Architecture des Ordinateurs

Mounir T. El Araki

mounir.elarakitantaoui@uic.ac.ma

CPI 1

Organisation du cours

- 12 séances de 2 heures
- 2 Contrôles Continus (40%)
- 1 Examen Final (50%)
- 1 note de présence (10%)
- 2 TD pendant les séances de cours

Plan du cours

- Historique
- Présentation de l'architecture des ordinateurs
- Représentation interne des informations
- Encodage/décodage de l'information
- Circuits logiques
- Mémoires
- Unité centrale de traitement

Objectif du cours

- Compréhension de l'organisation des ordinateurs.
- Modélisation du fonctionnement des ordinateurs.
- Représentation des données.
- Représentation du calcul arithmétique et logique.
- Compréhension des fondements des traitements des programmes par les ordinateurs.

Plan du cours

- **Historique**
- Présentation de l'architecture des ordinateurs
- Représentation interne des informations
- Encodage/décodage de l'information
- Circuits logiques
- Mémoires
- Unité centrale de traitement

Objectif de l’historique

- Prendre connaissance de l'évolution des ordinateurs depuis leurs débuts.
- Connaitre l'origine des différents systèmes de numération utilisés
  - binaire, décimal, etc;
- Comprendre l’utilisation du binaire en informatique;
- Connaitre les composantes essentielles de la Machine de von Neumann.

Références

- Unités de mesure de capacité
  - Kilo = $10^3 \approx 2^{10}$ = 1024
  - Méga = $10^6 \approx 2^{20}$ = 1 048 576
  - Giga = $10^9 \approx 2^{30}$ = 1 073 74 824
  - Tera = $10^{12} \approx 2^{40}$ = 1 099 511 627 776
  - Peta = $10^{15} \approx 2^{50}$ = 1 125 899 906 842 624

- Unités de mesure de temps
  - ms = milliseconde = 10^{-3} s = 0,001 s
  - _µs = microseconde = 10^{-6} s = 0,000 0001 s
  - ns = nanoseconde = 10^{-9} s = 0,000 000 001 s
  - ps = picoseconde = 10^{-12} s = 0,000 000 000 001 s

Historique

- L'ordinateur est né pour répondre à un besoin de calcul plus vite
  - Automatisation du calcul

- 18ième siècle et avant : les principes fondateurs
- 19ième siècle : les calculateurs
- 20ème siècle : théorie de l'information et la machine universelle
- 1945 : Architecture de Von Neumann et naissance de l'ordinateur
- Les années 1950 : 1ere génération : tubes a vides
- Les années 1960 : 2eme génération : transistors
- Les années 1970 : 3eme génération : circuits intègres
- Les années 1980 : 4eme génération : puces avec des millions de transistors

Les principes fondateurs

- Al Khawarizmi (870): Apparition de l'Algèbre

- John Napier (1614) : théorie de logarithmes permettant de transformer des multiplications en additions (Bâtons de Napier)

- Blaise Pascal (1642) : première machine a calculer, la Pascaline (principe de roues dentées). Machine pouvait additionner et soustraire des nombres en prenant en compte les retenues. (Principe de complément).

- Gottfried Leibniz (1673): améliore la machine de Pascal en y ajoutant un mécanisme permettant d'automatiser l'‘exécution répétitive d'additions et de soustraction.
  - Première machine avec les opérations de +, -, *, et \ arithmétiques.
  - Système binaire (basé sur des 0 et 1)
  - Puissance et la simplicité de l'arithmétique binaire, système utilisé par les ordinateurs actuels.

Les calculateurs (1/2)

- Joseph Jacquard (1801) : (le métier à tisser) des cartes perforées (programme) pour sélectionner les fils a tisser

- Charles Babbage (1822 -1833) : La machine analytique.
  - Père de l'ordinateur (rapprochement entre commande externe & machine calculer)
  - Moulin faisant les calculs (Processeur), Magasin stockant les chiffres (Mémoires), puis les résultats imprimés, des cartes perforées (métier à tisser) donnant les instructions (logiciel)
  - Réalisation de sa machine analytique avec l'aide Augusta Ada King (Ada Lovelace), l'ancêtre des ordinateurs.

- George Boole (1854) : Algèbre de Boole (booléenne)
  - **Système de logique symbolique** : Fonctions logiques permettant de modéliser des raisonnements logiques, en exprimant un « état » en fonction de conditions.
  - Fonction de Base : 'Complémentarité', 'ET', et 'OU'.

Les calculateurs (2/2)

- Herman Hollerith (1890) : Calculateur statistique : Cartes perforées
  - Premiers supports d'entrée-sortie et premières mémoires de masse.
  - Invention du premier clavier.
  - Les tabulatrices : Machines pour recensement (Bureau de recensement)
  - 1911 : 'Computing Tabulating Records (CTR) company'
  - 1924 : CTR devient IBM

Naissance de l'ordinateur

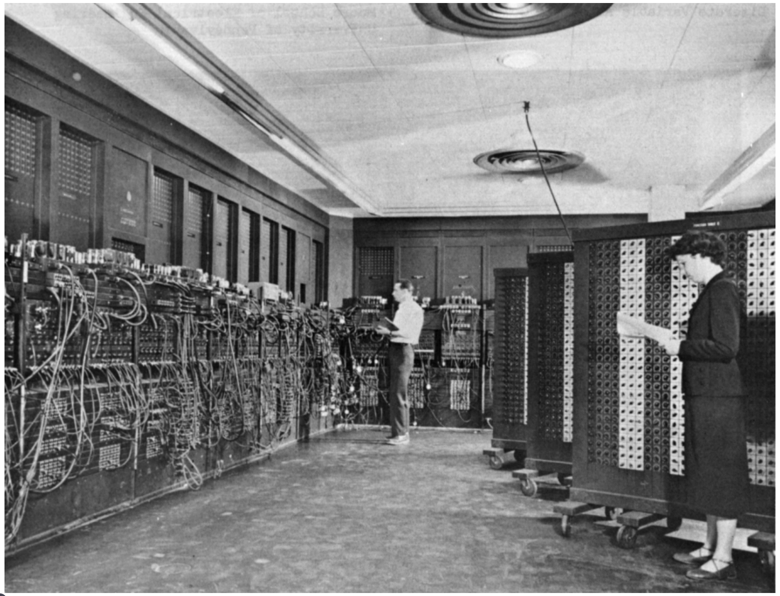
- Claude Shannon (1948) : Utilisation du binaire pour les calculs logiques et arithmétiques
  - Tous calculs sont réalises avec les 3 opérations logiques de base NOT, AND, OR.
  - Communication de Shannon : Source, Encodeur, Signal, Décodeur, Destinataire
  - Théories de l'information et de communication

- Alan Turing : Machine de Turing (ou Machine universelle)
  - Décrivant un modèle abstrait du fonctionnement des appareils mécaniques de calcul
  - Concepts de programmation et de programme

- John Von Neumann (1945) :
  - Enregistrer le programme en mémoire Architecture de l'ordinateur moderne : l'architecture de Von Neumann

Premier ordinateur (ENIAC)

- Electronic Numerical Integrator Analyser and Calculator –(ENIAC) 1945



- Construit a l'Université de Pennsylvanie

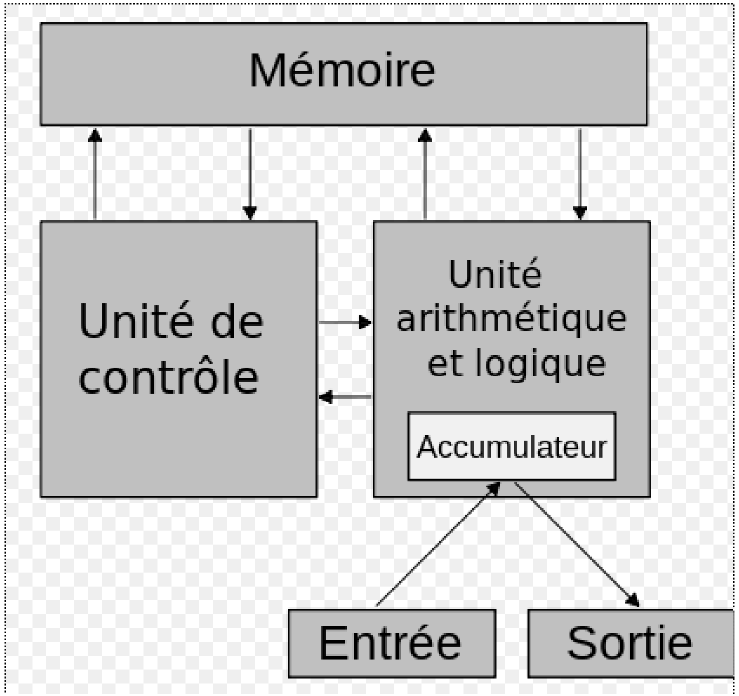
- Technologie des tubes a vide (17468), 7200 diodes, 70000 résistance, 10000 capacités
  - pesant 30 tonnes !
  - occupant167 m^2

- Construit pour être Turing-complet
  - Peut simuler toutes les machines de Turing à une Bande

- Multiplication de 2 nombres de 10 chiffres en 3ms !

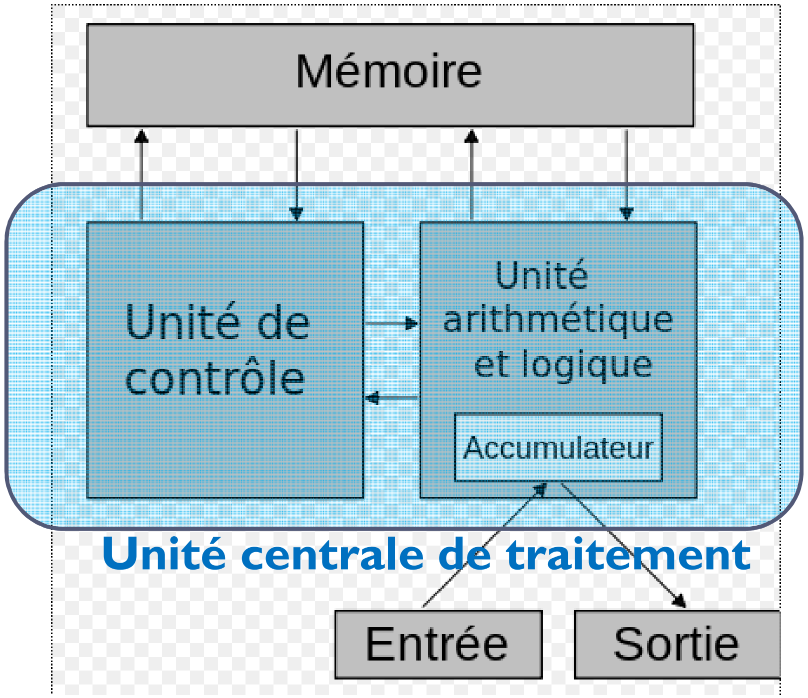
Architecture de Von Neumann (1/2)

- Machine universelle contrôlée par un programme.
- Les instructions du programme sont codées sous forme numérique binaire et enregistrées en mémoire.
- Les instruction sont exécutées en séquence mais peuvent être modifiées par le programme lui-même.
- Existence d'instructions permettant les ruptures de séquences.



Architecture de Von Neumann (2/2)

- La mémoire ou (mémoire centrale) qui contient **les données** et **les programmes** à exécuter
- L'unité centrale de traitement qui exécute les programmes chargés en mémoire
- Les unités d'entrée/sortie qui facilitent les échanges d'information avec tous types de périphériques (écran, clavier, souris, imprimante, etc.)



L’industrie informatique (1/2)

- 1950 : 1ère génération : tubes a vides
  - L’IBM 701 utilisait une mémoire à tubes cathodiques de 2 048 mots de 36 bits.
  - Effectuait 16 000 additions par seconde

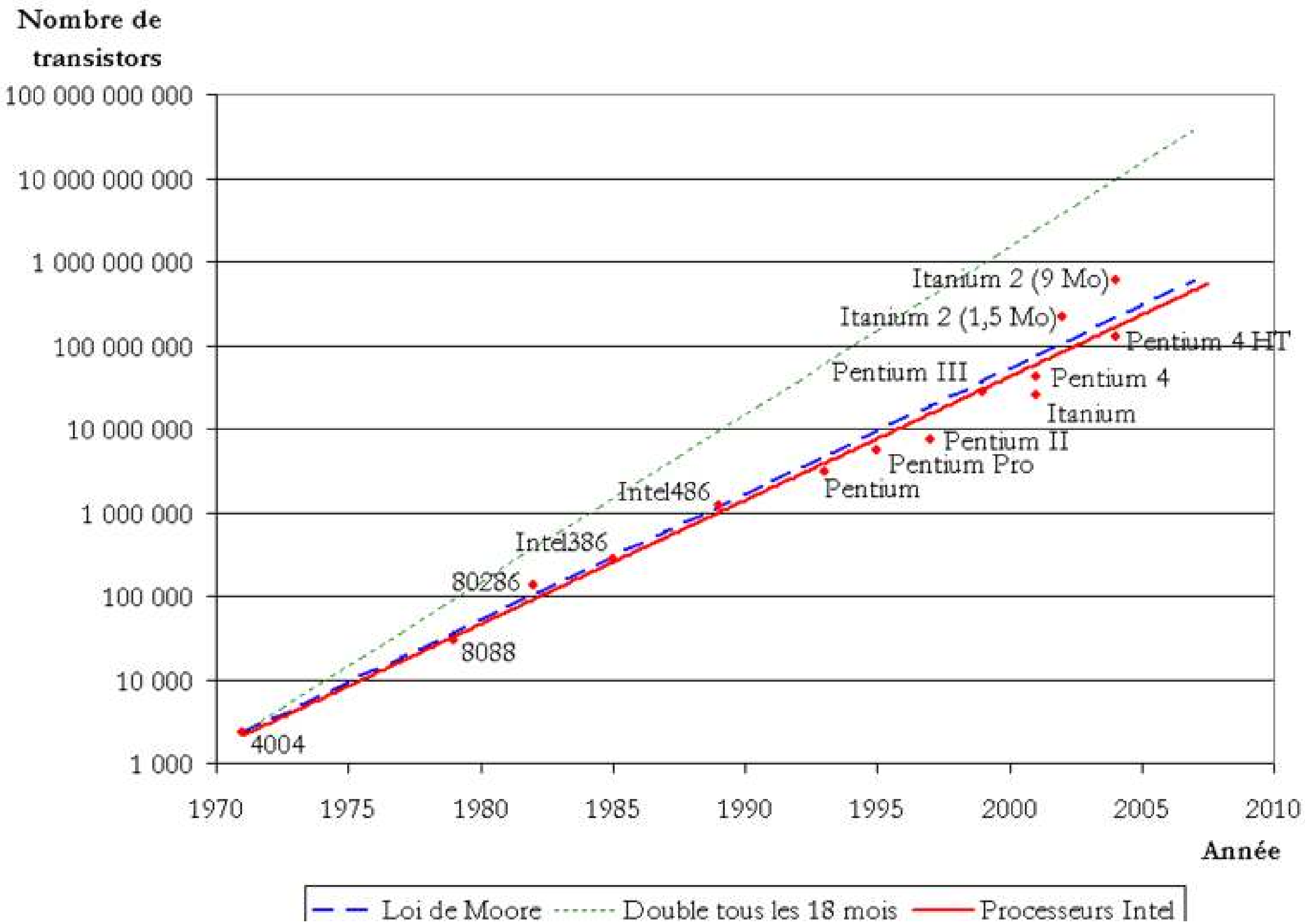
- 1960 : 2ième génération : transistors.
  - Moindre coût, bus unique pour interconnecter les différents composants
  - Le PDP-1 (Programmed Data Processor) de DEC premier ordinateur interactif (concept de mini-ordinateur).
  - Vitesse d’horloge de 0,2 MHz , et stockage de 4 096 mots de 18 bits.
  - Effectuait 100 000 opérations par seconde.
  - Vendu 120 000 $ environ

L'industrie informatique (2/2)

- 1970 : 3ième génération : circuits intégrés. (*Boom informatique*)
  - Utilisation du premier circuit intégrés dans les systèmes embarqués de la NASA
  - Multi-programmation (plusieurs programmes en mémoire)) dès qu'un programme est en attente d'une entrée-sortie, l'unité de commande poursuit l'exécution d'un autre programme.
  - IBM série 360, première gamme commercial et scientifique.
  - 14 000 ordinateurs IBM 360 vendus
  - HP-2116 , avec mémoire 16 bits, supportant l'Algol, le Fortran et le BASIC.
  - Mini-ordinateurs Série 3 d'IBM, Séries 30 puis AS/400
  - DEC PDP-11 (le langage C)

- 1980 : 4ième génération : puces avec des millions de transistors (Very-large-scale integration (VLSI)
  - multiplication des unités péripheriques (stockage, écran, imprimante, etc.)
  - langages évolués de deuxième génération (Pascal et C++ langage objets) et langages d'interrogation de très haut niveau comme SQL.

Loi de Moore


Nombre de transistors
100 000 000 000
10 000 000 000
1 000 000 000
100 000 000
10 000 000
1 000 000
100 000
10 000
1 000
1970
1975
1980
1985
1990
1995
2000
2005
2010
Année
Itanium 2 (9 Mo)
Itanium 2 (1,5 Mo)
Pentium 4 HT
Pentium III
Pentium 4
Itanium
Pentium II
Pentium Pro
Pentium
Intel486
Intel386
80286
8088
4004
Loi de Moore
Double tous les 18 mois
Processeurs Intel

Evolution des microprocesseurs Intel

Date	Nom	Nombre de transistors	Finesse de gravure (µm)	Fréquence de l'horloge	Largeur des données	MIPS
1971	4004	2 300		108 kHz	4 bits/4 bits bus	
1974	8080	6 000	6	2 MHz	8 bits/8 bits bus	0,64
1979	8088	29 000	3	5 MHz	16 bits/8 bits bus	0,33
1982	80286	134 000	1,5	6 à 16 MHz (20 MHz chez AMD)	16 bits/16 bits bus	1
1985	80386	275 000	1,5	16 à 40 MHz	32 bits/32 bits bus	5
1989	80486	1 200 000	1	16 à 100 MHz	32 bits/32 bits bus	20
1993	Pentium	3 100 000	0,8 à 0,28	60 à 233 MHz	32 bits/64 bits bus	100
1997	Pentium II	7 500 000	0,35 à 0,25	233 à 450 MHz	32 bits/64 bits bus	300
1999	Pentium III	9 500 000	0,25 à 0,13	450 à 1 400 MHz	32 bits/64 bits bus	510
2000	Pentium 4	42 000 000	0,18 à 0,065	1,3 à 3,8 GHz	32 bits/64 bits bus	1 700
2004	Pentium 4D « Prescott »	125 000 000	0,09 à 0,065	2.66 à 3,6 GHz	32 bits/64 bits bus	9 000
2006	Core 2™ Duo	291 000 000	0,065	2,4 GHz (E6600)	64 bits/64 bits bus	22 000
2007	Core 2™ Quad	2*291 000 000	0,065	3 GHz (Q6850)	64 bits/64 bits bus	2*22 000 (?)
2008	Core 2™ Duo (Penryn)	410 000 000	0,045	3,33 GHz (E8600)	64 bits/64 bits bus	~24 200
2008	Core 2™ Quad (Penryn)	2*410 000 000	0,045	3,2 GHz (QX9770)	64 bits/64 bits bus	~2*24 200
2008	Intel Core i7 (Nehalem)	731 000 000	0,045 (2008) 0,032 (2009)	2,66 GHz (Core i7 920) 3,33 GHz (Core i7 Ext. Ed. 975)	64 bits/64 bits bus	?
2009	Intel Core i5/i7 (Lynnfield)	774 000 000	0,045 (2009)	2,66 GHz (Core i5 750) 2,93 GHz (Core i7 870)	64 bits/64 bits bus	?
2010	Intel Core i7 (Gulftown)	1 170 000 000	0,032	3,33 GHz (Core i7 980X)	64 bits/64 bits bus	?

Ordinateur

256 MB , 1 GB, 2 GB

b : bit
B : Byte (= octet = 8 bits)
bps : bits par sec
Mbps : mégabits par sec
MB ou Mbytes ou Mo
GB ou Gbytes ou Go

(disque dur, clé USB)
250 MB à 10 GB

Stockage

Mémoire
RAM

Intel : Pentium
SGI-MIPS: R10000
Sun: UltraSparc
Motorola-IBM-Apple: PowerPC
200-500 MHz

CPU

Bus

SCSI: 4 Mbps
Firewire : 400 Mbps

Réseau

Logiciel

Ethernet: 10 Mbps
Fast Ethernet; 100 Mbps
FDDI: 100 Mbps
ATM: 155-622 Mbps
Gigabit Ethernet : 1 Gbps

Systèmes d’exploitation
Base de données
Navigateur Internet

Principaux acteurs du monde informatique

Cisco
3Com
Bay Networks
Novell

Réseau (Networking)

Microsoft
Oracle
IBM
SAS

Logiciel (Applications)

Traitement (Computing)

Intel
IBM
Motorola
Sun
HP
NEC
Hitachi
Fujitsu
Apple

Utilisation des ordinateurs

- **Développement de programme**
  - Programmes système : fonctions de base de l'ordinateur
    - Système d'exploitation
  - Programmes d'application
    - Calcul scientifique
    - Gestion
    - Conduite de processus

- **Développement de logiciel**
  - Langages de programmation
  - Compilation, édition de liens et chargement
  - Programme, algorithme
  - Cycle de vie du logiciel
    - Compréhension du problème
    - Spécification du système
    - Conception
    - Programmation
    - Tests et validation
    - Entretien ou maintenance

Plan du cours

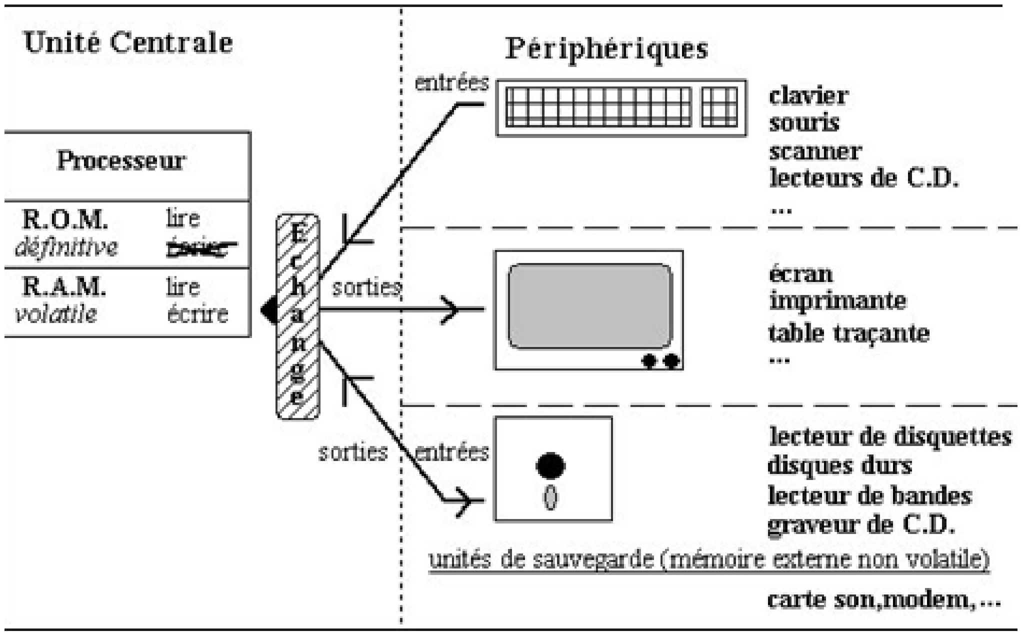
- Historique
- Présentation de l’architecture des ordinateurs
- Représentation interne des informations
- Encodage/décodage de l’information
- Circuits logiques
- Mémoires
- Unité centrale de traitement

Objectif

- Comprendre les composants d’un ordinateur
- Définir le vocabulaire des composants d’un ordinateur
- Comprendre les étapes d’exécution d’un programme informatique

Principaux éléments

- Unité centrale
- Cédérom ou DVD
- Disque dur
- Disquette
- Clavier
- Souris
- Écran ou projecteur
- Modem
- Scanner
- Carte de son
- Images et vidéo
- Fax modem
- Ports SCSI, USB, etc.



Type de Réseaux et Station de travail

Réseaux

- WAN
- MAN
- LAN
- WWW (Internet)

Définitions

- Ordinateur
  - Machine de traitement de l'information (acquiert l'information la stocke , la traite et la restitue).

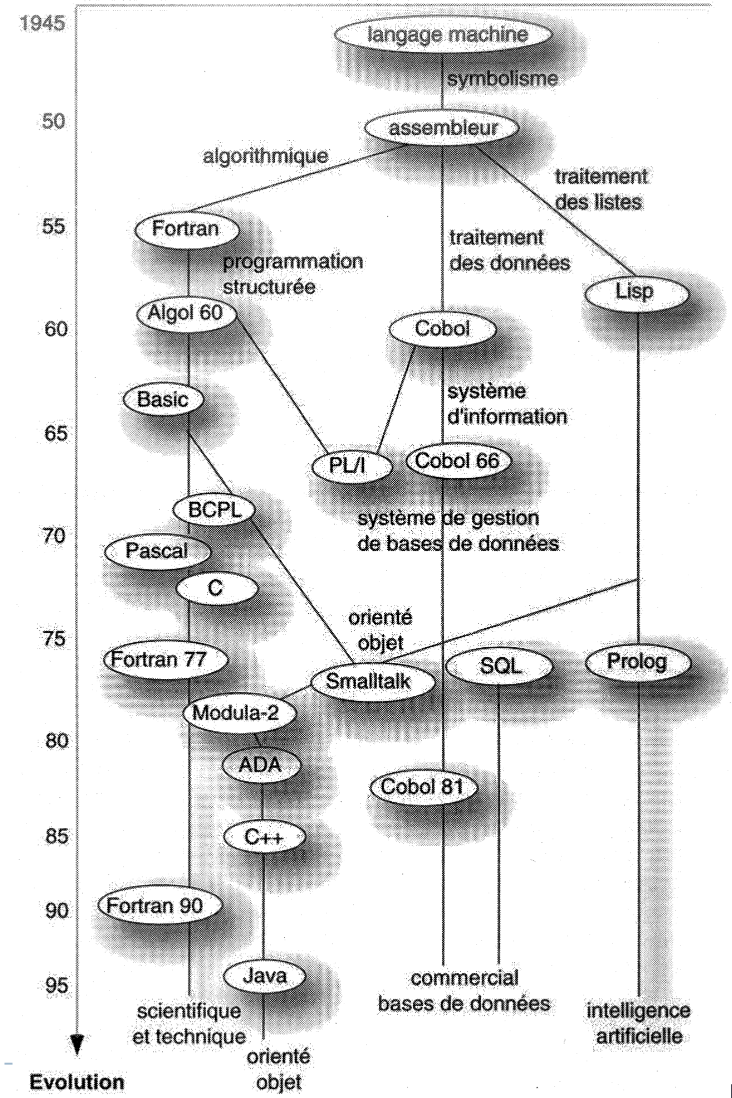
- Type d'informations
  - Données à valeurs numériques,
  - Données textuelles ,
  - Données numériques représentant des images, du son, et des vidéos.

- Système Informatique
  - Ensemble des moyens logiciels & matériels (ordinateur) nécessaires pour satisfaire les besoins informatiques des utilisateurs.

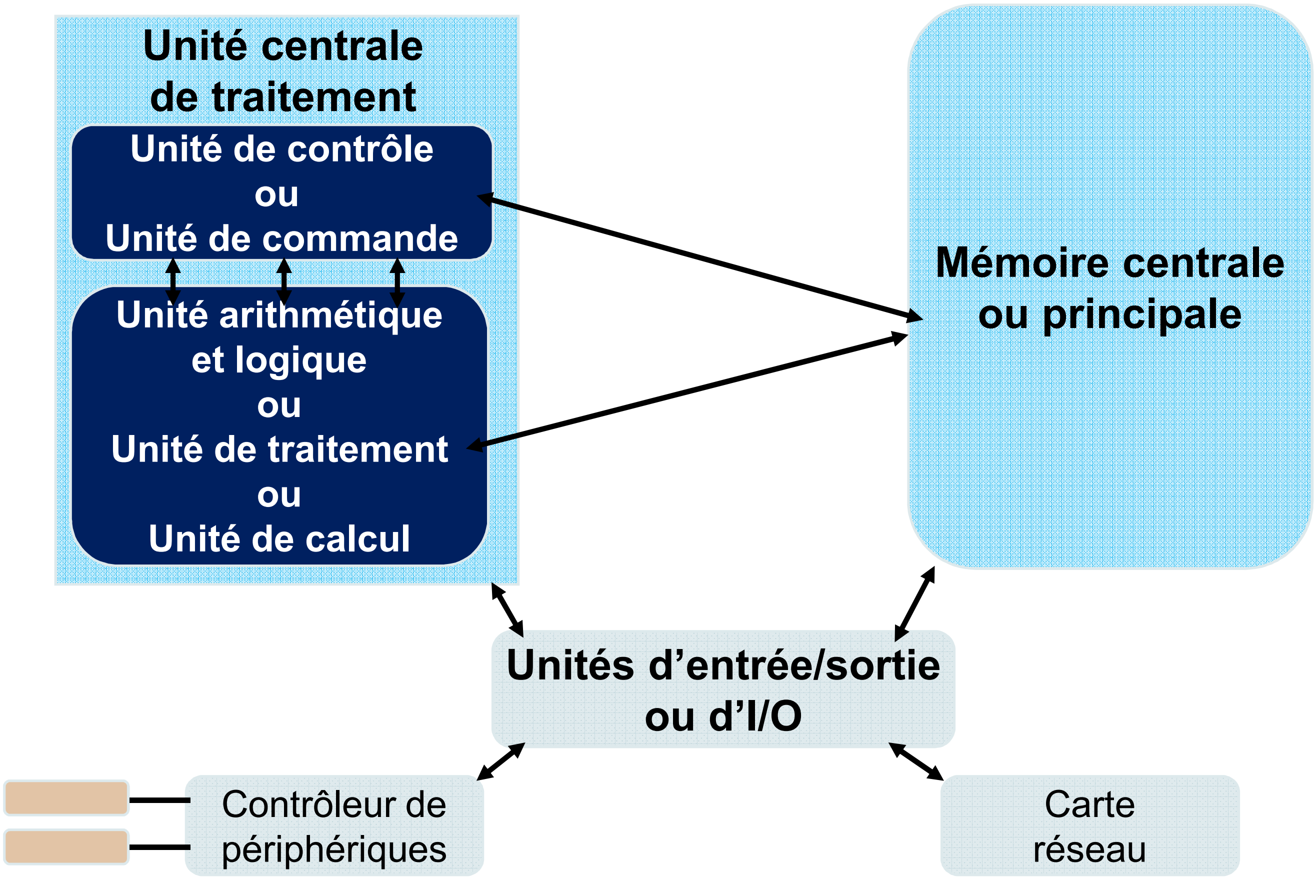
Utilisation des ordinateurs (Programmation)

- Programme
  - Suite d'instructions dans un langage de programmation, décrivant un traitement exécutable par un ordinateur
    - programmes systèmes
    - programmes d'application
- Système d'exploitation
  - Programme système qui gère les différentes ressources de l'ordinateur
- Programmation
  - Réalisation d'un programme dont l'exécution apporte une solution satisfaisante au problème donné
    - Langages de programmation (machine, assembleur, évolués)

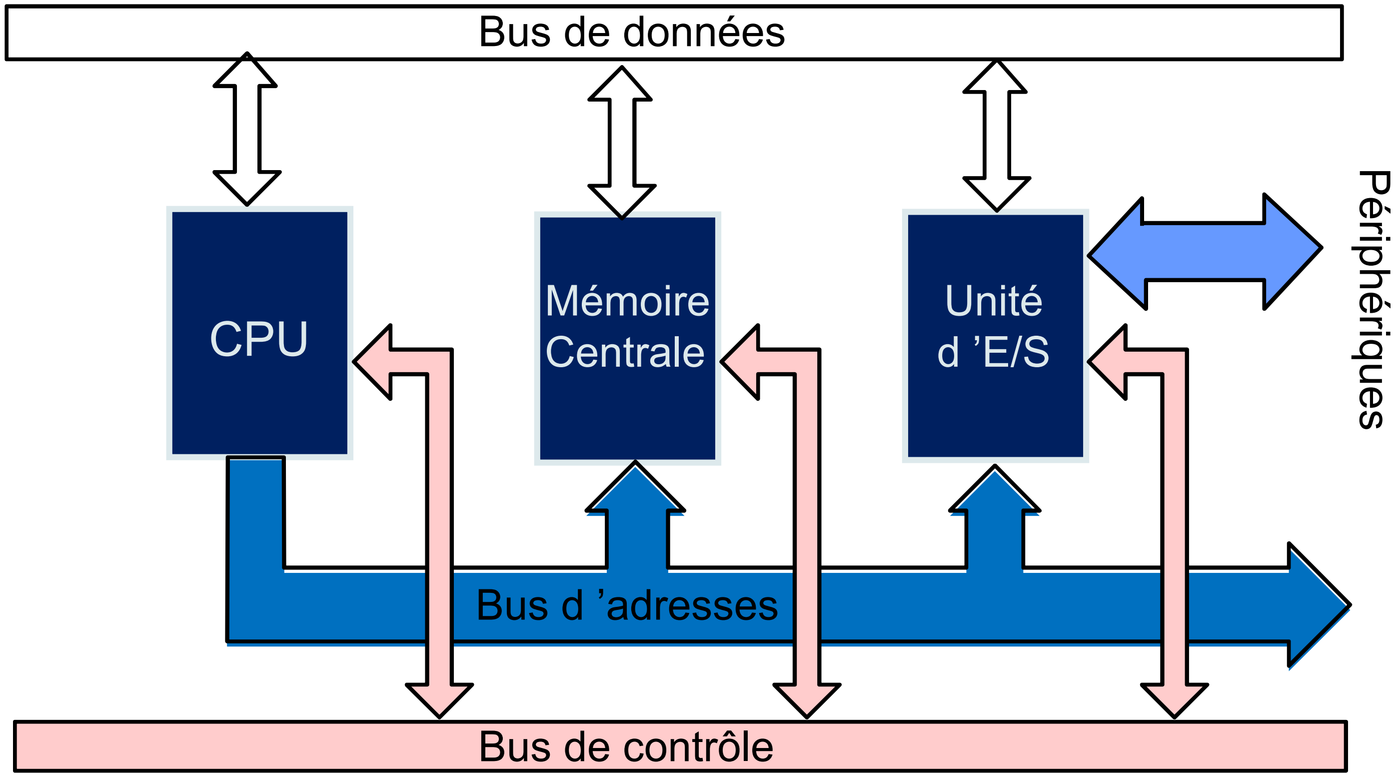
Evolution des langages de programmation



Principes de base (1/2)

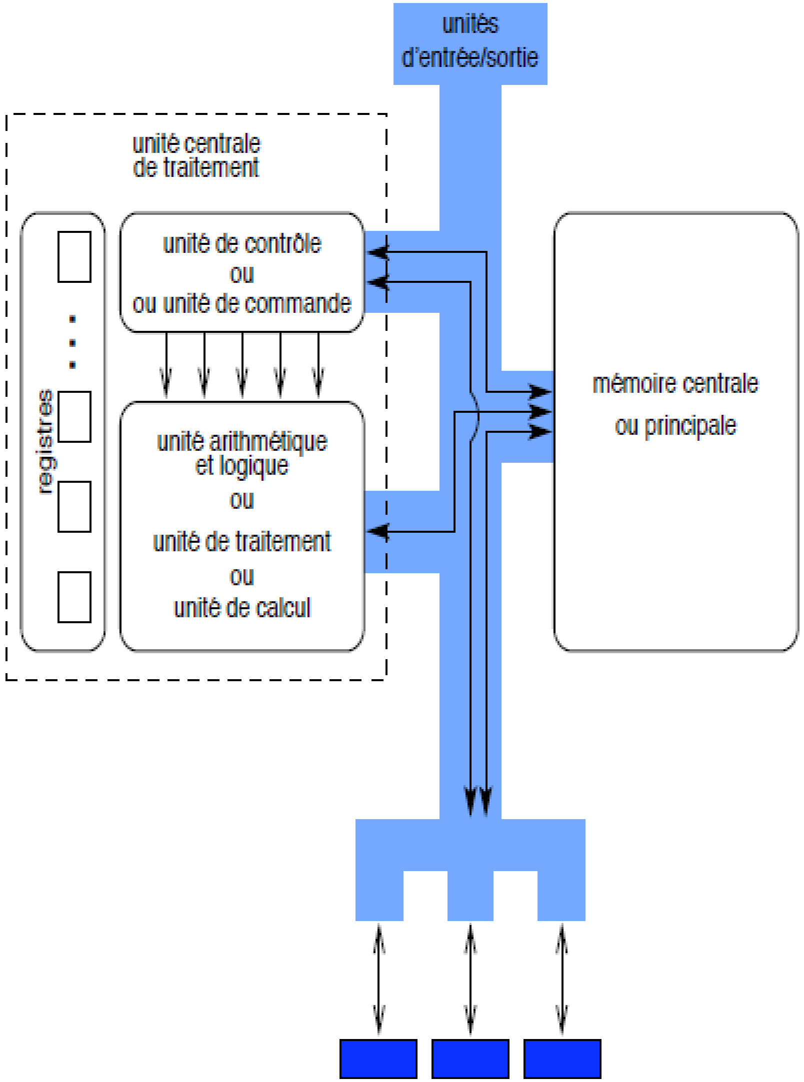


Principes de base (2/2)



Exécution d’un programme

1. Chargement du programme et des données depuis un périphérique dans la mémoire centrale
2. Chargement séquentiel des instructions du programme de la mémoire centrale dans l'unité de contrôle
3. Analyse par l'unité de contrôle de l'instruction et passage a l'UAL pour traitement
4. Traitement de l'instruction par l'UAL avec appel éventuel à la mémoire ou aux unités d'entrée-sortie.



Mémoire centrale (1/2)

- La mémoire peut contenir les données et les programmes
- Unité élémentaire d'information : bit (**b**inary dig**it**) = 0 ou 1.
- Octet = 8 bits.
- Mot mémoire = regroupement d'octets
  - (unité d'accès de base a la mémoire et unité de base de traitement).
- Utilisation de plusieurs bits pour coder une information.
  - Méthode de codage e.g., ASCII, ISO 8859-1, UTF-8

Mémoire centrale (2/2) ‘Mot mémoire’

- La mémoire est constituée de **cellules.** Chaque cellule correspond a un **mot-mémoire**. La longueur de ce mot constitue une caractéristique importante de l'architecture d'un ordinateur :
  - chaque mot possède sa propre adresse (sa position dans la mémoire)
  - c'est l'unité de base de traitement (taille des instructions)

- La capacité d'une mémoire s'exprime en fonction du nombre de mots-mémoire ainsi que du nombre de bits par mot.
  - Seulement la taille en octet est comptée. (En général)
  - Unité de mesure de la capacité de mémoire (Exemple : Mots)
    - Kilo (Ko) = 2^{10}=1.024 octets
    - Mega (Mo) = 2^{20}=1.048.576 octets
    - Giga (Go) = 2^{30}=1.073.741.824 octets
    - Tera (To) = 2^{40}=1.099.511.627.776 octets

Les registres

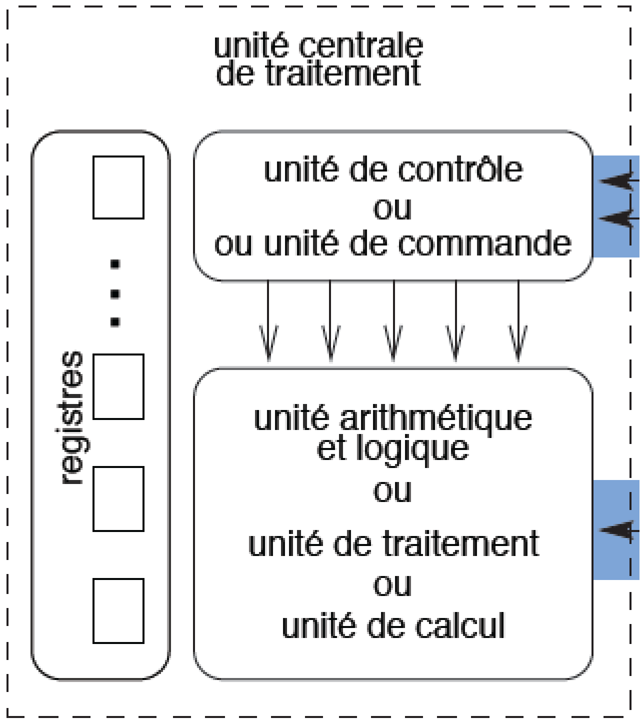
- Un registre est une cellule de mémoire ayant une fonction particulière. Il existe 2 types de registres pour interagir avec la mémoire centrale (non situé en MC) :
  - **registre adresse (RA)**, qui contient l'adresse d'un mot mémoire
  - **registre mot (RM)**, qui contient le contenu d'un mot mémoire

- Utilité : Exécuter les 2 opérations élémentaires en mémoire :
  - lecture : le registre d'adresse contient l'adresse du mot a lire qui est copié dans le registre mot.
  - écriture : le registre d'adresse contient l'adresse du mot dans lequel le contenu du registre mot va être écrit.

Unité centrale de traitement (CPU)

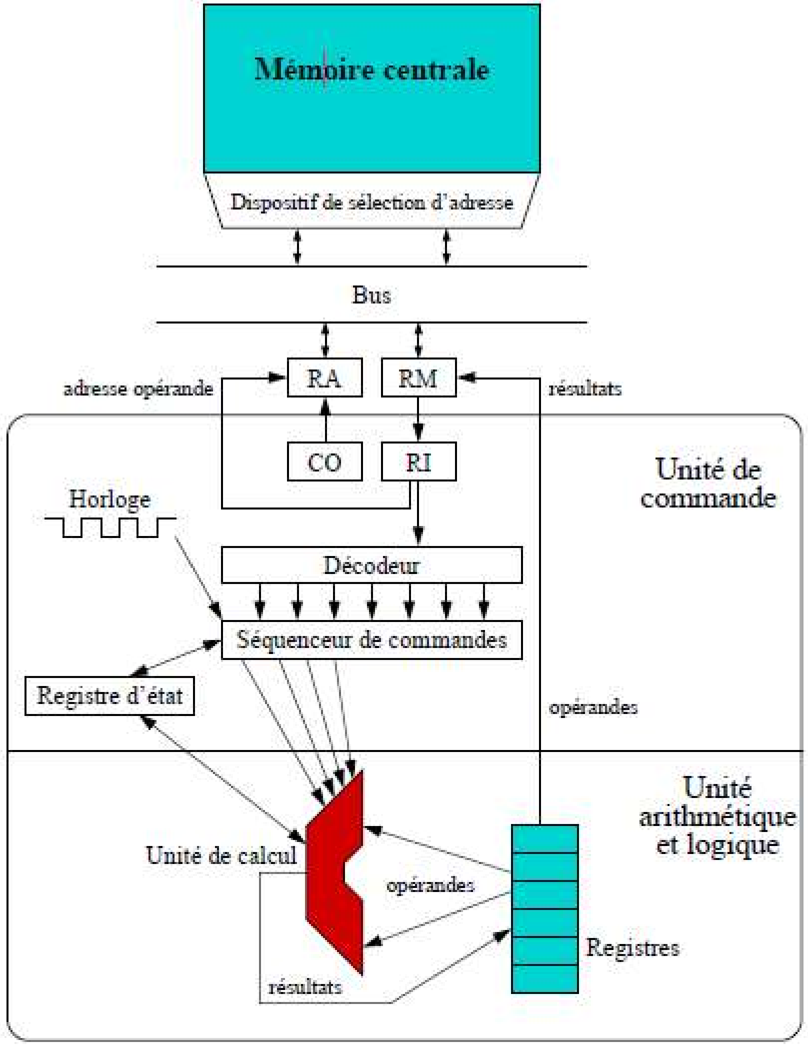
- ## Unité de commande
  - Prends les instructions en mémoire, les décode et les passe a l'UAL en fonction des cycles horloges.

- ## Unité Arithmétique et Logique (UAL)
  - Réalise effectivement les opérations arithmétiques (+,-,*,/) et logiques (NOT, AND, OR, XOR).



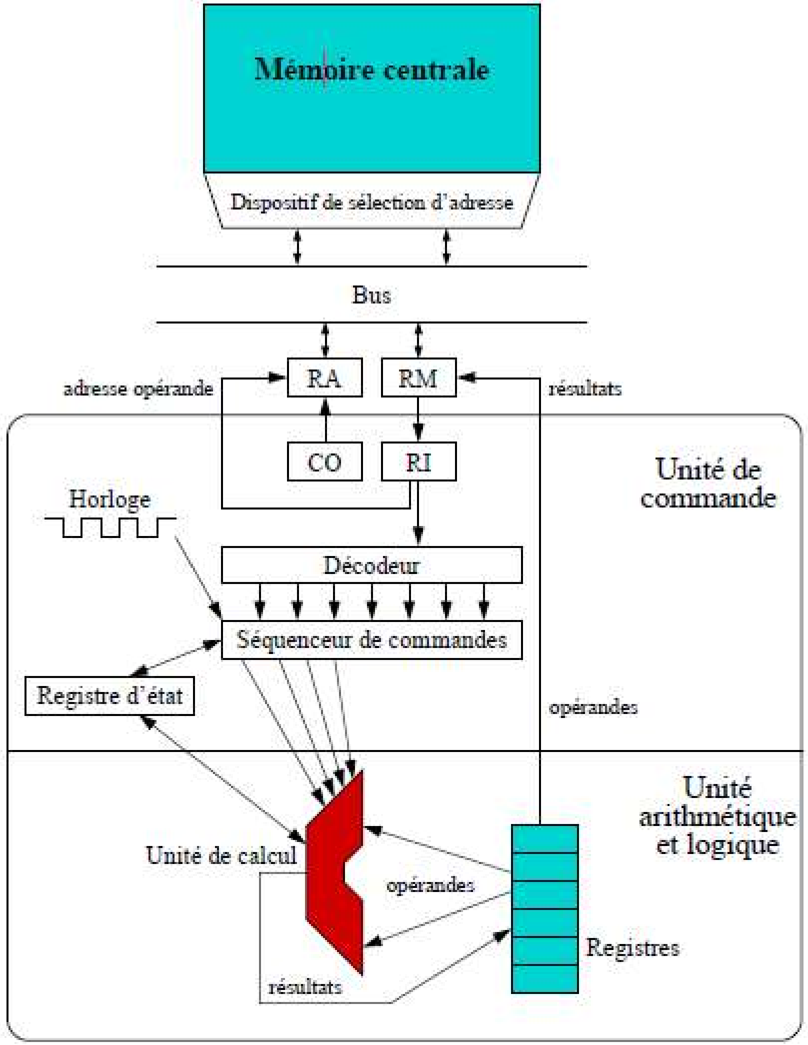
Unité de commande

- Registre d'instruction (RI) : contient l'instruction (opération + opérande) en cours d‘exécution
- Compteur ordinal (CO) : adresse de la prochaine instruction a exécuter
- Décodeur : décode les instructions
- Séquenceur : active les circuits nécessaires de l'UAL
- Horloge : rythme l'enchaînement des commandes (externe a l'unité de commande)



Exécution d'une instruction

1. Chargement de la prochaine instruction a exécuter depuis la mémoire jusque dans le RI.
2. Modification du CO.
3. Décodage de l'instruction (operateur).
4. Localisation dans la mémoire des données (opérande) utilisées par l'instruction.
5. Chargement des données dans les registres internes de l'unité centrale.
6. Exécution de l'instruction.
7. Stockage des résultats.
8. Retour a la première étape.



Entrées/Sorties et Périphériques

- Un ordinateur a besoin d‘échanger de l'information avec l'environnement extérieur. Ainsi il lui faut par exemple charger le programme et les données avec lesquels il va travailler, mais aussi communiquer avec l'utilisateur, visualiser des résultats.

- Unités d’entrées et sorties
  - Transfèrent les informations entre l'unité centrale et les unités périphériques.

- Unités Périphériques
  - Unités d‘échange de données avec le mon extérieur (écran, clavier, souris, imprimante, modem) et mémoires auxiliaires (disques) qui permettent de stocker de façon permanente. Chaque périphérique est associé a un contrôleur.

Plan du cours

- Historique
- Présentation de l'architecture des ordinateurs
- **Représentation interne des informations**
- Encodage/décodage de l'information
- Circuits logiques
- Mémoires
- Unité centrale de traitement

Objectif

- À la fin de cette unité,
  - Comment les caractères et les nombres entiers positifs et négatifs sont représentés dans la mémoire d'un ordinateur.
  - Comment effectuer les opérations arithmétiques addition et soustraction avec des entiers binaires.
  - Comment effectuer la multiplication et la division binaire

- Pour cela :
  - Effectuer des opérations arithmétiques sur des entiers dans n'importe quelle base, surtout en binaire et en hexadécimal;

Introduction (1/2)

- Les information traitées par l'ordinateur :
  - Nombres, instructions, images, séquences d'images animées, sons, etc., toujours représentées sous forme binaire. Une information élémentaire correspond donc à un chiffre binaire 0 ou 1 appelé **bit.**

- Comment traiter les informations : le **'codage'**
  - Fonction établissant une correspondance entre la représentation externe de l'information (e.g., 'CCVV', 36, une image, un son) et sa représentation interne qui est une suite de bits (e.g., 11011101, 11001010, ...)

- Les avantages du binaire :
  - facile à réaliser techniquement à l'aide de systèmes à deux états d'équilibre. (bistables)
  - En électronique ces 2 états corresponde a l'existence ou non d'une tension (+5V=1 et 0V=0).
  - opérations fondamentales simples à effectuer, sous forme de circuits logiques.

Introduction (2/2 : définitions)

- Types d'information traités : instructions et données.

  - Les instructions sont écrites en langage machine et représentent les opérations (exemple: addition, multiplication, etc.) effectuées par l'ordinateur. Elles sont composées de plusieurs champs :

    - Le code de l'opération à effectuer (opcode)
    - Les opérandes impliqués dans l'opération
    - Le codage dépend du processeur
    - Décodage par l'unité de commande
    - Nombre limité d'instructions {processeur CISC/RISC (Complex/Reduced Instruction-Set Computer)

  - Les données sont les opérandes sur lesquelles portent les opérations. On distingue les données numériques et les données non numériques (exemple : texte).

    - Non numériques (codage assez simple car aucune opération) arithmétique ou logique ne sera appliquée sur ces données, une table de correspondance suffit)
    - Numériques (codage complexe qui doit faciliter la mise en place de circuits réalisant les opérations arithmétiques)

Binaire

- Un système de numération utilisant la base 2. Toutes les informations sont codées avec des 0 et des 1.
  - 1 bits : 2^1 possibilités = 0, 1
  - 2 bits : 2^2 possibilités = 00, 01, 10, 11
  - 3 bits : 2^3 possibilité = 000, 001, 010, 011, 100, 101, 110, 111
  - n bits : 2^n possibilités
- Un mot = un ensemble de bit avec un poids $2^n; 2^{n-1}...2^1; 2^0$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	1	1	0	1

Binaire

- Le binaire

0	0	10	1010	31	1 1111
1	1	11	1011	32	10 0000
2	10	12	1100	63	11 1111
3	11	13	1101	64	100 0000
4	100	14	1110	127	111 1111
5	101	15	1111	128	1000 0000
6	110	16	1 0000	255	1111 1111
7	111	17	1 0001	256	1 0000 0000
8	1000	18	1 0010		
9	1001	19	1 0011		
		20	1 0100		
		24	1 1000		

Binaire

- En décimal, avec **n** chiffres, on obtient 10^n combinaisons possibles, i.e. on peut compter de 0 à 10^n-1.
  - Exemple : Avec 3 chiffres, on a $10^3 = 1000$ combinaisons possibles et on peut compter de 000 à 999.
-
- En binaire, avec **n** bits, on obtient 2^n combinaisons possibles, i.e. on peut compter de 0 à 2^n-1
  - Exemple : avec 8 bits, on a $2^8 = 256$ combinaisons possibles et on peut compter de 00000000 à 11111111, i.e. de 0 à 255.

Les données non numériques

- Afin de faciliter les échanges entre machines, des codages binaires normalisés ont été établis
  - BCD, (Binary Coded Decimal)
  - ASCII,
  - Unicode

  - Nombre variable de bits, 6, 7, 8 16, 32

  - Certains bit sont réservés au "contrôle" ou a la "correction" des autres

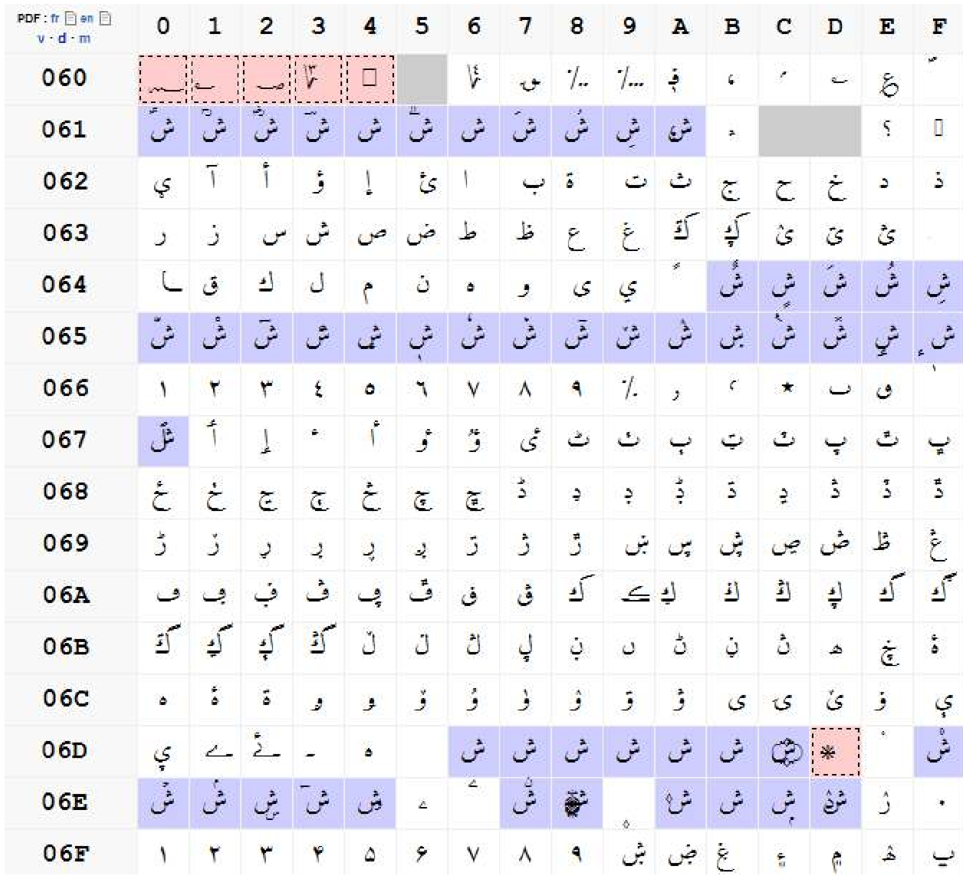
ASCII

- American Standard Code for Information Interchange
  - 7 bits (128 caractères)
  - 26 lettres majuscules A – Z
  - 26 lettres minuscule a – z
  - 10 chiffres 0 à 9
  - caractères de ponctuation
    - sp,! ” #$%& ’ ()*+,-. /< = >?@ [ ] ^_` { | } ~
  - caractères de contrôle :
    - null, etx, bel, bs, ht, lf, vt, ff, cr, …, del
  - Pas de caractères accentués
  - 1 bit supplémentaire utilisé pour le contrôle de parité
- ASCII étendu
  - 8 bits -> 256 caractères
    - caractères internationaux
    - caractères semi-graphiques

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0	0x00	(sp)	32	40	0x20	@	64	100	0x40	`	96	140	0x60
(soh)	1	1	0x01	!	33	41	0x21	A	65	101	0x41	a	97	141	0x61
(stx)	2	2	0x02		34	42	0x22	B	66	102	0x42	b	98	142	0x62
(etx)	3	3	0x03	#	35	43	0x23	C	67	103	0x43	c	99	143	0x63
(eot)	4	4	0x04	$	36	44	0x24	D	68	104	0x44	d	100	144	0x64
(enq)	5	5	0x05	%	37	45	0x25	E	69	105	0x45	e	101	145	0x65
(ack)	6	6	0x06	&	38	46	0x26	F	70	106	0x46	f	102	146	0x66
(bel)	7	7	0x07	'	39	47	0x27	G	71	107	0x47	g	103	147	0x67
(bs)	8	10	0x08	(	40	50	0x28	H	72	110	0x48	h	104	150	0x68
(ht)	9	11	0x09	)	41	51	0x29	I	73	111	0x49	i	105	151	0x69
(nl)	10	12	0x0a	*	42	52	0x2a	J	74	112	0x4a	j	106	152	0x6a
(vt)	11	13	0x0b	+	43	53	0x2b	K	75	113	0x4b	k	107	153	0x6b
(np)	12	14	0x0c	,	44	54	0x2c	L	76	114	0x4c	l	108	154	0x6c
(cr)	13	15	0x0d	-	45	55	0x2d	M	77	115	0x4d	m	109	155	0x6d
(so)	14	16	0x0e	.	46	56	0x2e	N	78	116	0x4e	n	110	156	0x6e
(si)	15	17	0x0f	/	47	57	0x2f	O	79	117	0x4f	o	111	157	0x6f
(dle)	16	20	0x10	0	48	60	0x30	P	80	120	0x50	p	112	160	0x70
(dc1)	17	21	0x11	1	49	61	0x31	Q	81	121	0x51	q	113	161	0x71
(dc2)	18	22	0x12	2	50	62	0x32	R	82	122	0x52	r	114	162	0x72
(dc3)	19	23	0x13	3	51	63	0x33	S	83	123	0x53	s	115	163	0x73
(dc4)	20	24	0x14	4	52	64	0x34	T	84	124	0x54	t	116	164	0x74
(nak)	21	25	0x15	5	53	65	0x35	U	85	125	0x55	u	117	165	0x75
(syn)	22	26	0x16	6	54	66	0x36	V	86	126	0x56	v	118	166	0x76
(etb)	23	27	0x17	7	55	67	0x37	W	87	127	0x57	w	119	167	0x77
(can)	24	30	0x18	8	56	70	0x38	X	88	130	0x58	x	120	170	0x78
(em)	25	31	0x19	9	57	71	0x39	Y	89	131	0x59	y	121	171	0x79
(sub)	26	32	0x1a	:	58	72	0x3a	Z	90	132	0x5a	z	122	172	0x7a
(esc)	27	33	0x1b	;	59	73	0x3b	[	91	133	0x5b	{	123	173	0x7b
(fs)	28	34	0x1c	<	60	74	0x3c	\	92	134	0x5c	|	124	174	0x7c
(gs)	29	35	0x1d	=	61	75	0x3d	]	93	135	0x5d	}	125	175	0x7d
(rs)	30	36	0x1e	>	62	76	0x3e	^	94	136	0x5e	~	126	176	0x7e
(us)	31	37	0x1f	?	63	77	0x3f	_	95	137	0x5f	(del)	127	177	0x7f

Unicode

- 1 a 4 octets (1114112 caractères)
- Codage unique quelque soit la plateforme, le logiciel, la langue.
- Code universel contenant, en plus de tous les caractères connus, 42 000 caractères asiatiques. Le code ASCII est contenu dans les 128 premiers caractères d'UNICODE.
- UNICODE est supporté par Windows NT, Windows 2000, Java, et certains systèmes UNIX.



- Unicode (UTF-7 , UTF-8, UTF-16, UTF- 32)
  - (Universal Character Set Transformation Format)
  - UTF-7
  - ï»¿Bonjour,
  - Je suis parti Ã l'Ã©cole

Les données numériques

- Nombres entiers positifs ou nuls : 0 ; 1 ; 315
- Nombres entiers négatifs : -1 ; -1255
- Nombres fractionnaires : 3,1415 ; -0,5

- Un algorithme de codage réalise la conversion en binaire. Les opérations arithmétiques (+, -, *, /) sont effectuées en arithmétique binaire.

Entiers positifs ou nuls

- Systèmes de numération
- Représentation pondérée d'un nombre N dans une base B :

- $N = a_n B^n + a_{n-1} B^{n-1} + \ldots + a_1 B + a_0 =$
  - où $a_n = 0, 1, \ldots B-1$
  - Les bases B les plus usitées sont :
  - B = 10, décimal
  - B = 2, binaire
  - B = 16, hexadécimal
  - B = 8, octal

- Exemples :
- $2341_{10} = 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 1 \times 10^0$
- $1\,0011_2 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 19_{10}$

Changement de base : binaire → décimale

- Additionner les puissances de 2 correspondants aux bits de valeur 1.
  - 101101011 = $2^8 + 2^6 + 2^5 + 2^3 + 2^1 + 2^0$ = 256+64+32+8+2+1 = 363

- Définitions:
  - bit de poids faible : le bit ayant la moindre valeur (i.e., celui de droite)
  - bit de poids fort : le bit ayant la plus grande valeur (i.e., celui de gauche)

Changement de base : décimale → binaire

- 363 / 2 reste 1
- 181 / 2 reste 1
- 90 / 2 reste 0
- 45 / 2 reste 1
- 22 / 2 reste 0
- 11 / 2 reste 1
- 5 / 2 reste 1
- 2 / 2 reste 0
- 1 / 2 reste 1

1 0 1 1 0 1 0 1 1

Addition de bits binaire

- 363+19 en base 2

$$
\begin{array}{ccccccccccc}
 & 1 & {}^{1}0 & {}^{1}1 & {}^{1}1 & 1 & {}^{1}0 & {}^{1}1 & {}^{1}0 & 1 \\
+ & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
$$

- => 382

- 363+256 en base 2

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

- => 619
- => Attention au bit de retenue (Carry)

Soustraction en nombre binaire

- 8-1 en base 2

$$
\begin{array}{ccccc}
 & 1_0 & 0_1 & 0_1 & {}_1 0 \\
- & 0 & 0 & 0 & 1 \\
 & 0 & 1 & 1 & 1
\end{array}
$$

- => 7 (Exercice : 100000 – 1111 (32-15))

- 13-6 en base 2

$$
\begin{array}{ccccc}
 & 1^0 & {}_1 1_0 & {}_1 0 & 1 \\
- & 0 & 1 & 1 & 0 \\
 & 0 & 1 & 1 & 1
\end{array}
$$

- => 7 (Exercice : 10110100 – 10000111)
- Quand le chiffre du bas est supérieur à celui du haut, on emprunte 2 au chiffre de gauche suivant et on ajoute ce 2 au chiffre du haut. On fait la soustraction. L'emprunt soustrait 1 au chiffre de gauche.
- 0 -0 = 0
- 1- 0 = 1
- 1- 1 = 0
- 0 – 1 = 1 et en emprunte 2, l'emprunt soustrait 1 au chiffre de gauche du haut.

Multiplication en binaire

- La multiplication binaire s'effectue comme la multiplication décimale ordinaire, mais est beaucoup plus simple, puisqu'il n'y a que des 1 et des 0.

```
      11011
    × 1101
  ---------
      11011
     00000
    11011
   11011
  ---------
  101011111
```

Multiplier 7 * 5 en binaire ?

Multiplier 8 * 5 en binaire ? Que remarquez-vous ?

Remarques (Multiplication/ Division par B)

- Remarques
  - $10_B = B$ quel que soit B :
  - $10_2 = 2$, $10_7 = 7$, $10_8 = 8$, $10_{16} = 16_{10}$
  - Ajouter un 0 à droite (décalage à gauche) = multiplication par B
  - Enlever le chiffre de droite (décalage à droite) =
  - division entière par B

Division en binaire

- Division
- La division binaire s'effectue comme la division décimale ordinaire, mais elle est beaucoup plus simple, puisque les facteurs sont 1 ou 0.
-

```
101111 / 0100
100          1
0011         0
   111
   100       1
   0111
     100     1
     011
```

- Résultat : 1011_2, reste 0011_2

Champs Fixe

- L'entier maximal pouvant être codé dépendra du nombre de bits que l'on réserve pour coder un nombre. En général les entiers sont codes sur un mot
  - Ex: pour un ordinateur 32 bits : $2^{32} - 1 = 4\ 294\ 967\ 295$.

- Dépassement de capacité (Overflow)
  - Lorsque par exemple le résultat d'une opération sur des nombres produit un nombre plus grand que la taille du mot prévu pour représenter ces nombres (ex: bit de retenue).

Entier négatif

- Il existe plusieurs façons de représenter un nombre négatif :

- Valeur absolue signée
- Complément a 1 (ou logique)
- Complément a 2 (ou arithmétique)

- Avantages & inconvénients ?

Valeurs absolues signées

- Les nombres sont codés comme des entiers positifs mais on sacrifie un bit (celui de poids fort) pour coder le signe :
  - Bit n-1 pour le signe (signe + = 0 , signe - = 1)
  - Bits n-2, n-3, n-4, …, 0 pour la valeur absolue
  - Ex: $(6)_2$ = 0110 / $(-6)_2$ = 1110

- Valeurs représentées : $[-2^{(n-1)}-1,.., 2^{(n-1)}-1]$
  - Ex: n=3 représente [-3, 3] , n=8 représente [-127,127]
- Avantage
  - Symétrie : autant de négatifs que de positifs
- Inconvénient
  - 2 représentations du 0 : 0000000 et 10000000
  - Bit de signe doit être traité de façon particulière => opérations arithmétiques non aisées

Compléments à 1 (se référer à l'Annexe)

- $A_{(Cà1)} = 2^n - 1 - A$
- Les nombres positifs sont codes comme précédemment, les négatifs sont obtenus en remplaçant tous les bits a 1 par 0 et vice-versa. :
  - Bit n-1 pour le signe (signe + = 0 , signe - = 1)
  - Bits n-2, n-3, n-4, …, 0 pour les positifs et leurs compléments
  - Ex: $(6)_2$ = 0110 / $(-6)_2$ = 1 0 0 1

- Valeurs représentées : $[-2^{(n-1)} - 1, .., 2^{(n-1)} - 1]$
  - Ex: n=3 représente [-3 ,3] , n=8 représente [-127,127]
- Avantage
  - Symétrie : autant de négatifs que de positifs
  - Une soustraction se réduit à l'ajout de son complément Ex : 3 – 2 = 3 + (-2)
- Inconvénient
  - 2 représentations du 0 : 0000000 et 11111111
  - Dépassement à éviter (overflow) Ex : pour n=4, 7 + 6
  - **Bit de retenue à reporter lors de l'addition**
  - 6 + (-1) = 5 → 0110 + 1110 = +1 0100 = 0101
  - 1 + (-2) = -1 → 0001 + 1101 = 1110
  - (-1) + (-1) = -2 → 1110 + 1110 = +1 1100 = 1101

Compléments à 2 (se référer à l'Annexe)

- $A_{(Cà1)} = 2^n - A$
- Les nombres positifs sont codes comme précédemment, les négatifs sont obtenus en ajoutant 1 au complément de 1. :
  - Bit n-1 pour le signe (signe + = 0 , signe - = 1)
  - Bits n-2, n-3, n-4, …, 0 pour les positifs et leurs compléments
  - Ex: $(6)_2$ = 0110 / $(-6)_2$ = 1001 + 1 = 1010

- Valeurs représentées : $[-2^{(n-1)}, \ldots, 2^{(n-1)} - 1]$
  - Ex: n=3 représente [-3, 3] , n=8 représente [-128,127]
- Avantage
  - **Pas de bit de retenue**
  - 6 + (-1) = 5 → 0110 + 1111 = 0101
  - 1 + (-2) = -1 → 0001 + 1110 = 1111
  - (-1) + (-1) = -2 → 1111+ 1111 = 1110
  - **Une seule représentation du 0**
  - Une soustraction se réduit à l'ajout de son complément Ex : 3 – 2 = 3 + (-2)
- Inconvénient
  - Dissymétrie : plus de négatifs que de positifs

Représentation sur 16 bits

16 bits $\Rightarrow 2^{16} = 65'536 = 2 \times 32'768$ valeurs possibles

décimal	valeur absolue et signe	complément à 2	complément à 1
+32767	0111...1...1111	0111...1...1111	0111...1...1111
+32766	0111...1...1110	0111...1...1110	0111...1...1110
...	...	...	...
+1	0000...0...0001	0000...0...0001	0000...0...0001
+0	0000...0...0000	0000...0...0000	0000...0...0000
−0	1000...0...0000	---------------	1111...1...1111
−1	1000...0...0001	1111...1...1111	1111...1...1110
...	...	...	...
−32766	1111...1...1110	1000...0...0010	1000...0...0001
−32767	1111...1...1111	1000...0...0001	1000...0...0000
−32768	---------------	1000...0...0000	---------------

Soustraction de nombre sur 4 bits

décimal	signe+val. absolue	comp. à 1	comp. à 2
+7	0111	0111	0111
-6	+1110	+1001	+1010
+1	?101	10000	10001
		$\hookrightarrow$ 1	$\Downarrow$
			0001
		0001	
	(a)	(b)	(c)

- (a) plus facile a lire, mais le bit de signe doit être traite a part ;
- (b) on effectue l'addition du complément, y compris le bit de signe, avec report de la retenue ;
- (c) on effectue une addition, y compris le bit de signe, mais sans report de la retenue.

Opérations arithmétique avec les nombres signés en complément à 2 (l'**Addition**)

- 4 cas sont possibles :
  - **Les deux nombres sont positifs** : addition binaire classique.
  - **Les deux nombres sont négatifs** : addition binaire classique et on oublie la dernière retenue (a gauche). La somme est négative.
  - **Le nombre positif est plus grand que le nombre négatif** : addition binaire classique et on 'oublie' la dernière retenue (a gauche). La somme est positive.
  - **Le nombre négatif est plus grand que le nombre positif** : addition binaire classique, la somme est négative et représentée directement dans le système complément à 2.

Opérations arithmétique avec les nombres signés en complément à 2 (**Soustraction**)

- La soustraction est considérée comme un cas particulier de l'addition :
  - A - B = A + (-B)
  - -A - B = (-A) + (-B)

- On prend donc le système complément a deux pour représenter les négatifs, et on effectue une addition.

Opérations arithmétique avec les nombres signés en complément à 2 (**Multiplication**)

- Les deux nombres doivent être représentés dans une forme sans complément (i.e., valeur absolue). On effectue la multiplication et on décide du signe du résultat :
  - opérandes sont de mêmes signes : le résultat est positif
  - opérandes signes différents : le résultat est négatif, on le représente avec son complément a 2

Opérations arithmétique avec les nombres signés en complément à 2 (**Division**)

- Dividende / Diviseur = Quotient

- Les deux nombres doivent être représentés dans une forme sans complément (i.e., valeur absolue) :

1. Déterminer si le dividende et le diviseur sont de mêmes signes ou de signes différents. Ceci va déterminer le signe du quotient ; Initialiser le quotient a zéro.

2. Soustraire le diviseur du dividende en utilisant l'addition avec complément a deux pour obtenir le premier reste partiel ; Incrémenter le quotient de 1. Si le reste partiel est positif aller a l'étape trois. Si le reste partiel est zéro ou négatif la division est terminée.

3. Soustraire le diviseur du reste partiel et incrémenter le quotient de 1. Si le résultat est positif, répéter l'opération pour le reste partiel trouvé. Si le résultat est zéro ou négatif la division est terminée.

Nombre réels

- Représentation de la virgule :

  - **virgule fixe** : nombre fixe de chiffres après la virgule ; Les bits a gauche (resp. a droite) de la virgule représentent la partie entière (resp. la partie décimale "binaire") du nombre et correspondent a des puissances de 2 (resp. l'inverse des puissances de 2).
    - Exemple : $(25,75)_{10} = (0011001,110)_2$
    - Sur 10 bits (7 partie entière, 3 partie décimale binaire) :
    - $0011001110 = 2^4+2^3+2^0+2^{-1}+2^{-2} = 16+8+1+0,5+0,25 = 25,75$

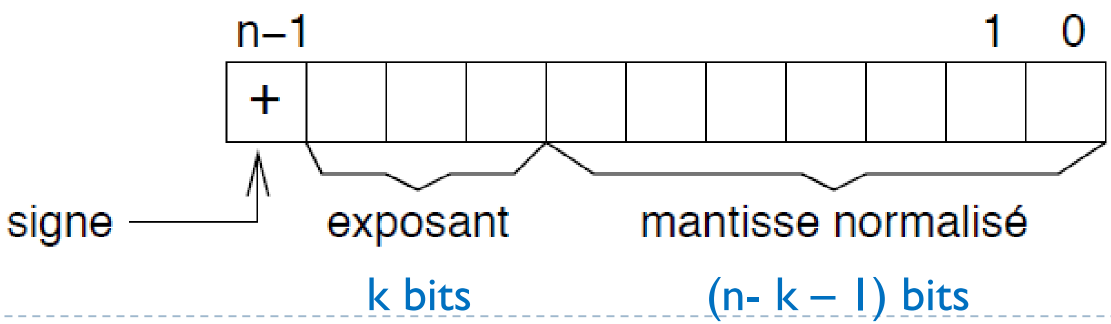
  - **virgule flottante :** la position de la virgule n'est pas fixe. Ces nombres sont des approximations de nombres réels.

Représentation de la virgule flottante

- Représentation d'un nombre sous la forme d'un produit de 2 facteurs
  - $N = (1)^s \times M \times B^E$
  - B : base
  - M : mantisse (nombre purement fractionnaire i.e., 0,xxx)
  - E : exposant
  - Ex: $0,12345 * 10^3 = 123,45$
  - Exposant et mantisse peuvent être signés

- Mantisse normalisé : premier chiffre significatif différent de zéro

- La représentation classique consiste a coder l'exposant en représentation biaisé (ou en excédent) sur k bits et la mantisse en valeur absolue signée sur (n - k -1) bits.



Codage de l’exposant

- Taille de l'exposant bornée.
- Codage par excédent n : on décale l'exposant on lui ajoutant n.
  - => Pas d'exposant négatifs
  - => Facilite les opérations de tri (pas besoin de conversion au décimal pour trier)
  - Ex: sur 3 bits, excédent a 4 :
    - +3 111 (on code $(3)_{10}$ = 11 puis on ajoute 100 = 111
    - +2 110
    - +1 101
    - 0 100
    - -1 011
    - -2 010
    - -3 001
    - -4 000
- Si la taille de l'exposant augmente alors l'intervalle des valeurs possibles représentables grandit.

Changement de base des fractions

- Binaire → Décimale
  - $N = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1.2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + \ldots$
  - $N = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1.2 + a_0 + a_{-1}(1/2) + a_{-2}(1/4) + \ldots$
  - Ex: $0{,}011_2 = 0^*\,(1/2) + 1^*(1/4) + 1^*\,(1/8)$
    $= 0{,}25_{10} + 0{,}125_{10} = 0{,}325_{10}$

- Décimale→ Binaire
  - $0{,}5625_{10}$
  - Réponse : $0{,}10010000_2$

0,	5625 x 2
1,	125 x 2
0,	25 x 2
0,	5 x 2
1,	0 x 2
0,	0

Codage de la mantisse

- Taille de la mantisse bornée.
- Changement de base (décimal → binaire) obtenu par multiplications successives par 2.
- Si cela ne converge pas vers 1 alors il n'y a pas de représentation exacte de ce nombre, on tronque alors suivant la taille de la mantisse.
- Si la taille de la mantisse augmente, la précision des valeurs possibles représentables grandit.

- Conversion de 0.375
  - 0,375 x 2 = 0,75
  - 0,75 x 2 = 1,5
  - 0,5 x 2 = 1,0
  - → $0{,}375 = 0 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} = 0 * 0{,}5 + 1 * 0{,}25 + 1 * 0{,}125$
  - → 0,375 = 0,011 en binaire
  - soit $0{,}11 \times 2^{-1}$ en forme binaire normalisée

  - (0,5); (0,25); (0,125); (0,0625) ; (0,03125) ; (0,015625) ; (0,0078125) ; (0,00390625)
  - (-1); (-2); (-3); (-4); (-5); (-6); (-7); (-8);

Opération arithmétique en virgule flottante

- Il est toujours possible de revenir a des opérations arithmétiques sur les nombres entiers :
  - multiplication : additionner les exposants, multiplier les mantisses et re-normaliser le résultat
  - division : soustraire les exposants, diviser les mantisses et re-normaliser le résultat
  - addition : de-normaliser la plus petite valeur d'exposant, additionner les mantisses, re-normaliser le résultat
  - soustraction : de-normaliser la plus petite valeur d'exposant, soustraire les mantisses, re-normaliser le résultat
- Il peut être nécessaire d'arrondir la mantisse (i.e., perte de précision).
- Dépassement de capacité et sous-passement de capacité peuvent se produire si l'exposant devient trop grand ou trop petit.

Norme IEEE 754

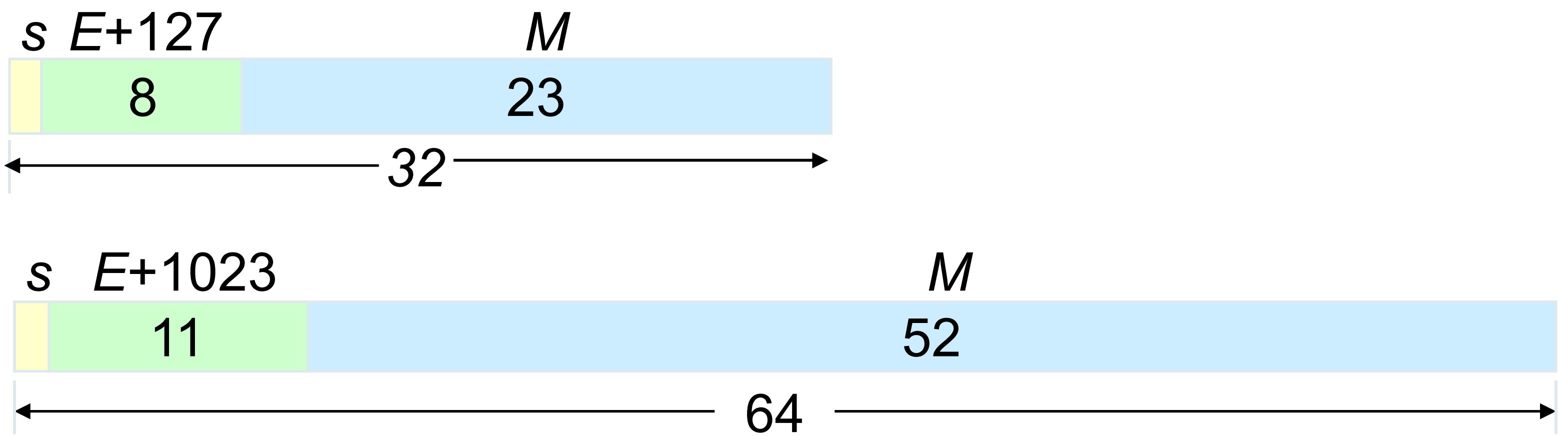
- Objectif : harmoniser les représentations en virgule flottante et définir le comportement en cas d'exception (dépassement, sous-passement)
- Bit de signe S (signe + = 0 , signe - = 1)
- Mantisse normalisée en base 2 avec un bit caché
  - Comme le 1 de tête doit toujours être présent, il n'est pas nécessaire de le stocker
- Exposant codé en excédent $2^{n-1} - 1$ (n = nombre de bit de l'exposant)

- Définition
- $x = (-1)S * (1,M) * 2^{Exp}$, avec $Exp = E - (2^{n-1} - 1)$

- Valeurs particulières :
- E=0 et M=0 → +/- 0
- $E=2^{n-1}$ et M=0 → +/- infini
- $E=2^{n-1}$ et M<>0 → NaN (Not a Number)

Précision des flottants

	simple précision 32 bits	double précision 64 bits
bit de signe	1	1
exposant	8	11
mantisse	23	52
codage de l'exposant	excédent 127	excédent 1023
variation de l'exp.	-126 à $+127$	-1022 à $+1023$
plut petit nombre	2^{-126}	2^{-1022}
plus grand nombre	environ 2^{+128}	environ 2^{+1024}
échelle des nombres décimaux	environ 10^{-38} à 10^{+38}	environ 10^{-308} à 10^{+308}



Exemple

- -0,75 en simple précision
  - 0,75 x 2 = **1**,5 ; 0,5 x 2 = **1**,0
  - 0,75 → 0,11 en binaire (0,50 + 0,25)
  - soit $1,1 \times 2^{-1}$ en forme normalisée
  - => $(-1)1 \times (1,100\ 0000\ 0000\ 0000\ 0000\ 0000) \times 2^{126-127}$
  - En simple présicion :
  - 1 01111110 10000000000000000000000

- 1 10000001 01000000000000000000000 en simple précision
  - signe = 1
  - exposant = 129
  - mantisse = $1 \times 2^{-2} = 1/4 = 0,25$
- => $(-1)S \times (1,M) \times 2^{E-127} = (-1)1 \times (1,25) \times 2^{129-127}$
- $= -1 \times 1,25 \times 2^2$
- = -5

Autre représentation binaire

- Décimaux codés binaire (BCD) : Chaque chiffre d'un nombre est code individuellement en son équivalent binaire sur 4 bits.
  - Ex, 15 = 0001'0101, 96 = 1001'0110
- Code excédent 3 : BCD+3 a chaque chiffre
- Code 2 dans 5
- Code biquinaire
- Avantage
  - Operations d'entrées / sorties plus faciles
- Inconvénient
  - Operations arithmétiques compliquées

Autre représentation binaire

décimal	BCD	excédent-3	2 dans 5	biquinaire
0	0000	0011	00011	01 00001
1	0001	0100	00101	01 00010
2	0010	0101	00110	01 00100
3	0011	0110	01001	01 01000
4	0100	0111	01010	01 10000
5	0101	1000	01100	10 00001
6	0110	1001	10001	10 00010
7	0111	1010	10010	10 00100
8	1000	1011	10100	10 01000
9	1001	1100	11000	10 10000
			63210	5043210

BCD

- Le BCD est un code dans lequel chaque chiffre d'un nombre décimal est codé en binaire sur 4 bits.
- Ces chiffres peuvent être représenté sur un octet individuel, c'est le BCD non compacté.

- Exemple :
  - $327_{10} \rightarrow$ 0000 0011 0000 0010 0000 0111

- Comme chaque chiffre n'utilise que 4 bits, on peut les grouper 2 par octet. C'est le BCD compacté.

- Exemple :
  - $53_{10} \rightarrow$ 0101 0011

Astuces (changement de base)

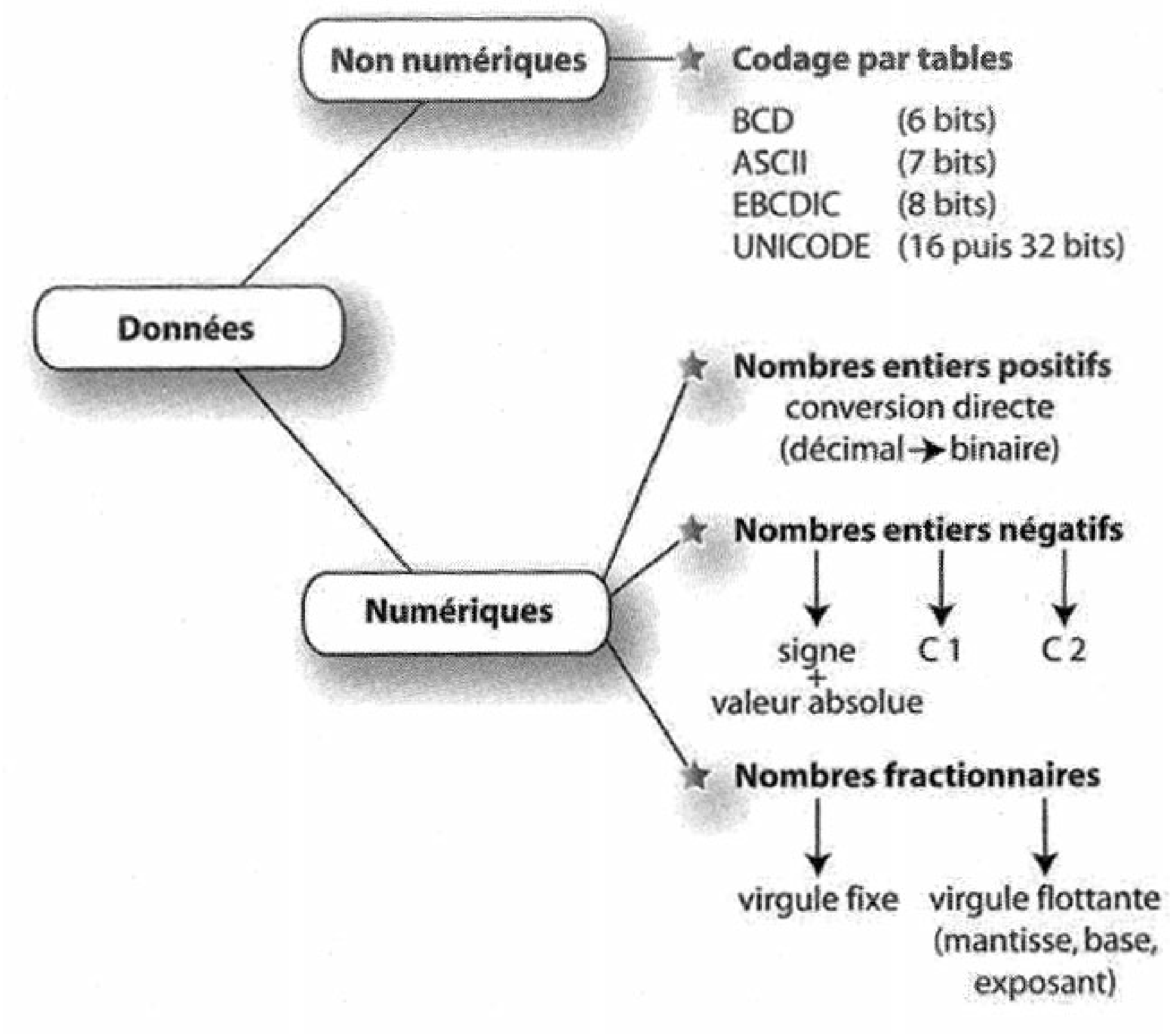
- Décimal → Binaire
- On peut effectuer les multiplications par 10 en remarquant que $10x = 8x + 2x$, et en se rappelant qu'un décalage à gauche de 1 bit est une multiplication par 2. C'est généralement plus rapide que la multiplication binaire.
  - 142 = (10 x 10) + (4 x 10) + 2
  - Ainsi, $1010_2 \times 1010_2 = 1010000_2 + 10100_2 = 110\ 0100_2$.
  - $100_2 \times 1010_2 = 100000_2 + 1000_2 = 10\ 1000_2$
  - On obtient finalement :
  - $142_{10} = 110\ 0100_2 + 10\ 1000_2 + 0010_2$

    Et
  - $142_{10} = 1000\ 1110_2$

Astuces (changement de base)

- Binaire → décimal
- Exemples : Convertir $1000\ 1110_2$ en base 10
- $1000\ 1110_2$ / 1010_2 = 1110_2, reste 0010_2
- 1110 / 1010 = 0001, reste 0100_2
- 0001 / 1010 = 0000, reste 0001_2
- $1000\ 1110_2$ = 0001 0100 0010 = 142_{10} en BCD compacté
- ou 0000 0001 0000 0100 0000 0010 en BCD non compacté

Représentation des données – un récapitulatif



Représentation de type sous Java

Primitive	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	2	valeur du jeu de caractères Unicode (65000 caractères possibles)
byte	Entier très court	1	-128 à 127
short	Entier court	2	-32768 à 32767
int	Entier	4	-2 147 483 648 à 2 147 483 647
long	Entier long	8	-9223372036854775808 à 9223372036854775807
float	flottant (réel)	4	$-1.4*10^{-45}$ à $3.4*10^{38}$
double	flottant double	8	$4.9*10^{-324}$ à $1.7*10^{308}$
boolean	booléen	1	0 ou 1 (en réalité, toute autre valeur que 0 est considérée égale à 1)

Annexe (complément à 1 à 2)

- Soit x un entier positif, on représente -x
- Complement à 1:
  - Formule: $2^n - 1 - x$
    - n=4, $2^4 - 1 - x = 15 - x$
    - En binaire: $(1\ 1\ 1\ 1) - (b_3\ b_2\ b_1\ b_0)$
    - Changer les bits.
- Complement à 2:
  - Formule: $2^n - x$
    - n=4, $2^4 - x = 16 - x$
    - En binaire: $(1\ 0\ 0\ 0\ 0) - (0\ \ b_3\ b_2\ b_1\ b_0)$
    - Changer les bits et ajouter 1.

Annexe (complément à 1)

- **Complément à 1**
- **Théorème 1:** Soit x un nombre positif (x_{n-1} , x_{n-2} , … , x_0), le nombre négatif est donné $(\bar{x}_{n-1}, \bar{x}_{n-2}, \ldots, \bar{x}_0)$ avec
$\bar{x} = 1 - x$

- Preuve
- (i). 2^n-1 en binaire est un vecteur de n bit (1,1, …, 1)
- (ii).2^n-1-x en binaire est (1,1, …,1) – (x_{n-1},x_{n-2}, …, x_0).
le résultat est

$$(\bar{x}_{n-1}, \bar{x}_{n-2}, \ldots, \bar{x}_0)$$

Annexe (complément à 2)

- **Complément à 2**
- **Théorème 1:** Soit x un nombre positif (x_{n-1} , x_{n-2} , … , x_0), le nombre négatif est la somme de son complément à 1 et 1

- Preuve
- $2^n – x = 2^n – 1 – x + 1.$

Annexe (Opération arithmétique en complément à 2)

Input: 2 entiers positifs x et y

1. On représente les opérandes en complément à 2
2. On somme les opérandes et ignore le bit n
3. Le résultat est en complément à 2

Arithmetic	Complément à 2
$x + y$	$x + y$
$x - y$	$x + (2^n - y) = 2^n+(x-y)$
$-x + y$	$(2^n - x) + y =2^n+(-x+y)$
$-x - y$	$(2^n - x) + (2^n - y) = 2^n+2^n-x-y$

Annexe (Opération arithmétique en complément à 1)

Input: 2 entiers positifs x et y

1. On représente les opérandes en complément à 2
2. On somme les opérandes et ignore le bit n
3. On supprime 2n- 1 s'il y a un 'Carry' à gauche
4 Le résultat est en complément à 1

Arithmetic	Complément à 1
$x + y$	$x + y$
$x - y$	$x + (2^n - 1 - y) = 2^n - 1 + (x - y)$
$-x + y$	$(2^n - 1 - x) + y = 2^n - 1 + (-x + y)$
$-x - y$	$(2^n - 1 - x) + (2^n - 1 - y) = 2^n - 1 + (2^n - 1 - x - y)$

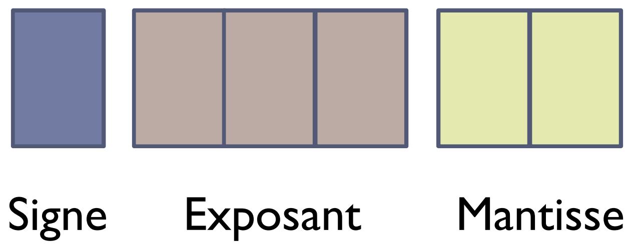
Annexe (recouvrement des nombres)

- Complément à 1
  - Soit $f(x) = 2^n - 1 - x$
  - Theorem: $f(f(x)) = x$
    - Preuve: $f(f(x))$
    - $= f(2^n - 1 - x) = 2^n - 1 - (2^n - 1 - x) = x$

- Complément à 2
  - Soit $g(x) = 2^n - x$
  - Theorem: $g(g(x)) = x$
    - Preuve: $g(g(x))$
    - $= g(2^n - x) = 2^n - (2^n - x) = x$

Exemple (virgule flottante)

- Supposons la représentation suivante:



- Le biais est égal à (2^{k-1} -1=3)
- Il y a un total de 64 symboles à représentés.

Valeurs (non normalisés et particulières)

- 0 000 00 = +0.0
- 0 000 01 = $+0.01\text{x}2^{-2}$ = +0.0001
- 0 000 10 = $+0.10\text{x}2^{-2}$ = +0.001
- 0 000 11 = $+0.11\text{x}2^{-2}$ = +0.0011
- 0 001 00 = $+1.00\text{x}2^{-2}$ = +0.01
- 0 001 01 = $+1.01\text{x}2^{-2}$ = +0.0101
- 0 001 10 = $+1.10\text{x}2^{-2}$ = +0.011
- 0 001 11 = $+1.11\text{x}2^{-2}$ = +0.0111
- 0 010 00 = $+1.00\text{x}2^{-1}$ = +0.1
- 0 010 01 = $+1.01\text{x}2^{-1}$ = +0.101
- 0 010 10 = $+1.10\text{x}2^{-1}$ = +0.11
- 0 010 11 = $+1.11\text{x}2^{-1}$ = +0.111
- 0 011 00 = $+1.00\text{x}2^{0}$ = +1.0
- 0 011 01 = $+1.01\text{x}2^{0}$ = +1.01
- 0 011 10 = $+1.10\text{x}2^{0}$ = +1.1
- 0 011 11 = $+1.11\text{x}2^{0}$ = +1.11
- 0 100 00 = $+1.00\text{x}2^{1}$ = +10.0
- 0 100 01 = $+1.01\text{x}2^{1}$ = +10.1
- 0 100 10 = $+1.10\text{x}2^{1}$ = +11.0
- 0 100 11 = $+1.11\text{x}2^{1}$ = +11.1
- 0 101 00 = $+1.00\text{x}2^{2}$ = +100.0
- 0 101 01 = $+1.01\text{x}2^{2}$ = +101.0
- 0 101 10 = $+1.10\text{x}2^{2}$ = +110.0
- 0 101 11 = $+1.11\text{x}2^{2}$ = +111.0
- 0 110 00 = $+1.00\text{x}2^{3}$ = +1000.0
- 0 110 01 = $+1.01\text{x}2^{3}$ = +1010.0
- 0 110 10 = $+1.10\text{x}2^{3}$ = +1100.0
- 0 110 11 = $+1.11\text{x}2^{3}$ = +1110.0
- 0 111 00 = $+\infty$
- 0 111 01 = NaN
- 0 111 10 = NaN
- 0 111 11 = NaN

Valeurs (non normalisés et particulières)

- 1 000 00 = -0.0
- 1 000 01 = -0.01×2^{-2} = -0.0001
- 1 000 10 = -0.10×2^{-2} = -0.001
- 1 000 11 = -0.11×2^{-2} = -0.0011
- 1 001 00 = -1.00×2^{-2} = -0.01
- 1 001 01 = -1.01×2^{-2} = -0.0101
- 1 001 10 = -1.10×2^{-2} = -0.011
- 1 001 11 = -1.11×2^{-2} = -0.0111
- 1 010 00 = -1.00×2^{-1} = -0.1
- 1 010 01 = -1.01×2^{-1} = -0.101
- 1 010 10 = -1.10×2^{-1} = -0.11
- 1 010 11 = -1.11×2^{-1} = -0.111
- 1 011 00 = -1.00×2^{0} = -1.0
- 1 011 01 = -1.01×2^{0} = -1.01
- 1 011 10 = -1.10×2^{0} = -1.1
- 1 011 11 = -1.11×2^{0} = -1.11
- 1 100 00 = -1.00×2^{1} = -10.0
- 1 100 01 = -1.01×2^{1} = -10.1
- 1 100 10 = -1.10×2^{1} = -11.0
- 1 100 11 = -1.11×2^{1} = -11.1
- 1 101 00 = -1.00×2^{2} = -100.0
- 1 101 01 = -1.01×2^{2} = -101.0
- 1 101 10 = -1.10×2^{2} = -110.0
- 1 101 11 = -1.11×2^{2} = -111.0
- 1 110 00 = -1.00×2^{3} = -1000.0
- 1 110 01 = -1.01×2^{3} = -1010.0
- 1 110 10 = -1.10×2^{3} = -1100.0
- 1 110 11 = -1.11×2^{3} = -1110.0
- 1 111 00 = $-\infty$
- 1 111 01 = NaN
- 1 111 10 = NaN
- 1 111 11 = NaN

Plan du cours

- Historique
- Présentation de l'architecture des ordinateurs
- Représentation interne des informations
- Encodage/décodage de l'information
- Circuits logiques
- Mémoires
- Unité centrale de traitement

Encodage

- Encoder une information en assurant son intégrité et sa compression avec des méthodes simples.
- Utilisation des codes pour représenter l'information pour résoudre les problèmes suivants:
  - **Assurer l'intégrité de l'information**
    - (détection et correction d'erreurs)
  - **Minimiser la taille de l'information (compression),**
  - **Garantir la sécurité de l'information** (encryptage/chiffrement).

Code détecteurs et correcteurs d'erreur

- Une information peut subir des modifications involontaires lors de sa transmission ou lors de son stockage en mémoire.
- → Utiliser des codes permettant de détecter ou même de corriger les erreurs.
- → Utiliser des bits supplémentaires (de contrôle) à ceux nécessaire pour coder l'information.
  - Codes auto-vérificateurs (e.g., contrôle de parité),
  - Codes auto-correcteurs (e.g., double parité, hamming, codes polynomiaux).

Contrôle de parité

- Code auto-vérificateur le plus simple.
- A un mot de taille 'm', on ajoute 1 bit de parité.
- "parité paire": la valeur du bit de parité est a 1 si le nombre de bit a 1 du mot 'm+1' est pair.
  - Exemple en parité paire : 7+1 bits
  - Valide :
    - **1** 1 0 0 1 1 0 0
    - **0** 1 0 0 1 1 0 1
  - Erreur :
    - **0** 1 0 0 1 1 0 0
- Si un bit est change par erreur la parité n'est plus vérifiée. L'erreur est détectée (mais pas corrigée).
  - →Retransmission de l'information.

Contrôle de double parité

- Code obtenu en effectuant un double contrôle de parité.
- A un mot de taille m, on ajoute 1 bit de parité transversal.
- Apres une série de mot, on ajoute 1 mot de parité (longitudinal).
  - **Exemple en double parité impaire : 7+1 bits et série de 4 mots.**

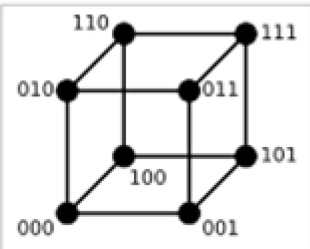
	No de bit 1	2	3	4	5	6	7	bit de parité	contrôle transversal
1. car.= 1	0	1	1	1	0	0	1	0	← faux
2. " = 9	0	1	1	1	0	0	1	1	OK
3. " = 6	0	1	1	0	1	1	0	1	OK
4. " = 8	0	1	1	1	0	0	0	0	OK
bit de parité	1	1	1	1	0	0	1		
contrôle longitudinal	OK	OK	OK	↑ faux	OK	OK	OK		

- Si un bit est change, l'erreur est détectée et corrigée.

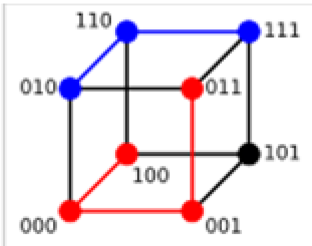
La distance de Hamming

- La distance de Hamming est le nombre de bits à changer pour passer d'une configuration de bis à une autre.
  - Exemple 10010101 & 10011001 à une distance de 2

- Pour n'importe quelle code incluant des membres ayant des distances de Hamming de 2, une erreur d'1 bit peut être détectée. Pourquoi?



Cube binaire de 3 bits pour trouver les distances de Hamming



100 → 011 a une distance de 3 (rouge)
010 → 111 a une distance de 2 (bleu)

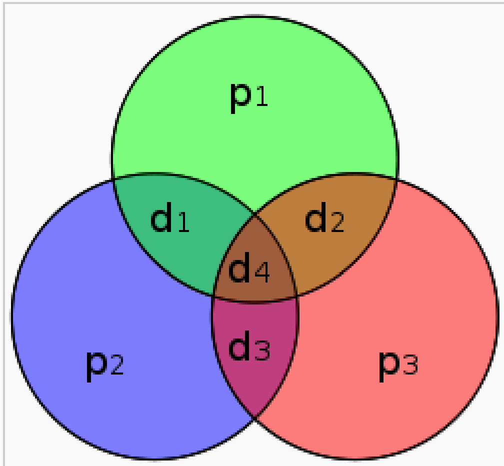
Codes de Hamming

- Les codes de Hamming sont utilisés télécommunication et en compression
- [7,4] codes de Hamming binaire
  - Soit notre mot (x_1 x_2 … x_7)
  - x_3, x_5, x_6, x_7 représente le message lui-même.
  - $x_4 := x_5 + x_6 + x_7 \pmod 2$
  - $x_2 := x_3 + x_6 + x_7$
  - $x_1 := x_3 + x_5 + x_7$

$$
\begin{array}{ccc}
(0\ 0\ 0\ 0) & \rightarrow & (0\ 0\ 0\ 0\ 0\ 0\ 0) \\
(0\ 0\ 0\ 1) & \rightarrow & (1\ 1\ 0\ 1\ 0\ 0\ 1) \\
(0\ 0\ 1\ 0) & \rightarrow & (0\ 1\ 0\ 1\ 0\ 1\ 0) \\
(0\ 0\ 1\ 1) & \rightarrow & (1\ 0\ 0\ 0\ 0\ 1\ 1) \\
(0\ 1\ 0\ 0) & \rightarrow & (1\ 0\ 0\ 1\ 1\ 0\ 0) \\
(0\ 1\ 0\ 1) & \rightarrow & (0\ 1\ 0\ 0\ 1\ 0\ 1) \\
(0\ 1\ 1\ 0) & \rightarrow & (1\ 1\ 0\ 0\ 1\ 1\ 0) \\
(0\ 1\ 1\ 1) & \rightarrow & (0\ 0\ 0\ 0\ 1\ 1\ 1) \\
 & \vdots &
\end{array}
$$

Le code de Hamming [7,4]

- Soit a = $x_4 + x_5 + x_6 + x_7$ (=1 Ssi un de ces bits est en erreur)
- Soit b = $x_2 + x_3 + x_6 + x_7$
- soit c = $x_1 + x_3 + x_5 + x_7$



Représentation graphique des quatre bits de données et des trois bits de parité

- Si erreur (assume en plus une) alors

'abc' est la représentation binaire

de l'indice de bit d'erreur.

- Si (y_1 y_2 … y_7) est le résultat et abc ≠ 000, alors on assume que le bit 'abc' est une erreur et on le change. Si 'abc'=000, on assume pas d'erreur.

Exemple utilisation de L_3

- Suppose (1 0 1 0 0 1 0) est réceptionné.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

100 est 4 en binaire, donc le message initial était (1 0 1 1 0 1 0).

Compression

- Objectif

  - Diminuer le nombre de bits utilisés pour le stockage et la transmission des informations. Les algorithmes de compression se caractérisent par les facteurs suivants :

    - **le taux de compression,**

    - **la qualité de compression** (avec ou sans perte d'information),

    - **le temps de compression.**

Codage de huffman

- Algorithme de compression sans perte, très connu.
- Réduit le nombre de bits utilisés pour représenter les caractères les plus fréquent,
- Augmente le nombre de bits utilisés pour représenter les caractères peu fréquent,
- Un arbre binaire donne le codage pour chaque caractère.

Codage de Huffman: Exemple simple

- Suppose on a un message qui consiste de 5 symboles, e.g. [►♣♣♠☻►♣☼► ☻]
- Comment peut on coder ce message utilisant des 0/1 pour que le message ait une longueur minimale (pour transmission ou sauvegarde)

- 5 symboles → au moins 3 bits
- Pour un encodage simple,
  - la longueur du code est 10*3=30 bits

►	000
♣	001
☻	010
♠	011
☼	100

Codage de Huffman: Exemple simple

- Intuition: Les symboles les plus fréquents doivent avoir des codes plus petits. Seulement, on doit distinguées chaque code, puisqu'ils n'auront pas la même longueur.

- Pour le code de Huffman

Le message sera ▶♣♣♠☻▶♣☼▶☻

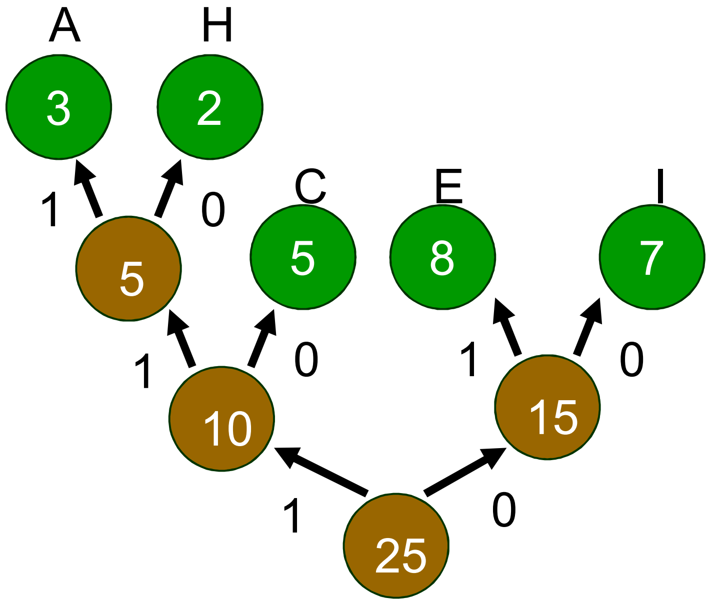
=3*2 +3*2+2*2+3+3=24bits

Symbol	Freq.	Code
▶	3	00
♣	3	01
☻	2	10
♠	1	110
☼	1	111

Codage de l’algorithme de Huffman

1. Prendre les symboles les mois probables de l’alphabet

2. Combiner ces deux symboles en un seul symbole , et on répète.



E	=	01
I	=	00
C	=	10
A	=	111
H	=	110

Codage de l'algorithme de Huffman

Caractère	Fréquence	Fixé	Huffman
E	125	0000	110
T	93	0001	011
A	80	0010	000
O	76	0011	001
I	73	0100	1011
N	71	0101	1010
S	65	0110	1001
R	61	0111	1000
H	55	1000	1111
L	41	1001	0101
D	40	1010	0100
C	31	1011	11100
U	27	1100	11101
Total	838	4.00	3.6229

Plan du cours

- Historique
- Présentation de l'architecture des ordinateurs
- Représentation interne des informations
- Encodage/décodage de l'information
- **Circuits logiques**
- Mémoires
- Unité centrale de traitement

Objectif

- Maîtriser les bases de l'algèbre booléenne.
- Synthétiser et analyser un circuit combinatoire .
- Connaître les circuits logiques les plus importants.
- Comprendre les principes des circuits séquentiels et des bascules.

Circuit logique

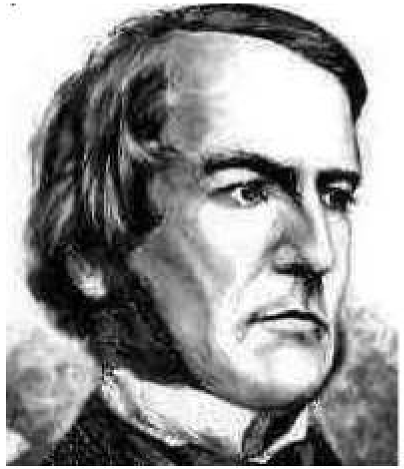
- Les circuits des machines électroniques modernes ont 2 états d'équilibre 0 et 1 (i.e., 2 niveaux de tension)) signal logique
- Une variable binaire est une variable qui ne peut prendre que deux états.

- **Circuit logique**
  - Représentation d'un circuit électronique. Exécute des opérations sur des variables logiques, transporte et traite des signaux logiques.

- **Circuit combinatoire**
  - circuit idéalisé
  - pas de prise en compte du temps de propagation des signaux
  - signaux de sortie ne dépendent que des signaux en entrée

- **Circuit séquentiel**
  - tiens compte du temps de propagation
  - mémoire
  - signaux de sortie dépendent également des signaux en entrée antérieurs

Algèbre de Boole



- Mathématicien britannique (1815-1864). Un des promoteurs de la logique mathématiques contemporaine
- George Boole a défini une algèbre qui s'applique a des fonctions logiques de variables logiques (variables booléennes).
  - Toute fonction logique peut être réalisée à partir de fonctions logiques de base.
  - Les opérations arithmétiques peuvent être réalisées à l'aide d'opérations logiques de base.
- Shannon découvrit que l'algèbre des classes de Boole était un outil puissant, qui permettait d'analyser et de représenter les circuits complexes, basés sur un fonctionnement à deux états.

Algèbre de Boole

- **Fonction logique**
  - Fonction définie par une table de vérité (i.e., tableau de correspondance entre les états d'entrée et les états de sortie)
  - Toutes les combinaisons possibles des variables d'entrées.
  - Représentée sous forme de diagramme ou d'expressions algébrique.
  - Trois operateurs de base : NON, ET, OU

- **Table de vérité**
  - La table de vérité d'une fonction de n variables a autant de ligne que d'états d'entrée, soit 2^n. Comme pour chacun de ces états d'entrées, on peut avoir deux valeurs de sorties (0 et 1), cela nous donne $2^{2pui(n)}$ fonctions possibles a n variables.

  - pour 1 variable, 4 fonctions pour 2 variables, 16 fonctions
  - pour 3 variables, 256 fonctions pour 4 variables, 65536 fonctions

Fonctions d'une variable

Entrée a	Z_0	Z_1	Z_2	Z_3
0	0	0	1	1
1	0	1	0	1

Z_0 = 0 constante

Z_1 = a constante

$Z_2 = \overline{a}$ constante

Z_3 = 1 constante

- Operateur NON
  - La seule fonction logique a une variable non triviale est la fonction de complémentation (Z_2) réalisée par l'opérateur logique NON (ou inverseur)

Table de vérité

Entrée a	NON $\overline{a}$
0	1
1	0

Fonctions à 2 variables

- Il existe 16 fonctions logiques a 2 variables. Les deux non triviales les plus importantes sont les fonctions de produit logique (intersection) et somme logique (réunion) réalisées par les operateurs **ET** et **OU**, notés respectivement **a.b** et **a + b**.

Entrée		ET
a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

Entrée		OU
a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

Fonctions à 2 variables

Entrée		NOR
a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

Entrée		NAND
a	b	$\overline{a.b}$
0	0	1
0	1	1
1	0	1
1	1	0

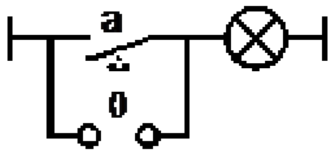
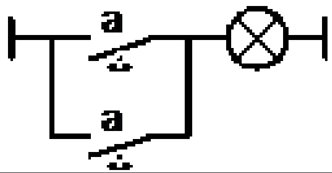
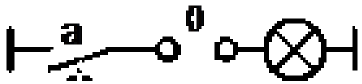
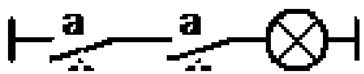
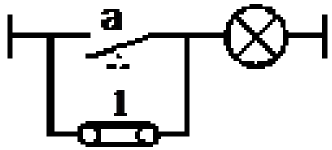
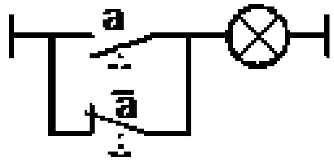
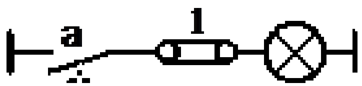
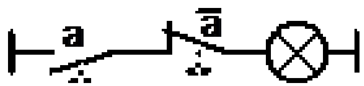
Entrée		XOR
a	b	a xor b
0	0	0
0	1	1
1	0	1
1	1	0

Fonctions à 2 variables

- Il y a 16 fonctions possibles de deux variables a,b

00	01	10	11	ab	
0	0	0	0	$F_0 = 0$	Constante 0
0	0	0	1	$F_1 = a.b$	Fonction ET
0	0	1	0	$F_2 = a.\overline{b}$	
0	0	1	1	$F_3 = a$	
0	1	0	0	$F_4 = \overline{a}.b$	
0	1	0	1	$F_5 = b$	
0	1	1	0	$F_6 = a \oplus b$	Fonction XOR
0	1	1	1	$F_7 = a+b$	Fonction OU

Relations particulières

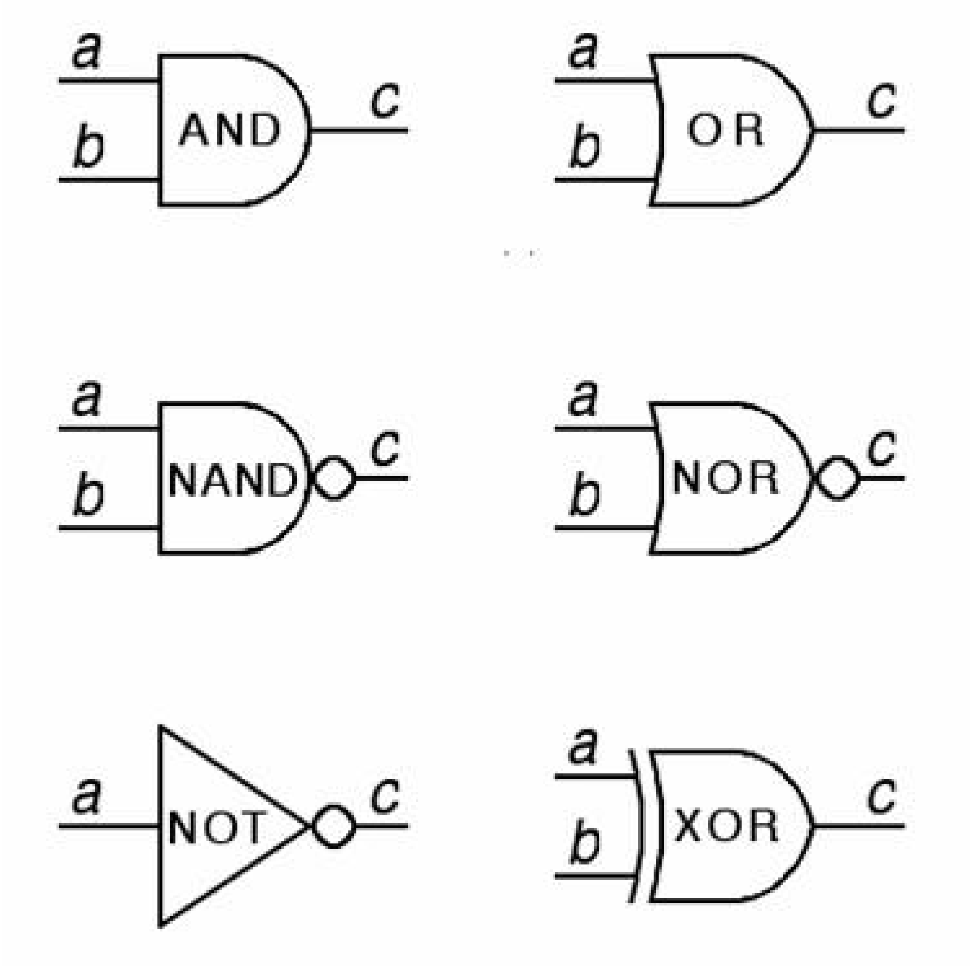
Représentation électrique	Equation	Représentation électrique	Equation
	$a + 0 = a$		$a + a = a$
	$a \cdot 0 = 0$		$a \cdot a = a$
	$a + 1 = 1$		$a + \bar{a} = 1$
	$a \cdot 1 = a$		$a \cdot \bar{a} = 0$

Opérateurs complets

- En pratique, [ET, OU , NON] permet bien d'exprimer tous les operateurs, mais il n'est pas minimal.
  - On peut réaliser la fonction ET avec des OU et des NON et la fonction OU avec des ET et des NON.

- Deux autres operateurs important du point de vue théorique dans l'algèbre de Boole :
  - les operateurs NAND (non et) et NOR (non ou).
  - Ces fonctions forment un ensemble complet ou minimal, c'est a dire qu'ils peuvent exprimer tous les operateurs.

Symboles des opérateurs logiques



Construire la table de vérité

- $f(a,b,c) = a + \overline{b}.c$

a	b	c	$\overline{b}.c$	f(a,b,c)
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

Théorème fondamentaux de l'algèbre de Boole

- Théorème des constantes
  - $a + 0 = a$ et $a.0 = 0$
  - $a + 1 = 1$ et $a.1 = a$
- Idempotence
  - $a + a = a$
  - $a.a = a$
- Complémentation
  - $a + \overline{a} = 1$
  - $a.\overline{a} = 0$
- Commutativité
  - $a + b = b + a$
  - $a.b = b.a$
- Distributivité
  - $a + (bc) = (a + b)(a + c)$
  - $a(b + c) = (ab) + (ac)$
- Associativité
  - $a + (b + c) = (a + b) + c = a + b + c$
  - $a(bc) = (ab)c = abc$

Théorème fondamentaux de l'algèbre de Boole

- Théorème de De Morgan
  - $\overline{ab} = \overline{a} + \overline{b}$
  - $\overline{a + b} = \overline{a}.\overline{b}$

- Autres relations
  - $\overline{\overline{a}} = a$
  - $a + (ab) = a$
  - $a + (\overline{a}b) = a + b$
  - $a(a + b) = a$
  - $(a + b)(a + b) = a + b$

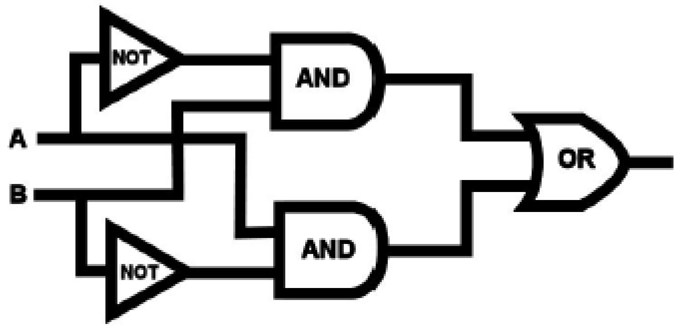
Absorptions

Propriétés du XOR

- $a \oplus b = \bar{a}b + a\bar{b}$
- $\overline{a \oplus b} = ab + \bar{a}\,\bar{b}$
- $a \oplus 0 = a$
- $a \oplus a = 0$
- $a \oplus 1 = \bar{a}$
- $a \oplus \bar{a} = 1$
- $a \oplus b = b \oplus a$
- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

Exemple l'opérateur XOR

- On veut exprimer la fonction XOR (ou exclusif) en n'utilisant que les fonctions ET, OU , NON :
- avec la méthode des minterms : (prochain slide)
  - $a \oplus b = \overline{a}b + a\overline{b}$
- avec la méthode des maxterms : (prochain slide)
  - $a \oplus b = (a + b)(\overline{a} + \overline{b})$



entrées		XOR
a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Méthode des minterms et des maxterms

- A l'aide des théorèmes précédents, il est possible d'exprimer toute fonction logique a l'aide des operateurs NON, ET, OU.
- Méthodes des minterms (i.e., somme logique des produits logiques)
  - La fonction peut être exprimée comme étant la ***somme logique*** des **minterms** correspondant a chaque sortie valant 1 dans la table de vérité. Chaque variable d'entrée est prise telle quelle si elle a la valeur 1, sinon elle est remplacée par son complément.
- Méthodes des maxterms (i.e., produit logique des sommes logiques)
  - La fonction peut être exprimée comme étant le ***produit logique*** des **maxterms** correspondant a chaque sortie valant 0 dans la table de vérité. Chaque variable d'entrée est prise telle quelle si elle a la valeur 0, sinon elle est remplacée par son complément.
- L'expression algébrique obtenu est dite forme normale (ou canonique).

Exemple des minterms

- minterm = produit logique de toutes les variables d'entrées correspondant a une sortie a 1.

a	b	c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$$f\,(a, b, c) = \overline{a}\,\overline{b}c + \overline{a}bc + a\overline{b}\,\overline{c}$$

Exemple des maxterms

- maxterm = somme logique de toutes les variables d'entrées correspondant a une sortie a 0.

a	b	c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$$f(a, b, c) = (a + b + c)(a + \overline{b} + c)(\overline{a} + b + \overline{c})(\overline{a} + \overline{b} + c)(\overline{a} + \overline{b} + \overline{c})$$

Simplification de fonction logique: méthode algébrique

- On utilise les théorèmes de l'algèbre de Boole vu précédemment pour simplifier l'expression algébrique.

- Exemple utilisant la distributivité et la complémentation
- $f(a, b, c) = abc + ab\overline{c} + a\overline{b}c + a\overline{b}\overline{c} + \overline{a}\overline{b}c$

$$= ab(c + \overline{c}) + a\overline{b}(c + \overline{c}) + \overline{a}\overline{b}c$$

$$= ab + a\overline{b} + \overline{a}\overline{b}c$$

$$= a(b + \overline{b}) + \overline{a}\overline{b}c$$

$$= a + \overline{a}\overline{b}c$$

$$= (a + \overline{a})(a + \overline{b})(a + c)$$

$$= a + \overline{b}c$$

Simplification de fonction logique : Table de Karnaugh

- Basée sur l'inspection visuelle de tables judicieusement construites (table de vérité a 2 dimensions).

  - On attribue la valeur 1 aux cases correspondantes aux états d'entrée ou la fonction est vraie, sinon on attribue 0.

  - Regroupement par blocs rectangulaires de 2, 4 ou 8, 2^n variables, des cases a 1 adjacentes.
    - Attention la table se referme sur elle-même.
    - Une case a 1 peut appartenir a plusieurs blocs.
    - Blocs les plus gros possibles (on utilise un bloc une seule fois).

  - Pour chaque bloc :
    - Si une variable prend comme valeur 0 et 1, on ne la prend pas en compte.
    - On garde les variables dont la valeur ne varie pas.
    - Operateur = ET.

  - **OU** de tous les termes de tous les blocs. (on somme tous les termes des blocs)

Table de Karnaugh à 2 variables

- Table de vérité :
  - Expression algébrique canonique (minterms) :
  - $f(a, b) = \bar{a}b + a\bar{b} + ab$

a	b	f(a,b)
0	0	0
0	1	1
1	0	1
1	1	1

b \ a	0	1
0	0	1
1	1	1

$f(a, b) = a + b$

Table de Karnaugh à 3 variables

- **Table de vérité :**
  - Expression algébrique canonique (minterms) :
  - $f(a, b) = \bar{a}\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c} + abc$

a	b	c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

c \ ab	00	01	11	10
0	0	0	1	1
1	1	0	1	1

$$f(a, b) = a + \bar{b}c$$

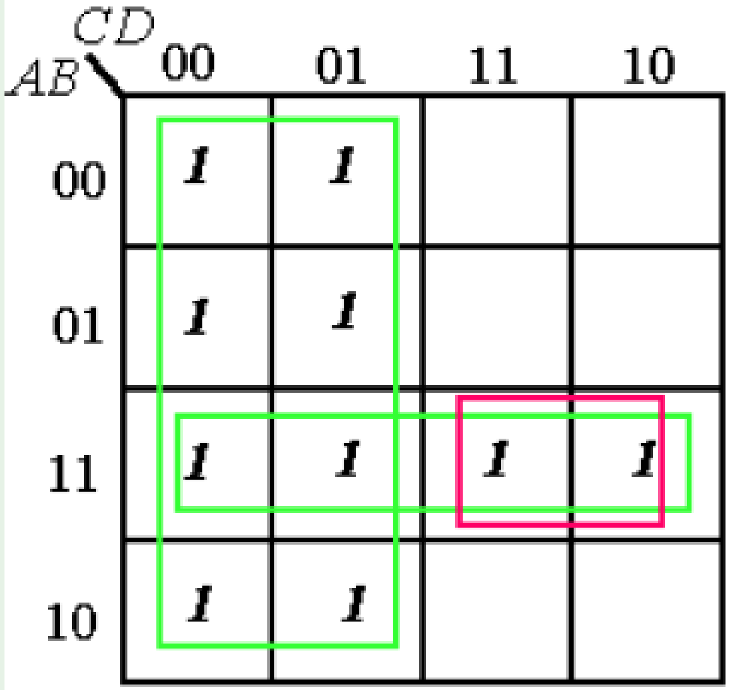
Table de Karnaugh à 4 variables

- Table de vérité :
- Expression algébrique canonique (minterms) :
  - $f(a; b; c; d) = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + ab\bar{c}d +$
    $a\bar{b}cd + a\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + a\bar{b}c\bar{d}$

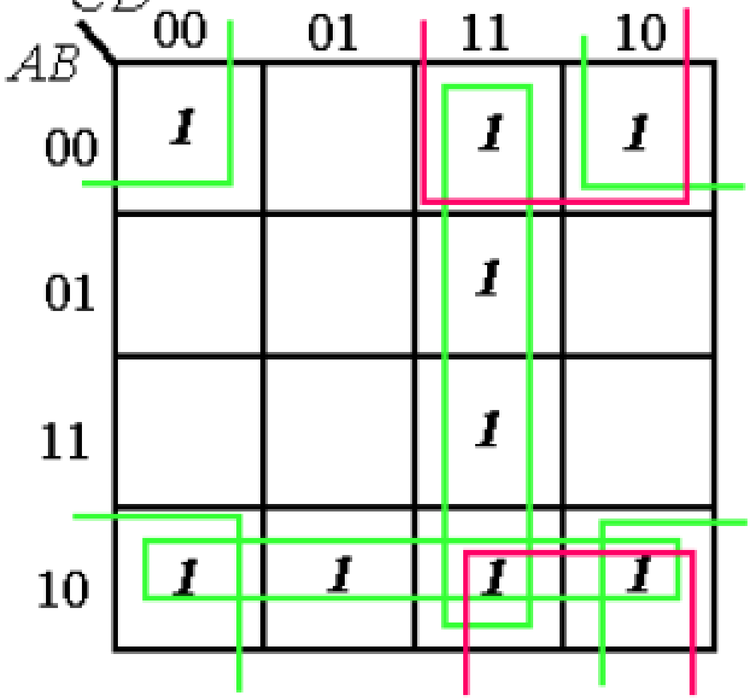
cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	1	1
11	0	0	0	1
10	1	0	0	1

$f(a, b) = a\bar{b} + \bar{b}\bar{d} + b\bar{c}d$

Autres exemples



$\bar{C} + AB$



$A\bar{B} + CD + \bar{B}\bar{D}$

Synthèse d'un circuit combinatoire

Méthode de synthèse

- A partir d'une fonction logique, déterminer un circuit logique réalisant cette fonction et obtenir le meilleur (i.e., le plus simple en nombre de portes, de connexions) :

1. construire la table de vérité de la fonction logique ;
2. dériver une expression algébrique (par exemple par la méthode des minterms) ;
3. simplifier cette expression (méthode algébrique ou tables de Karnaugh) ;
4. réaliser la fonction logique a l'aide d'operateurs divers (NON, ET, OU, XOR, NAND, NOR, etc.) pour obtenir un logigramme.

Analyse d’un circuit combinatoire

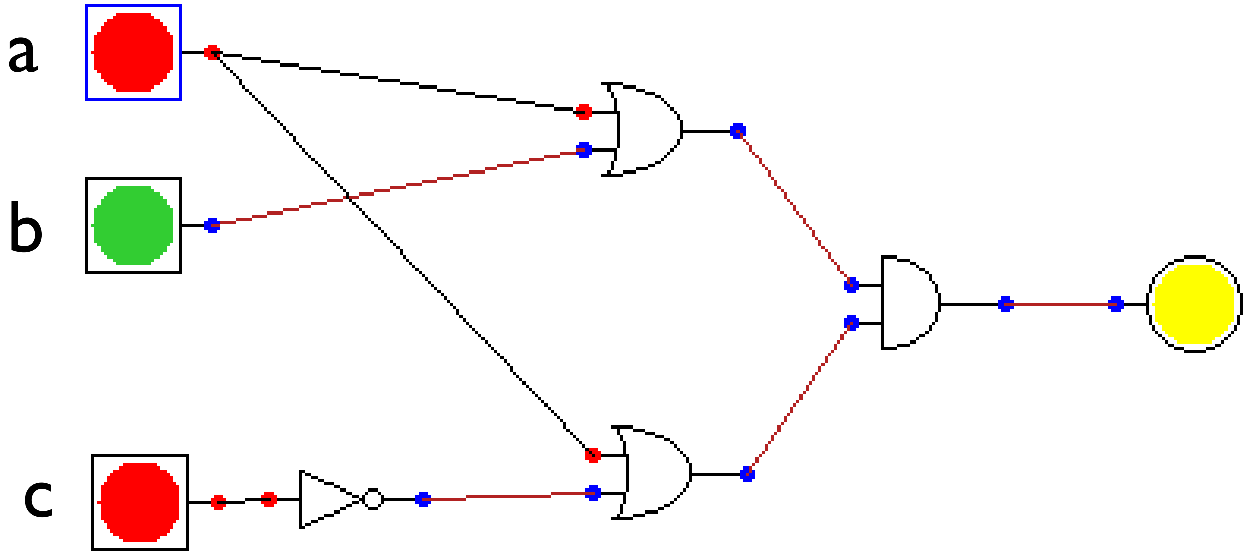
- L'analyse est l'opération inverse de la synthèse.

Méthode de d'analyse

- Retrouver la fonction d'un circuit dont on connaît uniquement le logigramme :

1. En procédant des entrées vers les sorties, donner, pour chaque operateur l'expression de sa sortie en fonction de ses entrées, jusqu'à obtention d'une expression pour chaque fonction réalisée par le circuit ;
2. Donner la table de vérité correspondante ;
3. En déduire le rôle du circuit.

Exemple d'analyse



$$f(a, b, c) = (a + b)(a + \bar{c})$$

Circuits logiques les plus importants

- Demi-additionneur (addition sans gestion de la retenue) et additionneur complet (addition avec gestion de la retenue) ;

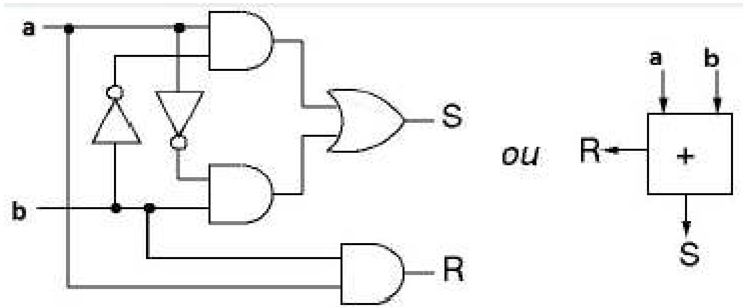
- Multiplexeur (plusieurs signaux en entrées, 1 seule sortie) et démultiplexeur (un seul signal en entrée et plusieurs sorties) ;

- Décodeur, codeur et transcodeur (e.g., conversion de base).

Synthèse d'un demi additionneur

- Circuit logique capable de faire la somme de 2 nombres binaires mais qui ne tient pas compte de la retenue éventuelle provenant d'une opération précédente.

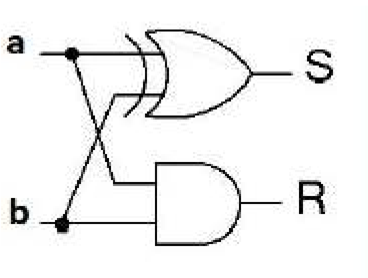
a	b	Sortie S	Retenue R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Expression algébrique canonique (minterms) :

$S = a\bar{b} + \bar{a}b = a \oplus b$

$R = ab$



Synthèse d’un étage additionneur

- Circuit logique capable de faire la somme de 2 nombres binaires et d'une retenue provenant d'une opération précédente.

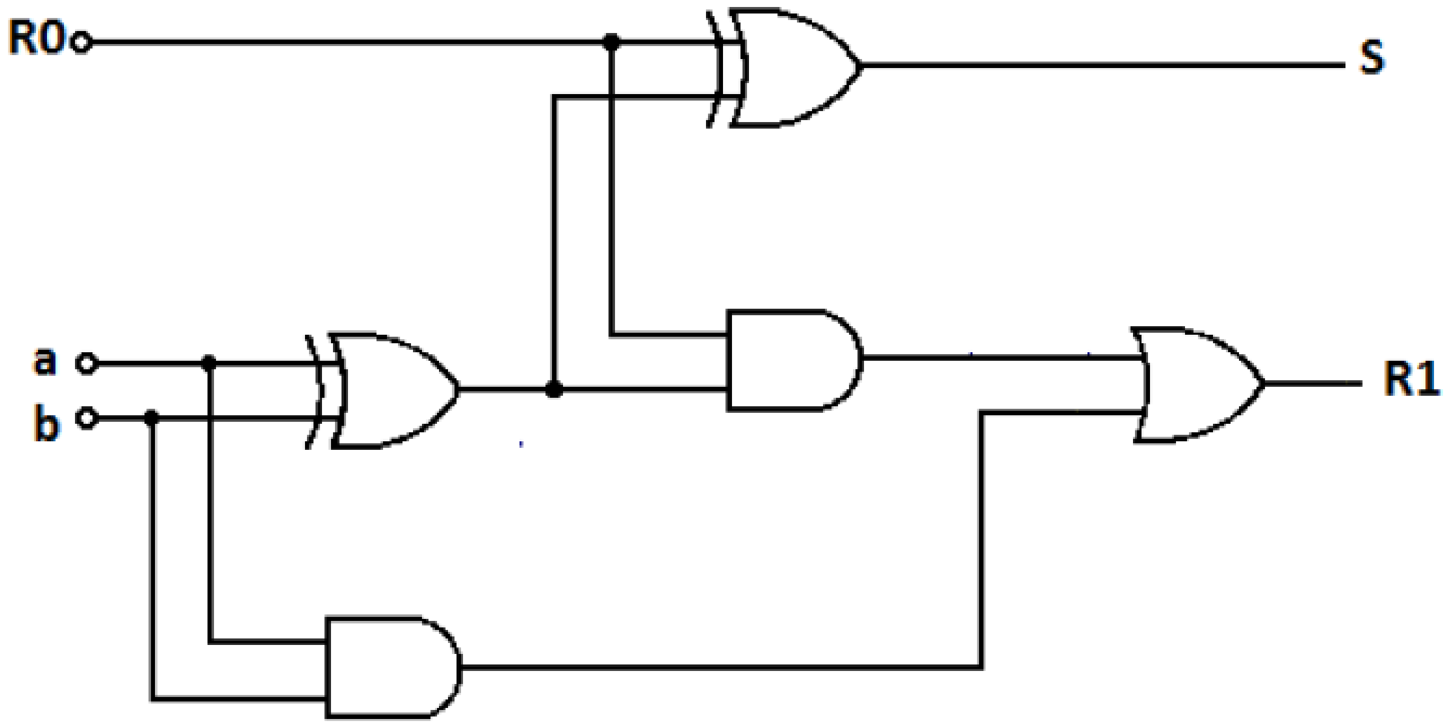
a	b	R0	Sortie S	R1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Expression algébrique canonique (minterms) :

$$
\begin{aligned}
S &= \overline{a}\,\overline{b}R_0 + \overline{a}b\,\overline{R_0} + a\overline{b}\,\overline{R_0} + abR_0 \\
&= R_0(\overline{a}\,\overline{b} + ab) + \overline{R_0}(\overline{a}b + a\overline{b}) \\
&= R_0(\overline{a \oplus b}) + \overline{R_0}(a \oplus b) \\
&= R_0 \oplus (a \oplus b)
\end{aligned}
$$

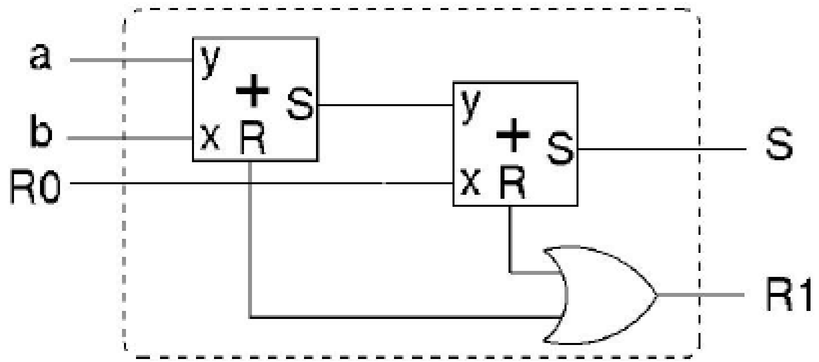
$$
\begin{aligned}
R_1 &= \overline{a}bR_0 + a\overline{b}R_0 + ab\overline{R_0} + abR_0 \\
&= R_0(\overline{a}b + a\overline{b}) + ab(R_0 + \overline{R_0}) \\
&= R_0(a \oplus b) + ab
\end{aligned}
$$

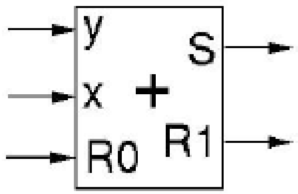
Logigramme d'un étage additionneur



Additionneur binaire complet

- L'étage d'additionneur est compose de 2 demi-additionneurs et d'un OU. Il fait la somme de 2 bits en tenant compte d'une éventuelle retenue.

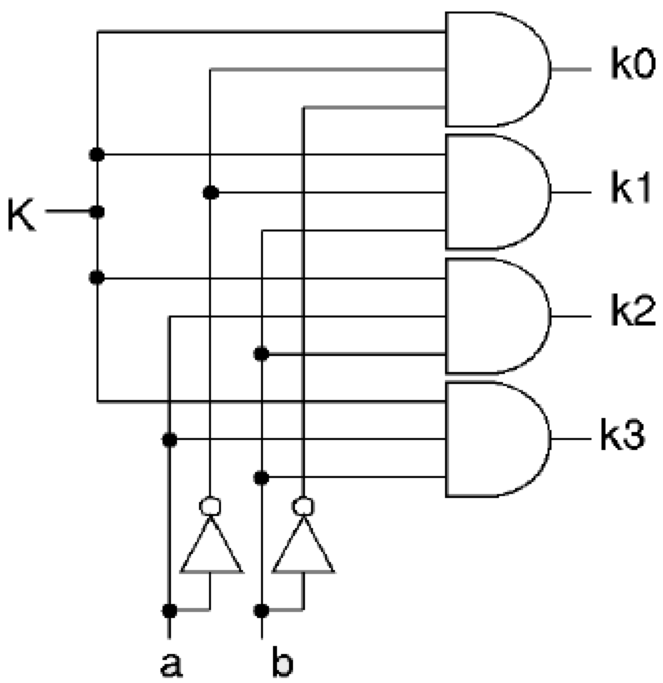


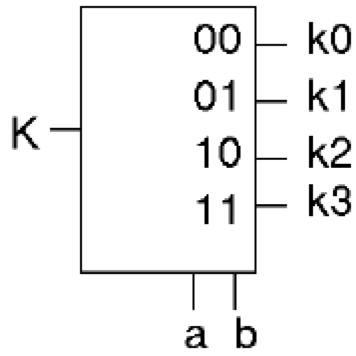


- L'additionneur complet est obtenu en utilisant en parallèle plusieurs étages additionneurs (il faut autant d'étages que de bits composants les nombre binaires a additionner).

Démultiplexeur

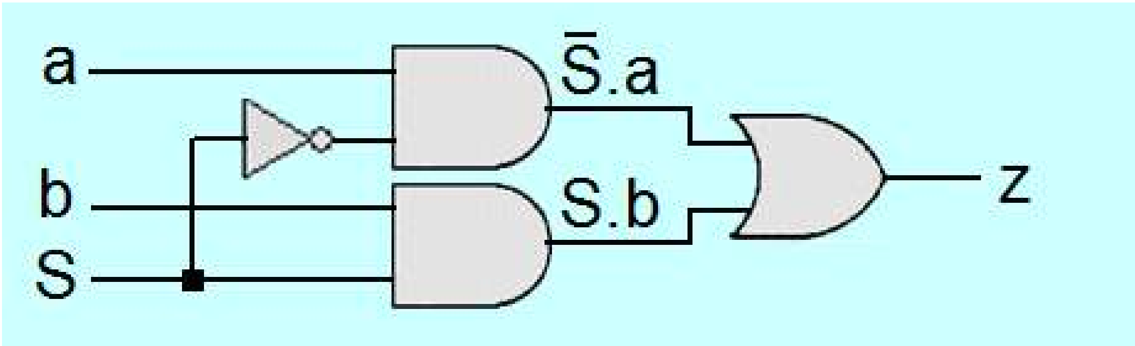
- 1 entrée, n variables, 2^n sorties
- Une des sorties prend la valeur de l'entrée (K) selon la valeur des n variables : la variable K est aiguillée sur l'une des 4 sorties.
- Utile pour choisir la source d'un signal





Multiplexeur

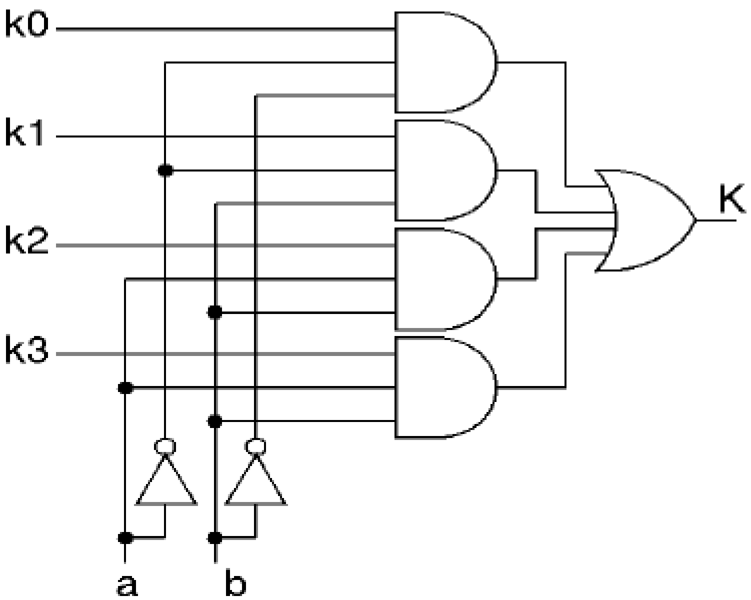
- Multiplexeur 2 bits ou 2 vers 1

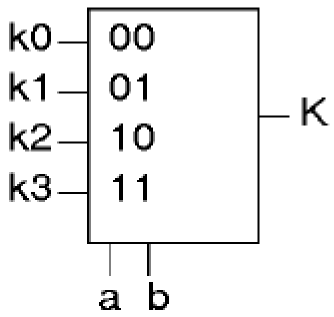


$$z = \bar{S}.a + S.b$$

multiplexeur

- 2^n entrées, n variables, 1 sortie
- La sortie (K) prend la valeur d'une des entrées selon la valeur des n variables : une des 4 entrées est aiguillée sur la sortie K.
- Utile pour choisir la source d'un signal



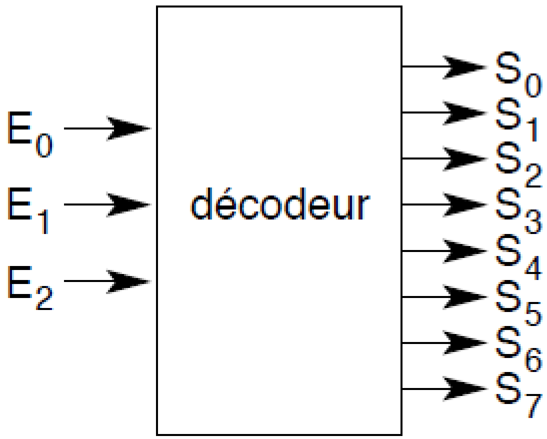


Application de multiplexeurs

- Fonction universelle (i.e., un multiplexeur a n variables peut réaliser les $2^{2\text{puis}(n)}$ fonctions logiques a n variables ;
- Multiplexage (i.e., concentrer plusieurs lignes en une seule ou faire l'opération inverse) ;
- Codage, décodage, transcodage.

Décodeur

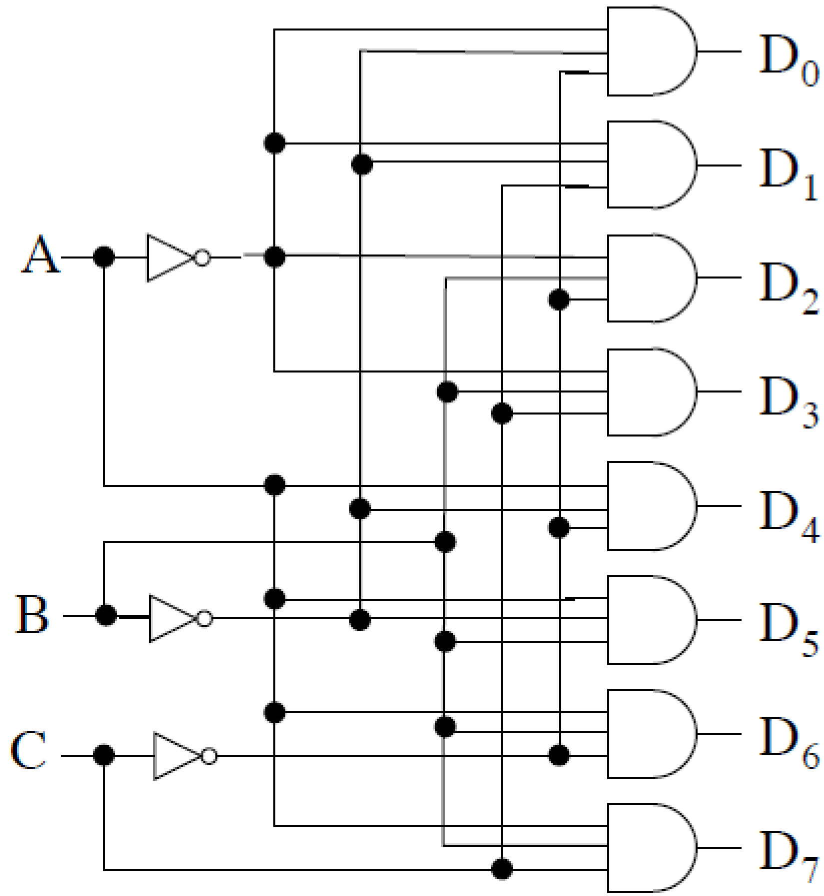
- Fait correspondre a un code en entrée (sur n lignes) une seule sortie active (i.e., a 1) parmi les 2^n sorties possibles.



entrées			sorties							
E_2	E_1	E_0	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

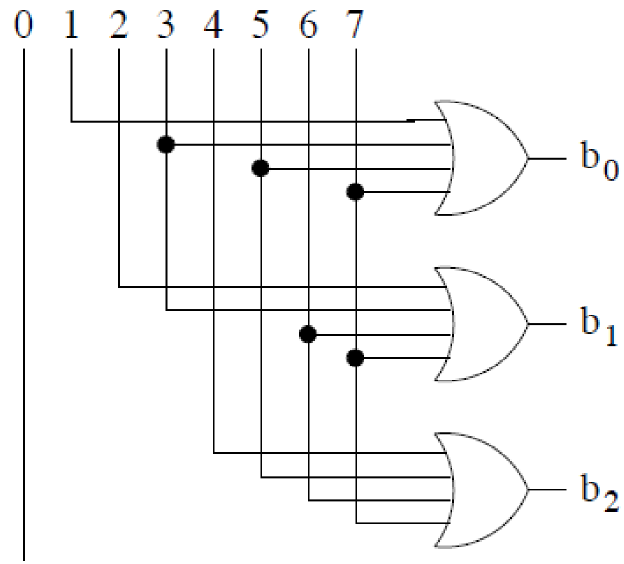
- Le décodeur peut être utilise pour convertir un nombre binaire en nombre décimal ou pour adresser une mémoire.

Décodeur



Codeur

- Fait correspondre a une entrée active, parmi les 2^n entrées, un code sur n lignes en sortie.



- Un transcodeur fait correspondre une entrée sur n lignes correspondant a un certain codage, une sortie sur m lignes correspondant a un autre codage.

Circuits séquentiels

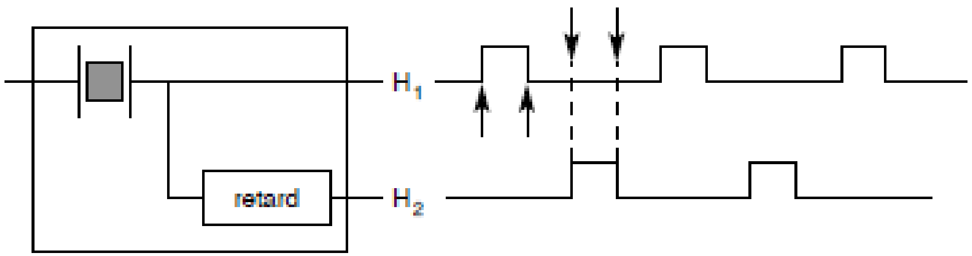
- Circuits combinatoires → pas de rétroactions
  - (i.e., de retours des sorties dans les entrées).

- Les **circuits séquentiels** possèdent des rétroactions :
  - les signaux de sortie ne dépendent pas uniquement des entrées mais aussi de leur séquence.
  - Le circuit se rappelle des entrées et des états précédents : il a une mémoire du passé.

  - Ajout des notions d'états et de mémoire.
  - Ajout de la notion de temps (i.e., horloge).
  - Repose sur la théorie des automates finis.

Horloge

- Besoin de séquentialiser les opérations, pour cela on utilise une horloge :
  - Système logique qui émet régulièrement une impulsion.
  - Deux impulsions = le temps du cycle.
- Ajout de circuit de retardement pour obtenir plusieurs impulsions décalées permettant de décomposer un cycle en plusieurs phases et de synchroniser ainsi les différentes phases.



Cycle compose de 4 phases :

1 le front montant de H1 ;
2 le front descendant de H1 ;
3 le front montant de H2 ;
4 le front descendant de H2 ;

Concept d'un automate fini

- Un **automate fini** (ou machine a états finie) est une machine abstraite constituée d'un nombre fini d'états et de transitions. Un automate fini est caractérisé, entre un temps t et t + 1 par :
  - sa réponse S,
  - son entrée E,
  - son état Q.

- Le comportement d'un automate fini est déterminé par :
  - Ses **fonctions de transfert** :
  - $S(t + 1) = \mathbf{f}(Q(t); E(t))$: sortie a t+1 dépend des entrées et états a t
  - $Q(t + 1) = \mathbf{g}(Q(t); E(t))$: état a t+1 dépend des entrées et états a t

  - Ses **tables de transitions** :
  - Valeurs de Q(t +1) et S(t +1) pour chaque combinaison de valeurs de E(t) et Q(t)

  - Diagrammes d'états (i.e., représentation graphique : état=rond, transition=flèche)

Exemple d'un automate fini (mémoire binaire)

- Mémoire binaire : stocke 1 bit
  - 1 entrée
    - Si 0, on mémorise la valeur 0
    - Si 1, on mémorise la valeur 1
  - 2 états : valeur 0 ou valeur 1

- Principe
  - A *t on est dans un état X, à t+1, on passe dans* un état Y selon la valeur de l'entrée à *t*
  - La sortie S à *t+1 prend la valeur de l'état à t*

Exemple d'un automate fini (mémoire binaire)

- Fonctions de transfert
  - S(t + 1) = Q(t) : la sortie a t+1 est égale à l'état à t
  - Q(t + 1) = E(t) : l'état a t+1 est égal à l'entrée à t
- Table de transitions (i.e., Valeurs de Q(t + 1) et S(t + 1) en fonction de E(t) et Q(t))

S(t+1)

Q(t) \ E(t)	0	1
0	0	0
1	1	1

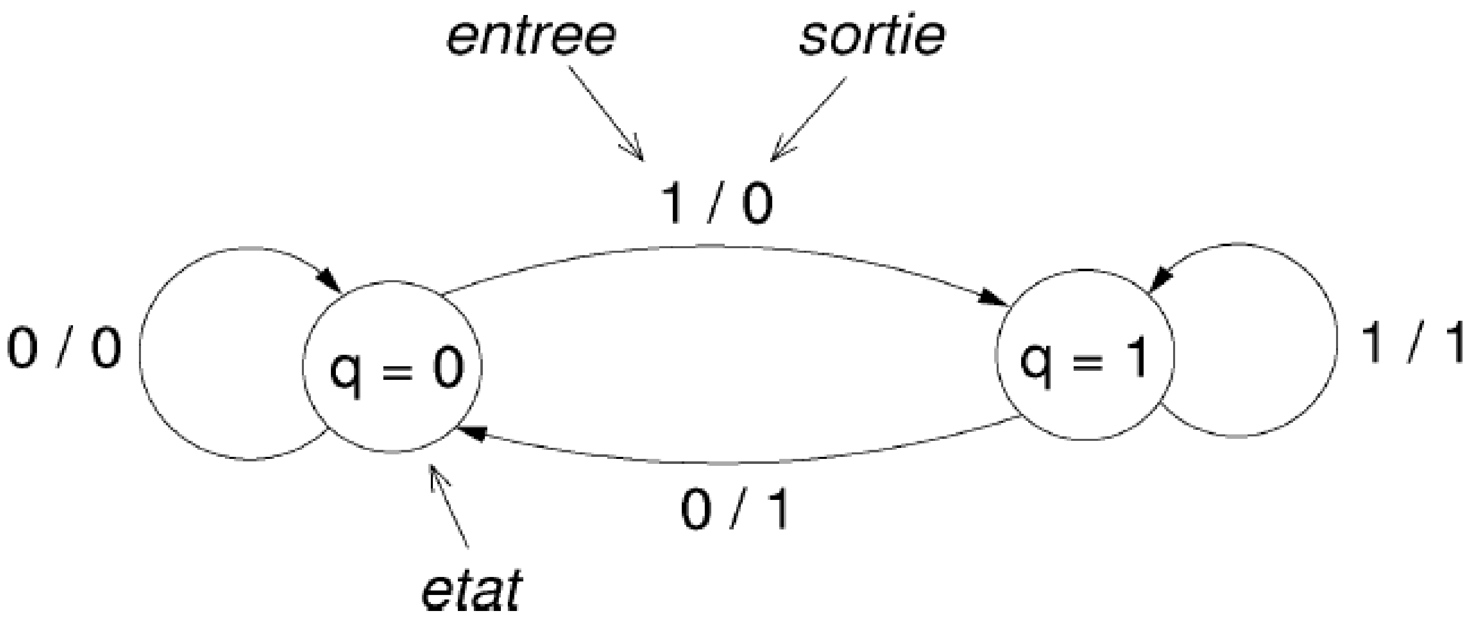
→ Quelque soit l'entrée, c'est l'état qui est renvoyé.

Q(t+1)

Q(t) \ E(t)	0	1
0	0	1
1	0	1

→ Quelque soit l'état, c'est l'entrée qui est mémorisée.

Exemple d’un automate fini (mémoire binaire)



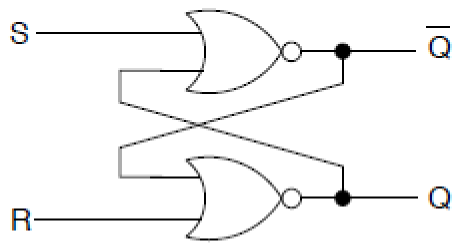
Bistables (Bascules Flip-Flop)

- Automate ayant 2 états stables.
  - Circuit permettant de mémoriser 1 bit.
    - Possède une variable codée sur 1 bit.
    - Valeur conservée et modifiable dans le temps.
  - Plusieurs types de bistables :
    - Asynchrone (bascule RS) : Les sorties sont recalculées a chaque changement des valeurs en entrées.
    - Synchrone (bascule RSh, D, flip-flop) : Les sorties sont recalculées en fonction d'un signal d'horloge en entrée (notée C ou ck pour clock).

Bascule RS

- Stocke 1 bit (→ un mot de n bits = n bascules)
- Accès aux informations/bascules
  - Lecture/écriture de tous les bits d'un mot en parallèle et en même temps (accès mot par mot et non pas bit par bit)
- 2 entrées
  - R (reset) pour la remise a 0.
  - S (set) pour la mise a l'état 1.
- 2 sorties
  - Q
  - $\overline{Q}$
- Réalisé a l'aide de 2 portes NOR

S^n R^n	Q_1^{n+1}	Q_2^{n+1}	
0 0	Q_1^n	$\overline{Q_1^n}$	Stable
1 0	1	0	Set
0 1	0	1	Reset
1 1	0	0	Interdit



la valeur de sortie ne dépend pas **uniquement** des valeurs d'entrées, mais aussi de la valeur antérieure de sortie. En temps normal *S et R valent 0.*

Bascule RS

- S (Set) est activée (=1) pour écrire un bit dans le bistable :
  - Si S = 1 et que Q = 0, alors Q est mis a 1.
    - $\overline{Q} = \overline{S + Q} = \overline{1 + 0} = 0 \rightarrow Q = \overline{R + \overline{Q}} = \overline{0 + 0} = 1$
  - Si S = 0 et que Q = 1, alors Q reste a 1.
    - $\overline{Q} = \overline{S + Q} = \overline{0 + 1} = 0 \rightarrow Q = \overline{R + \overline{Q}} = \overline{0 + 0} = 1$

- R (Reset) est activée (=1) pour effacer un bit dans le bistable :
  - Si R = 1 alors que Q = 1, alors Q est mis à 0
  - Si R = 1 alors que Q = 0, alors Q reste à 0.

- Dans les deux cas on est dans un état stable.

Bascule D

- Recopie sur sa sortie Q l'unique signal applique a son entrée D, avec un retard d'une période d'horloge.

- 2 entrées :
  - D pour la valeur en entrée
  - Ck pour l'entrée de contrôle.

- 1 sortie Q :
  - Qt+1 = D si ck = 1
  - Qt+1 = Q si ck = 0.

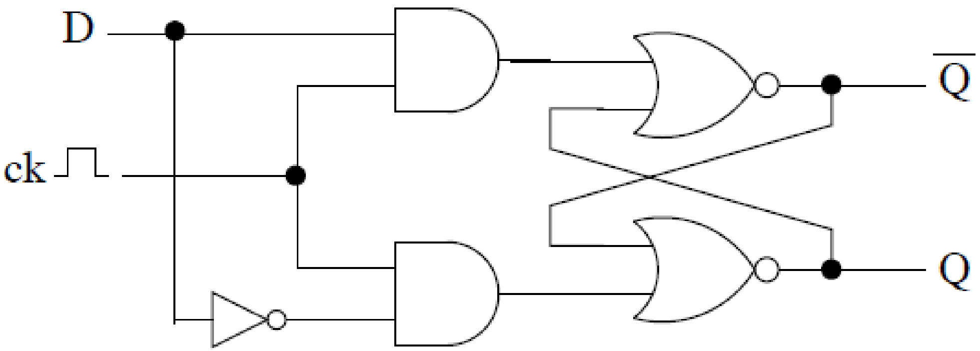
$Q^{n+1} = D^n$

ou

$Q^{n+1} = D^nC + Q^nC$

C	D^n	Q^{n+1}
0	0	Q^n
0	1	Q^n
1	0	0
1	1	1

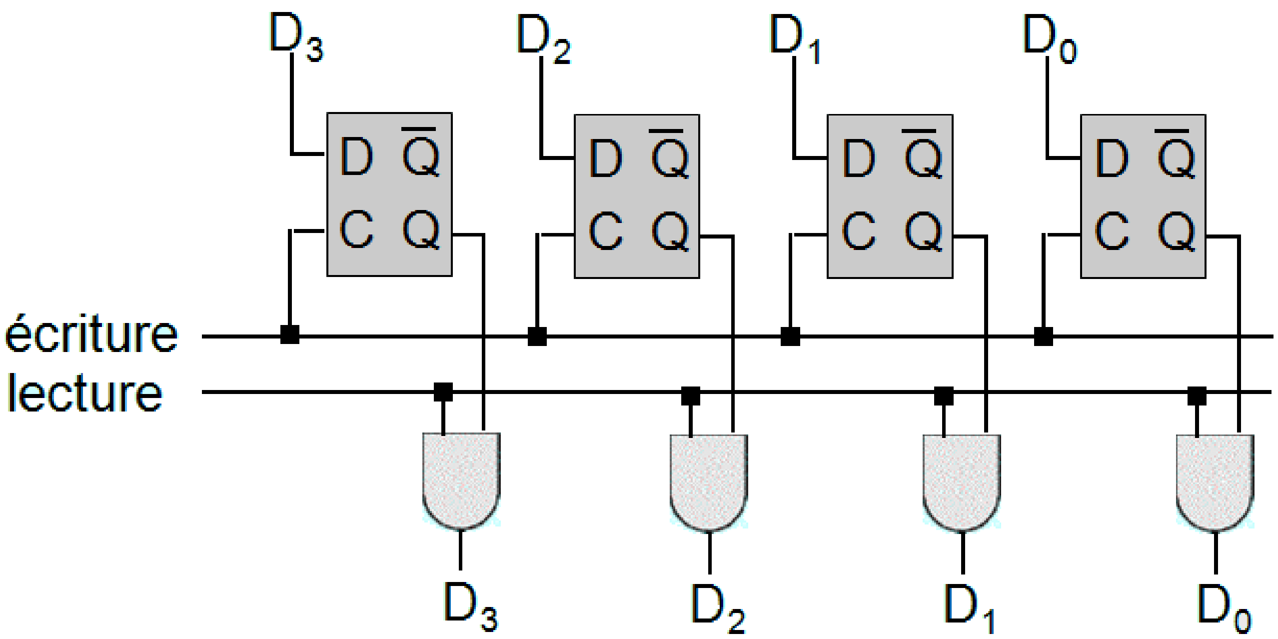
- Le signal d'horloge retarde le signal en entrée D d'une période d'horloge



L'inverseur élimine complètement la possibilité d'avoir la com-binaison 1-1 à l'entrée des NOR.

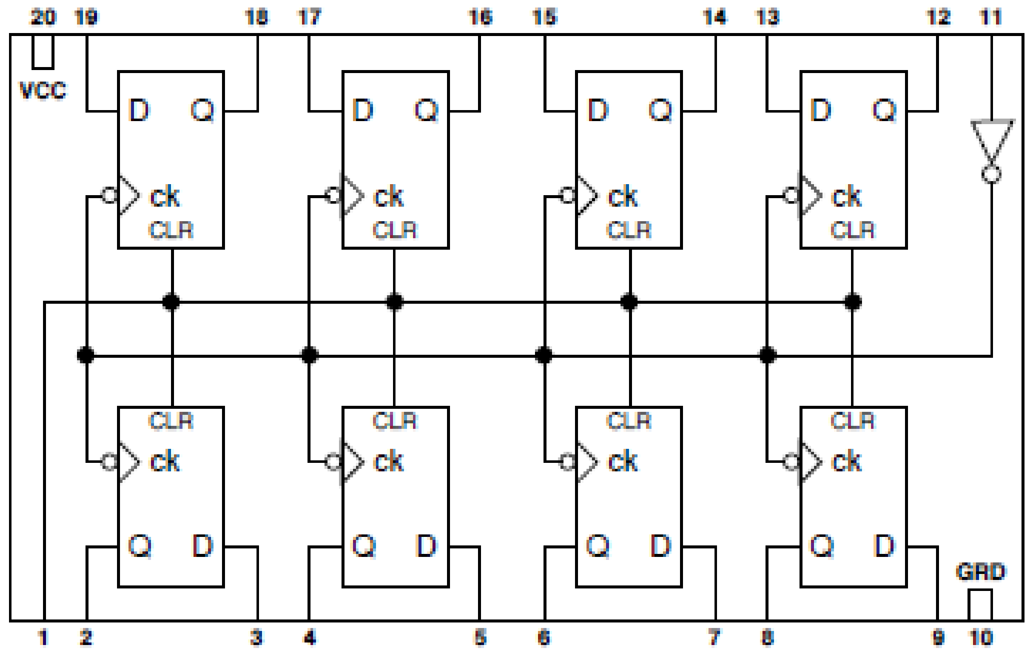
Applications des bistables : les registres

Plusieurs bistable en parallèle permettent de mémoriser plusieurs bits d'information. Ce sont des registres. Ils sont utilises dans un processeur pour stocker des valeurs lors de l'exécution d'un programme.



Applications des bistables : les registres

- Registre 8 bits
  - 8 bascules D.
  - 8 bits en entrée pour écrire le mot.
  - 8 bits en sortie pour récupérer la valeur du mot.



Applications des bistables : les mémoires

- Le problème du schéma du registre 8 bits est qu'il nécessite un nombre de broches trop élevé :
  - Chaque bit nécessite 2 broches (→ 16 broches),
  - L'alimentation électrique et la terre nécessitent 2 broches,
  - Les commandes nécessitent 2 broches

- → Pas de mémoires de forte capacité sur le même schéma ! Il faut économiser les broches.