USB

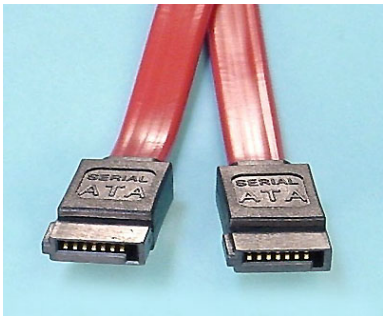


Le disque dur

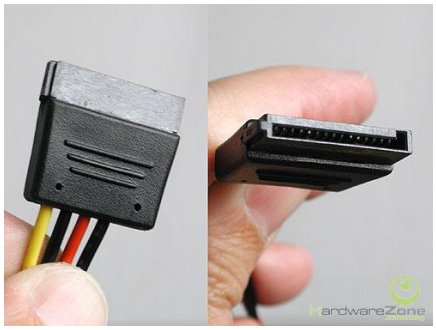
Le disque dur

- Taille de l'ordre du To
- Son contenu ne s'efface pas quand on éteint l'ordinateur
- Deux types
  - disque magnétique : lent (temps d'accès de plusieurs ms), peu cher, grande capacité, fragile mécaniquement
  - SSD (Solid State Disk) : pas de pièces mécaniques, plus rapide, nombre d'écritures limité, plus cher, taille plus limitée
- Branchement à la carte mère via un connecteur SATA
- Deux formats de disques : 3,5" et 2,5"

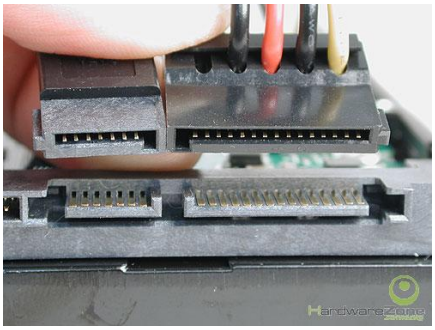
Connecteur SATA (data)



Connecteur SATA (power)



Connecter un disque SATA



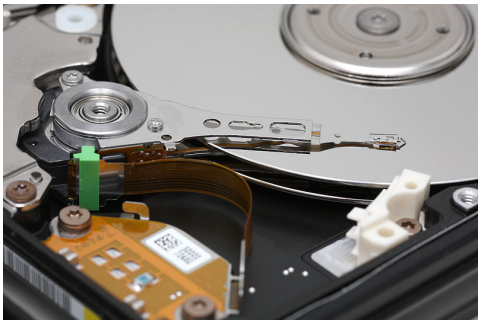
Disque M.2

- Les disques SSD M.2 doivent être branchés sur un connecteur spécifique de la carte mère

Disque dur magnétique

- Un disque contient plusieurs plateaux et têtes de lecture. Chaque plateau est découpé en pistes, chaque piste est découpée en secteurs lors du formatage bas niveau. Un secteur contient en général 512 octets (éventuellement 1024 ou 2048).
- Un secteur peut être ou devenir défectueux. Les données qu'il contenait sont alors perdues. La technologie SMART permet parfois d'anticiper ces pannes. L'information est alors déplacée à un autre endroit du disque.
- Le disque est sensible au magnétisme (ne pas le placer à côté d'un aimant (haut-parleur,...)).
- Il existe différents formats de disques dur (3.5", 2.5")

Le disque dur



Le système de fichiers

- Il est mis en place lors du formatage (haut niveau) du disque
- Il se charge d'allouer les blocs nécessaires à la sauvegarde de chaque fichier.
- Il se charge de mémoriser le nom des fichiers et permet d'organiser ces fichiers en répertoires.
- Le système de fichiers occupe une certaine place sur le disque dur (quelques %)
- Un fichier est fragmenté lorsque ses données ne sont pas stockées dans des blocs consécutifs. La fragmentation ralentit l'accès au fichier. Le système de fichiers essaye de limiter cette fragmentation.

Les partitions

- Un disque dur peut être découpé en partitions (qui simulent des disques indépendants).
- Les partitions permettent de séparer les différents systèmes d'exploitation, ou de séparer les données du système.
- Sur un PC, on peut avoir
  - Une table de partitions DOS (obsolète) : au maximum 4 partitions primaires. L'une de ces partitions peut être une partition étendue qui sera découpée en partitions logiques. Dans certains cas, le système d'exploitation doit se trouver sur une partition primaire.
  - Une table de partition GPT (introduit avec l'UEFI) : on peut avoir jusque 128 partitions !

Le lecteur de disques

Lecteurs de disques

- CDROM (700 Mo)
- DVD (4,7 Go par couche)
- Blu-ray (25 Go par couche)

L'UEFI / le BIOS

Il réside en ROM

- il contient le programme qui permet de charger le système d'exploitation
- il contient les fonctions de base qui permettent de gérer le matériel
- il propose une interface (setup) qui permet de
  - configurer le matériel
  - choisir sur quels périphériques on cherche le système d'exploitation (dvd, clef USB, réseau, disque dur,...)
  - préciser comment la machine peut s'allumer (que faire après une coupure de courant, allumage à une heure donnée, allumage par un périphérique (ex : WOL (wake on lan))
  - protéger la machine par un mot de passe
- il peut être mis à jour (flashage) car la ROM est en réalité une E^2PROM. Gare si l'opération échoue (panne de courant au mauvais moment,...) !

Le boot de la machine (BIOS)

À l'allumage ou après un reset,

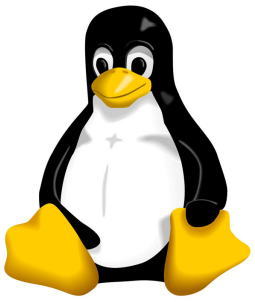
- le processeur exécute les instructions situées à une adresse fixe (nécessairement en ROM : le BIOS)
- le programme en ROM va initialiser les circuits de la carte mère
- le BIOS va réaliser certains tests
- le BIOS va rechercher un périphérique à partir duquel il peut charger le système d'exploitation
- Si le système est sur un disque dur, le BIOS va charger un petit programme stocké dans les premiers secteurs du disque dur. Ce programme va charger le noyau du système à partir de la partition "active"
- le noyau du système prend le contrôle et charge les pilotes de périphériques et les programmes systèmes

Le boot de la machine (UEFI)

À l'allumage ou après un reset,

- le processeur exécute les instructions situées à une adresse fixe (nécessairement en ROM : l'UEFI)
- le programme UEFI va initialiser les circuits de la carte mère
- il va réaliser certains tests
- il va rechercher une partition EFI System, et dans cette partition, il va lancer le progamme qui est indiqué dans la configuration en NVRAM (mémoire non volatile) charger le système d'exploitation
- Ce programme va charger le noyau du système.
- le noyau du système prend le contrôle et charge les pilotes de périphériques et les programmes systèmes

Le système d'exploitation



La virtualisation

- Elle permet de faire tourner plusieurs systèmes d'exploitation simultanément (de la même manière qu'on fait tourner plusieurs programmes en parallèle).
- Il y a un système principal qui contrôle l'utilisation des ressources de la machine (CPU, mémoire, disque, vidéo, ...) et qui fournit aux autres systèmes des ordinateurs virtuels (i.e. simulés).
- Intéressant pour héberger plusieurs serveurs sur une seule machine, ou simplement pour tester des systèmes.

Architecture d'un ordinateur et fonctionnement du système d'exploitation

Avertissement

- La présentation qui suit ignore volontairement de nombreux détails.
- Cette simplification a pour but de se focaliser sur les points essentiels, pour aboutir à une première compréhension du fonctionnement global d'une machine.

Le processeur

Le processeur contient essentiellement

- des registres : ce sont les mémoires les plus rapides de la machine, en nombre limité (quelques dizaines). Tous les calculs s'effectuent sur des valeurs stockées dans les registres.
- une unité arithmétique et logique (UAL ou ALU en anglais) : elle est chargée d'effectuer les calculs
- un circuit chargé de décoder et d'exécuter les instructions : il pilote la sélection des registres et indique à l'UAL quelle opération effectuer.
- un contrôleur mémoire et E/S : il pilote l'accès à la mémoire et aux périphériques

Le processeur

- Le processeur est connecté au reste de la machine par l'intermédiaire
  - d'un bus d'adresses (de 36 à 40 bits, ce qui permet d'accéder entre 64 Gio et 1 Tio de mémoire) : il indique à quelle position (adresse) de la mémoire on veut lire/écrire un mot (suite de 8, 16, 32 ou 64 bits)
  - d'un bus de données (64 bits actuellement) : il contient le mot que l'on est en train de lire/écrire en mémoire
  - d'un bus de contrôle : ce bus transmet divers signaux pour indiquer par exemple si l'on veut lire ou écrire en mémoire (R/W), permettre d'interrompre le processeur, etc...

Les registres

- un pointeur d'instruction (IP) : contient l'adresse de la prochaine instruction à exécuter
- un registre d'état : ses bits (flags) donnent des informations sur la dernière opération effectuée. Par exemple :
  - flag Zéro : indique si le dernier résultat obtenu était nul
  - flag Signe : indique si le dernier résultat obtenu était négatif
  - flag Carry (retenue) : indique si la dernière addition a généré une retenue
  - ...

  Le registre d'état est utilisé pour déterminer le résultat des différents tests (équivalent du *if*).
- un pointeur de pile (SP) : indique la position du dernier élément de la pile gérée par le processeur
- les registres généraux : servant essentiellement pour faire les différents calculs

Exécution d'une instruction

- Le processeur charge les octets qui représentent la prochaine instruction à exécuter à partir de la position désignée par le pointeur d'instructions (IP)
- Il ajoute à IP la taille de l'instruction en cours pour pointer sur la prochaine instruction.
- Il décode l'instruction et pilote ensuite le contrôleur mémoire, la sélection des registres et l'UAL pour l'exécuter.

Autres unités de calcul

Le processeur contient aussi d'autres unités de calcul et les registres dont elles ont besoin :

- l'unité de calcul en virgule flottante, spécialisé dans les calculs sur les réels (dont les calculs tels que exp, log, sin, ...)
- unité de calcul vectorielle (instructions MMX, SSE, ...), accélérant de traitement de données vectorielles telles que matrices, images, ... Ces unités sont utiles quand on applique le même traitement à des données différentes (data-parallélisme).

Schéma simplifié d'un ordinateur

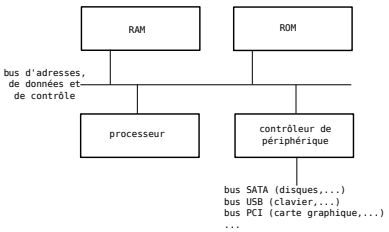


Schéma simplifié d'un ordinateur

- La ROM est une mémoire en lecture seule (Read Only Memory) qui ne s'efface pas à l'extinction de l'ordinateur. Elle contient le programme qui permet de démarrer l'ordinateur (BIOS ou UEFI).
- La RAM est la mémoire vive (Random Access Memory), en lecture/écriture. Elle s'efface quand on éteint l'ordinateur.
- Le processeur a normalement le contrôle du bus et peut lire/écrire librement en RAM, et seulement lire en ROM. Il peut aussi transférer des données aux périphériques.
- Certains périphériques peuvent prendre le contrôle du bus et lire/écrire directement en mémoire (DMA, Direct Memory Access) pour décharger le processeur d'un travail de copie de zones mémoires.

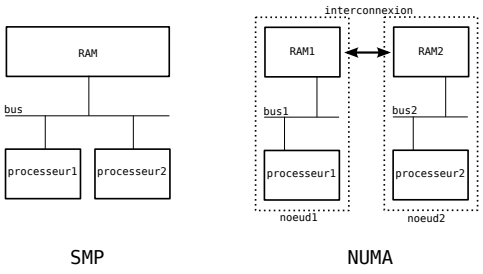
Alignement des variables

- Un processeur 32 bits (par exemple) aura un bus de données de 32 bits soit 4 octets.
- La RAM est alors aussi organisée pour pouvoir fournir 4 octets d'un seul coup.
- Le processeur peut donc lire en une seule fois
  - les informations de 32 bits qui sont situées à des adresses qui sont des multiples de 4,
  - les informations de 16 bits qui sont situées à des adresses qui sont des multiples de 2,
  - les informations de 8 bits qui sont situées à des adresses qui sont des multiples de 1.
- Quand ces conditions ne sont pas respectées, on a besoin de 2 accès mémoire pour lire la donnée. C'est plus lent.
- Donc, pour aller vite, il faut que les informations élémentaires soit placées à une adresse qui doit être un multiple de la bonne valeur. C'est l'**alignement**.
- Le compilateur insère des octets non utilisés dans les structures pour obtenir le bon alignement.

Architectures SMP et NUMA

- Sur une même carte-mère, on peut avoir plusieurs processeurs (donc plusieurs circuits intégrés avec leur propre radiateur). Chacun de ces processeurs peut bien sûr avoir plusieurs cœurs avec éventuellement de l'hyperthreading.
- Si les processeurs accèdent tous de la même manière à la mémoire et aux périphériques, le système est dit SMP (Symmetric Multiprocessing).
- Si chaque processeur a un accès privilégié à une partie de la mémoire, le système est dit NUMA (Non Uniform Memory Access). L'accès d'un processeur à la mémoire qui lui est rattachée est plus rapide, mais il peut néanmoins accéder (plus lentement) au reste de la mémoire.

Architectures SMP et NUMA



Processeurs graphiques

- Les cartes graphiques ont des centaines de (petits) processeurs, qui sont spécialisés pour effectuer des traitements vectoriels (une même opération est appliquée à des données différentes).
- Ces cartes graphiques ont leur propre mémoire.
- Ces processeurs permettent de traiter efficacement les opérations d'affichage (traitement des textures, etc...)
- Les programmes pour utiliser ces processeurs graphiques sont spécifiques.

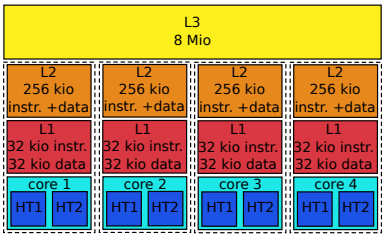
Mémoire cache

- Le processeur est plus rapide que la mémoire vive centrale (RAM), environ d'un facteur 10.
- Quand il a besoin d'obtenir une donnée de la mémoire, le processeur est forcé d'attendre que la mémoire transmette cette information. Pendant ce temps d'attente, il n'exécute pas d'instruction.
- La solution est d'avoir une mémoire plus rapide intégrée au processeur, et qui garde une copie des informations de la RAM. C'est la **mémoire cache**.
- Quand on accède à une donnée, on a de fortes chances d'accéder ensuite à la donnée qui suit en mémoire (principe de localité). On charge donc des blocs d'octets (lignes) dans le cache pour améliorer l'efficacité.

Niveaux de caches

- Idéalement, on voudrait avoir un gros cache et très rapide.
- Malheureusement, plus un cache est gros, plus il est lent (le cache est un tableau associatif qui associe une donnée à une adresse).
- On utilise donc différents niveaux de cache, du plus rapide (et petit) au plus gros (et lent).
- Exemple :
  - niveau 1 (level 1, L1) : 32 kio pour les instructions et 32 kio pour les données, vitesse de 90 Go/s.
  - niveau 2 (L2) : 256 kio, vitesse de 35 Go/s
  - niveau 3 (L3) : 8 Mio, vitesse de 25 Go/s
  - RAM : quelques Gio, vitesse de 12 Go/s

Exemple de niveaux de caches



Cache miss et cohérence des caches

- Quand un cache d'un certain niveau ne possède pas une donnée (cache miss), il la demande au niveau supérieur, en remontant si nécessaire jusqu'à la RAM. Pendant ce temps là, le processeur attend.
- Quand des cœurs ont des caches séparés, il faut s'assurer qu'une donnée écrite dans un cache sera correctement transmise (aussi vite que possible) aux autres caches de manière à ce que chaque processeur travaille sur la même valeur des variables. On dit alors que le cache est cohérent. Néanmoins, si on veut être sûr que la donnée a été transmise aux autres caches, il faut utiliser une instruction spéciale (memory barrier).
- Certains processeurs ne gère pas cette cohérence des caches, et le programmeur doit en tenir compte.

Fonctionnement simplifié du cache

- Quand il n'y plus de place dans un cache, il faut effacer les données les plus anciennes.
- Quand une donnée est écrite dans le cache de niveau 1, elle peut être :
  - directement écrite dans les caches de niveaux supérieurs (write through),
  - copiée en RAM uniquement avant de la faire disparaître du cache L3 (write back).