Architecture des ordinateurs

Henri Derycke

henri.derycke@univ-eiffel.fr

Université Gustave Eiffel
IGM, bureau 4B165

à partir du cours de Samuele Giraudo

2024–2025

L'informatique

Informatique : contraction d'« **infor**mation » et d'« auto**matique** ».

Science du **traitement automatique de l'information**.

Ordinateur : machine automatique de traitement de l'information obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.

On utilise des ordinateurs pour

1. **accélérer** des calculs complexes et difficiles ;
2. **regrouper** une masse importante de données ;
3. **traiter** de gros volumes de données.

Point de vue adopté

L'informatique est un vaste domaine qui comprend, entre autres,

- l'algorithmique ;
- la combinatoire ;
- la calculabilité ;
- la cryptographie ;
- l'intelligence artificielle ;
- le traitement d'images ;
- les réseaux ;
- l'étude des machines.

En effet,

> « *L'informatique n'est pas plus*
> *la science des ordinateurs*
> *que l'astronomie n'est celle des télescopes.* »
>
> — M. R. Fellows, I. Parberry

Ce cours s'inscrit cependant précisément dans **l'étude des machines**.

Objectifs du module

L'objectif du module est d'établir une approche simplifiée mais exacte du fonctionnement et de l'organisation d'un ordinateur.

Celui-ci est organisé en trois axes.

1. **Axe 1** : **ordinateurs et codage des données**.

   Évolution des ordinateurs depuis leur apparition à aujourd'hui, organisation logique des ordinateurs, représentation des données.

2. **Axe 2** : **programmation en assembleur**.

   Bases de la programmation en assembleur, compilation, fonctions. Mise en perspective pour comprendre le fonctionnement des langages actuels.

3. **Axe 3** : **ouverture et améliorations**.

   Concept de pipeline, de hiérarchies de mémoire. Architecture 64 bits.

Contenu du cours

Axe 1.

Histoire

Représentation

Axe 2.

Programmation

Axe 3.

Mémoires

Pipelines

`https://igm.univ-mlv.fr/~derycke/AO`

Pré-requis

Ce cours demande les pré-requis suivants :

- des bases en connaissance du **système** Linux (fonctionnement élémentaire, principales commandes du terminal, logiciels classiques) ;
- des bases en **algorithmique** (manipulation de structures de données élémentaires comme les tableaux et les chaînes de caractères).
- des bases en **programmation en** C (notion de programme, de compilation, de variable, de type, d'instruction, de fonction) ;

Bibliographie (minimale) :

- P. Carter, *PC Assembly Language*, 2006, `https://pacman128.github.io/static/pcasm-book-french.pdf` ;
- J. L. Hennessy, D. A. Patterson, *Architectures des ordinateurs : une approche quantitative*, 2003.

Axe 1 : ordinateurs et codage des données

Histoire

Représentation

Domaines de progression

Les progrès en la matière du développement des ordinateurs sont la conséquence d'avancées de trois sortes :

1. les **découvertes théoriques** (mathématiques, informatique théorique) ;
2. les **avancées technologiques** (physique, électronique) ;
3. les **réalisations concrètes d'ordinateurs** (ingénierie).

Les avancées théoriques majeures

-350 : Aristote fonda les bases de la **logique** ;

1703 : G. W. Leibniz inventa le **système binaire** ;

1854 : G. Boole introduisit l'**algèbre de Boole** ;

1936 : A. M. Turing introduisit la **machine de Turing** ;

1938 : C. Shannon découvrit comment **utiliser l'algèbre de Boole** dans des circuits ;

1945 : J. von Neumann définit l'architecture des ordinateurs modernes : la **machine de von Neumann** ;

1948 : C. Shannon introduisit la **théorie de l'information**.

Les avancées technologiques majeures

1904 : J. A. Fleming inventa la **diode à vide** ;

1948 : J. Bardeen, W. Shockley et W. Brattain inventèrent le **transistor** ;

1958 : J. Kilby construisit le premier **circuit intégré** ;

1971 : la société Intel conçut le premier **microprocesseur**, l'Intel 4004.

Les réalisations majeures

Antiquité : utilisation et conception d'**abaques** ;

1623 : W. Schickard conçut les plans de la première machine à calculer, l'**horloge calculante** ;

1642 : B. Pascal construisit la **Pascaline** ;

1801 : J. M. Jacquard inventa le premier **métier à tisser programmable** ;

1834 : C. Babbage proposa les plans de la **machine analytique** ;

1887 : H. Hollerith construisit une **machine à cartes perforées** ;

1949 : M. V. Wilkes créa un ordinateur à architecture de von Neumann, l'**EDSAC**.

Toutes ne sont **pas des ordinateurs** au sens de la définition précédente.

Le temps des machines à mécanismes

Cette période s'étend de l'antiquité et se termine dans les années 1930.

Souvent d'initiatives isolées, les machines construites à cette époque possédaient néanmoins souvent quelques points communs :

- utilisation d'**engrenages** et de **courroies** ;
- nécessité d'une **source physique de mouvement** pour fonctionner ;
- machines construites pour un **objectif fixé** *a priori* ;
- espace de **stockage très faible** ou inexistant.

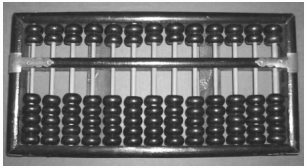
Ces machines faisaient appel à des connaissances théoriques assez rudimentaires, toute proportion gardée vis-à-vis de leur époque d'introduction.

Les abaques

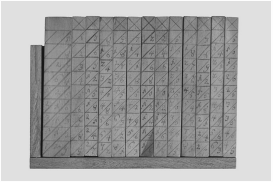
Abaque : instrument mécanique permettant de réaliser des calculs.

Exemples :

Cailloux



Bouliers

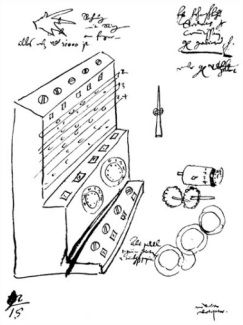


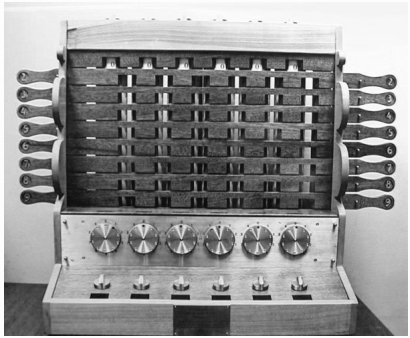
Bâtons de Napier

(J. Napier, 1617)

L'horloge calculante

En 1623, Schickard écrivit à Kepler pour lui présenter les plans de son horloge calculante.

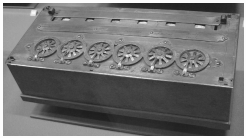




Elle permettait de faire des calculs arithmétiques, utilisait des roues dentées et gérait le report de retenue. Elle ne fut construite qu'en 1960.

La Pascaline

En 1642, Pascal conçu la Pascaline.



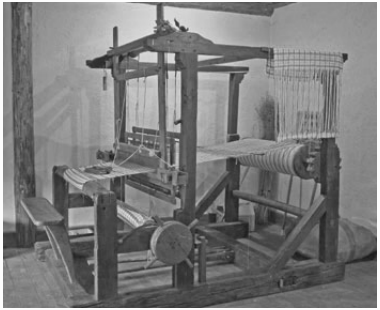
Elle permettait de réaliser des additions et soustractions de nombres décimaux jusqu'à six chiffres.

En 1671, Leibniz améliora la Pascaline de sorte à gérer multiplications et divisions.

Ces machines permettent de faire des calculs mais leur **tâche** est **fixée dès leur construction** : la notion de programme n'a pas encore été découverte.

Le métier à tisser programmable

En 1801, Jacquard proposa le premier métier à tisser programmable, le Métier Jacquard.

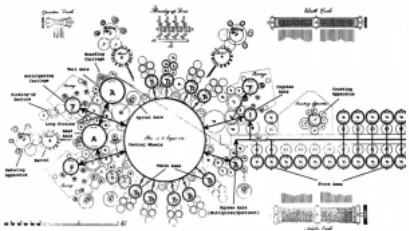


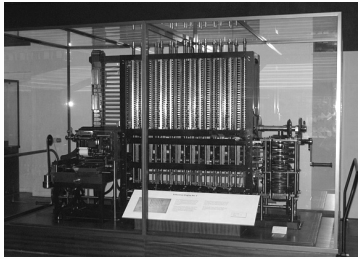
Il était piloté par des **cartes perforées**. Celles-ci dirigeaient le comportement de la machine pour tisser des motifs voulus.

Ces cartes jouaient le rôle de **programme** : une même machine pouvait exécuter des tâches différentes sans avoir à être matériellement modifiée.

La machine analytique

En 1834, Babbage proposa les plans d'une machine, la machine analytique.





Il ne put cependant pas la construire. Elle ne le fut finalement que dans les années 1990.

La machine analytique — quelques caractéristiques

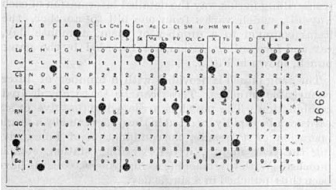
La machine analytique est étonnamment moderne pour avoir été imaginée si tôt dans l'histoire.

Elle comprend, de manière séparée et bien organisée,

- un lecteur de cartes perforées pour les instructions du **programme** ;
- un lecteur de cartes perforées pour les **données** passées en entrée ;
- une unité de **calcul** (le « moulin ») ;
- une **mémoire** (le « magasin »).

La machine à cartes perforées

En 1887, Hollerith construisit une machine à carte perforées pour faciliter le recensement de la population des États-Unis.



Les cartes perforées servaient à décrire les caractéristiques des individus en y perçant des trous.

La machine pouvait ainsi dénombrer les individus selon plusieurs critères.

Le temps des machines à lampes

Cette période débuta dans les années 1930 avec l'introduction de la machine (théorique) de Turing et se termina dans les années 1950.

D'un **point de vue matériel**, elle était basée sur

- les relais ;
- les tubes à vide ;
- les mémoires à tores de ferrite ;
- les bandes magnétiques.

D'un **point de vue théorique**, elle avait pour fondements le système binaire, la machine de Turing et l'architecture de von Neumann (pour la plupart).

Les machines de cette ère se programmaient en

- langage machine pour les toutes premières ;
- assembleur.

La diode à vide

En 1904, Fleming inventa la diode à vide.

Elle s'utilise principalement comme **interrupteur** ou comme **amplificateur** de signal électrique.

Ceci mena à l'invention des **tubes à vide**, composants électriques utilisés dans la conception des télévision, postes de radio et les premières machines électriques.

Ils sont encore utilisés aujourd'hui dans des domaines très précis : fours à micro-ondes, amplificateurs audio et radars entre autres.

Le système binaire, la logique et l'électronique

En 1703, Leibniz s'intéressa à la représentation des nombres en **binaire** et à leurs opérations.

Les bases de la logique étaient connues au temps d'Aristote mais il fallut attendre 1854 pour que Boole formalise la notion de **calcul booléen**.

Celui-ci est formé de deux valeurs, le faux (codé par le 0 binaire) et le vrai (codé par le 1 binaire) et par diverses opérations :

- le *ou* logique, noté $\vee$;
- le *et* logique, noté $\wedge$;
- le *ou exclusif*, noté $\oplus$;
- la *négation* logique, notée $\neg$.

Ces opérations donnèrent lieu aux **portes logiques** dans les machines et constituent les composants de base des unités chargées de réaliser des calculs arithmétique ou logiques.

ou

et

ou excl.

nég.

La machine de Turing — calculabilité

La machine de Turing est un concept mathématique qui fut découvert par Turing en 1936.

Son but est de fournir une abstraction des mécanismes de calcul.

Elle pose la définition de ce qui est calculable :

« tout ce qui est effectivement ***calculable***
est **calculable** par une machine de Turing ».

Des deux occurrences du mot « calculable »,

1. la 1re signifie ce que l'on peut calculer de manière **intuitive** (ce qu'un être humain peut calculer par un raisonnement) ;
2. la 2e signifie ce que la machine de Turing peut produire comme résultat à l'issue de l'**exécution** d'un programme.

La machine de Turing — fonctionnement et complétude

Une machine de Turing travaille sur un ruban infini (ou plutôt, arbitrairement grand) contenant des cases qui peuvent être soit vides (·), soit contenir la valeur 1, soit contenir la valeur 0.

Le ruban est la **mémoire** de la machine.

Une tête de lecture/écriture vient produire un résultat sur le ruban en modifiant le contenu de la case pointée et en se déplaçant d'un pas à gauche ou à droite.

Cette tête de lecture est pilotée par un **programme**.

Un machine est Turing-complète si elle peut calculer tout ce que peut calculer une machine de Turing.

L'architecture de von Neumann — caractéristiques

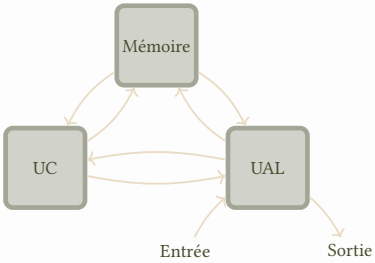
L'architecture de von Neumann est un modèle organisationnel d'ordinateurs décrit par von Neumann en 1945.

Quelques caractéristiques importantes :

- une machine de von Neumann possède diverses parties bien distinctes et dédiées à des tâches précises (mémoire, unité de calcul et unité de contrôle) ;
- le programme à exécuter se situe dans la mémoire interne de la machine (un programme est une donnée comme une autre). Une telle machine est dite machine à programme enregistré ;
- elle est pourvue d'entrées et de sorties qui permettent la saisie et la lecture de données par un humain ou bien par une autre machine.

Encore aujourd'hui la très grande majorité des ordinateurs sont des machines de von Neumann.

L'architecture de von Neumann — organisation



- La mémoire contient à la fois des données et le programme en cours d'exécution.
- L'UC, Unité de Contrôle, permet de traiter les instructions contenues dans le programme. Elle est chargée du **séquençage** des opérations.
- L'UAL, Unité Arithmétique et Logique, permet de réaliser des instructions élémentaires : opérations arithmétiques ($+$, $-$, $\times$, $/$, …), opérations logiques ($\vee$, $\wedge$, $\oplus$, $\neg$, …), opérations de comparaison ($=$, $\neq$, $\leqslant$, $<$, …).
- L'entrée permet de recevoir des informations.
- La sortie permet d'envoyer des informations.

Les générations d'ordinateurs

Le dernier siècle peut se découper en **générations**, suivant le matériel utilisé. Des quatre principales avancées technologiques découlent les quatre générations suivantes :

1re génération : de 1936 à 1956, emploi de **tubes à vide** ;

2e génération : de 1956 à 1963, emploi de **transistors** ;

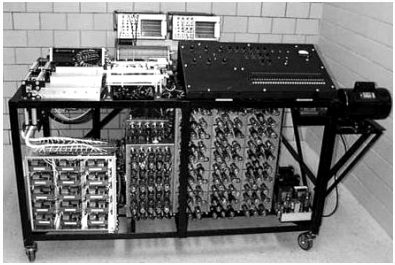
3e génération : de 1963 à 1971, emploi de **circuits intégrés** ;

4e génération : de 1971 à 2021 (et plus), emploi de **microprocesseurs**.

Parcourons maintenant l'évolution des machines en les rangeant en trois catégories : les machines à mécanismes, les machines à lampes et les machines à transistors.

L'ABC

L'Atanasoff–Berry Computer (ABC) fut le premier ordinateur numérique électronique. Il fut construit en 1937 par Atanasoff et Berry.



Il représentait les données en binaire et il adoptait une **séparation entre mémoire et unité de calcul**.

Il a été construit pour la résolution de systèmes d'équations linéaires (il pouvait manipuler des systèmes à vingt-neuf équations).

L'ASCC

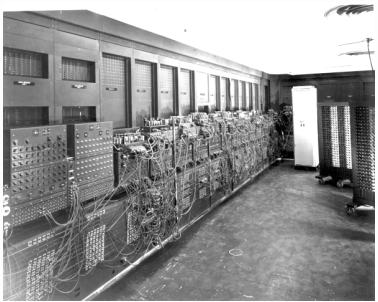
L'Automatic Sequence Controlled Calculator (ASCC), également appelé Harvard Mark I, fut construit par IBM en 1944. Il fut le premier ordinateur a **exécution totalement automatique**.

Il pouvait réaliser une multiplication de nombres de vingt-trois chiffres décimaux en six secondes, une division en quinze secondes et des calculs trigonométriques en une minute.

Il ne vérifie pas l'architecture de von Neumann car il fonctionne à cartes perforées (le programme n'est pas une donnée).

L'ENIAC

L'Electronic Numerical Integrator Analyser and Computer (ENIAC) fut achevé en 1946. C'est le premier ordinateur électronique **Turing-complet**.

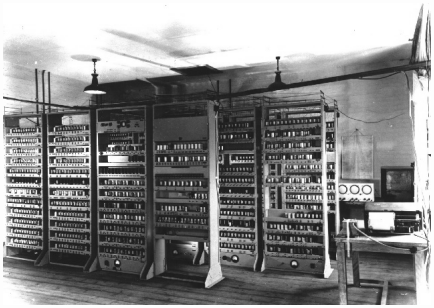


Il pouvait réaliser des multiplications de nombres de dix chiffres décimaux en trois millièmes de seconde.

Il contient 17468 tubes à vide et sa masse est de trente tonnes.

L'EDSAC

L'Electronic Delay Storage Automatic Calculator (EDSAC), descendant de l'ENIAC, fut contruit en 1949. C'est une **machine de von Neumann**.



Sa mémoire utilisable est organisée en 1024 régions de 17 bits. Une instruction est représentée par un code 5 bits, suivi d'un bit de séparation, de 10 bits d'adresse et d'un bit de contrôle.

Le temps des machines à transistors

Cette période débuta dans les années 1950 et se prolonge encore aujourd'hui (en 2021).

D'un **point de vue matériel**, elle se base sur

- les transistors ;
- les circuits intégrés ;
- les microprocesseurs ;
- les mémoires SRAM et flash.

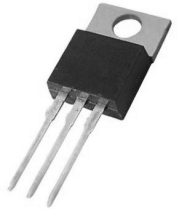
D'un **point de vue théorique**, il y a assez peu de différences par rapport à l'ère précédente. Les machines sont de von Neumann et Turing-complètes. Les principales avancées théoriques sont de nature algorithmique où de nombreuses découvertes ont été réalisées.

L'apparition des premiers langages de programmation est un signe annonciateur de cette nouvelle ère :

- le Fortran (Formula Translator) en 1957 ;
- le COBOL (Common Business Oriented Language) en 1959.

Le transistor

En 1948, Bardeen, Shockley et Brattain inventèrent le transistor.



C'est une **version améliorée du tube à vide** :

- élément moins volumineux et plus solide ;
- fonctionne sur de faibles tensions ;
- pas de préchauffage requis.

Ils peuvent être miniaturisés au point de pouvoir être assemblés en très grand nombre (plusieurs milliards) dans un très petit volume.

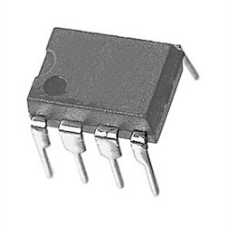
L'IBM 1401

L'IBM 1401 fut fabriqué de 1959 à 1965. Il fut l'une des machines à transistor les plus vendues de son époque.

Il pouvait réaliser 193000 additions de nombres de huit chiffres décimaux par seconde et disposait d'une mémoire d'environ 8 Kio.

Le circuit intégré

En 1958, Kilby inventa le circuit intégré.



C'est intuitivement un composant obtenu en connectant d'une certaine manière des transistors entre eux.

Son rôle est donc de regrouper dans un espace très réduit un très grand nombre de composants électroniques (transistors, portes logiques, *etc.*).

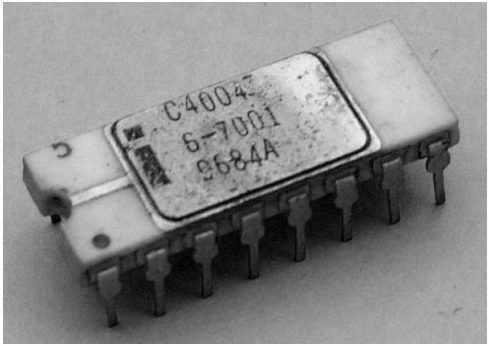
L'IBM 360

L'IBM 360 fut commercialisé dès 1966 et fut l'une des premières machines à circuits intégrés les plus célèbres de son époque.

Il pouvait accueillir jusqu'à 8 Mio de mémoire.

Le microprocesseur

Le premier microprocesseur fut conçut en 1971. Il s'agit de l'Intel 4004.



Il contenait 2300 transistors et sa fréquence était de 740 KHz.

Il fournissait une puissance équivalente à l'ENIAC.

Comparaison de quelques réalisations

Machine	Date	Fonctionnement	Base	T.-complet
Mach. analytique	1834	Mécanique	Décimal	Oui (!)
ABC	1937	Électronique	Binaire	Non
ASCC	1944	Électromécanique	Décimal	Non
ENIAC	1946	Électronique	Décimal	Oui
IBM 360	1966	Électronique	Binaire	Oui

Vers les ordinateurs d'aujourd'hui — matériel

Depuis l'invention du microprocesseur, les machines ont subi de nombreuses **évolutions matérielles** :

- **miniaturisation** de plus en plus poussée. Apparition des ordinateurs portables dans les années 1980, des PDA dans les années 1990, des smartphones et des tablettes dans les années 2010 ;
- meilleure **gestion de l'énergie**. La consommation énergétique d'un composant est une problématique importante. Les batteries offrent des capacités (mesurées en ampère-heure $A \cdot h$) de plus en plus grandes et les composants électroniques consomment de moins en moins ;
- plus grande **puissance de calcul** ;
- augmentation des quantités de **mémoire**. Ceci concerne les mémoires de stockage ainsi que les mémoires de travail ;
- meilleure **fiabilité**.

Vers les ordinateurs d'aujourd'hui — théorie

En parallèle, de nouvelles **connaissances théoriques** viennent se joindre à ces avancées :

- nouveaux **algorithmes**. De nombreuses découvertes théoriques améliorent, p.ex., l'efficacité des algorithmes de recherche de données, des calculs de chemins dans les graphes et des calculs algébriques pour la cryptographie ;
- nouveaux **langages de programmation** et paradigmes de programmation. Les langages deviennent de plus en plus abstraits (éloignés des considérations matérielles), ce qui permet de programmer des choses de plus en plus complexes et des projets de plus en plus conséquents ;
- apparition du **calcul parallèle** ;
- apparition des **réseaux** ;
- apparition des **machines virtuelles**.

Axe 1 : ordinateurs et codage des données

Histoire

Représentation

Bits

Pour qu'un ordinateur puisse traiter des données, il est nécessaire de les **représenter** de sorte qu'il puisse les « comprendre » et les manipuler.

Il est facile de représenter électroniquement deux états :

- l'état `0`, modélisé par l'**absence** de courant électrique ;
- l'état `1`, modélisé par la **présence** de courant électrique.

On appelle bit l'unité de base d'information, à valeur dans l'ensemble $\{0, 1\}$, symbolisant l'état `0` ou l'état `1`.

Suites de bits

Dans une suite de bits, on distingue deux bits particuliers :

Chaque bit d'une suite de n bits est implicitement **indicé** de 0 pour le bit de poids faible à $n-1$ pour le bit de poids fort :

Un bit d'indice plus grand (resp. petit) qu'un autre est dit de poids plus fort (resp. de poids plus faible).

Dans un ordinateur, les informations sont représentées par des **suites finies de bits**. On parle de représentation sur n bits lorsque l'on fixe le nombre de bits n pour représenter une donnée.

Représenter des informations

Pour qu'une suite de bits représente de l'information, il faut savoir comment l'**interpréter**. Une interprétation s'appelle un codage.

On placera, si possible pour chaque suite de symboles, le nom de son codage en indice.

Les principales informations que l'on souhaite représenter sont

- les entiers (positifs ou négatifs) ;
- les nombres réels ;
- les caractères ;
- les textes ;
- les instructions (d'un programme).

Pour chacun de ces points, il existe beaucoup de codages différents.

Leur raison d'être est que, selon la situation, un codage peut s'avérer meilleur qu'un autre (plus simple, plus économique, plus efficace).

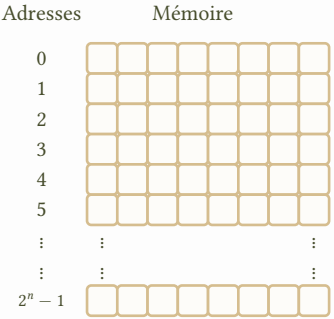
Types de données fondamentaux

Type	Taille
bit	1 bit
octet (byte)	8 bits
mot (word)	16 bits
double mot (dword)	32 bits
quadruple mot (qword)	64 bits
kibioctet (Kio)	$2^{10} = 1024$ octets
mébioctet (Mio)	$2^{20} = 1048576$ octets
gibioctet (Gio)	$2^{30} = 1073741824$ octets
kilooctet (Ko)	$10^3 = 1000$ octets
mégaoctet (Mo)	$10^6 = 1000000$ octets
gigaoctet (Go)	$10^9 = 1000000000$ octets

On utilisera de préférence les unités Kio, Mio et Gio à la place de Ko, Mo et Go.

Mémoire d'une machine

La mémoire d'un ordinateur est un tableau dont chaque case contient un octet.



Chaque octet de la mémoire est repéré par son adresse. C'est sa **position** dans le tableau.

Sur un **système *n* bits**, l'adressage va de 0 à $2^n - 1$ au maximum.

Boutisme

La structure octet par octet de la mémoire donne lieu au problème suivant : comment organiser en mémoire une suite u de bits constituée de plus de huit bits ?

Il existe pour cela deux conventions : le petit boutisme (*little-endian*) et le grand boutisme (*big-endian*). Les architectures qui nous intéressent sont en **petit boutisme**.

L'organisation se fait en trois étapes :

1. si u n'est pas constituée d'un nombre de bits multiple de huit, on ajoute des 0 à sa **gauche** de sorte qu'elle le devienne. On appelle u' cette nouvelle suite ;
2. on découpe u' en morceaux de huit bits
$$u' = u'_{k-1} \ \ldots \ u'_1 \ u'_0$$
où k est la longueur de u' divisée par 8 ;
3. en petit (resp. grand) boutisme, les morceaux sont placés, des petites aux grandes adresses, de u'_0 à u'_{k-1} (resp. de u'_{k-1} à u'_0) dans la mémoire.

Boutisme

– Exemple –

Pour

$$u := \texttt{11111 00000000 11111111},$$

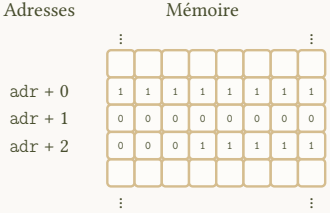
nous avons

$$u' = \texttt{00011111 00000000 11111111}$$

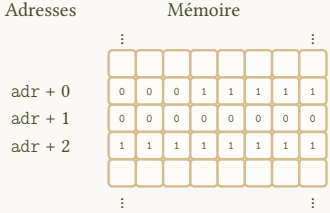
et

$$u'_2 = \texttt{00011111}, \qquad u'_1 = \texttt{00000000}, \qquad u'_0 = \texttt{11111111}.$$

Selon les deux conventions, le placement de u à l'adresse `adr` en mémoire donne lieu à



en petit boutisme



en grand boutisme

Codage unaire

Le moyen le plus simple de représenter un entier positif n consiste à le coder par une suite de n bits à 1. Ceci est le codage unaire de n.

– Exemple –

L'entier $(7)_{\text{dix}}$ est codé par la suite $(\texttt{1111111})_{\text{un}}$.

Avec ce codage, les **opérations arithmétiques** sont **très simples** :

- addition : concaténation des codages.

– Exemple –

$(7)_{\text{dix}} + (9)_{\text{dix}} = (\texttt{1111111})_{\text{un}} + (\texttt{111111111})_{\text{un}} = (\texttt{1111111 111111111})_{\text{un}}$

- produit de u et v : substitution de v à chaque 1 de u.

– Exemple –

$(3)_{\text{dix}} \times (5)_{\text{dix}} = (\texttt{111})_{\text{un}} \times (\texttt{11111})_{\text{un}} = (\texttt{11111 11111 11111})_{\text{un}}$

Avantages : opérations arithmétiques faciles sur les entiers positifs.

Inconvénients : mémoire en $\Theta(n)$ pour coder l'entier n ; codage des entiers positifs seulement.

Représentation Binary Coded Decimal (BCD)

Le codage BCD consiste à représenter un entier positif

$$(c_{n-1} \ \dots \ c_1 \ c_0)_{\text{dix}}$$

exprimé en base dix par la suite de $4n$ bits

$$(c'_{n-1} \ \dots \ c'_1 \ c'_0)_{\text{bcd}}$$

où chaque c'_i est le code BCD de c_i.

Celui-ci code chaque chiffre décimal par une suite de quatre bits au moyen de la table

$(0)_{\text{dix}} \to (0000)_{\text{bcd}}$	$(2)_{\text{dix}} \to (0010)_{\text{bcd}}$	$(4)_{\text{dix}} \to (0100)_{\text{bcd}}$	$(6)_{\text{dix}} \to (0110)_{\text{bcd}}$	$(8)_{\text{dix}} \to (1000)_{\text{bcd}}$
$(1)_{\text{dix}} \to (0001)_{\text{bcd}}$	$(3)_{\text{dix}} \to (0011)_{\text{bcd}}$	$(5)_{\text{dix}} \to (0101)_{\text{bcd}}$	$(7)_{\text{dix}} \to (0111)_{\text{bcd}}$	$(9)_{\text{dix}} \to (1001)_{\text{bcd}}$

– Exemple –

$(201336)_{\text{dix}} = (0010\ 0000\ 0001\ 0011\ 0011\ 0110)_{\text{bcd}}$

Représentation Binary Coded Decimal (BCD)

Avantages : représentation très simple des entiers et naturelle car très proche de la base dix.

Inconvénients : gaspillage de place mémoire (six suites de 4 bits ne sont jamais utilisées), les opérations arithmétiques ne sont ni faciles ni efficaces.

Ce codage est très peu utilisé dans les ordinateurs. Il est utilisé dans certains appareils électroniques qui affichent et manipulent des valeurs numériques (calculettes, réveils, *etc.*).

Changement de base — algorithme

Soient $b \geqslant 2$ un entier et x un entier positif. Pour écrire x en base b, on procède comme suit :

1. Si $x = 0$: renvoyer la liste $[0]$
2. Sinon :
   - 2.1 $L \leftarrow [\,]$
   - 2.2 Tant que $x \neq 0$:
     - 2.2.1 $L \leftarrow (x \,\%\, b) \cdot L$
     - 2.2.2 $x \leftarrow x/b$
   - 2.3 Renvoyer L.

Ici, $[\,]$ est la liste vide, $\cdot$ désigne la concaténation des listes, $\%$ est l'opérateur modulo et $/$ est la division entière.

Changement de base

– Exemple –

Avec $x := (294)_{\text{dix}}$ et $b := 3$, on a

x	$x \% 3$	$x/3$	L
$(294)_{\text{dix}}$	$(0)_{\text{dix}}$	$(98)_{\text{dix}}$	$[0]$
$(98)_{\text{dix}}$	$(2)_{\text{dix}}$	$(32)_{\text{dix}}$	$[2, 0]$
$(32)_{\text{dix}}$	$(2)_{\text{dix}}$	$(10)_{\text{dix}}$	$[2, 2, 0]$
$(10)_{\text{dix}}$	$(1)_{\text{dix}}$	$(3)_{\text{dix}}$	$[1, 2, 2, 0]$
$(3)_{\text{dix}}$	$(0)_{\text{dix}}$	$(1)_{\text{dix}}$	$[0, 1, 2, 2, 0]$
$(1)_{\text{dix}}$	$(1)_{\text{dix}}$	$(0)_{\text{dix}}$	$[1, 0, 1, 2, 2, 0]$
$(0)_{\text{dix}}$	–	–	$[1, 0, 1, 2, 2, 0]$

Ainsi, $(294)_{\text{dix}} = (101220)_{\text{trois}}$.

Valeur portée par une suite de chiffres

Soit $b \geqslant 2$ un entier, appelé base.

Soit

$$x := (x_{n-1}x_{n-2}\dots x_1x_0)_b$$

un nombre exprimé en base b. On a ainsi $x_{n-1} \neq 0$ et $0 \leqslant x_i \leqslant b-1$ pour tout $0 \leqslant i \leqslant n-1$.

Les x_i sont les chiffres de x. Il comporte donc n chiffres significatifs.

La valeur dénotée par x est l'entier naturel

$$\sum_{i=0}^{n-1} x_i \times b^i.$$

– Exemple –

La valeur dénotée par $(30042)_{\text{six}}$ est $2 \times 6^0 + 4 \times 6^1 + 3 \times 6^4 = (3914)_{\text{dix}}$.

Note : par convention, dans un calcul, les nombres sont exprimés en base dix.

Représentation binaire des entiers positifs

Le codage binaire consiste à représenter un entier positif en base deux.

En machine, si l'écriture d'un entier est plus longue que huit bits, il est d'usage de le représenter par tranches de huit bits.

Par ailleurs, si son écriture n'est pas de longueur multiple de huit, on complète généralement à gauche par des zéros de sorte qu'elle le soit.

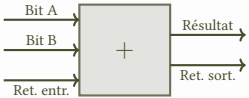
– Exemples –

- $(35)_{\text{dix}} = (100011)_{\text{deux}}$ est représenté par l'octet

- $(21329)_{\text{dix}} = (101001101010001)_{\text{deux}}$ est représenté par la suite de deux octets

L'addition d'entiers — l'additionneur simple

L'additionneur simple est un opérateur qui prend en entrée **deux bits** et une **retenue d'entrée** et qui renvoie deux informations : le **résultat** de l'addition et la **retenue de sortie**.



Il fonctionne selon la table

	Bit A	Bit B	Ret. entr.	Résultat	Ret. sort.
(0)	0	0	0	0	0
(1)	0	0	1	1	0
(1)	0	1	0	1	0
(2)	0	1	1	0	1
(1)	1	0	0	1	0
(2)	1	0	1	0	1
(2)	1	1	0	0	1
(3)	1	1	1	1	1

L'addition d'entiers — l'algorithme usuel

Un additionneur n bits fonctionne de la même manière que l'**algorithme d'addition usuel**.

Il effectue l'addition chiffre par chiffre, en partant de la droite et en reportant les retenues de proche en proche.

– Exemple –

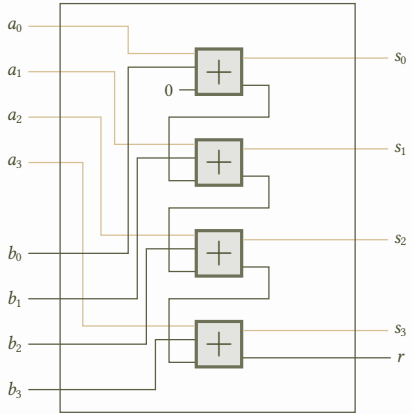
L'addition binaire de $(159)_{\text{dix}}$ et de $(78)_{\text{dix}}$ donne $(237)_{\text{dix}}$:

$$\begin{array}{cccccccccc} & & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ + & & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ \hline & & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}$$

La retenue de sortie est 0.

L'addition d'entiers — l'additionneur 4 bits

On construit un additionneur 4 bits en combinant quatre additionneurs simples :



La multiplication d'entiers — le multiplicateur simple

Le **multiplicateur simple** est un opérateur qui prend **deux bits** en entrée et qui renvoie un **résultat**.

Il fonctionne selon la table

Bit A	Bit B	Résultat
0	0	0
0	1	0
1	0	0
1	1	1

Note : ce multiplicateur simple binaire n'a pas de retenue de sortie, contrairement au multiplicateur simple décimal.

La multiplication d'entiers — l'algorithme usuel

Le multiplicateur simple permet de réaliser la multiplication bit à bit et ainsi, la multiplication de deux entiers codés en binaire, selon l'**algorithme de multiplication usuel**.

– Exemple –

La multiplication binaire de $(19)_{\text{dix}}$ et de $(44)_{\text{dix}}$ donne $(836)_{\text{dix}}$:

						1	0	0	1	1
$\times$					1	0	1	1	0	0
						0	0	0	0	0
					0	0	0	0	0	·
				1	0	0	1	1	·	·
			1	0	0	1	1	·	·	·
		0	0	0	0	0	·	·	·	·
$+$	1	0	0	1	1	·	·	·	·	·
	1	1	0	1	0	0	0	1	0	0

Bit de signe, codage de taille fixée et plage

Pour l'instant, nous avons vu uniquement la représentation d'entiers positifs.

On peut représenter des entiers négatifs et positifs à l'aide d'un bit de signe. C'est le **bit de poids fort** qui renseigne le signe de l'entier

- s'il est égal à 1, l'entier codé est négatif 

;
- s'il est égal à 0, l'entier codé est positif 

.

Dorénavant, on précisera toujours le **nombre n de bits** sur lequel on code les entiers.

Si un entier requiert moins de n bits pour être codé, on **complétera** son codage avec des 0 ou des 1 à gauche en fonction de son signe et de la représentation employée.

À l'inverse, si un entier x demande strictement plus de n bits pour être codé, on dira que x ne peut pas être codé sur n bits.

La plage d'un codage désigne l'ensemble des valeurs qu'il est possible de représenter sur n bits.

Représentation en magnitude signée

Le codage en magnitude signée permet de représenter un entier en le codant en binaire et en lui adjoignant un **bit de signe**.

– Exemples –

En se plaçant sur 8 bits, on a $(45)_{\text{dix}} = (00101101)_{\text{ms}}$ et $(-45)_{\text{dix}} = (10101101)_{\text{ms}}$.

Avantage : calcul facile de l'opposé d'un entier.

Inconvénient : l'addition de deux entiers ne peut pas se faire par l'algorithme classique d'addition.

Il y a de plus deux encodages différents pour zéro (qui est positif et négatif) :

$$(0)_{\text{dix}} = (00\ldots0)_{\text{ms}} = (10\ldots0)_{\text{ms}}.$$

Plage (sur n bits) :

plus petit entier : $(11\ldots1)_{\text{ms}} = -(2^{n-1}-1)$;

plus grand entier : $(01\ldots1)_{\text{ms}} = 2^{n-1}-1$.

Représentation en complément à un

Le complément à un d'une suite de bits

$$u_{n-1} \dots u_1 u_0$$

est la suite

$$\overline{u_{n-1}} \dots \overline{u_1}\ \overline{u_0},$$

où $\overline{0} := 1$ et $\overline{1} := 0$.

Le codage en complément à un consiste à coder un entier négatif par le complément à un de la représentation binaire de sa valeur absolue.

Le codage d'un entier positif est son codage binaire habituel.

Le bit de poids fort doit être un bit de signe : lorsqu'il est égal à 1, l'entier représenté est négatif ; il est positif dans le cas contraire.

Remarque : sur n bits, le complément à un revient à représenter un entier négatif x par la représentation binaire de $(2^n - 1) - |x|$.

Représentation en complément à un

– Exemples –

Sur 8 bits, on a $(98)_{\text{dix}} = (\texttt{01100010})_{\text{c1}}$ et $(-98)_{\text{dix}} = (\texttt{10011101})_{\text{c1}}$.

Avantage : calcul facile de l'opposé d'un entier.

Inconvénient : l'addition de deux entiers ne peut pas se faire par l'algorithme classique d'addition.

Il y a de plus deux encodages différents pour zéro (qui est positif et négatif) :

$$(0)_{\text{dix}} = (\texttt{00}\ldots\texttt{0})_{\text{c1}} = (\texttt{11}\ldots\texttt{1})_{\text{c1}}.$$

Plage (sur n bits) :

plus petit entier : $(\texttt{10}\ldots\texttt{0})_{\text{c1}} = -(2^{n-1}-1)$;

plus grand entier : $(\texttt{01}\ldots\texttt{1})_{\text{c1}} = 2^{n-1}-1$.

Représentation avec biais

On fixe un entier $B \geqslant 0$ appelé biais.

Soit x un entier (positif ou non) que l'on souhaite représenter.

Posons $x' := x + B$.

Si $x' \geqslant 0$, alors le codage de x avec un biais de B est la représentation binaire de x' sur n bits. Sinon, x n'est pas représentable (sur n bits et avec le biais B donné).

– Exemples –

En se plaçant sur 8 bits, avec un biais de $B := 95$, on a

- $(30)_{\text{dix}} = (\texttt{01111101})_{\text{biais}=95}$. En effet, $30 + 95 = 125$ et $(\texttt{01111101})_{\text{deux}} = (125)_{\text{dix}}$.
- $(-30)_{\text{dix}} = (\texttt{01000001})_{\text{biais}=95}$. En effet, $-30 + 95 = 65$ et $(\texttt{01000001})_{\text{deux}} = (65)_{\text{dix}}$.
- l'entier $(-98)_{\text{dix}}$ n'est pas représentable car $-98 + 95 = -3$ est négatif.

Représentation avec biais

Note : ce codage n'est pas compatible avec le bit de signe. En effet, le bit de poids fort d'un entier codé en représentation avec biais ne renseigne pas sur le signe de l'entier.

Avantages : permet de représenter des intervalles quelconques (mais pas trop larges) de $\mathbb{Z}$.

Inconvénient : l'addition de deux entiers ne peut pas se faire par l'algorithme classique d'addition.

Plage (sur n bits) :

plus petit entier : $(0 \dots 0)_{\text{biais=B}} = -B$;

plus grand entier : $(1 \dots 1)_{\text{biais=B}} = (2^n - 1) - B$.

Représentation en complément à deux

On se place sur n bits.

Le complément à deux (puissance n) d'une suite de bits u se calcule en

1. complémentant u à un pour obtenir u' ;
2. incrémentant u' (addition de u' et $0 \dots 01$) pour obtenir u''.

– Exemple –

Sur 8 bits, avec $u := 01101000$, on obtient

1. $u' = 10010111$;
2. $u'' = 10011000$.

Ainsi, le complément à deux de la suite de bits 01101000 est la suite de bits 10011000.

Remarque : l'opération qui consiste à complémenter à deux une suite de bits est involutive (le complément à deux du complément à deux d'une suite de bits u est u).

Représentation en complément à deux — codage/décodage

Le codage en complément à deux consiste à **coder** un entier x de la manière suivante.

- Si $x \geqslant 0$, le codage de x est simplement son codage binaire habituel.
- Sinon, $x < 0$. Le codage de x est le complément à deux du codage binaire de la valeur absolue de x.

Pour **décoder** une suite de bits u représentant un entier codé en complément à deux, on procède de la manière suivante.

- Si le bit de poids fort de u est 0, on obtient la valeur codée par u en interprétant directement sa valeur en binaire.
- Sinon, le bit de poids fort de u est 1. On commence par considérer le complément à deux u'' de u. La valeur codée par u est obtenue en interprétant la valeur de u'' en binaire et en considérant son opposé.

Représentation en complément à deux — remarques

Le codage en complément à deux fait que le **bit de poids fort** est un **bit de signe** : lorsque le bit de poids fort est 1, l'entier représenté est négatif. Il est positif dans le cas contraire.

Un entier x est codable sur n bits en complément à deux si et seulement si la suite de bits obtenue par le codage de x en complément à deux sur n bits possède un **bit de signe cohérent** avec le signe de x.

Remarque : sur n bits, le complément à deux revient à représenter un entier strictement négatif x par $2^n - |x|$.

Représentation en complément à deux

– Exemples –

- **Représentation de $(98)_{\text{dix}}$ sur 8 bits.** On a $(98)_{\text{dix}} = (01100010)_{\text{deux}}$. Cohérence avec le bit de signe. Ainsi, $(98)_{\text{dix}} = (01100010)_{\text{c2}}$.
- **Représentation de $(-98)_{\text{dix}}$ sur 8 bits.** Le complément à deux de 01100010 est 10011110. Cohérence avec le bit de signe. Ainsi, $(-98)_{\text{dix}} = (10011110)_{\text{c2}}$.
- **Représentation de $(130)_{\text{dix}}$ sur 8 bits.** On a $(130)_{\text{dix}} = (10000010)_{\text{deux}}$. Incohérence avec le bit de signe : il est à 1 alors que l'entier est positif. Cet entier n'est pas représentable sur 8 bits.
- **Représentation de $(-150)_{\text{dix}}$ sur 8 bits.** On a $(150)_{\text{dix}} = (10010110)_{\text{deux}}$. Le complément à deux de cette suite est 01101010. Incohérence avec le bit de signe : il est à 0 alors que l'entier est négatif. Cet entier n'est pas représentable sur 8 bits.
- **Représentation de $(-150)_{\text{dix}}$ sur 10 bits.** On a $(150)_{\text{dix}} = (0010010110)_{\text{deux}}$. Le complément à deux de cette suite est 1101101010. Cohérence avec le bit de signe. Ainsi, $(-150)_{\text{dix}} = (1101101010)_{\text{c2}}$.

Représentation en complément à deux

– Exemples –

- **Décodage de $(00110101)_{c2}$ sur 8 bits.** Le bit de poids fort est 0. Ainsi, l'entier codé est positif et $(00110101)_{c2} = (00110101)_{deux}$. On a donc $(00110101)_{c2} = (53)_{dix}$.
- **Décodage de $(10110101)_{c2}$ sur 8 bits.** Le bit de poids fort est 1. Ainsi, l'entier codé est négatif. On doit considérer le complément à deux de la suite 10110101 qui est 01001011. On a maintenant $(01001011)_{deux} = (75)_{dix}$. Ainsi, $(10110101)_{c2} = (-75)_{dix}$.
- **Décodage de $(11111111)_{c2}$ sur 8 bits.** Le bit de poids fort est 1. Ainsi, l'entier codé est négatif. On doit considérer le complément à deux de la suite 11111111 qui est 00000001. On a maintenant $(00000001)_{deux} = (1)_{dix}$. Ainsi, $(11111111)_{c2} = (-1)_{dix}$.

Représentation en complément à deux

Avantage : l'addition de deux entiers se calcule par l'algorithme classique d'addition.

Inconvénient : représentation des entiers négatifs légèrement plus complexe que par les autres codages.

Plage (sur n bits) :

plus petit entier : $(\texttt{10}\ldots\texttt{0})_{\text{c2}} = -2^{n-1}$;

plus grand entier : $(\texttt{01}\ldots\texttt{1})_{\text{c2}} = 2^{n-1} - 1$.

Dans les ordinateurs d'aujourd'hui, les entiers sont habituellement encodés selon cette représentation.

Représentation en complément à deux — calcul rapide

Il existe une **méthode plus rapide** pour calculer le complément à deux d'une suite de bits u que de la complémenter à un et de l'incrémenter.

Elle consiste en les deux étapes suivantes :

1. repérer le bit à 1 de poids le plus faible de u ;
2. complémenter à un les bits de u qui sont de poids strictement plus forts que ce bit.

– Exemple –

L'application de cette méthode sur la suite 01101000 donne :

u	0	1	1	0	1	0	0	0
bit à 1 à droite	0	1	1	0	**1**	0	0	0
cpl. à 1 bits à gauche	1	0	0	1	**1**	0	0	0

La dernière ligne est le complément à deux de u.

Dépassement de capacité

En représentation en complément à deux sur n bits, l'addition de deux entiers x et y peut produire un entier qui sort de la plage représentable. On appelle ceci un dépassement de capacité (*overflow*).

Fait 1. Si x est négatif et y est positif, $x + y$ appartient toujours à la plage représentable.

Fait 2. Il ne peut y avoir dépassement de capacité que si x et y sont de **même signe**.

Règle : il y a dépassement de capacité si et seulement si le **bit de poids fort** de $x + y$ est **différent** du bit de poids fort de x (et donc de y).

– Exemple –

Sur 4 bits, l'addition

$$\begin{array}{rcccc} & 0 & 1 & 0 & 0 \\ + & 0 & 1 & 0 & 1 \\ \hline & 1 & 0 & 0 & 1 \end{array}$$

engendre un dépassement de capacité.

Retenue de sortie

En représentation en complément à deux sur n bits, on dit que l'addition de deux entiers x et y produit une **retenue de sortie** si l'addition des bits de poids forts de x et de y renvoie une retenue de sortie à 1.

– Exemple –

Sur 4 bits, l'addition

$$\begin{array}{cccccc} & & 1 & 1 & 0 & 0 \\ + & & 0 & 1 & 0 & 1 \\ \hline & (1) & 0 & 0 & 0 & 1 \end{array}$$

engendre une retenue de sortie.

Attention : ce n'est pas parce qu'une addition provoque une retenue de sortie qu'il y a dépassement de capacité.

Dépassement de capacité et retenue de sortie

Le dépassement de capacité **(D)** et la retenue de sortie **(R)** sont deux choses disjointes. En effet, il peut exister les quatre cas de figure.

– Exemple –

Sur 4 bits :

- non **(D)** et non **(R)** :

		1	0	0	0
+		0	0	1	0
	(0)	1	0	1	0

- non **(D)** et **(R)** :

		0	1	1	0
+		1	1	0	0
	(1)	0	0	1	0

- **(D)** et non **(R)** :

		0	1	1	0
+		0	1	1	1
	(0)	1	1	0	1

- **(D)** et **(R)** :

		1	0	1	0
+		1	0	0	0
	(1)	0	0	1	0

Résumé des représentations des entiers

- **Représentation en magnitude signée**.
  Plage : de $-2^{n-1}+1$ à $2^{n-1}-1$.
  Avantage : codage simple ; calcul facile de l'opposé.
  Inconvénients : addition non naturelle ; deux codages de zéro.
- **Représentation en complément à un**.
  Plage : de $-2^{n-1}+1$ à $2^{n-1}-1$.
  Avantage : codage simple ; calcul facile de l'opposé.
  Inconvénients : addition non naturelle ; deux codages de zéro.
- **Représentation avec biais** (de $B \geqslant 0$).
  Plage : de $-B$ à $(2^n-1)-B$.
  Avantage : possibilité d'encoder un intervalle arbitraire.
  Inconvénient : addition non naturelle.
- **Représentation en complément à deux**.
  Plage : de -2^{n-1} à $2^{n-1}-1$.
  Avantage : addition naturelle.
  Inconvénient : codage moins intuitif.