Notions d'Architecture des Ordinateurs (A.O.)

Université de La Réunion
Cours préparé par Dr Philippe Martin

http://www.phmartin.info/cours/ao/
(→ supports de cours, TDs, exemples de QCM, corrigés, ...)

L'**objectif** de ce cours est de faire connaître à ses étudiants
- les composants et principes majeurs de fonctionnement d'un ordinateur ou d'un réseau d'ordinateurs, depuis l'électronique (transistors, ...) jusqu'à la programmation,
- les critères principaux d'évaluation de performance (capacité, rapidité, …),
- les termes français et anglais relatifs à tout ceci.

Pile des objets/processus nécessaires au traitement de l'information :

- programmes applicatifs
- systèmes d'exploitation et réseaux (partie 5 du cours)
- langages, interpréteurs et compilateurs (partie 2)
- microprocesseurs (parties 2 et 4)
- circuits (partie 3) et encodage de l'information (partie 1)
- portes logiques (parties 3 et 4)
- diodes et transistors (partie 4)

Contenu:

Partie 1. Introduction, 1) Définitions générales, 2) Historique, 3) Numération (site offrant des informations complémentaires sur cette Partie 1)
Partie 2. Architecture de Von Neuman: CPU, bus, mémoires, …
Partie 3. Circuits logiques (combinatoires ; plus ceux séquentiels)
Partie 4. Composants électroniques; parallélisme; entrées/sorties

Une bonne part des graphiques de ce cours reprennent ceux du livre "Architecture et technologie des ordinateurs" de P. Zanella, Y. Ligier et E. Lazard (616 pages ; ISBN: 978-2-10-078459-2 ; dernière édition : 2018).
La plupart des définitions de ce cours sont dérivées de ce livre et, souvent, centralisent des informations éparses dans ce livre (afin de simplifier l'accès à ces informations et d'accélérer la compréhension des notions définies).
La bibliothèque universitaire contient plusieurs livres sur l'Architecture des Ordinateurs, e.g. quelques uns de Philippe Darche.
Ce cours est un préalable nécessaire à de nombreux autres cours
→ il nécessite de comprendre beaucoup de termes (ceux en caractères gras) et les techniques de résolution de problèmes vus en cours et TD.

Organisation du cours

Ce cours est *organisé* comme une *suite intuitive* de définitions
→ **plus de précision et facilite la mémorisation**
→ elles doivent être relues périodiquement pour être bien comprises;
(les évaluations nécessitent une bonne compréhension,
toute tentative de "par-cœur" est contre-productive);
→ ce cours prépare à de nombreux autres cours et n'a lui-même **pas d'autres pré-requis que des notions mathématiques élémentaires (cf. page 3)**.

Lire les liens pointés dans ce cours est important (et cela sera donc aussi testé dans la mesure du possible) mais, au moins pour les évaluations, vous n'avez pas besoin de lire d'autres documents (pages Web, livres, cours, …) que ceux pointés dans ce cours.

Vous (étudiant) devez venir à chaque TD **en ayant étudié le cours** *(ceci sera testé)*, **en ayant préparé le TD** (i.e., avoir tenté de résoudre les exercices), et avec une version (papier ou sur ordinateur) du cours et du TD. Si ce n'est pas le cas, l'intervenant pourra vous exclure du TD.

Pour chaque définition ou technique que vous ne comprenez pas, vous devez
- tout d'abord chercher à comprendre par vous-même (via un dictionnaire, le Web ou d'autres passages du cours) puis en posant des questions (dès que possible mais pas par e-mail ; pour des questions sur un CM, faites-le avant le début de la "petite évaluation de début de TD" sur ce CM) ;
- signaler toute ambiguïté ou autre problème dans le cours (→ points bonus).

Ceci fait partie du **rôle de l'étudiant** car, souvent, un enseignant ne peut deviner ce que l'étudiant ne comprend pas si celui-ci ne l'exprime pas.

Le **rôle d'un enseignant** est
i) de vous aider à apprendre (→ donc de vous fournir un cours précis et structuré et de répondre à vos questions),
ii) de vérifier que vous apprenez suffisamment, et
iii) de vous entraîner à répondre – ou, plus généralement, exposer des informations – de manière précise, organisée et convaincante.

Plus de détails (et leurs fondements) sont dans la page
http://www.phmartin.info/cours/devoirsEnseignantEtudiant.html

Travail personnel

Chaque heure de cours ou de TD doit entraîner entre 1/2 heure et 4h de travail travail personnel (4h selon la fin de ce document de l'UFR ST pour les licences). Ceci inclut

- la relecture du CM relatif au cours (pour optimiser la compréhension et la rétention, relire le soir même, puis 2 ou 3 jours après, puis la semaine suivante) ; **rechercher dans un dictionnaire chaque mot non compris** ;
- **la préparation des TDs, la compréhension des corrigés, l'entraînement aux QCMs**, l'entraînement à refaire des questions de TD en temps limité.
  Note : plus il y a de concepts composants/reliés à un concept compris et mémorisés, plus ce dernier concept est facilement compris et donc mémorisé
  -> lisez tout ce que l'on vous demande de lire ; ne sautez pas de parties.

Vous devez bien-sûr maîtriser les notions mathématiques élémentaires (-> si besoin revoir ces notions sur le Web, e.g. via Wikipedia)**, par exemple** :

- $x^a * x^b = x^{a+b}$ et donc aussi $x^a / x^b = x^{a-b}$
- $\log_2(2^x) = 2^{\log_2(x)} = x$ $\quad$ $\text{sqrt}(x) = x^{1/2}$
- la/les règle(s) de proportionnalité (règle de 3, ...) ; testez-vous sur les problèmes listés dans https://fr.wikipedia.org/wiki/Règle_de_trois.

Question de présence et d'évaluation (notée donc ; effectuez là maintenant en allant sur la page Moodle du cours **puis, dans cette page Moodle,** en cliquant sur "ao1wooclap" *(si Wooclap à un problème, "ao1cmQ1" sera mis à disposition).* **Quel est le résultat de** $\mathbf{((\text{sqrt}(\log_2(2^{16})) / 2^{-1}) / 2^0) * 2^{3*-2}}$ **?**

Notes / rappels :

- ***chaque question Wooclap vous permet d'évaluer tout de suite si vous n'avez pas compris une notion, n'êtes pas à niveau, ... ; votre succès ou pas à une question a extrêmement peu d'influence sur votre note finale***
- *dans ce cours (CMs, TDs, tests), les calculatrices sont inutiles et interdites !*
- *sauf cas exceptionnel (voir avec moi au prochain TD), toute absence non administrativement justifiée (auprès de moi et de la scolarité, avec un certificat médical en pièce jointe) entraînera des points négatifs sur une note de grosse évaluation (les CMs et les TDs sont officiellement obligatoires).*

1. Définitions générales

E.g. (latin: exempli gratia): "par exemple".

I.e. (latin: id est): "c'est-à-dire".

***** : "multiplie" (dans ce cours, "multiplie" sera représenté par "*" ou ".", et "*" représentera toujours "multiplie").

Information: données numériques (e.g., 314) ou symboliques (e.g., Vrai, Très chaud, Rouge, "Un chat est sur une table", ...); e.g., données multimédia.

Numérisation: représentation de données avec des *nombres* (i.e. des *codes* numériques ou non numériques).

Méthodes de numération: méthodes pour écrire les nombres.

Informatique [I.T.: Information Technology] (terme issu de la contraction de "information" et "automatique"): science du traitement de l'information.

Ordinateur [computer]: machine capable d'exécuter des programmes.

Programme: suite d'instructions arithmétiques ou logiques ou de contrôle (ces dernières définissent l'ordre d'exécution d'autres instructions).

Logiciel [software]: programme ou ensemble de programmes.

Système informatique [information system]: système aidant la réalisation de tâches/décisions et composé de matériels [hardware] (ordinateur, périphériques, ...), logiciels [software], information (données ou bien connaissances) et personnes.

1. Définitions générales

Chiffres [digit]: symboles formant les nombres,
e.g., 28 est formé des symboles 2 et 8.

Représentation en base 10 (décimale):
avec 10 symboles différents (e.g., chiffres de 0 à 9).

Représentation en base 2 (binaire): avec 2 symboles différents
(e.g., 0 et 1, vrai et faux, ouvert et fermé, 0 volt et 5 volts).
Avantages:
- de nombreux systèmes ont deux états (en électronique, optique, ...)
- facile à distinguer et à manipuler.

Bit (contraction de "binary digit"): 0 ou 1. N'utilisez pas l'abréviation "b"
car "octet" [byte] (8 bits) est abrévié par "o", "B" ou "b" (d'où l'ambiguïté).
Dans ce cours, "bit" n'est jamais abrévié.

Octet [byte]: ensemble de 8 bits.
Dans ce cours, l'abréviation "B" n'est pas utilisée.

Implémenter (une idée/technique/méthode):
réaliser un artefact (machine ou programme)
mettant en œuvre cette idée/technique/méthode.

Incrémenter (d'un nombre N une variable ou un compteur):
ajouter N à cette variable ou à ce compteur.

2. Historique

Vous n'avez pas à vous souvenir des dates exactes (ce serait du "par cœur") mais vous devez avoir compris ce à quoi chaque terme réfère.

- Préhistoire: calcul avec des cailloux (latin: calculi) ou des doigts (latin: digiti)

- Vers -500, premiers **outils de calcul manuels** :
  abaques, bouliers, règles à calcul

- Diverses bases:
  10 (doigts), 12 (heures), 60 (temps, angles), 7 (musique), …
  Page à lire : https://fr.wikipedia.org/wiki/Système_sexagésimal

- **Numération positionnelle**:
  chaque chiffre a un poids dans une base, e.g.,
  $1803 = 1*10^3 + 8*10^2 + 0*10^1 + 3*10^0$
  $= 1*1000 + 8*100 + 0*10 + 3*1$
  → nécessite le chiffre 0 (inventé/connu par certains érudits ou commerçants indiens et arabes 2 siècles après J.-C.; adopté par certains Européens au 11ème siècle)

- Numération romaine, e.g., $MDCCCLXXIII_{romain} = 1873_{10}$

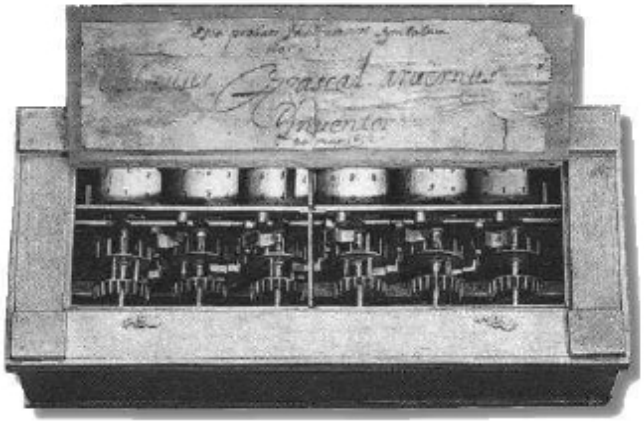
2. Historique

- 1614: tables de logarithmes de John Neper

- 1617: bâtons de Neper
  (sorte de règle à calcul)

- 1632 : Invention de la règle à calcul (Oughtred)
  et Francis Bacon invente le premier codage de l'alphabet

- 1642: Blaise Pascal créé la Pascaline qui additionne et soustraie des nombres de 6 chiffres
  (roues codeuses + notion de décalage des retenues)



- 1673: Leibniz étend la Pascaline pour les 4 opérateurs (+, -,*,/).
  Il invente aussi le système binaire moderne ainsi que le calcul différentiel et intégral

- 1805: Joseph Jacquard utilise des cartons troués (1ères cartes perforées) pour "programmer" les métiers à tisser

2. Historique

- 1833: Charles Babbage débute la conception théorique de la **Machine de Babbage (**→ 4 opérations arithmétiques de base) puis de la **Machine Analytique**, 1er modèle de calculateur programmable :
  - 4 parties: mémoire, unité de calcul,
    entrée (lecteur de cartes perforées), sortie (perforation)
  - 4 opérations arithmétiques, test et branchement conditionnel

  → en 1840, Ada Augusta Byron (fille du poète Lord Byron) invente et écrit les premières itérations successives (→ algorithme ; 1ers programmes informatiques non exécutés)

- 1854: Georges Boole publie l'algèbre booléenne
  (calcul sur les valeurs booléennes: Vrai et Faux)

- 1890: Hermann Hollerith construit un calculateur de statistiques à cartes perforées, l'utilise pour le recensement américain, et fonde la Tabulating Machines Company qui devient IBM en 1924

- 1904: John Fleming invente la diode
  (le premier tube à vide)

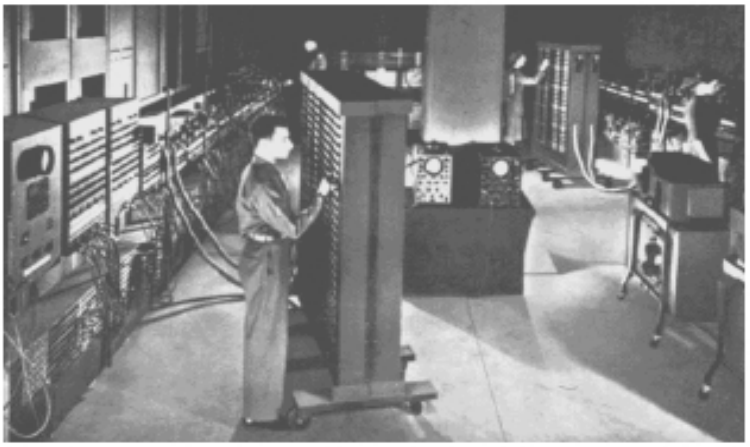
- 1938: conception de la **Machine de Turing** qui modélise/formalise les principes élémentaires du fonctionnement de toute machine et de toute opération mentale

- 1940: invention du circuit imprimé
  (plaquette comportant des pistes pour relier les composants)

- 1920-1944: **calculateurs électro-mécaniques** avec relais ou tubes
  (~ implémentations de "machines de Babbage")

2. Historique

- début 1945: ENIAC, 1er **calculateur électronique** programmable mais nécessitant de rebrancher des centaines de câbles pour chaque calcul car sa mémoire interne était trop petite; 1ers "bogues" (← bug = insecte)
  Le terme "**calculateur électronique programmable"** n'est pas utilisé pour référer – entre autres – aux ordinateurs, il est utilisé pour référer aux "ancêtres des ordinateurs" qui étaient déjà des calculateurs électroniques programmables, e.g. l'ENIAC.



- fin 1945 : EDVAC, **1er véritable ordinateur** (← programme en mémoire)
  → en 2023, l'ordinateur a 78 ans !

- fin 1945: publication de la **machine de von Neumann**:
  unité arithmétique et logique (UAL) + unité de commande + mémoire centrale + unité d'entrée + unité de sortie

- 1947: invention du transistor (Bell Telecom)

- 1949: EDSAC, 1er véritable ordinateur suivant la "machine de von Neumann" (mais on peut considérer que l'ENIAC avait aussi un tel modèle)

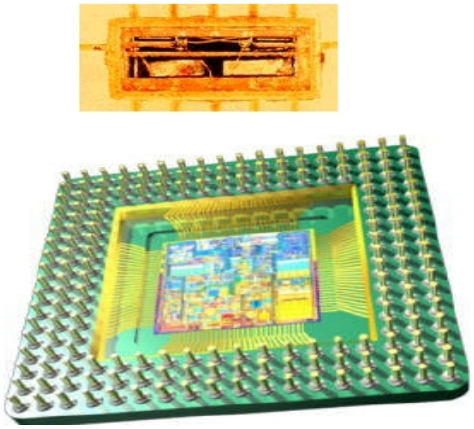
- Deux petits extraits de "Timeless S1-E8 (Space Race)" montrant le matériel informatique pilotant la mission Apollo 11 (1969) :
  - extrait 1 : une mémoire de 2 mégabytes !
  - extrait 2 : ruban perforé, "interpretative opcode", "fixed-point arithmetic"

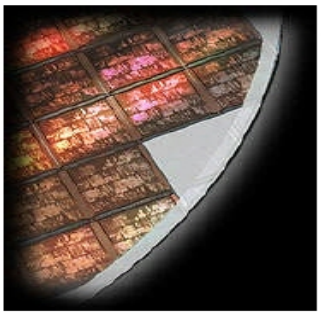
2. Historique

- 1950-1960: ordinateurs de 1ère génération (**électrique**), basés sur des tubes à vides; 1ères mémoires de masse (mémoires magnétiques, accès séquentiel)

- 1960-1970: 2ème génération (**électronique**), basés sur des transistors; mini-ordinateurs; disques durs (accès direct) ;
  1ers SGBDs [DBMSs] (système de gestion de bases de données) ;
  1ers circuits intégrés;
  1ers langages de programmation (1960: Lisp, Cobol, Fortran; 1964: Basic)

- 1970-1980: 3ème génération (**micro-électronique**), basés sur une puce (circuit intégré basé sur un microprocesseur) intégrant des milliers de transistors; 1ers ordinateurs personnels;
  1er systèmes d'exploitation multi-utilisateurs: Multics (1969), Unix (1972);
  1971: Arpanet (ancêtre d'internet), 1er microprocesseur (4004 d'Intel)
  1972: Intel sort le 8008 (8 bits, 200 KHz, 3500 transistors);
  1974: François Moreno invente la carte à puce





- 1980-1990:
  - 4ème génération (puce intégrant des centaines de milliers de transistors)
  - ordinateurs personnels; périphériques (souris, CD-ROM, ...); internet
  - 1980: une branche de IBM adopte le futur MS-DOS (développé, abandonné, et vendu à Microsoft par une autre branche de IBM)
    → monopole des logiciels de Microsoft sur la machine la plus vendue
  - 1991: Linus Torvalds créé Linux en ré-écrivant+allégeant le noyau d'Unix

2. Historique et analyse

- 1990-...:
  - parallélisme (dans le microprocesseur, plusieurs microprocesseur, ...)
  - mémoires
  - WWW (1990 : réseaux de *documents* liés par des liens hypertextes; ce "Web" est différent de internet qui, lui, est un réseau de *communication* via des câbles, des ondes, ...)
  - début de la fusion de l'informatique, des télécommunications et du multimédia (1999: iBook)

20ème siècle: siècle de la physique, chimie, ... et de l'informatique
21ème siècle: siècle de la physique, biologie, ... et de l'informatique ;
ordinateur électronique+photonique+quantique+biologique ?
À lire : "Informatique naturelle", "Ordinateur biologique, e.g., à ADN : 1 et 2"

1965-1975: Gordon Moore *remarque* que le nombre de transistors intégrables sur une puce de circuit intégré double tous les 24 mois.
→ **loi de Moore** ("loi/observation empirique" : vérifiée de 1971 à 2021):
*la puissance des nouveaux microprocesseurs (*pas celle de leurs CPUs*)*
et la capacité des nouvelles mémoires
doublent tous les 18 mois (environ / au plus)
La progression s'affaiblit et cette loi est prévue ne plus être vraie en 2025 (mais, pour les évaluations, c'est l'énoncé ci-dessus qui est à retenir).

Allez sur Wooclap (depuis "ao1wooclap" sur la page Moodle du cours) pour la question d'évaluation suivante (1 seule réponse à sélectionner) :
Selon la loi de Moore la puissance des microprocesseurs ...
A) quadruple tous les trois ans (environ / au plus)
B) quintuple tous les trois ans (environ / au plus)
C) double tous 9 mois (environ / au plus)
D) triple tous 18 mois (environ / au plus)
E) aucune des 4 dernières réponses n'est juste

Plan de la section 3 (Numération) de la partie 1

3.1. **Définitions (rappel)**

3.2. **Codes numériques**
3.2.1. ***Entiers positifs***
3.2.1.1. Changement de base
3.2.1.2. Énumération
3.2.1.3. Addition
3.2.1.4. Unités
3.2.2. ***Entiers signés en binaire***
3.2.3. ***Nombres fractionnaires***

3.3. **Codes non numériques** (pour décimaux, caractères, …)
("non numérique" dans le sens "non basé sur la numération positionnelle)
3.3.1. Codes pour caractères
3.3.2. Sérialisation des caractères: endianisme

3.4. **Avantages du numérique par rapport à l'analogique**

Sur Wooclap, pour évaluer votre compréhension des notions du plan ci-dessus (rappel: 1 seule réponse juste) :
La numérisation est la représentation ...
A) d'un chiffre (seulement)
B) d'une donnée avec un code numérique (seulement)
C) d'une donnée avec un code numérique et non numérique
D) d'une donnée avec, possiblement, un code non numérique
E) d'une donnée avec un nombre (i.e. un code numérique ou non numérique)
F) les 2 dernières réponses (sont justes)

3.1. Numération - définitions (rappel)

Chiffres [digit]: symboles utilisés dans les nombres.

Numérisation: représentation de données avec des *nombres* (i.e. des *codes* numériques ou non numériques) .

Numération: méthodes pour écrire les nombres.

Numération positionnelle: la valeur d'un chiffre dépend de sa position, chaque chiffre a un poids dans une base, e.g.,

$431{,}01_{\mathbf{10}} = 4*\mathbf{10}^2 + 3*\mathbf{10}^1 + 1*\mathbf{10}^0 + 0*\mathbf{10}^{-1} + 1*\mathbf{10}^{-2}$

$= 4*100 + 3*10 + 1*1 + 0*1/10 + 1*1/100$

$= 431{,}01 = 431{,}01_{\mathbf{10}}$

$431{,}01_{\mathbf{5}} = 4*\mathbf{5}^2 + 3*\mathbf{5}^1 + 1*\mathbf{5}^0 + 0*\mathbf{5}^{-1} + 1*\mathbf{5}^{-2}$

$= 4*25 + 3*5 + 1*1 + 0*1/5 + 1*1/25$

$= 100 + 15 + 1 + 0 + 0{,}04 = 116{,}04 = 116{,}04_{10}$

$1{,}01_{\mathbf{2}} = 1*\mathbf{2}^0 + 0*\mathbf{2}^{-1} + 1*\mathbf{2}^{-2}$

$= 1*1 + 0*1/2 + 1*1/4 = 1 + 0 + 0{,}25 = 1{,}25 = 1{,}25_{10}$

Représentation en base 10 (décimale):
avec 10 symboles différents (e.g., chiffres de 0 à 9).

Représentation en base 2 (binaire): avec 2 symboles différents (e.g., 0 et 1, vrai et faux, ouvert et fermé, 0 volt et 5 volts).
$431{,}01_2$ n'a pas de sens ('4' et '3' ne sont pas des symboles de la base 2).

Bit (contraction de "binary digit"): 0 ou 1. N'utilisez pas l'abréviation "b" car "octet" [byte] (8 bits) est abrévié par "o", "B" ou "b" (d'où l'ambiguïté). Dans ce cours, "bit" n'est jamais abrévié.

Octet [byte]: groupe de 8 bits. Dans ce cours, l'abréviation "B" n'est pas utilisée.

3.2.1. Codes numériques - entiers positifs

$N_{10} = a * 10^n + b * 10^{n-1} + \ldots + z*10^0$

Exemple: $12_{10} = 1*10^1 + 2*10^0$

$453_{10} = 4*10^2 + 5*10^1 + 3*10^0$

De même: $N_2 = a * 2^n + b * 2^{n-1} + \ldots + z * 2^0 = N'_{10}$

E.g.: $1100_2 = 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 8_{10} + 4_{10} + 0 + 0 = 12_{10}$

Conversion de 12_{10} en 1100_2 :

- *1ère méthode*: décomposer en puissances de 2 (e.g., $8 = 2^3$)
  comme dans l'exemple ci-dessus
  $12_{10} = 8_{10} + 4_{10} + 0 + 0 = 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 1100_2$
- *2ème méthode*: diviser par deux jusqu'à arriver à 0 et noter les restes:
  12 div 2 = 6 (reste: **0**); 6 div 2 = 3 (reste: **0**) ; 3 div 2 = 1 (reste: **1**);
  1 div 2 = 0 (reste: **1**).
  Suite des restes, de droite à gauche → 1100_2

Vous devez utiliser la 1ère méthode: elle est plus pratique et plus générique.
Connaissez au moins les puissances de 2, 8 et 16 suivantes:

$\mathbf{2^0=1}$, $2^1=2$, $2^2=4$, $2^3=8$, $2^4=16$, $\mathbf{2^5=32}$, $\mathbf{2^{10}=1024}$,

$\mathbf{2^{16}} = 2^{10+6} = 2^{10} * 2^6 = 1024 * 64 = \mathbf{65536}$,

$2^{-1}=0{,}5$, $2^{-2}=0{,}25$, $2^{-3}=0{,}125$, $2^{-4}=0{,}0625$

$8^{-1}=0{,}125$, $16^{-1}=0{,}0625$

3.2.1.1. Entiers positifs - changement de base

décimal	binaire	octal	hexadécimal
0	0	0	0
1	1	1	1
2	10 <-	2	2
3	11	3	3
4	100 <-	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000 <-	10 <-	8
9	1001	11	9
10 <-	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000 <-	20	10 <-

Cette table est à comprendre, pas à apprendre/copier (c'est contre-productif).

Convertir en binaire de l'octal ou de l'hexadécimal – ou tout autre nombre d'une base *b* qui est une puissance de 2 – se fait par groupe de *b* bits, en partant du bit de poids le plus faible.

Exemples, sachant que $8 = 2^3$ (→ groupes de 3 bits) et $16 = 2^4$:

$101101_2 = 101'101_2 = 55_8$

$101101_2 = 10'1101_2 = 2D_{16}$

Pour convertir des nombres entre des bases puissances de 2, est donc souvent plus rapide de passer par le binaire.

Sur Wooclap : **22_3 est égal à ...**

A) 11_8 B) 9_{10} C) 13_5 D) les 3 dernières réponses (sont justes)

E) aucune des 4 (autres réponses n'est juste)

Sur Wooclap : **$FA82_{16}$ est égal à ...**

A) 765010_8 B) 175202_8 C) 176202_8 D) 175102_8 E) aucune des 4

3.2.1.2. Entiers positifs - énumération

(cette page est - pour vos notes dans ce cours d'AO - la plus importante de ce cours car de nombreuses questions de ce cours sont directement ou indirectement liées à cette page)

Avec **1** bit (-> avec 1 chiffre en base 2),
on peut représenter $2^{\mathbf{1}}$ entiers: 0 et 1.
Avec **2** bits, on peut représenter
$2^{\mathbf{2}}$ entiers: 00_2, 01_2, 10_2, 11_2.
Avec **3** bits, on peut représenter
$2^{\mathbf{3}}$ (= 8) entiers: 000_2, 001_2, 010_2, ... 111_2.

Mémoire de $2^{\mathbf{2bits}}$ = 4 mots

$3 = 11_2 \rightarrow$ |_____|
$2 = 10_2 \rightarrow$ |_____|
$1 = 01_2 \rightarrow$ |_____|
adresse $0 = 00_2 \rightarrow$ | 1er mot |

Avec n bits, on peut représenter 2^n entiers positifs
(preuve: 2^{n+1} bits → les 2^n possibilités des n bits * les 2 possibilités du bit supplémentaire → 2^{n+1} entiers → formule respecté au rang n+1)
→ **2^n adresses: 2^n cases mémoires ("mots") peuvent être adressées.**
Exemple d'application: **avec un bus/registre d'adresse de n bits, on peut avoir 2^n adresses et donc adresser 2^n "mots" (cases mémoire).**

En base 10, avec 1 chiffre, on peut représenter 10^1 entiers positifs : 0 à 9,
avec 2 chiffres, ... 10^2 entiers positifs : 0 à 99,
avec n chiffres, ... 10^n entiers positifs : 0 à $10^n - 1$.

Dans une base b, avec n chiffres, on peut représenter de 0 à $b^n - 1$.

En base 2, avec 3 chiffres, on peut représenter de **000 à**
$\mathbf{2^3 - 1} = 7_{10} = 7_8 = 111_2 = 1'000_2 - 1 = 10_8 - 1$

En base 16, avec 4 chiffres, on peut représenter de **0000 à**
$\mathbf{16^4 - 1} = 65535_{10} = FFFF_{16} = 1111'1111'1111'1111_2$
$= 1'0000'0000'0000'0000_2 - 1 = 2^{16} - 1$

Ces notions sont évaluées plus de 50 fois dans ce cours (TDs, applications, ...).
Malgré cela, une bonne partie de vos prédécesseurs des années passées n'y ont pas prêtés attention et leur taux de réussite pour les questions relatives à ces notions est resté faible.

Exemples de question d'évaluation: **1) Dans une base B, avec N chiffres, on peut représenter les entiers positifs de 0 à ...**

A) $B^N - 1$ B) B^N C) $N^B - 1$ D) 2^B E) aucune des 4 autres réponses

2) En base 16, avec 2 chiffres, le nombre/mot le plus grand (ou l'adresse la plus grande) qui peut être représenté(e) est ...

A) 256 B) 2^{16} C) 511 D) $4^4 - 1$ E) aucune des 4 autres

Les notions liées à la numération sont suffisamment simples pour i) être communicables/explicables via des égalités, et ii) être aussi des moyens de communiquer des sens de l'égalité utilisée et de symboles liés à l'égalité. Par exemple, le texte en police à chasse fixe ci-dessous pourrait être envoyé à des "agents intelligents" (matériels ou logiciels, terriens ou extraterrestres) pour leur leur communiquer des sens de "=", de symboles pour la numération positionnelle en différentes bases, de symboles pour la numérotation romaine, de symboles de groupements. Les messages envoyés par le SETI à d'éventuels extraterrestres intelligents, dont le message d'Arecibo et le message de Dutil et Dumas (explication pointée par ce lien) envoyés en 1999 sont bien plus compliqués.

$$
\begin{array}{rcrcrcrcrcl}
{}_{w@} & = & {}_{w.} & = & 0 & = & 00_2 & = & 0_8 & = & 0_2 \\
@_{w@} & = & ._{w.} & = & 1 & = & 01_2 & = & 1_8 & = & I_R \\
@@_{w@} & = & .._{w.} & = & 2 & = & 10_2 & = & 2_8 & = & II_R \\
@@@_{w@} & = & ..._{w.} & = & 3 & = & 11_2 & = & 3_8 & = & III_R \\
 & & ...._{w.} & = & 4 & = & 100_2 & = & 4_8 & = & IV_R \\
 & & ....._{w.} & = & 5 & = & 101_2 & = & 5_8 & = & V_R \\
... ..._{w.} & = & ...._{w.} & = & 6 & = & 110_2 & = & 6_8 & = & VI_R \\
. ..._{w.} & = & ...._{w.} & = & 7 & = & 111_2 & = & 7_8 & = & VII_R \\
...._{w.} & = & ...._{w.} & = & 8 & = & 1000_2 & = & 10_8 & = & VIII_R \\
...._{w.} & = & ...._{w.} & = & 9 & = & 1001_2 & = & 11_8 & = & IX_R \\
...._{w.} & = & ...._{w.} & = & 10 & = & 1010_2 & = & 12_8 & = & X_R \\
. ..._{w.} & = & ...._{w.} & = & 11 & = & 1010_2 & = & 13_8 & = & XI_R
\end{array}
$$

Pour montrer sa compréhension/interprétation des notions de ce texte (ce message), un lecteur de celui-ci pourrait par exemple émettre le message suivant.

$$12 = ..._{w.} = 1100_2 = 1'100_2 = 14_8 = XII_R$$

3.2.1.3. Entiers positifs - addition

```
  X    1101'1001
  Y    1001'0010
X+Y  1'0110'1011
```

Si les registres ou mots-mémoire de la machine sont de
1 octet [byte] (8 bits),
comme X+Y doit normalement s'étendre sur 9 bits,
le processeur ne conserve que $0110'1011_2$ mais
signale au programmeur un "dépassement de capacité" [overflow]

Notes: 1) dorénavant, les réponses proposées pour les questions d'évaluation sur Moodle sont automatiquement mélangées, lorsque possible ;
2) suivant le temps restant, les questions suivantes sont à effectuer maintenant ou au début du 1er TD.

Sur Wooclap : **(BF_{16} + $0001'1110_2$) est égal à ...**

A) $1110'1101_2$ B) $1111'1101_2$ C) $1101'1101_2$ D) $1101'1111_2$

E) aucune des 4 autres réponses

Sur Wooclap : **(12_8 + 11_6) est égal à ...**

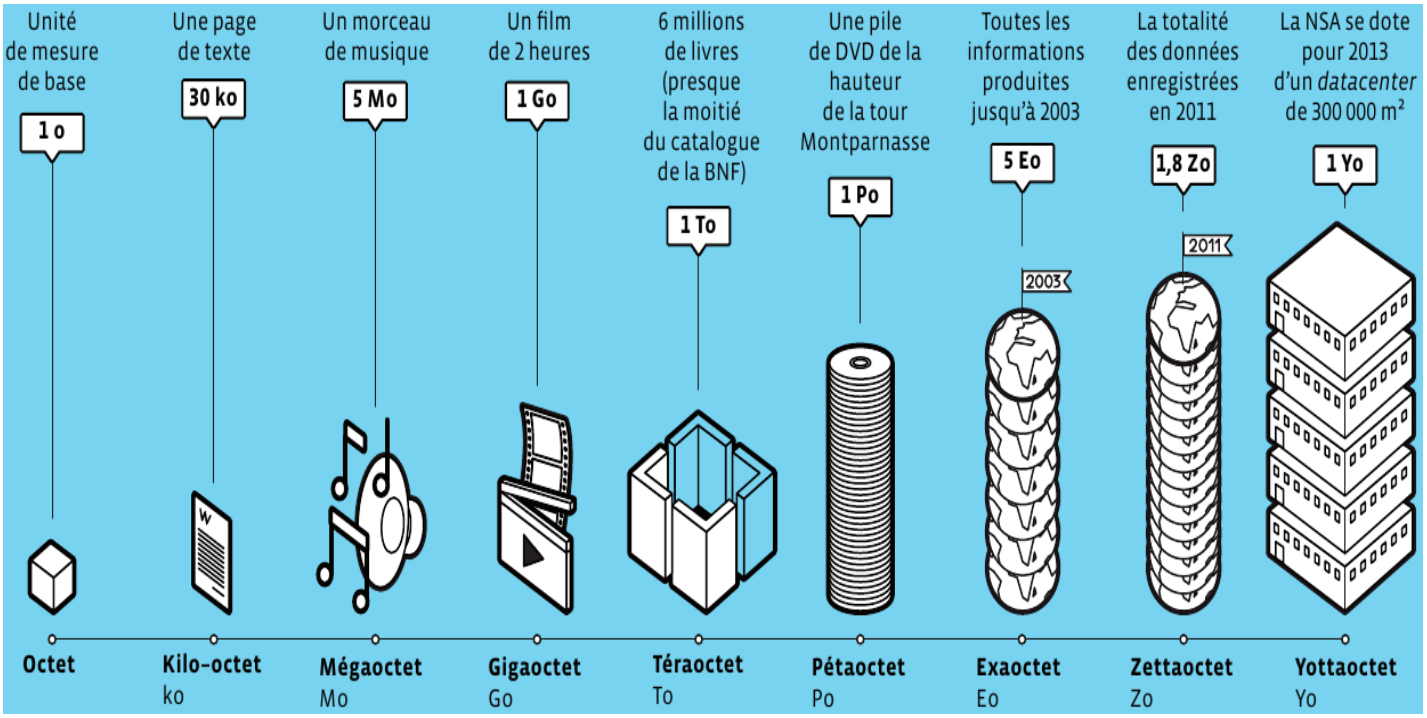
A) 1111_2 B) 11_{13} C) 11_{16} D) 11_{17}

E) aucune des 4 autres réponses

3.2.1.4. Entiers positifs - unités

Pour la vitesse d'un réseau, **1 Ko** (Kilo-octet) = 10^3 octets = **1000 octets**.
Pour la taille d'une mémoire, **1 Ko** = 2^{10} octets = 1024 octets = 1 Kio (kibi-octet).
De même (cf. Wikipedia's Préfixe_binaire et Metric_prefix) :

- **1 Méga-octet (Mo)** = 1000 Ko (10^6 octets) ou bien 1024 Ko (2^{20} octets).
  ** Fichier généré par un appareil photo numérique: entre 0,2 et 6 Mo (-> stockable en 0,1 à 0,6 Mo avec peu de perte de qualité).*
  ** Taille d'une base de donnée de 100 employés ne contenant que leurs noms, prénoms, dates de naissance et numéros de S.S.: 0,006 Mo.*
  ** Taille d'un fichier contenant la chaîne de caractères "Hello World!": 12 caractères UTF-8 (→ 12 octets) en texte pur, au moins 28 en HTML, au moins 4900 en PDF, au moins 19450 en MS Word.*
  ** Taille du Petit Robert (dictionnaire de ~2600 pages): ~22,5 Mo.*
- **1 Giga-octet (Go)** = 1000 Mo (10^9 octets) ou bien 1024 Mo (2^{30} octets).
  ** Taille d'un film actuel (au format mp4) de 2 heures: ~1 Go ou moins.*
  ** Taille d'une mémoire principale (MP) d'ordinateur portable : 32 ou 64 Go.*
  ** Pour stocker les noms de 8 milliards de personnes: 500 Go*
- **1 Téra-octet (To)** = 1000 Go (10^{12} octets) ou bien 1024 Go *(2^{40} octets)*.
  ** Taille d'un disque dur d'ordinateur portable: entre 250 Go et 1 To.*
  ** Texte brut de tous les livres écrits jusqu'à nos jours: ~200 To.*
- **1 Péta-octet (Po)** = 1000 To (10^{15} octets) ou bien 1024 To *(2^{50} octets).*
  Taille des plus gros(ses) "jeux/collections de données" [datasets] stockées.
  Le collisionneur de particules du CERN génère ~100 Po/min de données.
  Le stockage d'un cerveau humain pourrait être évalué à environ 2,5 Po.
  Un centre de données stocke des Po ou Eo, sur des disques de plusieurs To.
- **1 Exa-octet (Eo)** = 1000 Po (10^{18} octets) ou bien 1024 Po *(2^{60} octets).*
  Quantité d'informations stockées/numérisées *: 300 Eo / 2 Zo en 2011, 33 Zo en 2018 (← téléphones portables), au moins 180 Zo en 2025 ; seulement 15% de ces données sont structurées ; l'ADN humain stocke* 700 Mo *ou* 3,4 Go *dans chaque cellule;* chaque gramme d'ADN pourrait à l'avenir être utilisé pour stocker au maximum 455 Eo.
  Le système de fichiers ZFS *pourrait gérer 2^{128} octets mais cela* "ferait bouillir les océans" (vu l'énergie que cela demanderait).
- **1 Zetta-octet (Zo)** = 1000 Eo (10^{21} octets) ou bien 1024 Eo *(2^{70} octets), 2012: 1 Zo de trafic annuel sur Internet ; 2021: 3,3 Zo (20 Zo stockés).*
- **1 Yotta-octet (Yo)** = 1000 Zo (10^{24} octets) ou bien 1024 Zo *(2^{80} octets)*.



Pour plus de détails, lisez le chapitre "La déferlante des octets" (pp. 20-27) du "journal du CNRS No262, nov.-dec. 2012". La figure ci-dessus en est extraite.

En 2010, Eric Schmidt (PDG de Google de 2001 à 2011) affirmait :

Tous les deux jours, nous produisons autant d'informations que nous en avons générées depuis l'aube de la civilisation jusqu'en 2003.

(Pour culture, voici aussi ce qui, à différentes échelles de grandeurs, existe physiquement : https://www.youtube.com/watch?v=NrRPg0pH9xc)

Questions liées aux puissances de 2 sur Wooclap :

* Supposez que vous ayez une bande de papier-toilette de 1024 mètres de long et que vous puissiez la plier en deux (parties égales et parfaitement plates) 12 fois de suite (→ ? couches), quelle longueur obtiendriez-vous en centimètres ? (largeur constante ; pliures supposées parfaites : non arrondies, …)

Notes : 1) le record avec du papier du type "couche de papier-toilette" est bien de 12 fois, avec une longueur initiale de 1219m, en 2002, par une lycéenne de 16 ans qui trouva préalablement la formule lui permettant de calculer la longueur nécessaire en fonctions de l'épaisseur du papier le plus fin qu'elle avait trouvé ; 2) si vous pouviez plier 42 fois une feuille de papier, vous obtiendriez une **épaisseur très environ égale à la distance Terre-Lune**.

* Question(s) liée(s) à l'isopoint génétique (lire ces articles : **1**, **2** et **3**)

3.2.2. Numération - entiers signés en binaire

Exemple: représentation en binaire de
+7 (= 0111_2) et de -7 (= $????_2$)

Trois méthodes principales
(que nous allons voir et qui donnent des résultats différents):

- Représentation du signe et de la valeur absolue
- Représentation par complément à 2
- Représentation par complément à 1

Avec ces 3 méthodes, le bit le plus à gauche est toujours
1 pour un entier négatif et 0 pour un entier positif

3.2.2.1. Entiers signés - représentation par "signe et valeur absolue"

Le bit le plus à gauche représente le signe (0: positif; 1: négatif).
Les autres bits représentent la valeur absolue de l'entier signé.

Peu utilisé car avec cette méthode

- 1 bit est spécialement dédié au signe, ce qui fait que
  - cette méthode est moins compacte que les autres méthodes,
  - les additions et multiplications sont plus compliquées
- +0 et -0 sont encodés différemment. E.g.:
  +0 = 000_2(*entier positif sur 3 bits*)
  +0 = 0000_2(*entier positif sur 4 bits*)
  -0 = 100_2(entier négatif avec représentation par signe et valeur absolue *sur 3 bits*)
  -0 = 1000_2(entier négatif avec représentation par signe et valeur absolue *sur 4 bits*)
  +4 = 100_2(*entier positif sur 3 bits*)
  +8 = 1000_2(*entier positif sur 4 bits*)

De manière générale (pas seulement pour cette sous-section), une suite de bits (ou de chiffres) ne peut être interprétée par l'homme ou la machine tant que son type (e.g., entier, caractère(s), nombre fractionnaire avec représentation par virgule flottante, image au format JPEG, ...) et le nombre de bits utilisés ne sont pas précisés par ailleurs (e.g., via le typage de variables dans programme et les sortes d'encodage utilisés par chaque machine pour chaque type).
Dans ce cours, ces précisions sont données via le contexte (titre de la section, sujet de l'exercice, ...) et/ou dans les indices comme ci-dessus.
Si un nombre est en base 10 ou s'il s'agit d'un entier positif, il est optionnel de préciser cela : ce seront des précisions "par défaut".
De manière similaire, pour les entiers positifs il est inutile de préciser le nombre de bits.
Pour les nombres négatifs, le nombre de bits sera supposé être celui impliqué par le nombre de chiffres utilisés dans ces nombres.
Ainsi, dans les exemples ci-dessus, les parties en italique sont facultatives.

3.2.2.2. Entiers signés - représentation par complément à 2 ou à 1

Soit x un entier positif que en binaire il est possible d'exprimer sur n bits,
"bin" une fonction qui renvoie la représentation binaire d'un nombre,
complément_à_2(x) le complément à 2 de bin(x)
qui peut être utilisé pour représenter -x en binaire, et
complément_à_1(x) le complément à 1 de bin(x)
qui peut aussi être utilisé pour représenter -x en binaire

complément_à_2(x) = bin(2^n - x) = bin((2^n - x - 1) + 1)
= bin(2^n - 1 - x) + 1 = complément_à_1(x) + 1

Complémenter à 1 revient à changer tous les 0 en 1, et vice versa.
Exemples **sur 4 bits** (résultats différents avec 3 bits ou 5 bits):

$$\text{complément\_à\_2}(1_{10}) = \text{complément\_à\_1}(0001_2) + 1$$
$$= 1110_{2(\text{complément à 1 sur 4 bits})} + 1$$
$$= 1111_{2(\text{complément à 2 sur 4 bits})}$$
$$= -1_{10}$$
$$//\text{note: } 1111_2 = 15_{10(\text{entier positif})} = 2^4-1$$

$$\text{complément\_à\_2}(7_{10}) = \text{complément\_à\_1}(0111_2) + 1$$
$$= 1000_{2(\text{complément à 1 sur 4 bits})} + 1$$
$$= 1001_{2(\text{complément à 2 sur 4 bits})}$$
$$= -7_{10}$$
$$//\text{note: } 1001_2 = 9_{10(\text{entier positif})} = 2^4-7$$

$$\text{complément\_à\_2}(8_{10}) = \text{complément\_à\_1}(1000_2) + 1$$
$$= 0111_{2(\text{complément à 1 sur 4 bits})} + 1$$
$$= 1000_{2(\text{complément à 2 sur 4 bits})}$$
$$= -8_{10}$$
$$//\text{note: } 1000_2 = 8_{2(\text{entier positif})} = 2^4-8$$

Pour le "complément à 2" (complément arithmétique ; cf. notes ci-dessus),
comme pour le "complément à 1" (complément restreint ou logique),
soustraire deux termes <=> additionner le premier terme au complément (arithmétique/logique)

3.2.2.2. Entiers signés - représentation par complément à 2 ou à 1

Exemples sur 3 bits:

décimal	signe+val.	complément à 1		complément à 2
+4	100	100		100
+3	011	011		011
+2	010	010		010
+1	001	001		001
+0	000	000		000
-0	**100**	**111**	+1 =	(1)**000**
-1	**101**	**110**	+1 =	**111**
-2	**110**	**101**	+1 =	**110**
-3	**111**	**100**	+1 =	**101**
-4	**---**(100)	**---**(011)	+1 =	**100**

Ainsi: $-0 = 111_{2(\text{complément à 1 sur 3 bits})} = 000_{2(\text{complément à 2 sur 3 bits})}$

En complément à 2 sur n bits, un dépassement au delà de n bits est ignoré.

En complément à 1 sur n bits, un dépassement au delà de n bits est ajouté au résultat (dépassement causé par une addition).

Plus généralement, avec n bits, en "complément à 2", on peut représenter des entiers signés compris entre $-(2^{n-1})$ et 2^{n-1}.

Mais, avec n bits, en "signe + valeur absolue" et "complément à 1", on ne peut représenter que des entiers signés compris entre $-(2^{n-1}-1)$ et (2^{n-1}), i.e., entre $(-2^{n-1}+1)$ et 2^{n-1}.

Sur Wooclap : **Que représente $11111_{2(\text{complément à 2 sur 5 bits})}$?**

A) -1_{10} B) -0_{10} C) -15_{10} D) $+31_{10}$

E) aucune des 4 précédentes réponses

Quelle "expression sur les bits en C" calcule la puissance de deux maximum qui divise un entier quand les entiers signés utilisent un complément à deux ?

3.2.3. Numération - nombres fractionnaires

Exemples: $0{,}01_2 = 0*2^0 + 0*2^{-1} + 1*2^{-2} = 0*1 + 0*0{,}5_{10} + 1*0{,}25_{10} = 0{,}25_{10}$

$0{,}125_{10} = 0{,}001_2$ car $((0{,}125 * 2) * 2) * 2 = 1$

Sur Wooclap : **$0{,}75_{10}$ est égal à ...**

A) $0{,}11_2$ B) $0{,}3_4$ C) $0{,}6_8$ D) les 3 dernières réponses E) aucune

Représentation en machine classique (pas via des "nombres universels" - cf. approche ici - ni en "précision-infinie|multi-précision via le stockage de nombres via des tableaux de chiffres, e.g. le BigInt de JavaScript") :

- La virgule d'un nombre N peut être représentée (virtuellement, i.e., implicitement) et gérée en ***virgule fixe*** (→ nombre fixe de chiffres après la virgule, e.g., traitement de N comme un nombre entier) -> gestion par le programmeur des traductions, tronquage, débordement de capacité, ...
  À lire : https://hackr.io/blog/float-vs-double

- La virgule d'un nombre N peut être représentée (virtuellement) et gérée en ***virgule flottante***, sous la forme **$N = M * B^E$**, **B** étant appelée la base, **M** la mantisse et **E** l'exposant.
  En binaire, la représentation de E et M est ***généralement*** par "**signe et valeur absolue**" (mais pas dans le format IEEE 754) et ***M est*** souvent ***normalisée*** afin qu'un certain bit soit toujours à 1 (ceci permet de ne pas le représenter physiquement – i.e. de ne pas le stocker – et donc d'utiliser ce bit autrement afin d'augmenter la précision) :
  * au format IEEE 754 (cf. page suivante), le bit de signe est séparé de la mantisse, l'exposant est biaisé pour être positif, et M = 1,...
    Exemples de conversions pour avoir M = 1, ... :

    $0{,}01_2 = 1_2 * 2^{-2}$ $8{,}25_{10} = 1000{,}01_2 = 1{,}00001 * 2^3$

    $-6{,}625_{10} = -\ \mathit{1}{,}10101_2 * 2^2$ (le '1' en italique n'est pas représenté)
  * dans le TD (uniquement), on doit avoir 0,5 =< M < 1,

    -> $-6{,}625_{10} = -\ 110{,}101_2 = -\ 0{,}\mathit{1}10101_2 * 2^3$

Multiplication: ajout des exposants et multiplication des mantisses.
Division: soustraction des exposants et division des mantisses.
Addition: addition des mantisses avec des exposants de même valeur.
Soustraction: soustraction des mantisses avec des exposants de même valeur.

Représentation des réels **normalisée** au format IEEE 754 → 3 choix :

Signe	Exposant biaisé	Mantisse
1	1000 0001	101 0100 0000 0000 0000 0000

a) sur 32 bits (8 bits pour E, 1 bit pour le signe de M, 23 bits pour |M|)

b) sur 64 bits (2 * plus précis: 1 bit de signe, 11 bits pour E, 52 bits pour |M|)

c) sur 80 bits (précision étendue).

Plus précisément, E est appelé "exposant réel". Ce qui est stocké (sur n bits) est appelé E', l'exposant "biaisé". Généralement, E' = E + (2^n/2 -1) pour que E' soit un nombre positif (→1 seule méthode d'encodage). E.g., sur 8 bits, le *biais*, 2^n/2 - 1= 127, les valeurs possibles de E vont de -127 à 128 (→ palette équilibrée de nombres positifs et négatifs ; cf. TD) et les valeurs possibles de E' sont entre 0 et 255 (on dit que E' est biaisé à 127). Note : dans la correction du TD, un biais de 128 (au lieu de 127) est utilisé, ce n'est pas usuel. Pour les QCMs, le biais sera de 2^n/2 - 1 (donc 127 lorsque E est sur 8 bits).

Quand E' est sur 11 bits, il est aussi courant qu'il soit biaisé à 1023.

Dans les programmes, les "flottants" sont des représentations des réels. Leur précision étant limitée, les opérations sur les flottants peuvent être fausses (e.g., "0.2 + 0.3 = 0.5" est faux dans les flottants) et ceci conduit à des bogues [bugs] → "Écran Bleu de la Mort" de Windows, explosion de la 1ère fusée Arianne 5, … ; cliquez ici pour un peu de théorie.
Il faut donc être prudent : ordonner/simplifier les opérations pour réduire les erreurs et gérer l'accumulation des erreurs ou prouver que le programme n'en génère pas (-> enseignements dans les années suivantes).

Exemple de question d'évaluation: **Dans une gestion normalisée de nombres en "virgule flottante" sous la forme M * B^E, ...**

A) au moins un nombre à gauche de la virgule est *toujours* représenté physiquement (→stocké) dans la mantisse

B) la représentation de E et M se fait *généralement* (mais pas dans le cas où le format IEEE 754 est utilisé) par *"signe et valeur absolue"*

C) l'exposant est biaisé à 128 si représenté sur 8 bits au format IEEE 754

D) les 3 dernières réponses E) aucune des 4 dernières réponses

3.3. Numération - codes non numériques

("non numérique" dans le sens "non basé sur la numération positionnelle)

Les codes sont soit numériques, soit non numériques.
Nous venons de voir *des* codes numériques pour des nombres.
Les codes non numériques sont basés sur des tables.
Ils peuvent être pour des décimaux, des caractères, ...

Il existe de nombreux codes binaires non numériques pour décimaux.
En voici quelques uns.

- **Décimal Codé Binaire** (DCB) [Binary Coded Decimal (BCD)]:
  4 bits pour chaque digit décimal.
  Exemple: $129_{10} = (0001\ 0010\ 1001)_{BCD}$
  Entrées-sorties faciles mais opérations arithmétiques compliquées (e.g., pour l'addition, il faut ajouter 6 chaque fois que le résultat est supérieur à 9).

- **Excédent-3** (comme DCB mais, pour simplifier les opérations arithmétiques, +3 (= 11_2) est ajouté à chacun des groupes de 4 bits.

- **Code 2 dans 5**: codage sur 5 bits, avec exactement 2 bits ayant la valeur 1 pour permettre la détection d'un nombre impair d'erreurs.
  Exemple: $129_{10} = 00101'00110'11000_{code2dans5}$

- **Code biquinaire** (un autre code pour la correction d'erreur): codage sur 7 bits, avec exactement 1 bit à 1 dans les 2 positions de gauche et exactement 1 bit à 1 dans les 5 positions de droite.

Exemple de question d'évaluation: **Le code biquinaire ...**
A) est un code non numérique B) est un code sur 5 bits
C) a exactement 2 bits à 1 dans les 2 positions de gauche
D) les 3 dernières réponses E) aucune des 4 autres réponses

3.3.2. Numération - Codes pour caractères

Voici quelques codes binaires non numériques pour caractères
– alphanumériques (A..Z, a..z, 0..9), spéciaux (?,!, @, ", ...), etc. – :

- **ASCII** [American Standard Code for Information Exchange] (7 bits)
  EBCDIC [Extended Binary Coded Decimal Internal Code] (8 bits)

- UNICODE **- ISO/IEC 10646** (UTF-8, UTF-16, UTF-32) :
  2^{21} *points de code* (cf. Wikipedia : "Unicode", "code point", "code unit", "caractère abstrait", par opposition à "glyphe", et leurs rationale) **via** 2^4+1 [0-10_{16}] "plans" de 2^{16} [0-$FFFF_{16}$] codes), e.g. U+00C7 pour 'Ç', plus qu'assez pour pour chaque caractère de chaque langue du monde. Adopté par les standards récents et tous les systèmes d'exploitation.
  Quelques sous-catégories de UNICODE :
  * **UTF-8** [Unicode Transformation Format 8 bits] (→ 97% des pages Web)
    - 1 octet pour représenter les 127 caractères Ascii → $[00\text{-}7F]_{16}$
    - 2 octets pour un *point de code* des caractères $[80\text{-}07FF]_{16}$. E.g., pour 'é' (U+00E9; donc même si 1 octet suffirait: $\mathbf{1110'1001}_2$) :
      110*(="début de 2 octets")*0'00**11** 10*(="la suite")***10'1001**$_2$ = C3 $A9_{16}$
      → constante 0xC3A9 en C, chaîne de caractères "\xC3A9" en C et JS
    - 3 octets pour $[0800\text{-}FFFF]_{16}$; 4 octets pour $[1000\text{-}10FFFF]_{16}$.
    → Codes de tailles variables ; en programmation, usage d'un type "Wide_character" (e.g. wchar_t en C) et de fonctions travaillant sur ces types, e.g., en C, wcschr au lieu de strchr ;
  * **UTF-16**, utilisé par Java et Windows
    - 2 octets pour les 2^{16} caractères du plan 0 (→ hors langues d'Asie)
    - 2 autres octets pour les autres caractères ;
  * **UTF-32** (sur 32 bits), utilisé par de nombreux systèmes Unix, 2^{21} codes même si 2^{32} aurait été possible.

Exemple de question d'évaluation: **L'Unicode ...**
A) est un code numérique B) inclut le code ASCII et le code EBCDIC
C) inclut les langues d'Extrême Orient pour UTF-16 dans son plan 0
D) les 3 dernières réponses E) aucune des 4 dernières réponses

3.3.3. Sérialisation des caractères: endianisme

Little-endian storage (stockage par le petit bout, en mémoire vive): stockage des octets (d'une valeur multi-octets) dans l'ordre croissant des poids, i.e., les bits, octets et mots de *poids les plus faibles* se situent en 1er dans la mémoire, "en haut (et/ou à gauche)", *aux adresses les plus hautes*, donc inversement à une lecture de gauche à droite de la valeur hexadécimale. Exemple: pour le caractère U+8F (caractère Unicode dont le code hexadécimal est $8F_{16}$), le premier octet stocké en mémoire est F et le second est 8.
C'est le cas dans les microprocesseurs de Intel ou pour UTF-16LE.

Big-endian storage: stockage des octets dans l'ordre inverse des poids, i.e., les bits, octets et mots de *poids les plus forts* se situent en 1er dans la mémoire, "en haut (et/ou à gauche)", aux *adresses les plus hautes*, comme dans une lecture de gauche à droite de la valeur hexadécimale.
C'est le cas dans les microprocesseurs de Motorola, pour UTF-16BE ou après le caractère U+FEFF.

Sur Wooclap :
A quoi réfère l'expression "little-endian" ?
A) à une méthode qui consiste à commencer par "le petit bout"
B) au stockage des octets (d'une valeur multi-octets) dans l'ordre des poids
C) au stockage des octets normalement utilisé pour UTF-16LE
D) les 3 dernières réponses E) aucune des 4 autres réponses

3.3.4. Avantages du numérique – et surtout de la base 2 – par rapport à l'analogique

- Facilité d'implémentation (0 et 1: ouvert et fermé, 0 volt et 5 volts, …)

- Fiabilité (signal facile à distinguer: 2/quelques états seulement à distinguer, pas des centaines de valeurs proches les unes des autres):
  - peu de déformations du signal dues au magnétisme, à la chaleur, ...
  - possibilité de codes détecteurs/correcteurs d'erreur ; exemple :

0	1	1	0	0
0	1	0	1	0
1	0	0	0	1
0	1	0	0	1
1	0	0	1	0
0	1	1	0	0
0	1	1	1	1
0	1	0	1	0

Bit contrôle

caractère contrôle

Vi1_2631

- Facilité et rapidité de manipulation (mémorisation, opérations et programmation de ces opérations). Il faut noter qu'un système numérique basé un peu plus que 2 états peut être plus efficace: lire http://en.wikipedia.org/wiki/Multi-valued_logic#Applications

Mais les entrées-sorties des processeurs sont en analogique
→ nécessité de convertisseurs entre numérique et analogique

Sur Wooclap :

Les avantages du numérique par rapport à l'analogique sont ...

A) sa facilité d'implémentation B) sa fiabilité
C) sa faible sensibilité aux déformations du signal dues au magnétisme
D) ses possibilités supplémentaires de manipulation
E) les 4 autres réponses

Un domaine impliquant nécessairement de l'encodage en binaire est ...

A) la numérisation B) l'informatique C) l'architecture des ordinateurs
D) les 2 dernières réponses E) aucune des 4 précédentes réponses

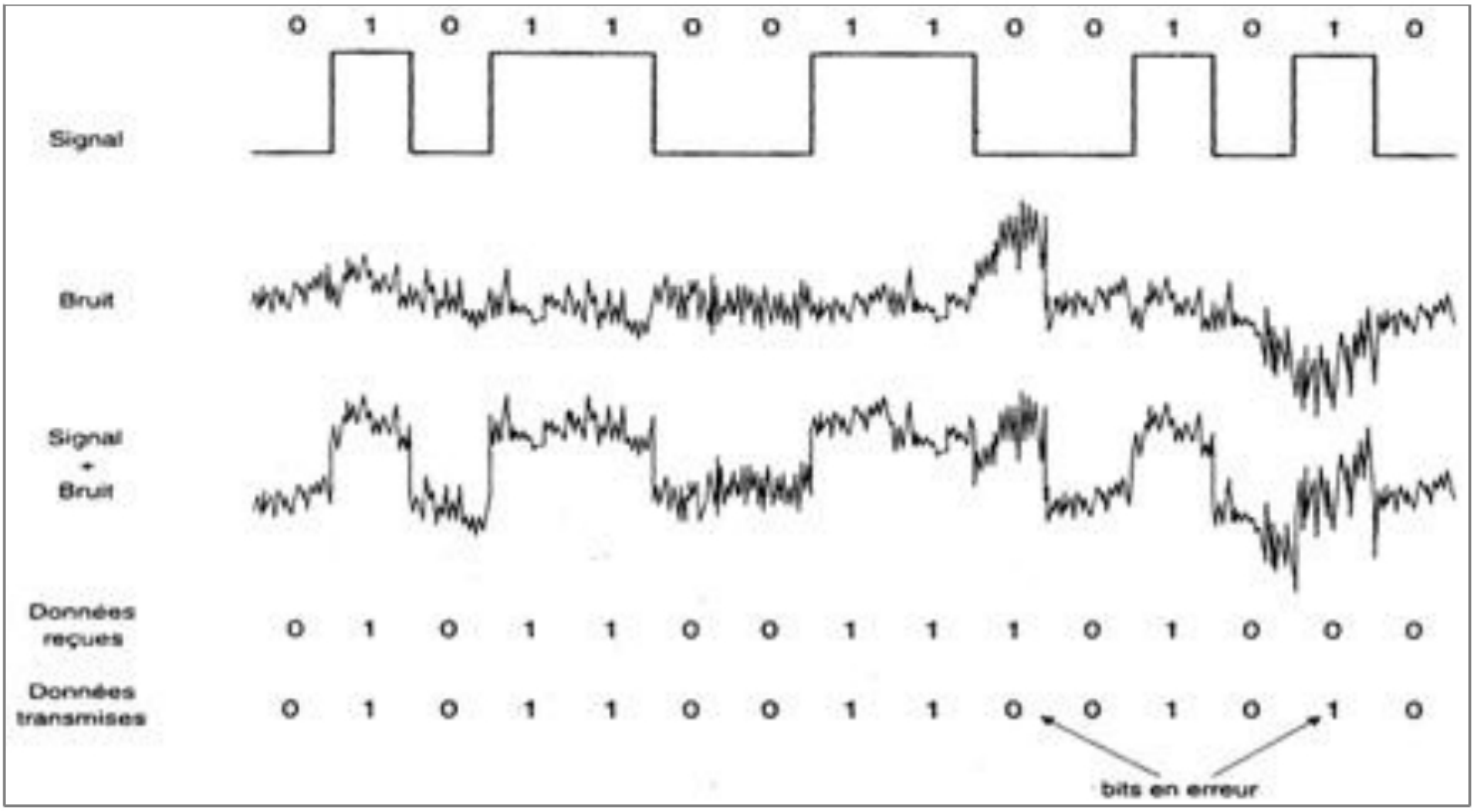
Rendu d'une image avec différentes précisions/définitions/compressions :

Codage 3 octets

Codage 1 octets

Codage 1 DIGIT

Effet du bruit sur un signal :



4. Tout ou partie du TD pour la partie 1 du cours d'AO

Si la section "3.2.1. Entiers positifs" de la partie 1 n'est pas finie,
ce TD commencera par finir cette section.
Il y aura en fait 2 "TDs réels" pour ce "TD pour la partie 1" (et, durant ceux-ci, quelques questions Wooclap via "ao1wooclap" sur Moodle).
Pour chaque exercice (question Wooclap ou pas), un étudiant passera au tableau pour l'effectuer et l'exercice sera corrigé au tableau.

I. Exemples de questions de contrôle pour le début du 1er TD réel

1. Un exemple de chiffre ni octal ni binaire est ...
   a) 0 b) 2 c) 5 d) 8 e) aucune des autres réponses

2. 10_{10} est égal à ... ?
   a) 1100_2 b) 1010_2 c) 11_8 d) 11_{16}

3. 10101_2 est égal à ... ?
   a) 21_{10} b) 23_{10} c) 17_{16} d) 21_{16}

4. L'addition des représentations en base 2 de 25_{10} et de 14_{10} donne … ?

5. Exprimer dans les bases suivantes le nombre de minutes dans "1h04min" (1:04) : 10, 60, 30, 2, 8, 16.

II. Exemples de questions de conversion/calcul

(si cela n'a pas encore été fait, le début de la section 3.2.3 et la section 3.2.2 seront présentées ; les questions après la 7.a sont optionnelles)

1. Exprimer en binaire, en octal et hexadécimal les nombres réels suivants : 32,625 et 128,75.

2. Convertir en décimal les nombres suivants, la base étant indiquée en indice: $DADA,C_{16}$, $270,14_8$ et $11011,01111_2$

3. Donner, sur 8 bits, les représentations "signe + valeur absolue", "complément à 1" et "complément à 2" des valeurs décimales suivantes: -32 et -128.

4. Faire, en binaire puis en décimal, l'addition de 001_8 à 377_8 puis de 177_8 à 200_8, en supposant tout d'abord que 377_8 et 200_8 sont des représentations sur 8 bits en complément à 1 de nombres négatifs, puis en supposant que ce sont des représentations sur 8 bits en complément à 2 de nombres négatifs.

5. On dispose d'une machine où les nombres réels sont représentés sur 32 bits (numérotés de droite à gauche de 0 à 31) avec:
   - une quantité fractionnaire sur 23 bits (les bits 0 à 22) correspondant à la mantisse m normalisée (0,5 =< m < 1);
   - un exposant biaisé, représentant une puissance de 2, codé sur 8 bits (les bits 23 à 30);
   - un bit pour le signe de la mantisse (0 si m >= 0, 1 si m < 0).

   Donner, sous la forme $\pm a * 2^b$ (a et b décimaux), la valeur qui correspond aux 32 bits suivants (sous forme octale): 27632000000.

6. La représentation des nombres réels correspond à celle décrite précédemment, sauf que le premier bit de la mantisse normalisée, premier bit qui est donc toujours à 1, n'est pas représenté sur la machine. Cet artifice complique un petit peu les manipulations mais permet de gagner en précision.
   Donner, sous forme octale, la représentation, sur une telle machine, des nombres décimaux suivants: 278 et -6,53125.

7. Soit une machine où les nombres réels sont représentés sur 12 bits, numérotés de droite à gauche de 0 à 11, avec:
   - une mantisse m normalisée (0,5 =< m < 1) sur 7 bits (les bits 0 à 6);
   - un exposant biaisé, représentant une puissance de 2, codé sur 4 bits (les bits 7 à 10);
   - un bit pour le signe de la mantisse (le bit 11).

   a) Trouver l'intervalle fermé des valeurs strictement positives représentables sur cette machine. Les bornes seront mises sous la forme $\pm a * 2^b$, a et b étant des entiers décimaux. Simplifier autant que possible.

   b) Mettre sous la forme $\pm a * 2^b$, où a et b sont des entiers décimaux, les 2 nombres réels suivants donnés sous forme hexadécimale: X=AE8 et Y=9DO.

   c) Calculer Z = y - X. Mettre le résultat sous la forme $\pm a * 2^b$, entiers décimaux, en simplifiant au maximum.

   d) Donner, sous forme octale, la représentation correspondant au nombre décimal suivant: -32,625.

8. Que représente en décimal l'information 37724000000_8 si on la considère comme un nombre entier, en complément à 1, en complément à 2 et comme un nombre réel, avec une représentation analogue à celle décrite dans l'exercice 7 ?