Architecture des ordinateurs

Gilles TALADOIRE
Université de La Nouvelle-Calédonie
Juin 2012

UE :	UE3 Info 2.3			
Elément constitutif :	Architecture des ordinateurs			
Objectifs :	Comprendre la conception et le fonctionnement des ordinateurs.			
Prérequis :	Aucun			
ECTS :	**3**			
Volume horaire	Cours	TD	TP	**Total**
	14	6	10	**30**
Programme :	Architecture en couches. Structure d'un ordinateur : processeurs, mémoires , bus et E/S. Couche physique. Couche micro-architecture. Couche ISA – Langage assembleur et de haut niveau.			
Modalités d'évaluation	CC + ET (durée 1,5h)			

Références

• Cours Architecture des ordinateurs - Emmanuel Viennet
http://www-gtr.iutv.univ-paris13.fr/Cours/Mat/Architecture
• Support de cours de Robert Racca - http://bv.univ-nc.nc groupe architecture
• Architecture de l'ordinateur - Andrew Tanenbaum - Dunod
• Architecture et technologie des ordinateurs - Paolo Zanella, Yves Ligier – Dunod
• Organisation et architecture de l'ordinateur – William Stallings
• Microprocesseurs - Henri Lilen – Dunod
• Assembleur x86 – Kip Irvine – CampusPress
Assembly Language for Intel-Based Computer - http://kipirvine.com/asm/

pour les images :
• Fou de PC - Anatole d'Hardancourt - Sybex
• Le micro … Comment ça marche ? - Ron White - Pearson
• Les microprocesseurs … Comment ça marche ? - Gregg Wyant, Tucker Hammerstrom - Dunod

Objectifs

Connaître les principes de fonctionnement d'un ordinateur :

Représentation / codage de l'information
Composants d'un ordinateur :
- processeurs : instructions et adressage
- mémoires,
- périphériques d'entrées/sorties

TP : Introduction au langage Assembleur

Exposés et rapports (par groupe de 2 ou 3) sur des thèmes précis

1 - Représentation des informations

Toute information d’un ordinateur (texte, nombres, images, son, vidéo, …) est représentée et manipulée sous forme binaire c’est à dire comme une suite de 0 ou de 1.

Représentation des informations

L'unité de base d'un ordinateur est le "bit".

Un bit ne peut prendre que 2 valeurs "0" ou "1".

Les bits sont regroupés par 8 pour former des octets.

Le binaire (base 2) ---> 0, 1

L'hexadécimal (base 16) ---> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Le code ASCII ---> codage des caractères (sur 1 octet)

Bases de numération

En base *b*, on utilise *b* symboles

Exemples :

En binaire, base 2, $a_i \in \{0, 1\}$

En décimal, base 10, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

En hexadécimal, base 16, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Un nombre entier s'écrit comme une suite de symboles :

$x = a_n a_{n-1} \ldots a_2 a_1 a_0$ ce qui signifie : $x = \sum_{i=0}^{n} a_i b^i$

Exemples :

En binaire, base 2, $(1010)_2 = 1 * 8 + 0 * 4 + 1 * 2 + 0 * 1$

$= 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$

En décimal, base 10, $(1881)_{10} = 1 * 1000 + 8 * 100 + 8 * 10 + 1 * 1$

$= 1 * 10^3 + 8 * 10^2 + 8 * 10^1 + 1 * 10^0$

En hexadécimal, base 16, $(BB)_{16} = 11 * 16 + 11 * 1 = 11 * 16^1 + 11 * 16^0$

Changements de base

Base *b* à 10 ==> multiplications et additions (cf. page précédente)

Base 10 à *b* ==> divisions successives, prise en compte du reste

Exemple :

44 div 2 = 22, reste = 0 = a_0 bit de poids faible

22 div 2 = 11, reste = 0 = a_1

11 div 2 = 5, reste = 1 = a_2

5 div 2 = 2, reste = 1 = a_3

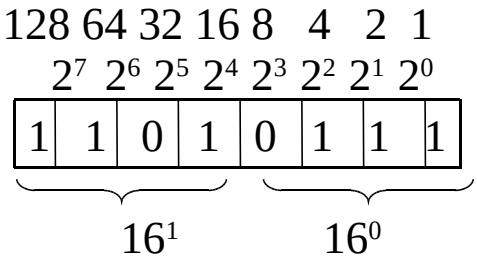
2 div 2 = 1, reste = 0 = a_4

1 div 2 = 0, reste = 1 = a_5 bit de poids fort

$(44)_{10} = (00101100)_2 = (2C)_{16} = (054)_8$

Base 2	base 10	base 16
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Exemple



11010111 en base 2
= 128 + 64 + 16 + 4 + 2 + 1
= 215 en base 10
= D7 en base 16
= 13 × 16 + 7 × 1

Bases de numération pour les nombres fractionnaires

x = $a_n a_{n-1} \dots a_2 a_1 a_0 , a_{-1} a_{-2} \dots a_{-p}$ ce qui signifie : $x = \sum_{i=-p}^{n} a_i b^i$

Exemple en base 10 :

$(12,346)_{10} = 1*10^1 + 2*10^0 + 3*10^{-1} + 4*10^{-2} + 6*10^{-3}$

Exemple en base 2 :

$(54,25)_{10}$ = ?

$(54)_{10} = (00110110)_2$ par divisions successives

0,25 * 2 = 0,5 ==> $a_{-1} = 0$

0,50 * 2 = 1,00 ==> $a_{-2} = 1$

0,00 * 2 = 0,00 ==> $a_{-3} = 0$

$(54,25)_{10} = (00110110,010)_2$

Codage de l’information

Le codage d’une information consiste à établir une correspondance entre la représentation externe (A ou 36) et sa représentation interne (suite de 0 et 1).

Codage des nombres entiers

La représentation des nombres s’effectue sur un nombre de bits fixes.
==> Tous les nombres ne peuvent être représentés
==> Risques de débordement lors des opérations

Entiers naturels (positifs ou nul)
représentation sur 1, 2, 4 ou 8 (+ rare) octets

codage sur *n* bits : représentation des nombres de 0 à 2^n - 1

Exemple :
sur 1 octet, 8 bits : codage des nombres de 0 à 2^8 - 1 = 255
sur 2 octets, 16 bits : codage des nombres de 0 à 2^{16} - 1 = 65535
sur 4 octets, 32 bits : codage des nombres de 0 à 2^{32} - 1 = 4 294 967 295

Entiers relatifs (négatifs, positifs ou nul)
représentation sur 1, 2, 4 ou 8 (+ rare) octets

Plusieurs méthodes :

• **Signe et valeur absolue :**
le bit de poids fort = 1 bit de signe (0 ==> +, 1 ==> -)
les autres = valeur absolue de la valeur

- il existe +0 et -0)
- calculs difficiles) ==> non utilisé

Exemple : $(-2)_{10} = (10000010)_2$

• **BCD ou DCB** (Décimal Codé Binaire)
objectif : se rapprocher de la représentation externe des nombres
indication sur le signe
codage de chaque chiffre sur 4 bits (min. pour compter de 0 à 9)

- utilisé pour la gestion
- cité pour l'histoire …

Exemple : 1881 = 0001 1000 1000 0001

• **Complément à 1 :**
nombre négatif = l'inverse de chaque bit de la valeur absolue

- le bit de poids fort = 1 bit de signe (0 ==> +, 1 ==> -)
- $x + (-x) = 2^n - 1$

Exemple : $(2)_{10} = (00000010)_2$
$(-2)_{10} = (11111101)_2$

• **Complément à 2 :**
nombre négatif = le complément à 1 de la valeur + 1

- le bit de poids fort = 1 bit de signe (0 ==> +, 1 ==> -)
- $x + (-x) = 2^n = 0$ sur n-1 bits
- le plus utilisé, facilité pour l'addition

Exemple : $(2)_{10} = (00000010)_2$
complément à 1 = $(11111101)_2$
$(-2)_{10} = (11111110)_2$

codage sur *n* bits : représentation des nombres de -2^{n-1} à $2^{n-1} - 1$

Exemple :
sur 1 octet, 8 bits : codage des nombres de $-2^7 = -128$ à $2^7 - 1 = +127$
sur 2 octets, 16 bits : codage des nombres de $-2^{15} = -32768$ à $2^{15} - 1 = +32767$
sur 4 octets, 32 bits : codage des nombres de $-2^{31} = -2\,147\,483\,648$
à $2^{31} - 1 = 2\,147\,483\,647$

Codage des caractères

Caractère stocké sur un octet

Code pour l'équivalence entre la valeur numérique et le caractère :

- EBCDIC (en voie de disparition - caractères alphabétiques non contiguë)
- ASCII (7 bits) ou ASCII étendu (8 bits)
- UNICODE

Remarques :

- important de conserver la relation d'ordre 'alphabétique'
- un texte ou une chaîne de caractères est une suite de caractères soit terminée par le caractère 0 (langage C), soit avec indication du nombre de caractères (langage Pascal)

Exemples :

B O N J O U R ∅ ou 7 B O N J O U R

Table des caractères ASCII

DECIMAL VALUE		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	HEXA DECIMAL VALUE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	BLANK (NULL)	►	SP	0	@	P	‘	p	Ç	É	á	▒			∝	≡
1	1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒			ß	±
2	2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▒			Γ	≥
3	3	♥	!!	#	3	C	S	c	s	â	ô	ú				π	≤
4	4	♦	¶	$	4	D	T	d	t	ä	ö	ñ				Σ	⌠
5	5	♣	§	%	5	E	U	e	u	à	ò	Ñ				σ	⌡
6	6	♠	▬	&	6	F	V	f	v	å	û	ª				μ	÷
7	7	BEL	↨	’	7	G	W	g	w	ç	ù	º				τ	≈
8	8	BS	↑	(	8	H	X	h	x	ê	ÿ	¿				Φ	°
9	9	HT	↓	)	9	I	Y	i	y	ë	Ö	⌐				Θ	∙
10	A	LF	→	*	:	J	Z	j	z	è	Ü	¬				Ω	·
11	B	VT	←	+	;	K	[	k	{	ï	¢	½				δ	√
12	C	FF	FS	,	<	L	\	l	¦	î	£	¼				∞	n
13	D	CR	GS	–	=	M	]	m	}	ì	¥	¡				φ	²
14	E	♫	RS	.	>	N	^	n	~	Ä	₧	«				∈	■
15	F	☼	US	/	?	O	_	o	⌂	Å	ƒ	»				∩	BLANK

Codage des nombres réels

- **Virgule fixe**

  position fixe de la virgule dans la chaîne de bits

  …. , ….

  utilisation du codage binaire vu précédemment

  calculs et gestion de la virgule difficile ==> peu utilisé

- **Par fraction a / b**

  stockage de deux entiers (on ne stocke que des rationnels)

  Difficile à gérer ==> peu utilisé

- **Virgule flottante**

$$x = M * B^E$$

où M est la mantisse avec le maximum de chiffres significatifs,
B la base (2, 4, 8, 10, 16, …)
et E l'exposant (signé)

Exemple :

$(3,25)_{10} = (11,01)_2$ en virgule fixe

$= (1,101)_2 * 2^1$

codage en base 2 sous la forme

avec SM le signe de la mantisse
E l'exposant
M la mantisse

mais différentes manières de coder E et M ==> standardisation

Standard IEEE 754 (1985)

simple précision sur 32 bits :
- 1 bit de signe de la mantisse
- 8 bits pour l'exposant
- 23 bits pour la mantisse

1bit 8 bits 23 bits

double précision sur 64 bits :
- 1 bit de signe de la mantisse
- 11 bits pour l'exposant
- 52 bits pour la mantisse

1bit 11 bits 52 bits

précision étendue sur 80 bits

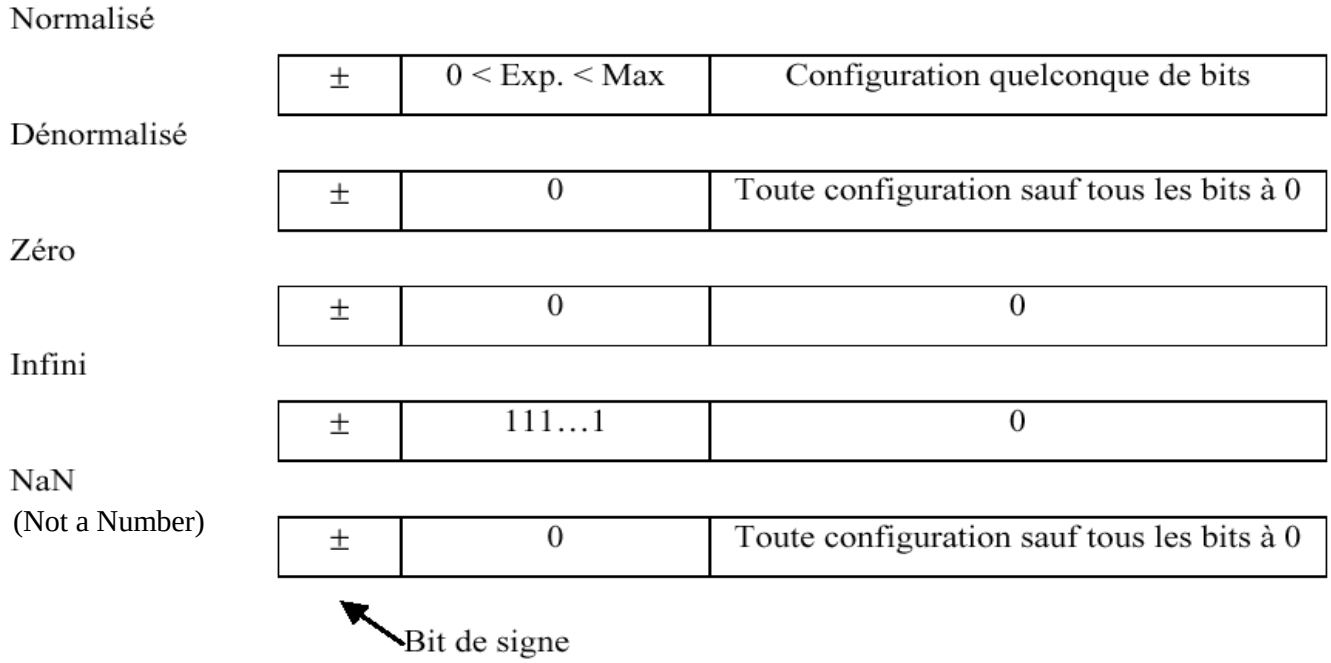
SM = 0 pour +, 1 pour -
E = codage de l'exposant par excédant càd en fait de E+127 ou E+1023
M = mantisse normalisée, le 1er bit étant toujours à 1, il est inutile et supprimé ==> bit caché

En simple précision, $x = (-1)^{SM} * 2^{(E-127)} * 1,M$
En double précision, $x = (-1)^{SM} * 2^{(E-1023)} * 1,M$

Caractéristiques des nombres flottants au standard IEEE

	Simple précision	Double précision
Bit de signe	1	1
Bit d'exposant	8	11
Bit de mantisse	23	52
Nombre total de bits	32	64
Codage de 1'exposant	Excédant 127	Excédant 1023
Variation de 1'exposant	-126 à +127	-1022 à +1023
Plus petit nombre normalisé	2^{-126}	2^{-1022}
Plus grand nombre normalisé	Environ 2^{+128}	Environ 2^{+1024}
Echelle des nombre décimaux	Environ 10^{-38} à 10^{+38}	Environ 10^{-308} à 10^{+308}
Plus petit nombre dénormalisé	Environ 10^{-45}	Environ 10^{-324}

Les divers formats de nombres du standard IEEE



Opérations arithmétiques

Multiplication
addition des exposants, produit des mantisses, normalisation
Division
soustraction des exposants, division des mantisses, normalisation
Addition (ou soustraction)

dénormalisation pour avoir les deux exposants à la même valeur (du plus grand)

addition (ou soustraction) des mantisses

normalisation du résultat

Attention aux dépassement de capacité :

- débordement supérieur (overflow)
- débordement inférieur (underflow) en fait proche de 0

Ce ne sont que des bits !!!

0100 0110 0100 1001 0100 1110 0000 0000 binaire
FIN∅ caractères
46 49 4E 00 hexadécimal
70 73 78 00 octets en décimal
70 73 78 00 entiers signés sur 1 octets
17993 19968 entiers sur 16 bits en décimal
17993 19968 entiers signés sur 16 bits en décimal
1179209216 entier sur 32 bits en décimal
1179209216 entier signé sur 32 bits en décimal
0 100 0110 0 100 1001 0100 1110 0000 0000 flottant simple précision
SM E M
$= + 2^{(140-127=13)}$ 1,100 1001 0100 1110 0000 0000
= 1100 1001 0100 11,10 0000 0000
= 12883 + 0,5 = 12883,5

Tout est bit !!!

2 - Architecture en couches

Architecture en couches

- Ordinateur = machine exécutant instructions élémentaires. Jeu d'instructions dépend du processeur reste très limité :
  add 2 nombres
  voir si un nbre est nul,
  modifier emplacement mémoire d 'une donnée…
  Ce langage machine très primitif est fastidieux à utiliser ==> nécessité de complexifier pour se rapprocher d'un langage humain.

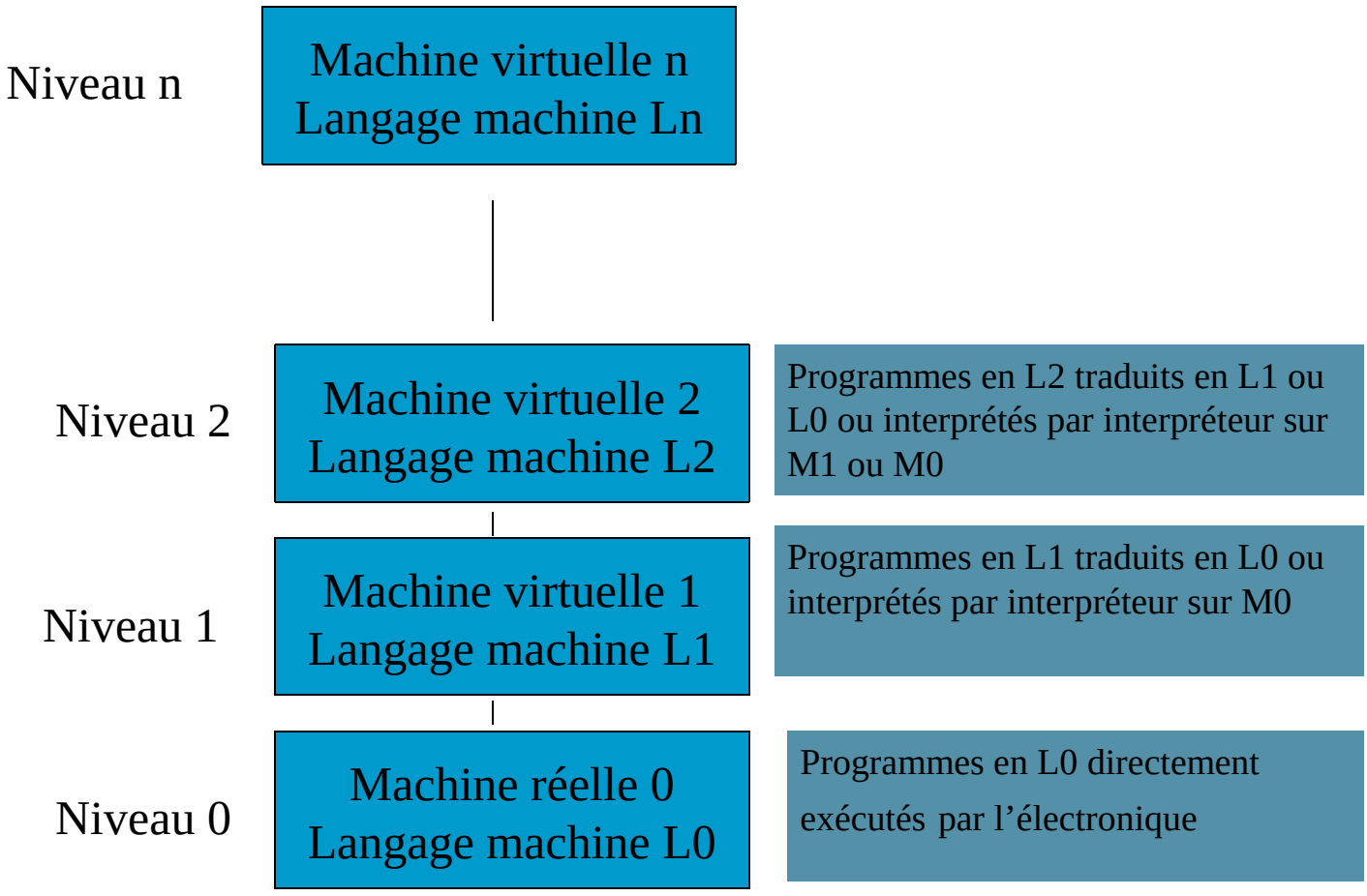
- Solution choisie: structurer les ordinateurs en couches, chaque couche ou abstraction reposant sur l'abstraction précédente

Les différents langages

- Voici une instruction en langage évolué :
  A:=A+1
- Cette instruction est incompréhensible pour le processeur. Elle est traduite (compilée ou interprétée) et s'écrit en Langage d'assemblage, proche du Langage machine
- load a
  inc
  store a
  où a est l'adresse de la variable A
- Pour l'exécuter, le processeur devra effectuer plusieurs cycles de base. Ainsi, chaque instruction machine sera traduite en un certain nombre de commandes du chemin de données

Langages, couches et machines virtuelles

- L0 : langage machine.
  L1 : nouveau langage + facile a utiliser.
  Nécessité de traduire (compiler) ou interpréter les programmes en L1. On peut imaginer l'existence d 'une machine virtuelle M1 qui exécuterait directement L1 (possible à construire mais + chère) . L1 doit être relativement proche de L0 (compilation) donc L1 pas encore idéal .
- Succession de langages et de machines virtuelles associées (et d 'interpréteurs de ces machines virtuelles) . Chaque langage Li tourne sur une machine virtuelle Mi ou est traduit en L_{i-k} ou interprété par un émulateur de Mi sur une M_{i-k} réelle.



Langages et Machines

- Un langage définit Une Machine et réciproquement.

- Tout langage (C++ , Java …) définit donc la machine qui peut exécuter tous les programmes écrits dans ce langage. Cette machine peut être construite (à un prix très élevé).

- Quand on programme dans un langage, on dispose d ’une machine virtuelle. Peu importe comment cette machine est interprétée et si elle l’est en plusieurs étapes.

Les machines multicouches actuelles

- Niveau 5 — Couche langages d ’applications
  ↓ traduction (compilateurs)
- Niveau 4 — Couche langage d ’assemblage
  ↓ traduction (assembleur)
- Niveau 3 — Couche Système d ’Exploitation
  ↓ interpr. Partielle
- Niveau 2 — Couche architecture du jeu d’instructions (ISA)
  ↓ interp (micro programmes) ou exécution directe
- Niveau 1 — Couche micro architecture
  Matériel
- Niveau 0 — Couche logique numérique

Le niveau zéro

- Les circuits électroniques du niveau 0 exécutent les programmes en langage machine du niveau 1.
- Ces circuits sont constitués de portes logiques (fabriquées a partir de composants analogiques transistors etc). Une porte a 2 entrées logiques et une sortie logique.
- En combinant qq portes on peut fabriquer une mémoire de 1 bit , regroupées par 16, 32 ou 64 pour faire des registres.
- Les portes peuvent être combinées pour réaliser le cœur même de l'ordinateur (voir étude de la couche physique)

Le niveau 1

- Ensemble de registres (de 8 à 32) = mémoire locale + un circuit (U A L) Unité Arithmétique et logique.
- Les registres sont reliés à l'UAL pour former le chemin des données (commandé par micro programme ou matériel) . Ce chemin des données sélectionne les registres pour les données et range le résultat dans un registre.
- Le niveau 1 exécute les instructions du niveau 2 . Il charge, analyse et exécute les instructions une à une en utilisant le chemin des données. Ceci est réalisé par un programme (exécuté par le niveau 0) ou par matériel (le plus souvent maintenant)

Machine virtuelle M1

Voici une machine virtuelle L1 (d ’une machine appelée MIC1) : elle est définie par le langage de niveau 1 qui se présente ainsi:

00000000 000001110110000100101001 00

Cette instruction s ’interprète connaissant sa structure

adresse suivante	Jam			UAL								C								Mémoire			B
	JMPC	JAMN	JAMZ	SLL8	SRA1	F0	F1	ENA	ENB	INVA	INC	H	OPC	TOS	CPP	LV	SP	PC	MDR	MAR	write	read	fetch	
0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	1	0	0	1	0	1	0	4

Le sens de cette instruction est le suivant:
décrémenter de 1 le registre de sommet de pile, recopier cette adresse dans le registre « adresse mémoire » et lancer une lecture en mémoire à cette adresse

Machine virtuelle M1

La machine virtuelle M1 (de la machine MIC1) sait effectuer les opérations suivantes:

- Choisir un registre et recopier sa valeur dans le bus B de l ’UAL
- Choisir une opération arithmétique ou logique parmi 16
- Exécuter cette opération entre l ’entrée A et l ’entrée B de l ’UAL
- Recopier le résultat dans 1 ou plusieurs registres

Chaque instruction du langage L1 précise ces 3 points

Le niveau 2 : la couche ISA

- Le niveau 2 est celui du jeu d ’instruction machine. (Instruction Set Architecture).
- C ’est l’ensemble des instructions qui seront exécutés par le matériel du niveau 1 ou interprétées par le micro programme .
- Voici un exemple de langage ISA (extrait)

Hexa	Mnémonique	Signification
10	BIPUSH octet	Push octet dans la pile
59	DUP	Duplique le mot du sommet et le met dans la pile
A7	GOTO offset	Branchement inconditionnel
60	IADD	pop 2 mots de la pile, push le résultat
7E	IAND	pop 2 mots de la pile, push leur ET dans la pile
99	IFEQ offset	pop un mot de la pile, branchement s'il est égal à 0
9B	IFLT offset	pop un mot de la pile, branchement s'il est < 0
9	IF_ICMPEQ offset	pop 2 mots de la pile, branchement si égaux

La machine virtuelle M2 (ISA)

Voici un exemple de machine virtuelle M2 (cas de MIC1).
Cette machine dispose d ’une mémoire gérée en pile et sait:

•Empiler ou depiler le sommet de la pile
•faire la somme des 2 valeurs du sommet de la pile et empiler le résultat
•faire la différence ……. // ……
•Tester si le sommet de la pile vaut zéro et si oui sauter de xx instructions dans le code exécutable
•charger la valeur du mot à l’adresse aa de la mémoire sur la pile (ou dépiler et stocker à l’adresse aa le sommet)
•appeler une routine (saut dans le code à une adresse)
•etc.

Le niveau 3 Système d’exploitation

- Niveau hybride: certaines instructions du niveau 2 y apparaissent de nouveau + des instructions spécifiques (gestion mémoire, périphériques, interruptions, etc.)
- Les services offerts a ce niveau sont exécutés par un interpréteur s’exécutant au niveau 2 appelé S.E. . Les instructions de niveau 2 présentent au niveau 3 s’exécutent directement au niveau 2. Elles sont traitées par le micro programme et non par le Système d’exploitation.
- La machine virtuelle 3 sait donc en plus écrire sur le disque, sauter à une page mémoire, lancer une interruption…

Niveau 4: langage d’assemblage

- Les niveaux 1 à 3 ne concernent pas le programmeur d ’applications : écrits par des programmeurs systèmes, ils sont là pour supporter les traducteurs des niveaux supérieurs
- Le langage fourni au niveau 4 est constitué de mots (mnémoniques) et non plus de nombres (niveau 1-3). C’est une forme symbolique des langages sous-jacents.
- Les programmes en langage d ’assemblage sont traduits en langage de niveau 3 par un « assembleur » puis exécutés par la machine virtuelle ou réelle inférieure

Niveau 5 langages de haut niveau

- On trouve ici des langages conçus pour être utilisés par des programmeurs d ’applications. Il en existe des centaines.
- Parmi ces langages il existe des langages généralistes (C, java, pascal, basic, fortran,…) ou spécialisés (autocad, matlab, …)
- Les programmes écrits dans ces langages sont généralement compilés pour être traduits en niveau 3 ou 4.
- Une machine virtuelle Pascal (par exemple) est une machine qui sait exécuter toutes les instructions Pascal (if then else, write, affectation, while do, …) et qui connaît la notion de type, de variable, …

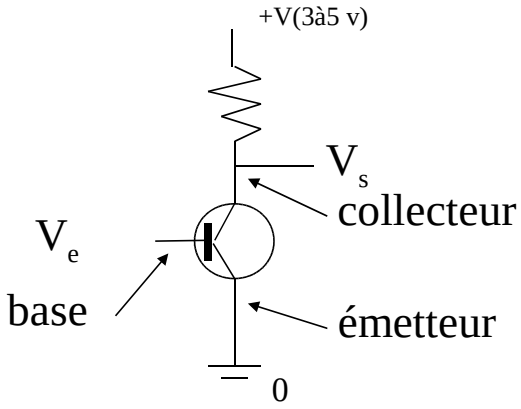
En résumé

- Un ordinateur peut être vu comme une suite de couches construites les unes sur les autres.
- Une couche représente un certain niveau d ’abstraction et comporte divers objets et opérations sur ces objets.
- L’ensemble des types de données, des opérations et des caractéristiques de chaque niveau s’appelle l’architecture de ce niveau. L’architecture d’un niveau c’est l’ensemble de ce dont l’utilisateur de ce niveau a conscience.

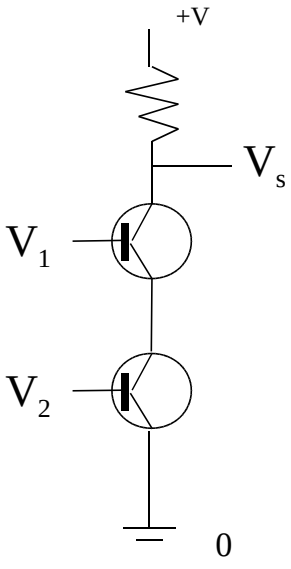
3 - La couche physique

Portes logiques et Algèbre de Boole

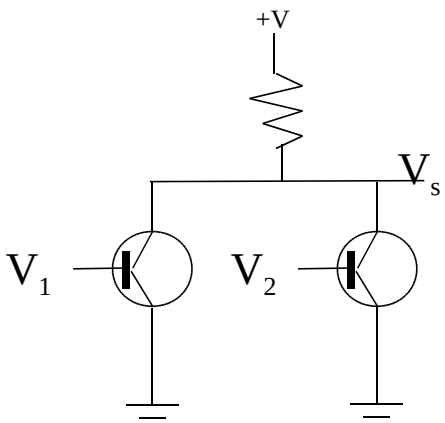
- Transistors-> circuits logiques (n entrées binaires, une sortie binaire)



Transistor = inverseur
temps commutation=2ns

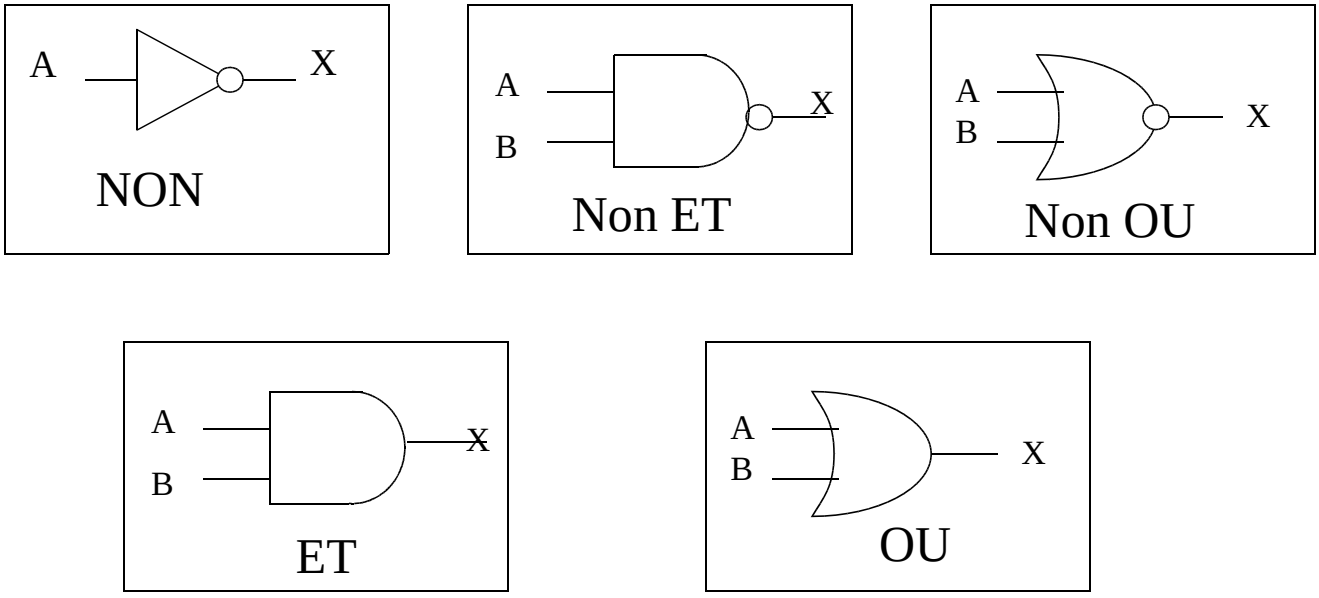


Non-ET (NAND)



Non-OU (NOR)

Les différentes portes logiques



Algèbre de Boole

- Toute fonction booléenne de n variables est définie par sa table de vérité.
- Toute fonction booléenne f s'écrit de manière unique comme une somme de facteurs (produits) des variables d 'entrée et de leurs négations. (forme canonique ou équation)
- exemple: la fonction majorité a pour table:

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Son équation est :
$M=\overline{A}BC+A\overline{B}C+AB\overline{C}+ABC$

remarque: Non-et= $\overline{AB} = \overline{A}+\overline{B}$

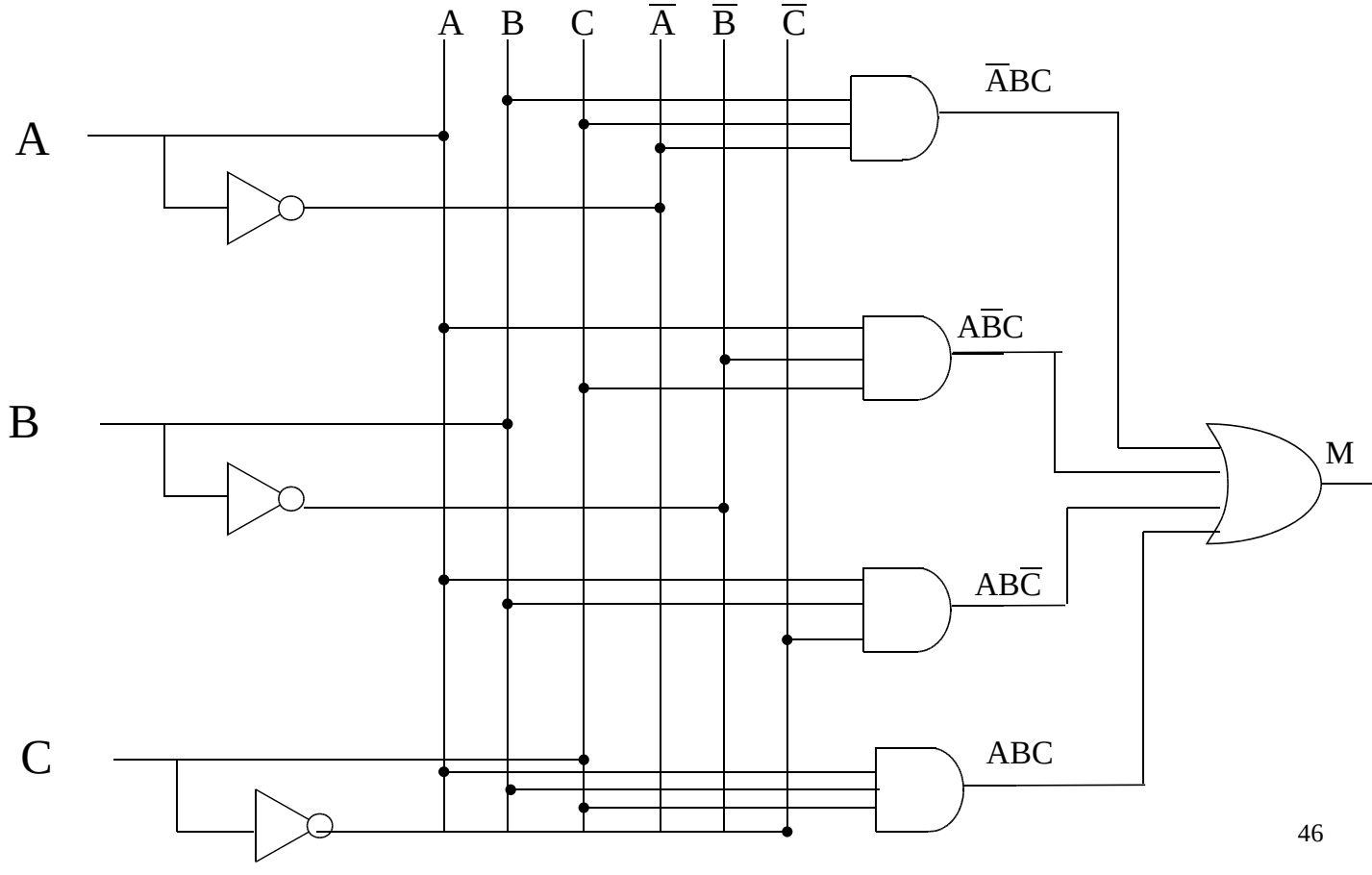
Réalisation d'une fonction logique

Pour réaliser une fonction logique quelconque :

- Ecrire l'équation à partir de la table de vérité
- Réaliser l'inversion de toutes les entrées pour disposer des complémentaires
- Construire une porte ET pour chacun des termes de l'équation
- Etablir le câblage des portes ET avec les entrées appropriées
- Réunir l'ensemble des sorties des portes ET vers une porte OU dont la sortie sera le résultat de la fonction.

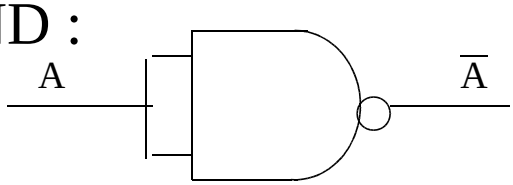
Exemple: la fonction Majorité

$$M=\overline{A}BC+A\overline{B}C+AB\overline{C}+ABC$$

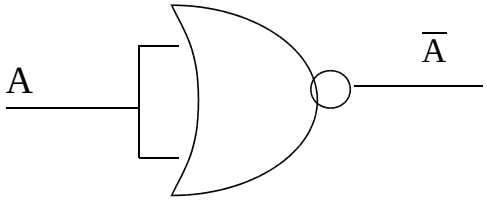


Fonctions Booléennes de base

- Remarque: la fonction NAND (ou NOR) est dite complète : toute fonction booléenne peut s'écrire uniquement avec des portes NAND ou avec des portes NOR.
- Réalisation de NON avec NAND :



- Réalisation de NON avec NOR :



Règles de l'algèbre de Boole

Nom de la loi	forme ET	forme OU
identité	$1\ A = A$	$0 + A = A$
nullité	$0\ A = 0$	$1 + A = 1$
idempotence	$A\ A = A$	$A + A = A$
Inversion	$A\ \overline{A} = 0$	$A + \overline{A} = 1$
Commutativité	$A\ B = B\ A$	$A + B = B + A$
Associativité	$A(BC) = (AB)C$	$A + (B + C) = (A + B) + C$
Distributivité	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption	$A(A + B) = A$	$A + AB = A$
De Morgan	$\overline{A\ B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\ \overline{B}$

Réalisation de ET avec NAND ou NOR

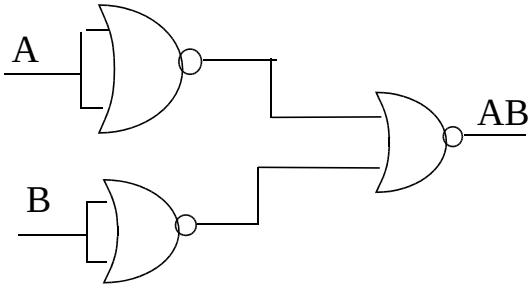
- Rappel : A ET B = AB A OU B = A + B
- $AB = \overline{\overline{AB}}$ = non ($\overline{A} + \overline{B}$) = $\overline{A}$ NOR $\overline{B}$

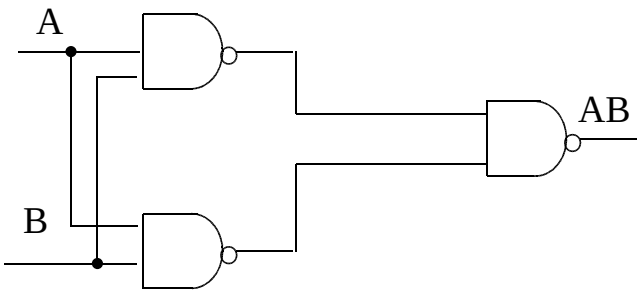
  or non A = $\overline{A}$ = $\overline{A + A}$ =A NOR A

  donc AB = (A NOR A) NOR (B NOR B)
- de même $AB = \overline{\overline{AB}}$ = $\overline{A \text{ NAND } B}$

  or non A = $\overline{A}$ = $\overline{AA}$ =A NAND A

  donc AB = (A NAND B) NAND (A NAND B)
- dessiner les circuits correspondants au circuit AB





Réalisation de OU avec NAND ou NOR

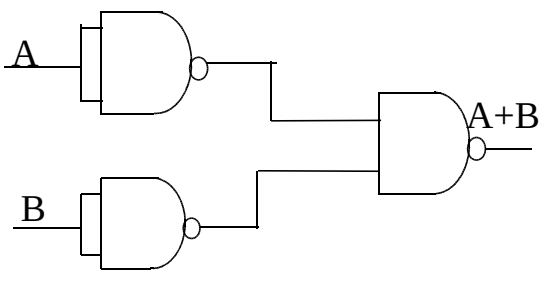
- Rappel : A ET B = AB A OU B = A + B
- $A + B = \overline{\overline{A + B}}$ = non ($\overline{A}\ \overline{B}$) = $\overline{A}$ NAND $\overline{B}$

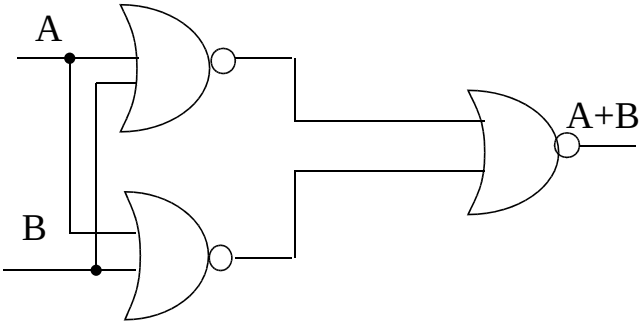
  or non A = $\overline{A}$ = $\overline{AA}$ =A NAND A

  donc A + B = (A NAND A) NAND (B NAND B)
- de même $A+ B = \overline{A + B}$ = $\overline{A \text{ NOR } B}$

  or non A = $\overline{A}$ = $\overline{A + A}$ =A NOR A

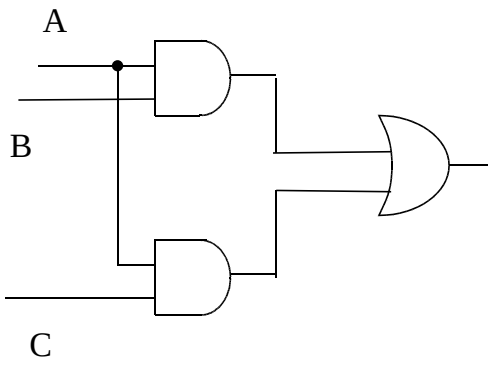
  donc A + B = (A NOR B) NOR (A NOR B)
- dessiner les circuits correspondants au circuit A+B

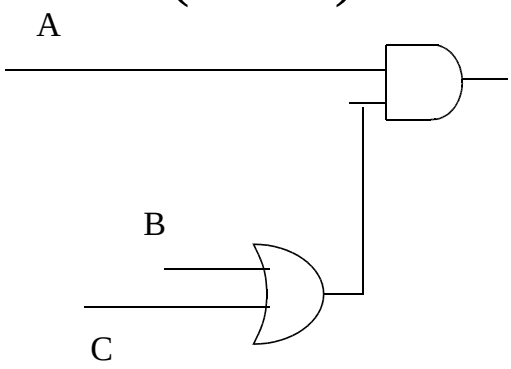




Exercice

- En utilisant la règle de distributivité, réalisez de deux manières différentes le circuit A(B + C) avec des portes ET et OU.
- AB + AC
- A(B + C)

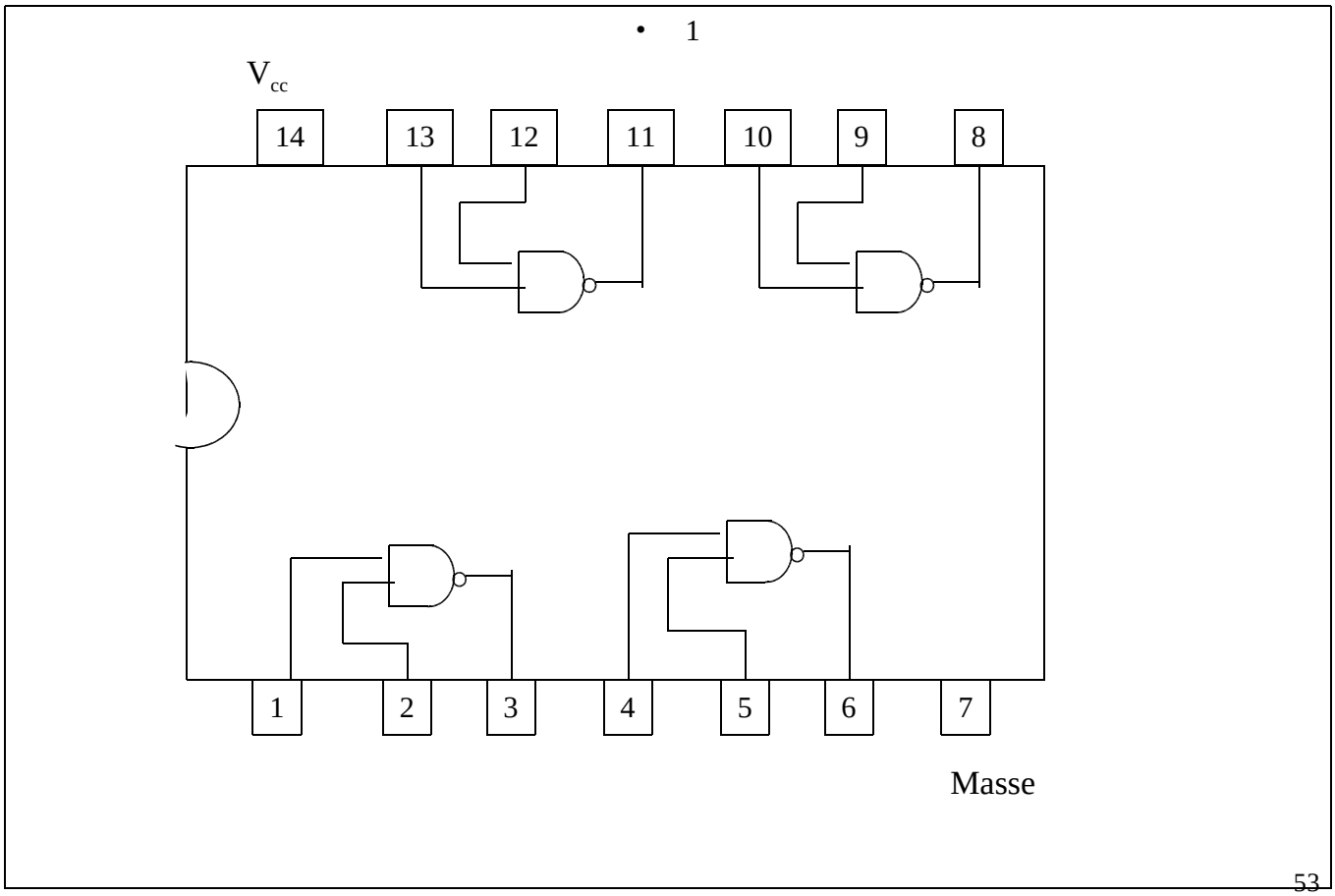




Circuits Intégrés Logiques (CIL)

- Les portes de base ne sont pas fabriquées individuellement. On les regroupe sur des circuits intégrés comportant un certain nombre de broches (entrées et alim de qq centimètres de long.) : CI, CIL ou puces ou chip.
- Ils sont encapsulés dans un boîtier céramique ou résine et disposent de pattes de connexion sur les 2 bords longs du rectangle (de 14 à 64) on les nomme DIP (Dual Inline Package)
- on les classe en
  - SSI (small scale Integration) pour 1 à 10 portes
  - MSI pour 10 à 100 portes par circuit
  - LSI pour 100 à 100 000 portes / circuit
  - VLSI plus de 100 000 portes par circuit

Exemple de circuit SSI



53

Caractéristiques des CIL

- On trouve sur le marché pour un prix modiques des circuits SSI. L'alimentation et la masse sont communes à toutes les portes.
- Le temps de réaction d'un circuit n est pas nul. Il varie de qq fractions de ns à 10 ns
- Intégration : Aujourd'hui on atteint 5 millions de portes sur une seule puce. Si toutes les connexions étaient présentes il faudrait 15 000 000 de broches!! À raison de 2,3 mm par broche le circuit mesurerait 18 Km de long!!! On construit en fait des circuits où le ratio portes/nb de broches est le plus élevé possible.

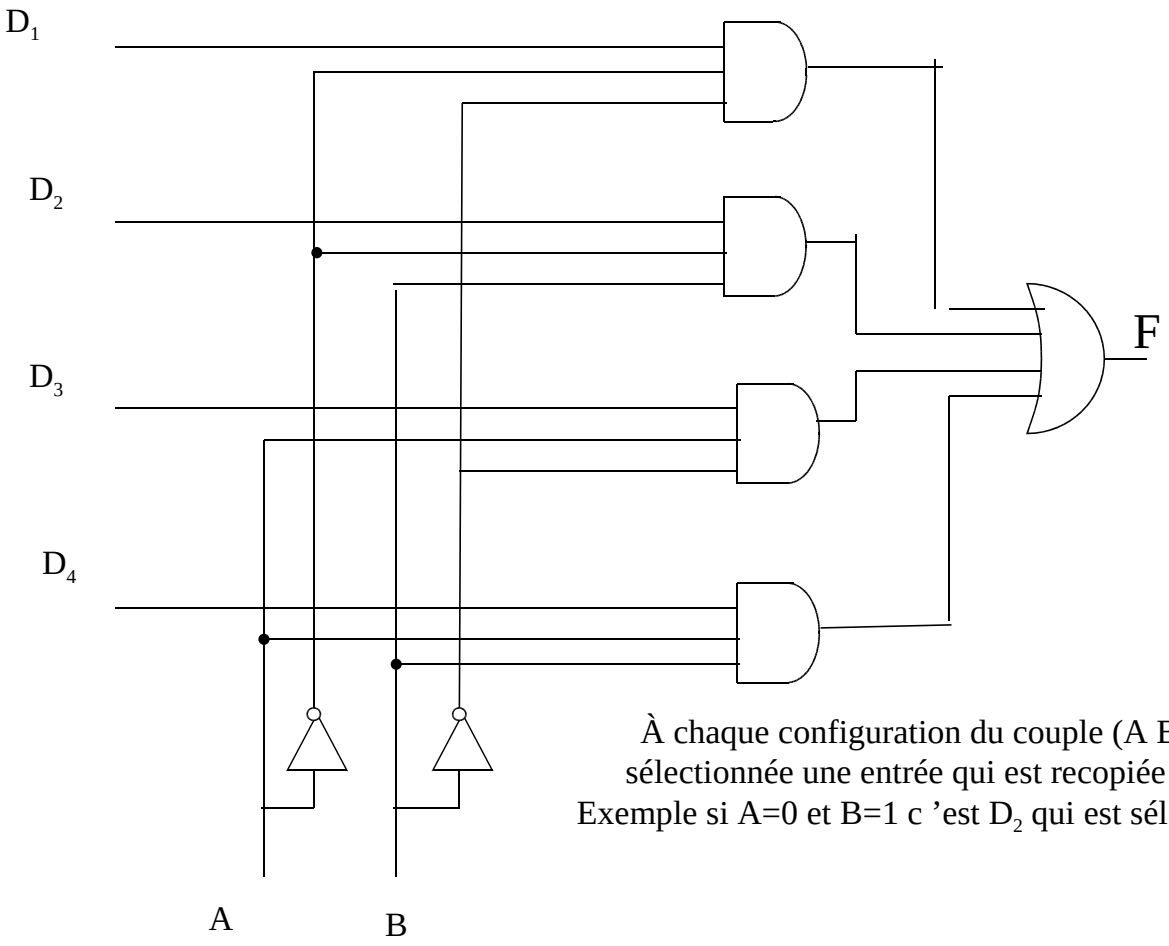
Circuits MSI combinatoires

- On appelle circuit combinatoire un circuit à n entrées et p sorties où les sorties s ’expriment uniquement en fonction des entrées. Le circuit « majorité » est un exemple de circuit combinatoire. Certains de ces circuits sont très utilisés. Nous en étudions quelques exemples.

- Tous les circuits ne sont pas combinatoires : dans certains circuits les sorties ne dépendent pas uniquement des entrées mais aussi des états antérieurs du circuits. Ce sont des circuits séquentiels ou à mémoire.

Circuit combinatoire: le multiplexeur

- Ce circuit possède 2^n entrées, une sortie et de n lignes de sélection (entrées particulières). La configuration des n lignes de sélection permet de choisir quelle entrée sera dirigée vers la sortie.
- Exemple de multiplexeur à 4 entrées et 2 lignes de sélection:



À chaque configuration du couple (A B) est sélectionnée une entrée qui est recopiée en F.
Exemple si A=0 et B=1 c 'est D_2 qui est sélectionné.

Circuit combinatoire: le multiplexeur

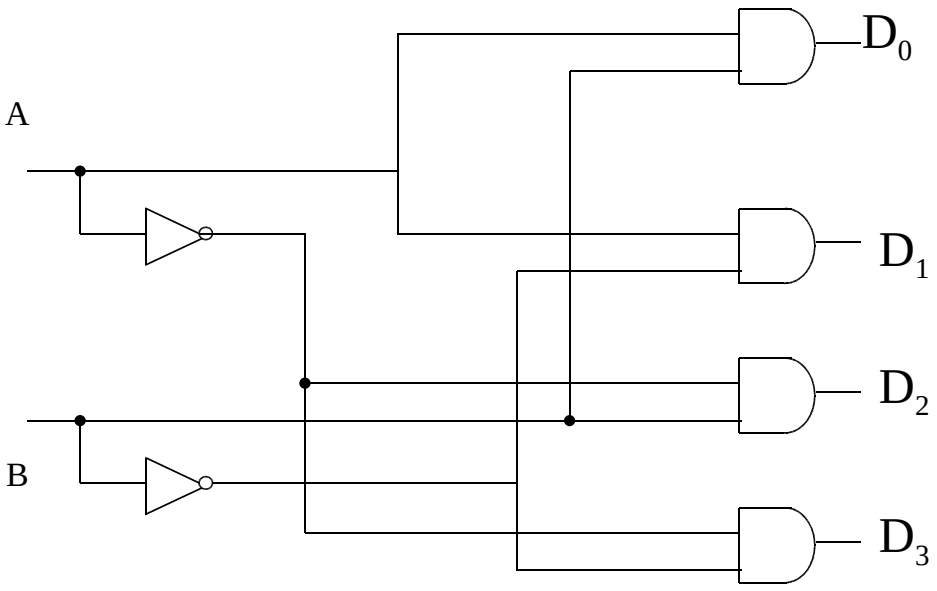
- Ce circuit possède 2^n entrées, une sortie et de n lignes de sélection (entrées particulières). La configuration des n lignes de sélection permet de choisir quelle entrée sera dirigée vers la sortie.
- Exemple de multiplexeur à 4 entrées et 2 lignes de sélection
- On trouve couramment des multiplexeurs à 8 entrées et 3 sélections . Si on ajoute 1 alimentation et 1 masse on obtient 13 broches. Il tient sur un circuit 14 broches.
- Réalisation d'une fonction de 3 variables par un multiplexeur : l'alim de D_i est égal à la valeur de la ligne i de la table de vérité.
- Exemple : réalisez le câblage des broches d'un Multiplexeur 14 broches pour avoir « majorité ».
- Remarque : le **démultiplexeur** effectue la fonction inverse càd la connexion d'une entrée à une des 2^n sorties.

Application du multiplexeur

- Une application intéressant du multiplexeur est la conversion parallèle/série.
  Supposons qu'une information de 8 bits arrive sur les entrées D_1 à $D_{8.}$ Si on diffuse séquentiellement sur les lignes de sélection les valeurs binaires de 000 à 111 on transmet sur la sortie les informations de l'entrée une après l'autre.
- Application : un clavier est scruté. Chaque touche correspond à un octet. Le multiplexeur permettra par exemple de transmettre sur une ligne téléphonique cet octet (en série)

Le décodeur

- Le décodeur traduit l'information binaire présente sur ses n lignes d'entrée pour rendre active l'une des ses 2^n sorties. Voici un exemple de décodeur 2 vers 4.

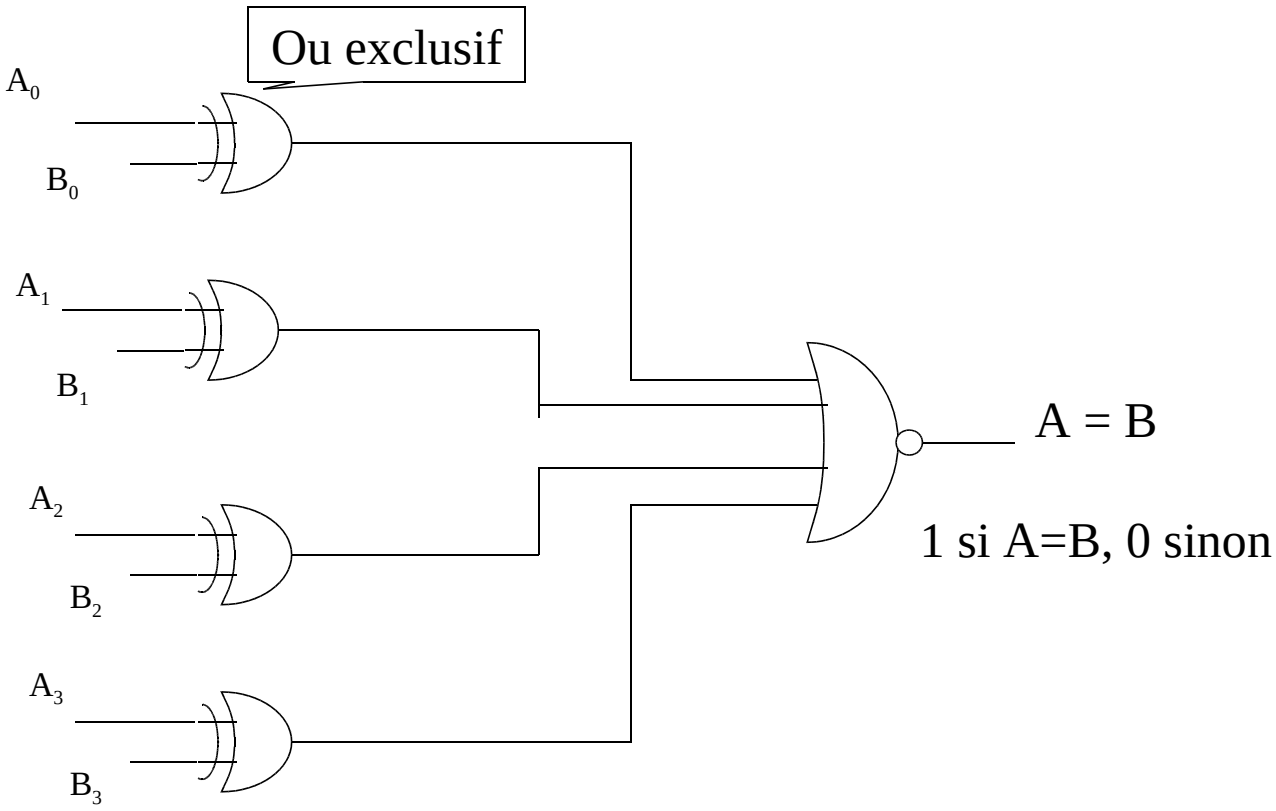


Décodeur …suite

- La sélection des circuits mémoires est une application typique du décodeur :

  supposons qu'une mémoire soit constituée de 8 boîtiers de 1Mo chacun. Le boîtier N°0 correspond aux adresse de 0 à 1Mo, le boîtier N°1 aux adresses entre 1Mo et 2Mo etc. Les 3 bits de poids fort de l'adresse sont utilisés pour sélectionner le boîtier : ils sont transmis aux entrées A B et C du décodeur qui désignent le n° du boîtier à utiliser.

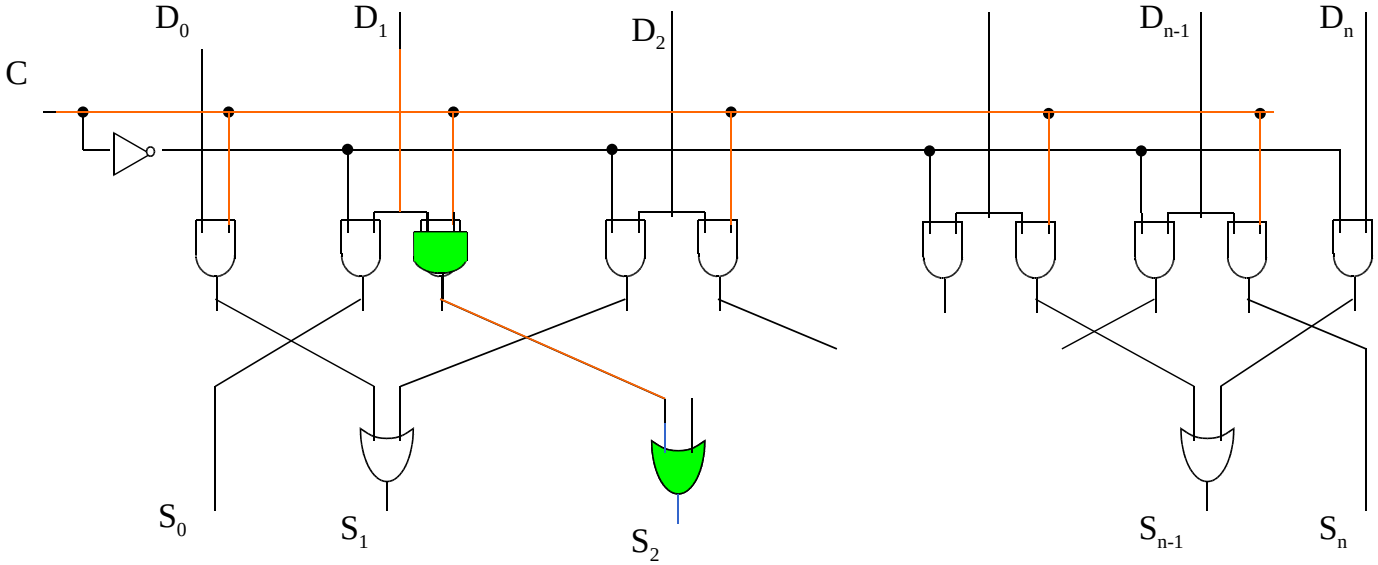
Le comparateur



Les circuits arithmétiques

- Ces circuits MSI sont utilisés dans les unités de traitement et de calcul. Nous allons examiner le décaleur qui possède n bits en entrée et n bits en sortie et réalise le décalage de 1 bit sur la droite ou sur la gauche, puis un additionneur binaire et enfin l'UAL.
- Le décaleur : comporte n bits d'entrée (typiquement n=8 ou 16, 32, 64) notés D_0, D_1, ..., D_n et n bits de sortie notés S_0, S_1, …, S_n qui prennent les valeurs des entrées décalées. Un signal de commande C indique le sens du décalage. Si C=0 on décale vers la gauche, C=1 vers la droite. Il est basé sur des paires de portes ET. Quand C=1, c'est la partie droite des portes qui est active, la gauche quand C=0.

Le décaleur (shifteur)

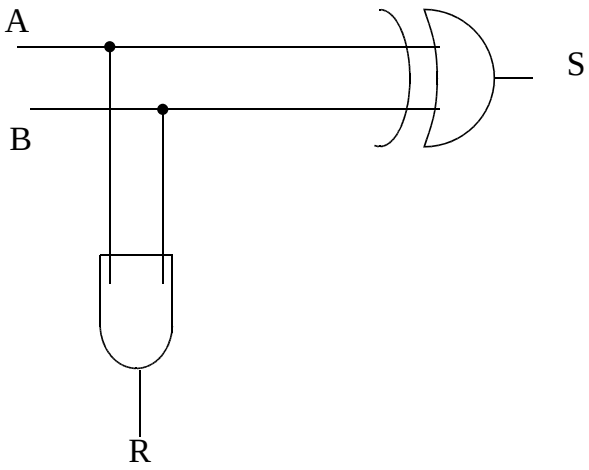


Si C=1, la porte ET N°1 est active et D_0 passe sur S_1 ,D_1 sur S_2 etc (D_n perdu)

L’additionneur

- Voici la table de vérité d’une fonction réalisant l’addition de deux mots de 1 bit. Remarquer les 2 sorties, S (somme) et R (retenue).

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



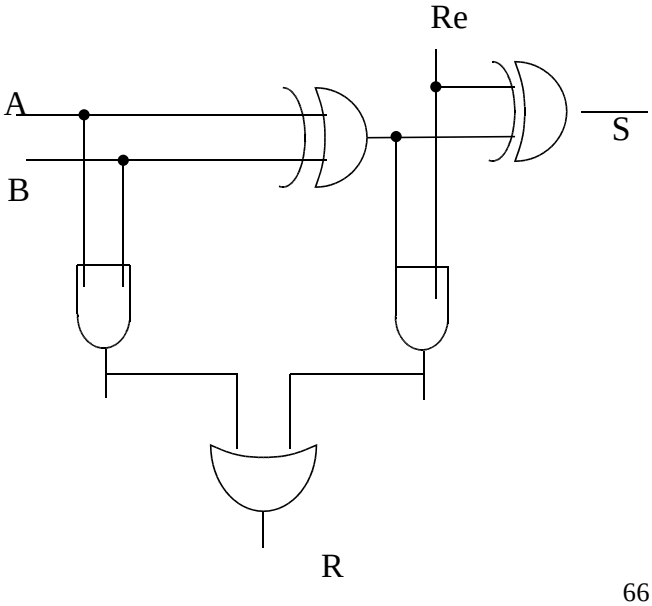
Quelles fonctions reconnaissez vous?

Réalisez le circuit.

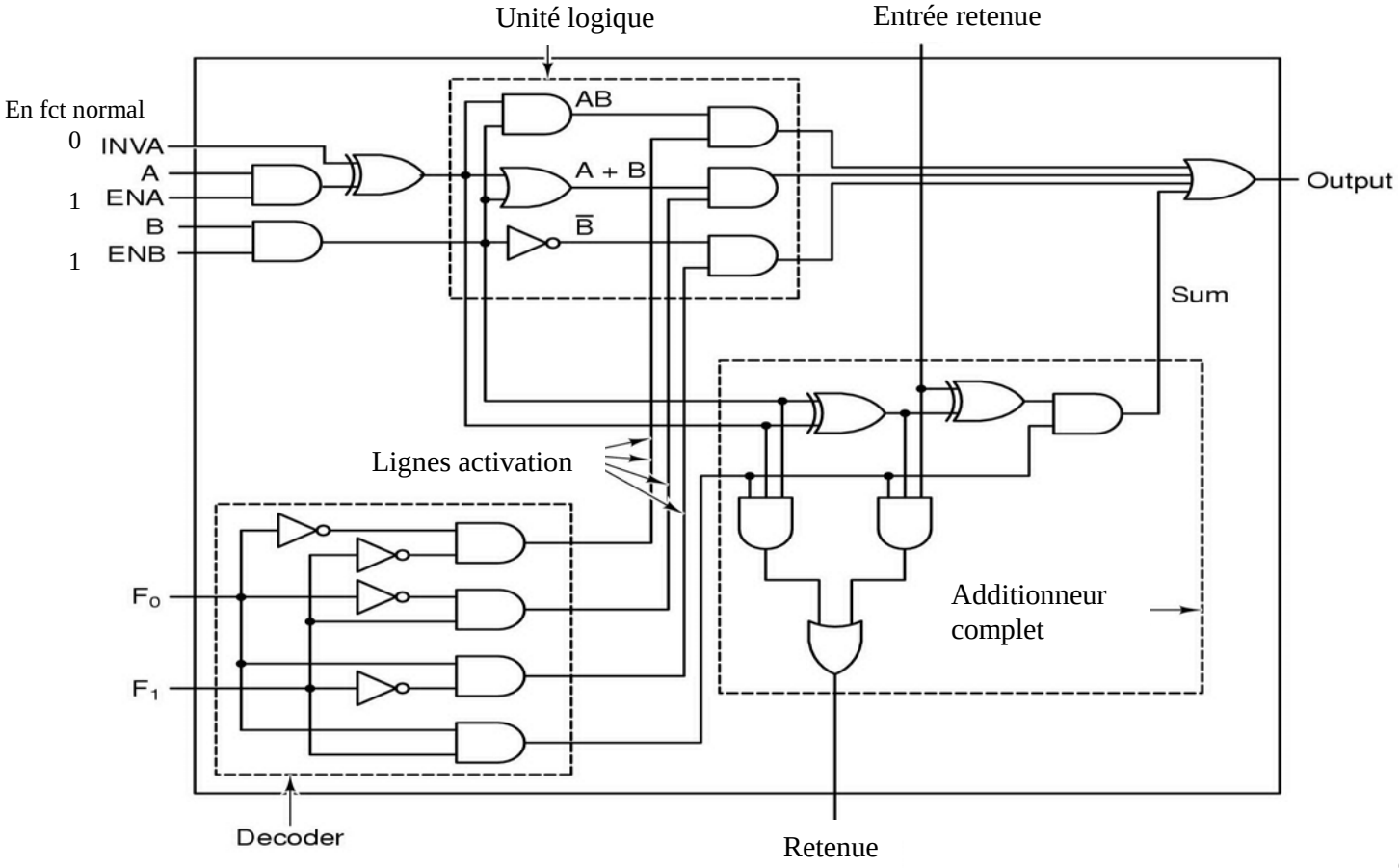
L’additionneur complet

- Pour réaliser l’addition complète de 2 bits, il faut tenir compte de l’éventuelle retenue sur les deux bits précédents.
- Donnez la table de vérité et le schéma d’un additionneur de 2 bits.

A	B	Re	S	R
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

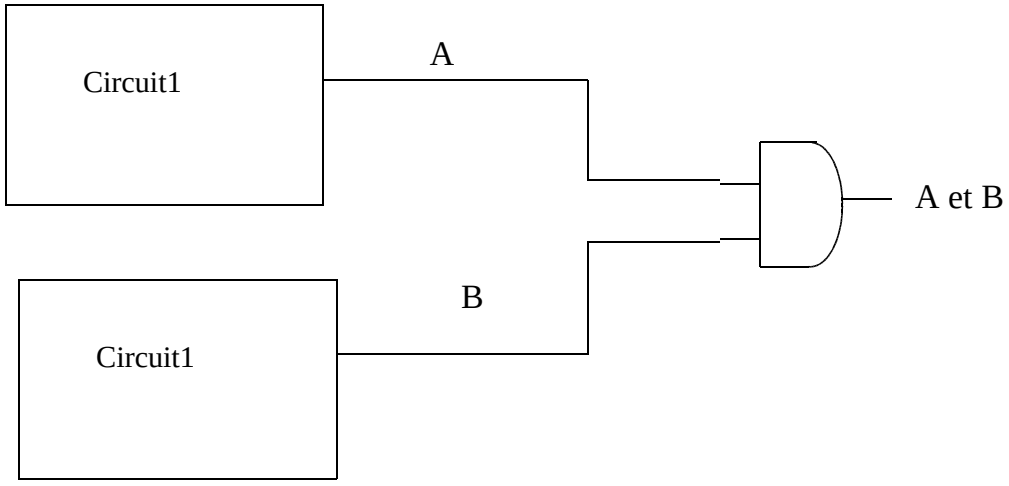


Une Unité Arithmétique et Logique



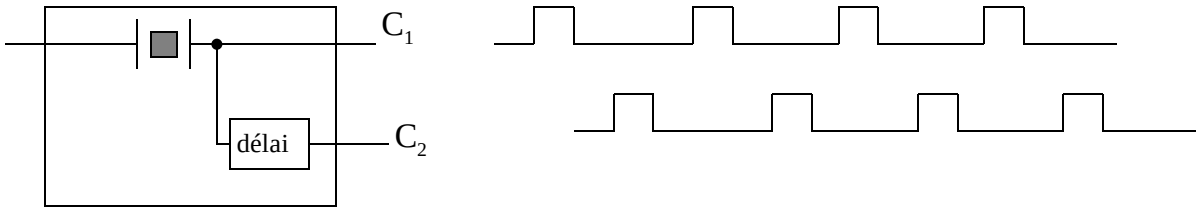
Les horloges

- Dans un circuit arithmétique et logique, il est souvent nécessaire de synchroniser l'arrivée des variables. En effet, le temps de réaction des circuits n'est pas négligeable et il se peut qu'une variable arrive en retard, ce qui perturberait le fonctionnement du circuit suivant.



Horloges ... suite

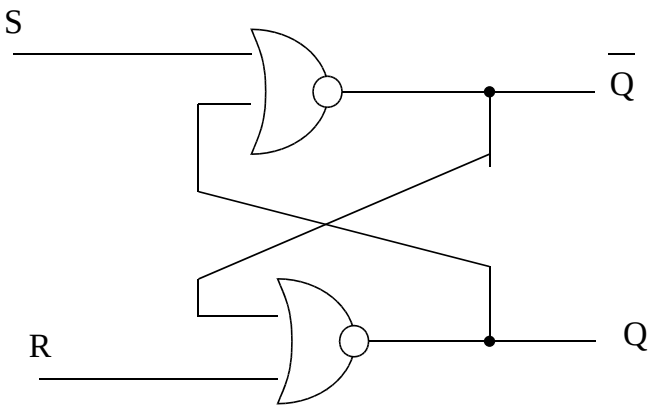
- Une horloge est un circuit qui émet régulièrement une suite d'impulsions calibrées (fréquence de 1 à qq Ghz). Si plusieurs évènements arrivent dans le même cycle d 'horloge, il faut les synchroniser et donc diviser le cycle en sous cycles. On y parvient en décalant le signal (en le faisant passer dans un circuit de temps de réaction connu)



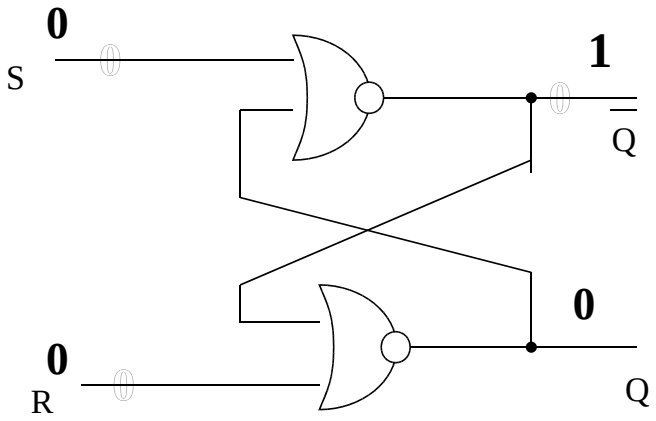
En réalisant des fonctions logiques sur C_1 et C_2 on peut déclencher certains évènements à des tops inférieurs au cycle d 'horloge.

Les circuits à mémoire

- Voici les composants qui expliquent le fonctionnement des circuits capables de mémoriser des données
- Les bascules : (mémorisation d'1 bit)
  Une bascule RS possède 2 entrées : S pour la mise à 1 de la bascule (S pour Set) et R pour la mise à 0 (R pour Reset).



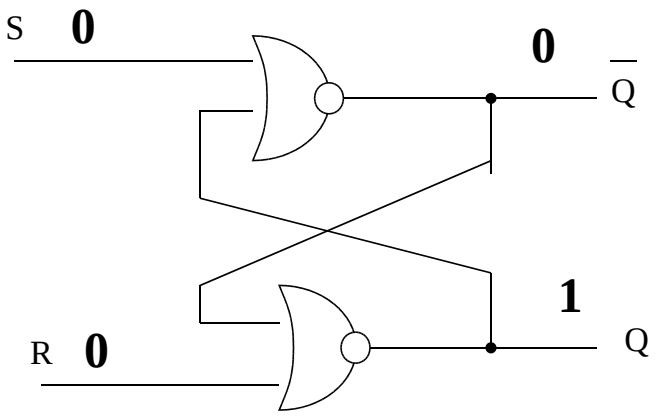
Fonctionnement d ’une bascule RS



Supposons que S=0, R=0 et Q=0. Calculez $\overline{Q}$

Le dispositif est il stable ?

Et si maintenant Q valait 1 au départ ?



Pourquoi a-t-on appelé les sorties Q et $\overline{Q}$?

Que se passe-t-il si S passe à 1 ? Et si S repasse à 0 ?

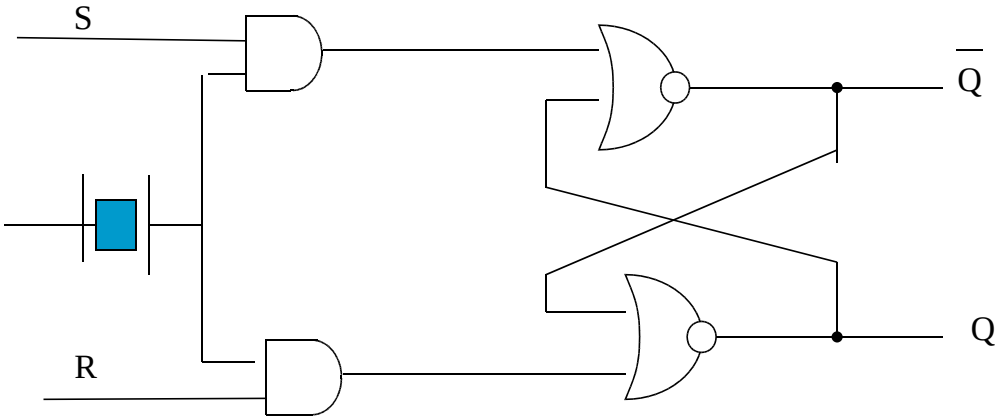
De même, que se passe-t-il si R passe à 1 ?

Bascule RS

- En résumé:
  si S passe momentanément de 0 à 1, la bascule passe (ou se maintient) à l'état Q=1. Quand c'est R qui passe de 0 à 1, la bascule passe ou se maintient à la valeur Q=0. On peut donc dire que la bascule se souvient de l'action antérieure de R ou S. C'est l'effet de mémorisation. La valeur de $\overline{Q}$ ne dépend pas seulement de S et R mais aussi de Q (donc de l'ancien $\overline{Q}$)

Bascules commandée par une horloge

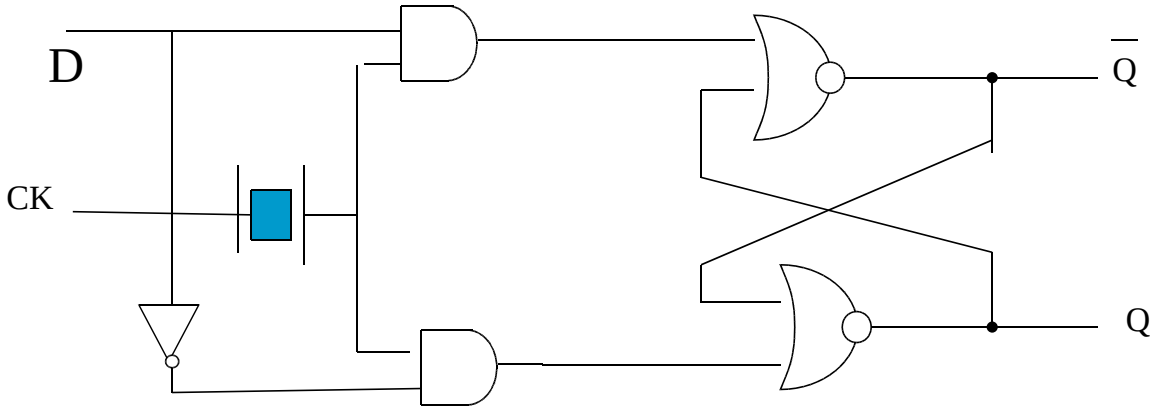
- Pour commander les changements d'état de bascules à des tops horaires précis, il suffit d'ajouter une horloge et 2 portes ET qui « désactivent » les entrées S et R :



Lorsque l'horloge est à 0, les entrées S et R sont inactives

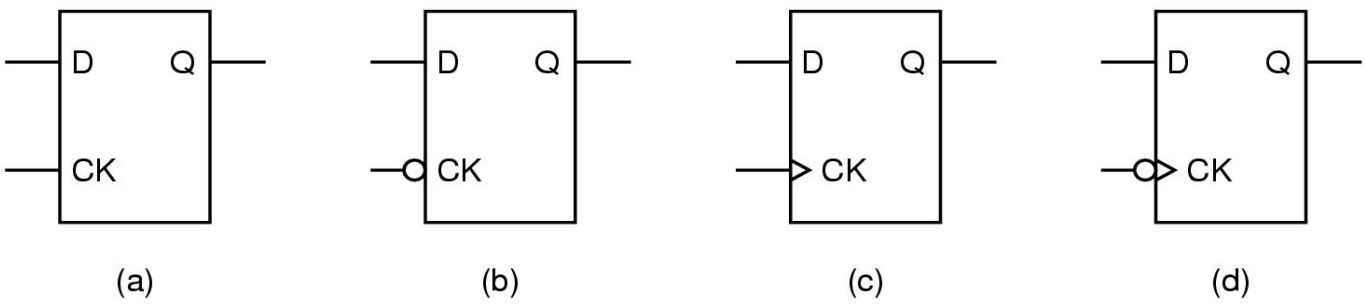
Bascules D

- Quand S et R valent 1 ensemble, Q=0 et $\overline{Q}$ =0. Si les 2 entrées S et R repassent à 0 simultanément, une bascule RS prend un état indéterminé.
- Pour empêcher ce fait, on construit des bascules à une seule entrée D et on crée son contraire. On a ainsi S et R toujours opposés :

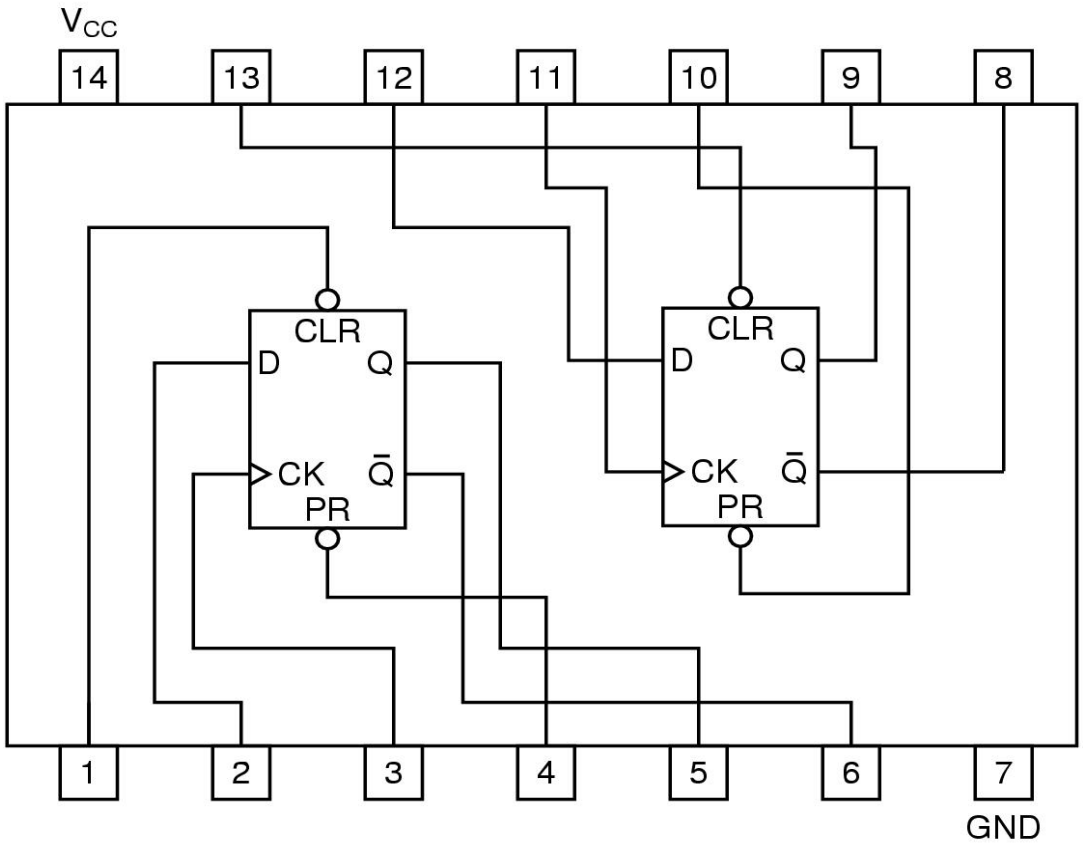


Les registres

- La plupart des bascules du commerce disposent des sorties Q et $\overline{Q}$. Certaines comportent une ou deux entrées supplémentaires (Reset ou Clear) pour remettre Q à 0, et Preset ou Set pour forcer Q à 1

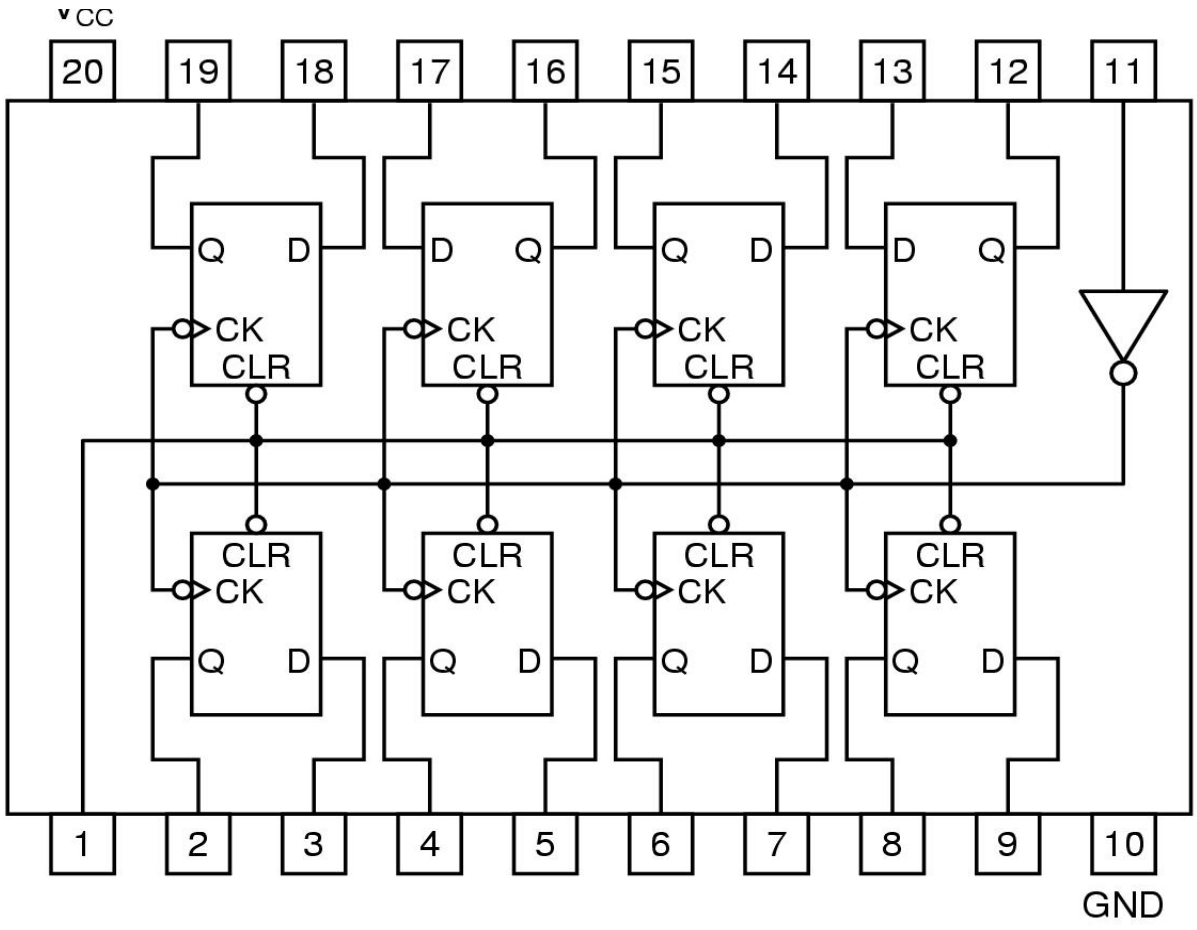


Registres…. suite



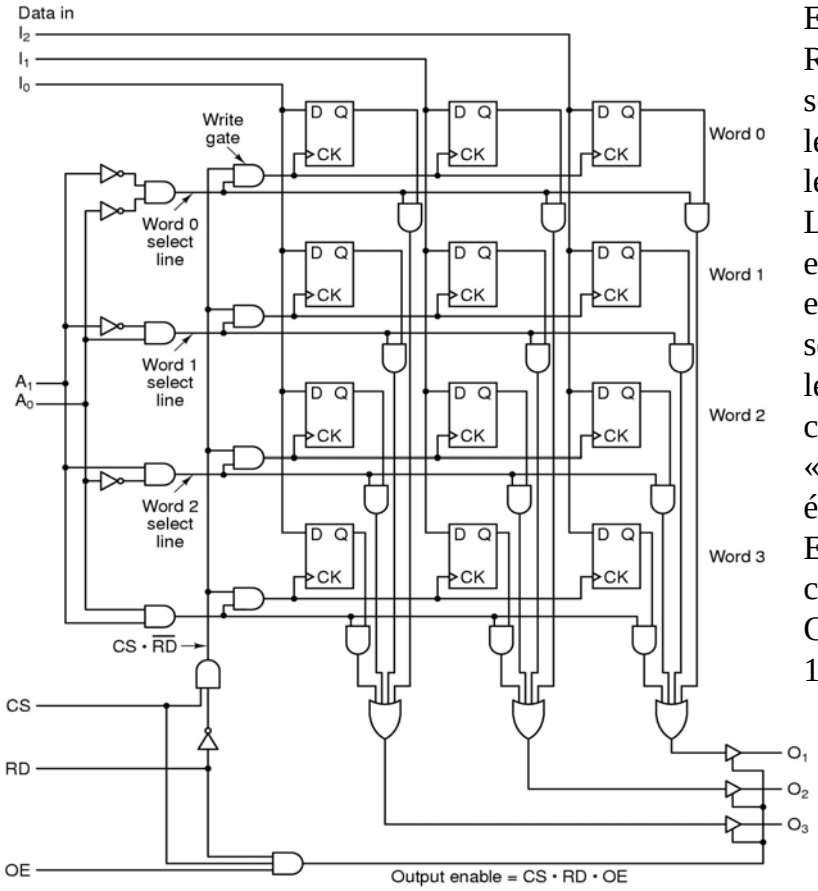
(a)

Registre de 8 bits, tous les CLR et horloges sont reliées, 20 broches nécessaires



(b)

Une mémoire de 4 mots de 3 bits

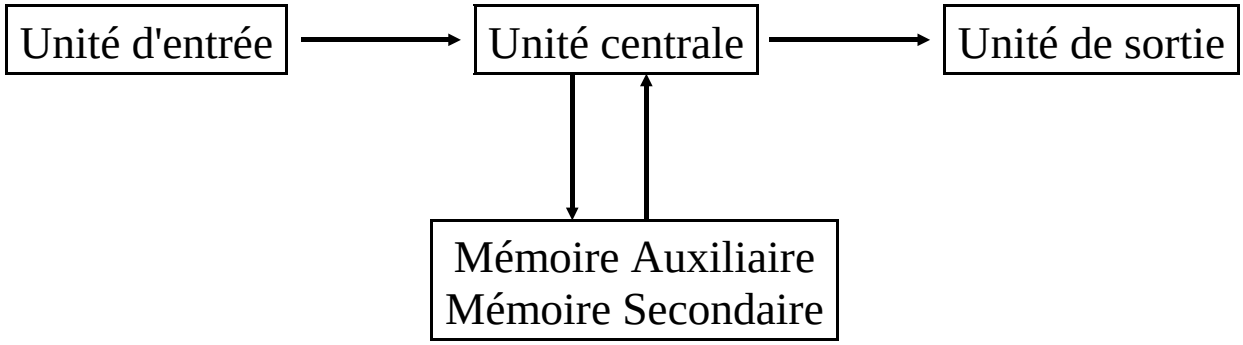


Ecriture: CS=1 (circuit sélectionné) RD=0: les entrées A0 et A1 sélectionnent le mot à écrire et les valeurs I sont écrites dans ce mot, les autres inchangés.
Lecture: CS=1, RD=1, OE=1 (output enable) Le mot est sélectionné par les A et la valeur des sorties Q est envoyée en sortie
les inverseurs situés à la sortie se comportent comme des interrupteurs et « isolent » les sorties pendant les écritures .
En pratique entrées et sorties sont confondues.
Cette mémoire de 12 bits n utilise que 14 broches .

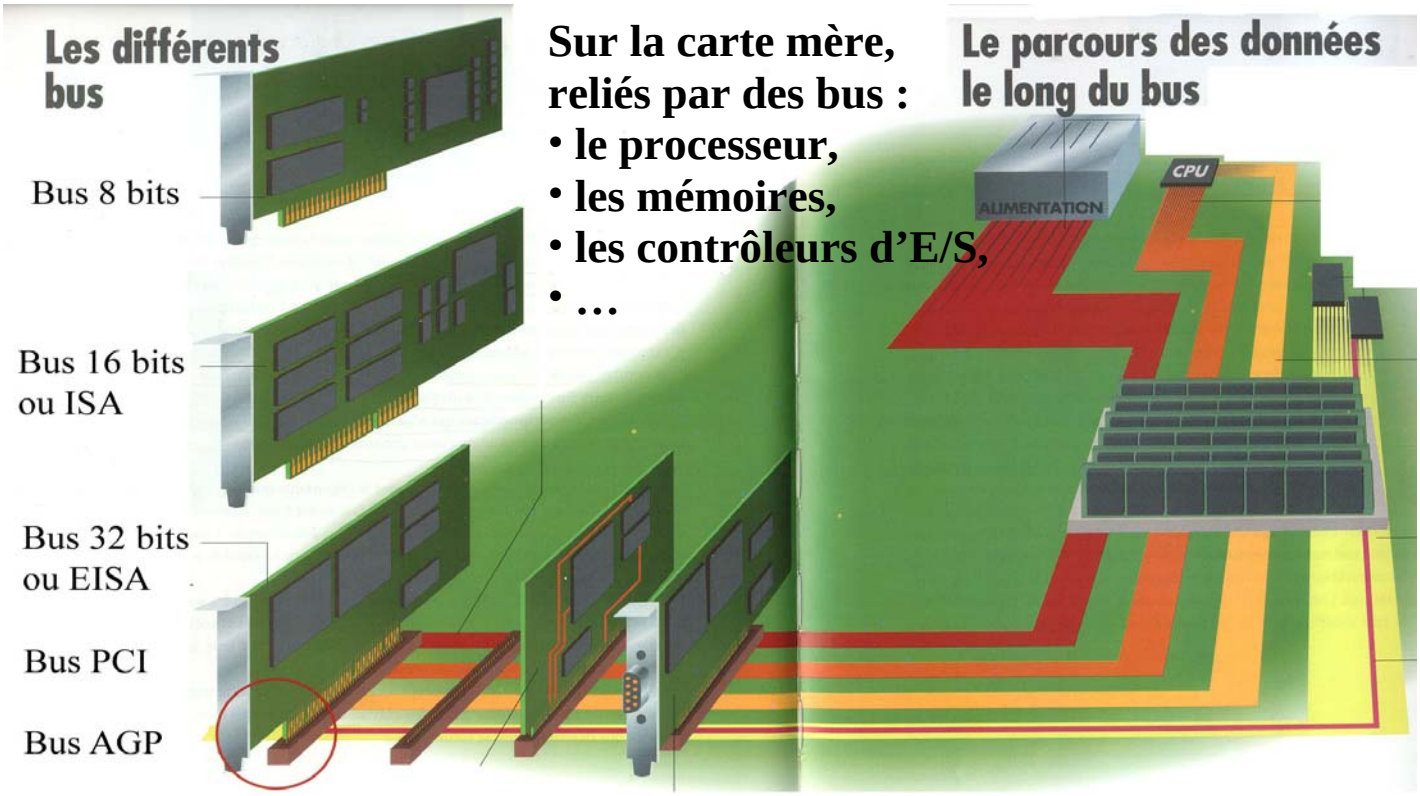
4 - Fonctionnement d'un ordinateur

Les différents éléments d'un ordinateur :
- le matériel (Hardware) : les éléments physiques,
- les logiciels (Software) : les programmes.

Le matériel :



Unité centrale

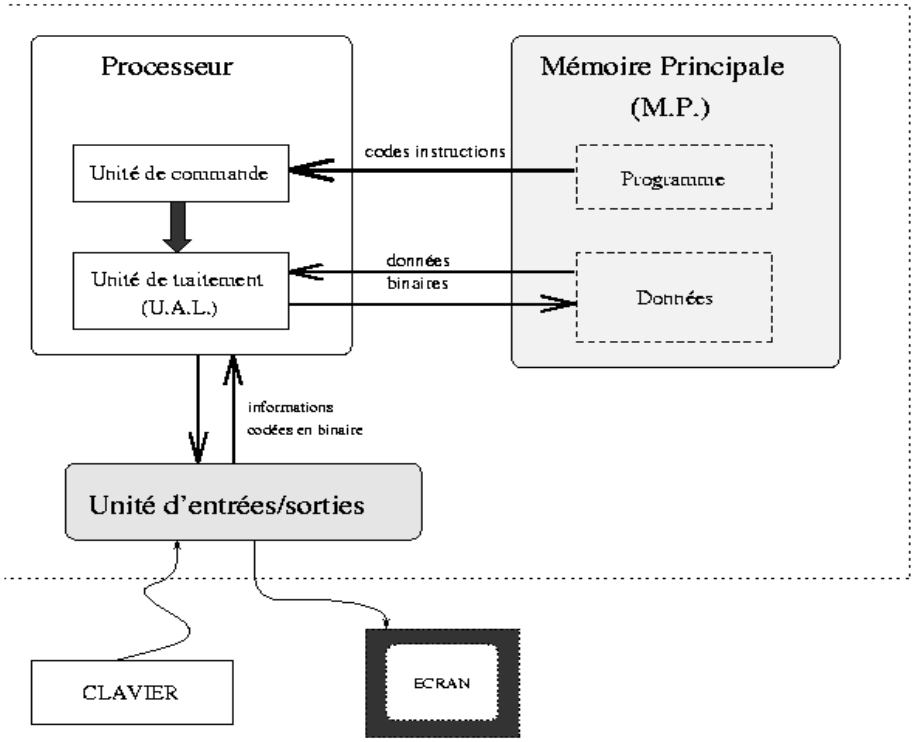


Sur la carte mère, reliés par des bus :
- **le processeur,**
- **les mémoires,**
- **les contrôleurs d'E/S,**
- **…**

Le micro-processeur

ou CPU (Central Processing Unit) ou MPU (Micro Processing Unit)

2 parties :
- unité de commande (U.C.) (ou unité de contrôle)
- unité de traitement (ou Unité Arithmétique et Logique U.A.L. ou A.L.U.)



Déroulement de l’exécution d’une instruction

- lire en mémoire l’instruction à exécuter,
- effectuer le traitement correspondant (U.A.L. + U.C.),
- passer à l’instruction suivante (U.C.) .

Exemple : *« ajouter 5 au contenu de la case mémoire 180 »*

- 1. le processeur lit et décode l'instruction;
- 2. le processeur demande à la mémoire la contenu de l'emplacement 180;
- 3. la valeur lue est rangée dans l'accumulateur;
- 4. l'unité de traitement (UAL) ajoute 5 au contenu de l'accumulateur;
- 5. le contenu de l'accumulateur est écrit en mémoire à l'adresse 180.

Organisation d’un processeur

Un processeur travaille avec des registres :

- Accumulateur,
- Registre d’état,
- Registre instruction,
- Pointeur d’Instruction,
- ...

Exemple : le 80x86

8088 :
Micro-processeur 16 bits
Bus 8 bits

8086 :
Micro-processeur 16 bits
Bus 16 bits

80x86 (80286, 80386, 80486, …) :
compatibilité ascendante

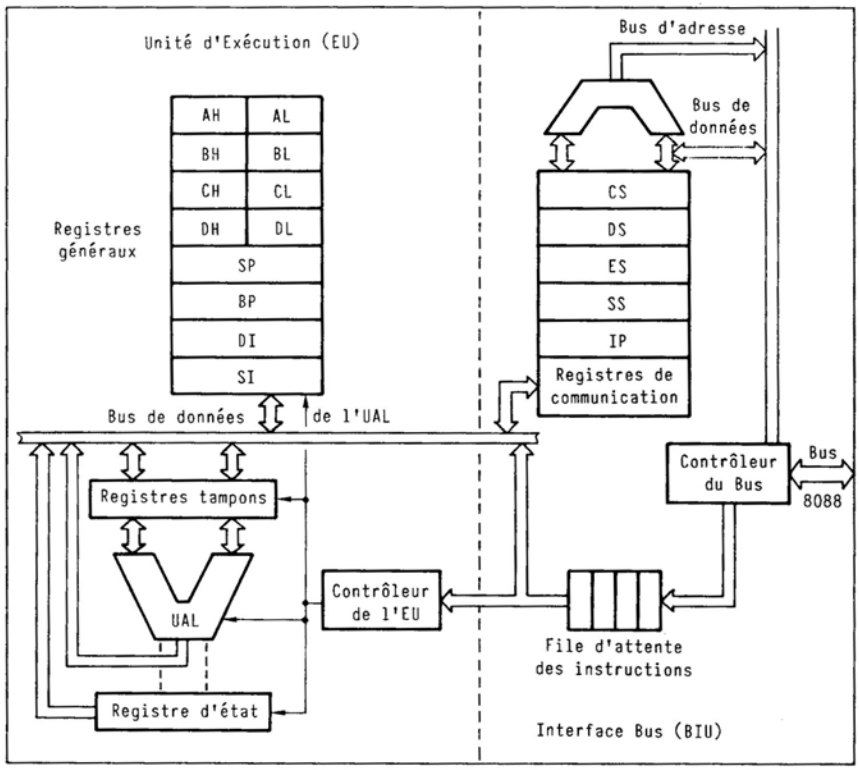
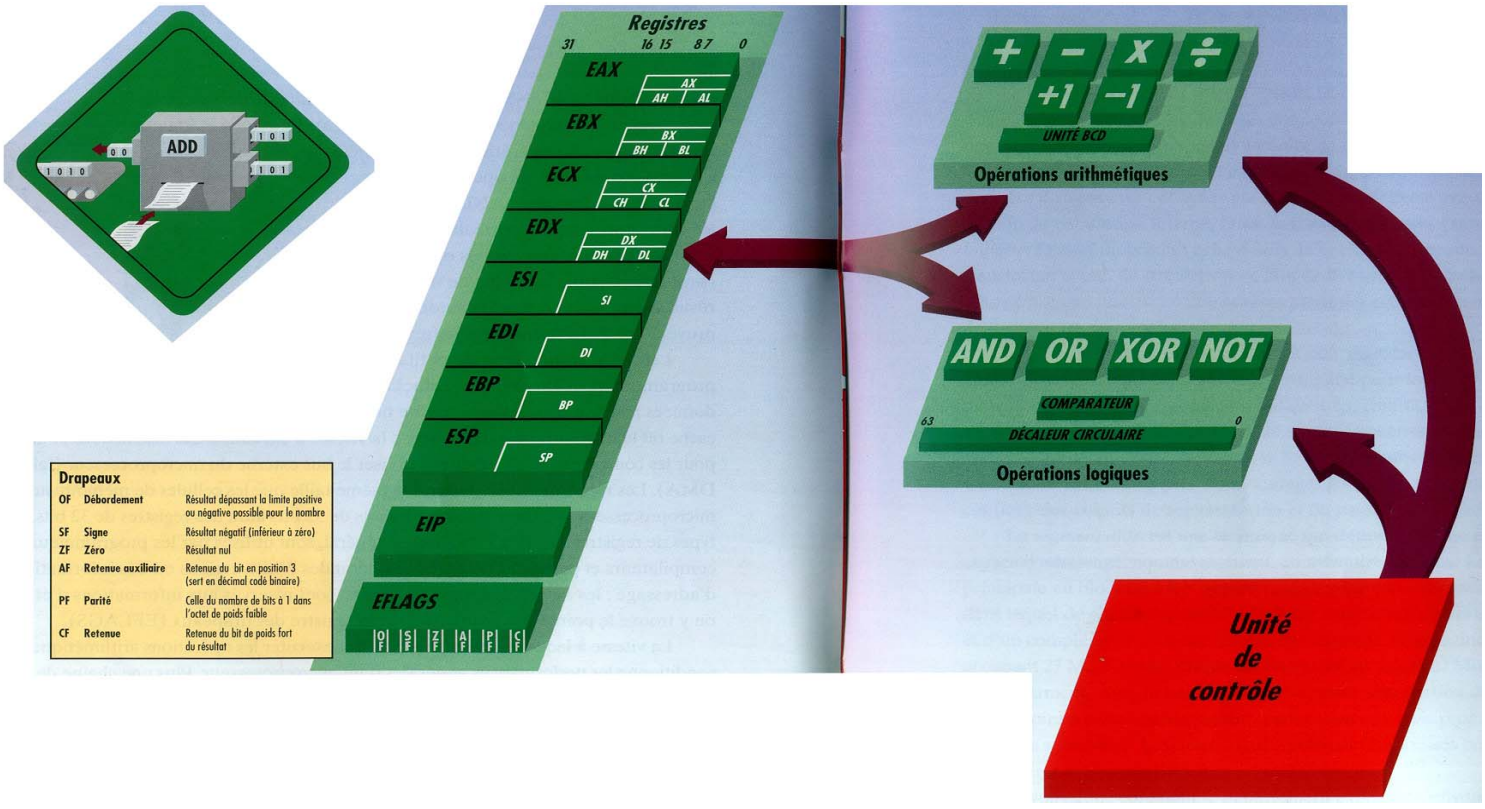


Figure 1.1. Organisation interne du 8088.

Exemple : U.A.L. 32 bits



Le langage machine du 8086

Micro-processeur à 40 broches
dont 20 pour le bus d'adresse
==> adressage de 1 Mo (2^{20})

Registres de 16 bits
==> adressage de 64 Ko (2^{16})
==> utilisation de registres de segments pour accéder à toute la mémoire

==> mêmes principes que les processeurs actuels (compatibles)
mais plus simple

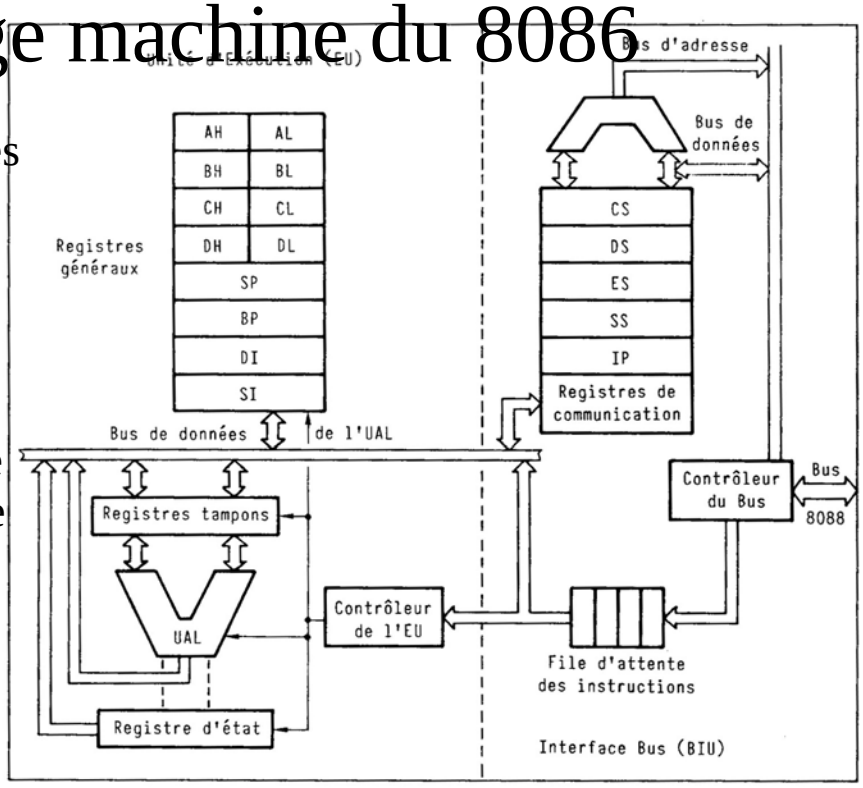


Figure 1.1. Organisation interne du 8088.

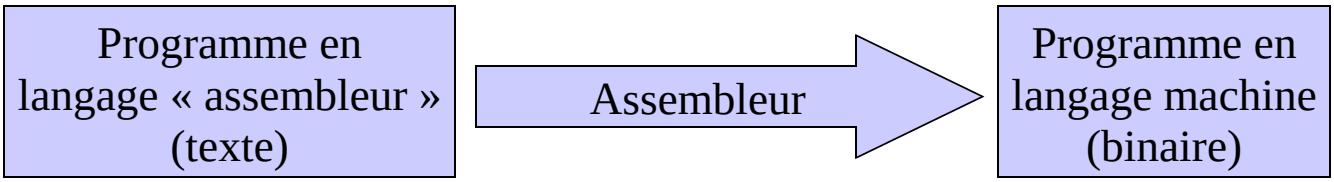
Le langage machine

Le langage binaire :
00000111 pour faire une addition avec AX
00101101 pour faire une soustraction avec AX

Le langage hexadécimal (base 16) :
07H pour faire une addition avec AX
2DH pour faire une soustraction avec AX

Le langage assembleur (utilisation de mnémoniques) :
ADD AX pour faire une addition avec AX
SUB AX pour faire une soustraction avec AX

Le langage assembleur



1 instruction assembleur ====> 1 instruction binaire

Différent pour chaque machine, pour chaque processeur
===> programmes très peu portables

Pourquoi utiliser le langage assembleur ?

• Aux débuts de l'informatique : seul moyen de programmation
• Aujourd'hui : utilisé pour être plus prés de la machine, pour savoir exactement les instructions générées (pour contrôler ou optimiser une opération)

On retrouve l'assembleur dans :
- la programmation des systèmes de base des machine (le pilotage du clavier, de l'écran, des disques durs et disquettes, ...),
- certaines parties du système d'exploitation,
- le pilotage de nouveaux périphériques (imprimantes, scanners, disques optiques, écrans graphiques, ...),
- l'accès aux ressources du système,
- ...

Avantages : génération de programmes efficaces et rapides (à l'exécution)
Inconvénients : développement et mise au point long et coûteux

====> Utilisé ponctuellement
====> pour nous, l'objectif est de comprendre le fonctionnement du cœur de la machine

Rappels :
8 bits
= 1 octet
= 1 byte (anglais)
16 bits
= 1 mot (de 2 octets)
= 1 word (anglais)

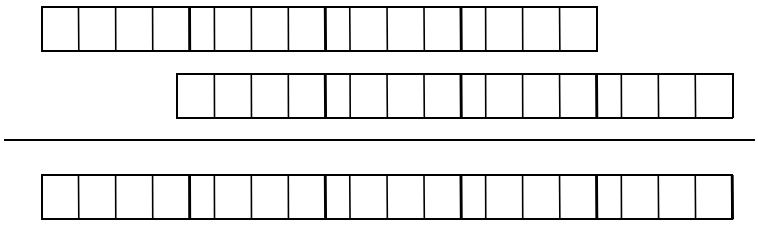
Assembleur 8086

Gestion des adresses mémoires sur deux mots :
- 1 mot pour le segment,
- 1 mot pour le déplacement dans le segment : l'offset ou adresse relative.

Une adresse est notée : `segment:offset`

On peut calculer l'adresse réelle par la formule suivante :

adresse = 16 * Segment + Offset



Segment

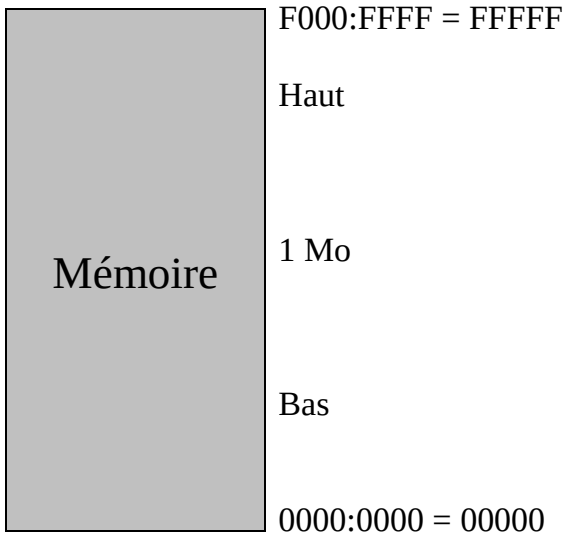
Offset

Adresse (20 bits)

Remarque:
Deux segments consécutifs ne sont pas disjoints.

Exemple :
Ces deux adresses indiquent le même octet en mémoire :

0000:0010 et 0001:0000
= 00010 = 00010



Zones mémoires réservées par le micro-processeur :
- les 16 octets les plus hauts sont réservés à la mise en service
(Que faire quand la machine se met sous tension ?)
(F000:FFF0 --> F000:FFFF),
- les 1024 octets les plus bas sont réservés aux vecteurs d'interruption
(0000:0000 --> 0000:03FF).

Zones mémoire réservées par l'architecture, l'organisation interne de la machine :
- une partie de la mémoire est constituée de ROM (Read Only Memory) et contient les opérations de base de la machine,
- une partie était réservée à la mémoire écran :
+ à partir de B000:0000 pour un écran monochrome,
+ à partir de B800:0000 pour un écran couleur.

Les registres

On distingue quatre groupes de registres :
- les registres de données,
- les registres de segments,
- les registres pointeurs et index,
- le registre d'état.

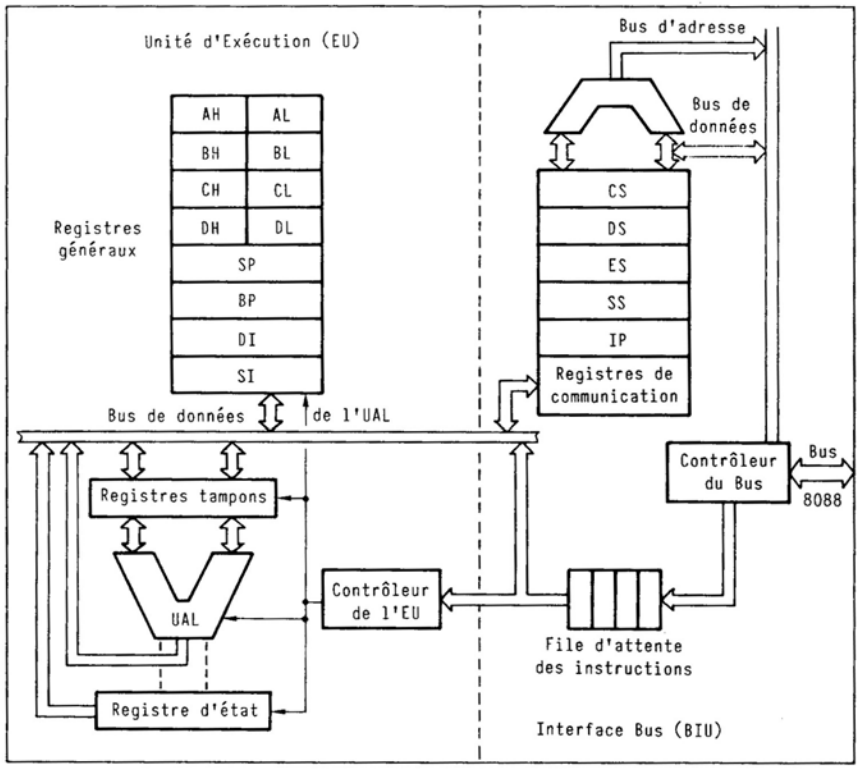


Figure 1.1. Organisation interne du 8088.

Les registres de données

Ces registres, auquel un rôle privilégié a été attribué, peuvent être considérés comme un registre de 16 bits ou deux de 8 bits :

+ AX (ou AL et AH pour respectivement la partie basse (Low) et haute (High)) appelé Accumulateur, utilisé pour les opérations arithmétiques,

+ BX (ou BL et BH) appelé registre de base ou base (Base register), utilisé pour l'adressage de données en mémoire,

+ CX (ou CL et CH) appelé compteur (Count register), utilisé comme compteur de boucle,

+ DX (ou DL et DH) appelé registre de données (Data register), utilisé en complément d'opération arithmétique.

Les registres de segments

Ces registres indiquent les segments des zones standards d'un programme :

+ CS (Code Segment) : Segment de code c'est à dire contenant les instructions en cours d'exécution,

+ SS (Stack Segment) : Segment de pile c'est à dire contenant la pile,

+ DS (Data Segment) : Segment de données c'est à dire contenant les données en cours d'utilisation,

+ ES (Extra Segment) : Segment de données supplémentaire.

Les registres Pointeurs et Index

Ces registres vont contenir des offsets pour adresser la mémoire dans les segments vus précédemment :

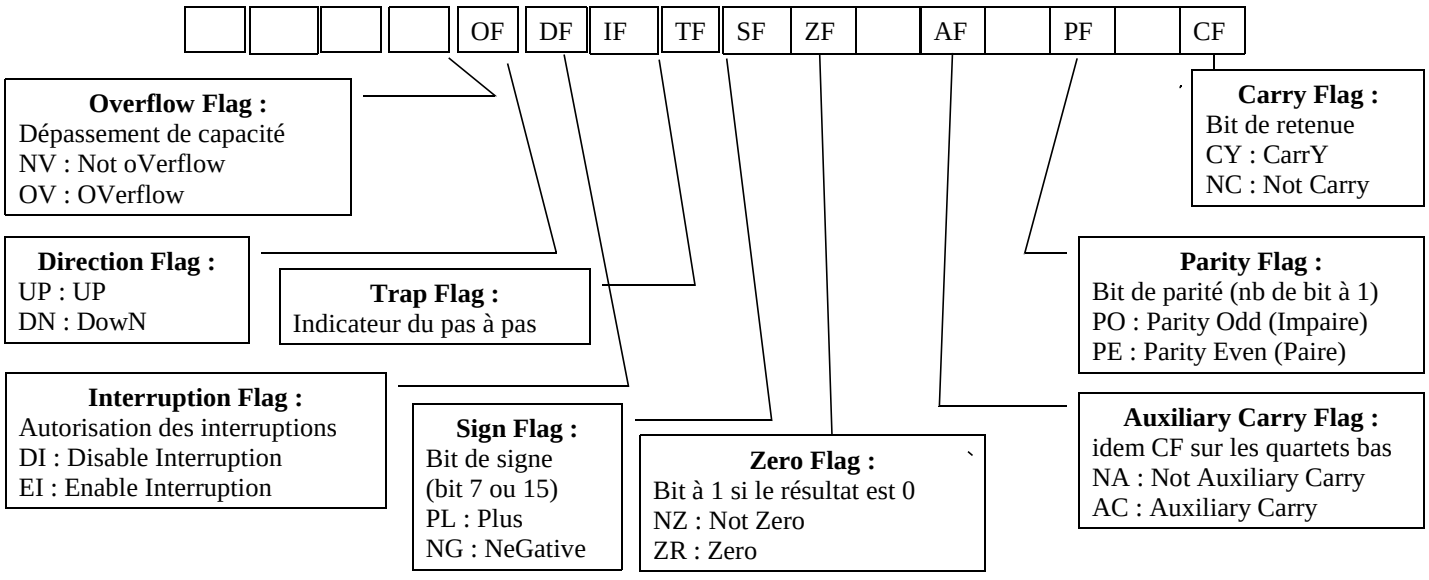
+ IP (Instruction Pointer), associé à CS, CS:IP donne l'adresse de la prochaine instruction à exécuter,

+ BP (Base Pointer) et SP (Stack Pointer), associés à SS, SS:SP indique le sommet de la pile,

+ SI (Source Index) et DI (Destination Index), registres d'index, utilisés pour parcourir des tableaux, en standard, SI est associé à DS, DI à ES.

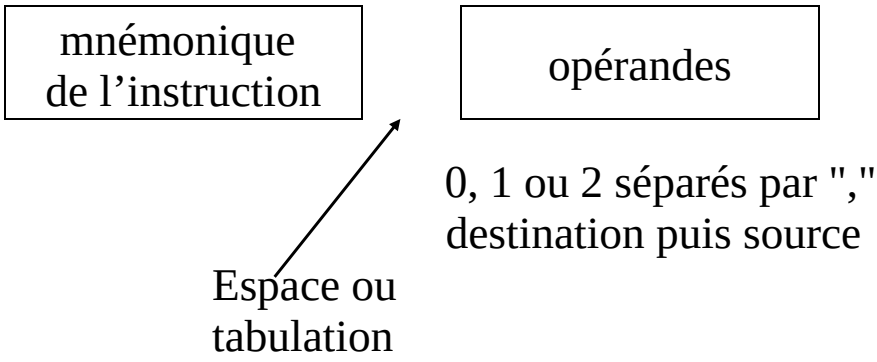
Le registre d'état

Ce registre de 16 bits contient des indicateurs (flags ou drapeaux) sur l'état du micro-processeur, en particulier à la suite d'opérations arithmétiques ou de comparaison.



Les instructions

Les instructions sont constituées de deux zones :



Exemple :
MOV dest,source pour effectuer un transfert de source vers dest

Les modes d'adressage

Il existe différentes manières d'accéder aux informations, différents modes d'adressage :

- adressage par registre :

Utilisation des noms des registres (AX, BX, CX, ...)

Exemples : MOV AX,BX
MOV AL,BL

- adressage immédiat :

Utilisation d'une valeur donnée, d'une constante

Exemples : MOV AX,460H
MOV AL,-30

Les modes d’adressage (2)

- les adressages en mémoire :
+ adressage direct :
Utilisation de l'adresse mémoire où aller chercher la donnée

Exemple : MOV AX,[300H]
--> Offset dans DS

+ adressage indirect par registre :
Utilisation de l'adresse mémoire contenue dans un registre (BX, BP, SI, DI)

Exemple : MOV BX,300H
MOV AX,[BX]

Les modes d’adressage (3)

+ adressage relatif à une base :
Utilisation de l'adresse mémoire obtenue par la somme d'un registre de base (BX ou BP) et un déplacement (dep)

Il existe différentes notations :
[BX + dep]
[BX] + dep
dep[BX]

Exemple : MOV BX,2F0H
MOV AX,[BX+10H]

Les modes d’adressage (4)

+ adressage direct indexé :
Utilisation de l'adresse mémoire obtenue par la somme d'un déplacement (dep) et un registre d'index (SI ou DI)

Exemple : MOV DI,10H
MOV AX,[2F0H+DI]

+ adressage indexé par rapport à une base :
Utilisation de l'adresse mémoire obtenue par la somme d'un registre de base (BX ou BP), d'un registre d'index (SI ou DI) et d'un déplacement (dep).

Exemple : MOV DI,15H
MOV BX,0BH
MOV AX,[BX+DI+1E0H]

Les instructions

On distingue 6 types d'instructions :
- les instructions de transfert de données,
- les opérations arithmétiques et logiques,
- les instructions de contrôle du programme,
- les instructions de travail sur les chaînes,
- les interruptions,
- les instructions de contrôle du processeur et du registre d'état.

Une instruction nécessite un certain nombre de cycles pour s’exécuter.

La durée d ’un cycle dépend de la fréquence d’horloge.

Les instructions de transfert

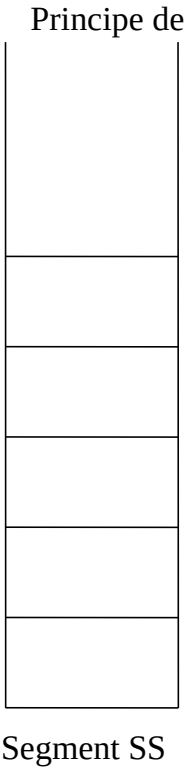
La principale instruction de transfert :

MOV destination,source

Transfert de l'octet ou du mot de source dans destination

impossible de faire les transferts mémoire/mémoire, registre de segment/immédiat,
impossible d'utiliser IP

Les instructions de transfert (2)



PUSH source

Empiler un mot sur la pile

SP = SP - 2
(SP+1; SP) = source

POP destination

Dépiler un mot sur la pile

destination = (SP+1; SP)
SP= SP + 2

Les opérations logiques

AND destination,source
ET logique entre destination et source, le résultat est mis dans destination

OR destination,source
OU logique entre destination et source, le résultat est mis dans destination

XOR destination,source
OU exclusif entre destination et source, le résultat est mis dans destination

NOT destination
Négation logique de destination, le résultat est mis dans destination

TEST destination,source
ET logique entre destination et source pour positionner les indicateurs sans modification de destination

Les opérations arithmétiques

CBW
Convertir un octet en un mot en tenant compte du signe (AL --> AX)

Addition

ADD destination,source
destination = destination + source

ADC destination,source
Addition avec retenue (Add with carry)
destination = destination + source + CF

INC destination
Incrémentation
destination = destination + 1

Les opérations arithmétiques (2)

Soustraction

SUB destination,source
destination = destination - source

SBB destination,source
Soustraction avec retenue (Subtract with Borrow)
destination = destination - source - CF

DEC destination
Décrémentation
destination = destination - 1

Les opérations arithmétiques (3)

NEG destination
Négation
destination = - destination

CMP destination,source
Comparaison
Soustraction (destination - source) pour le positionnement des indicateurs, sans modification de destination

Les opérations arithmétiques (4)

Multiplication

MUL source
Multiplication non signée
si source est un octet, AX = AL * source
si source est un mot, (DX; AX) = AX * source

IMUL source
Multiplication signée
si source est un octet, AX = AL * source
si source est un mot, (DX; AX) = AX * source

Les opérations arithmétiques (5)

Division

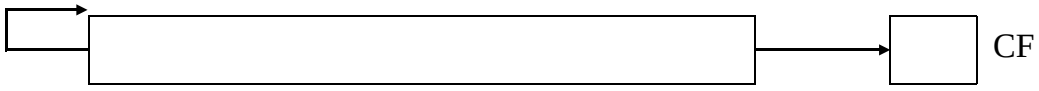
DIV source
Division non signée
si source est un octet, AL = AX DIV source
AH = AX MOD source
si source est un mot, AX = (DX; AX) DIV source
DX = (DX; AX) MOD source

IDIV source
Division signée
si source est un octet, AL = AX DIV source
AH = AX MOD source
si source est un mot, AX = (DX; AX) DIV source
DX = (DX; AX) MOD source

Les opérations de décalage

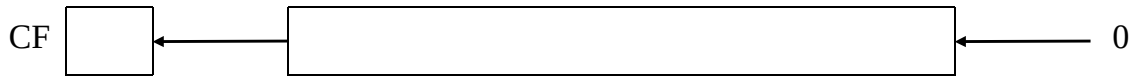
SAR destination,count
(où count = 1 ou CL pour toutes les opérations de décalage et de rotation)

Shift Arithmetic Right : Décalage arithmétique à droite
Si count = 1, division entière par 2, avec le reste dans CF
Le bit de poids fort est laissé à son ancienne valeur (conservation du signe).
Le bit éliminé est mis dans CF.

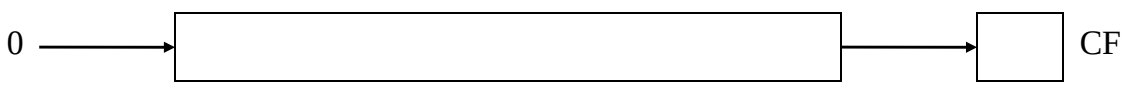


Les opérations de décalage (2)

SHL destination,count ou **SAL destination,count**
Shift Logical Left ou Shift Arithmetic Left : Décalage à gauche
Si count = 1, multiplication par 2
Le bit de poids faible est mis à 0.
Le bit éliminé est mis dans CF.

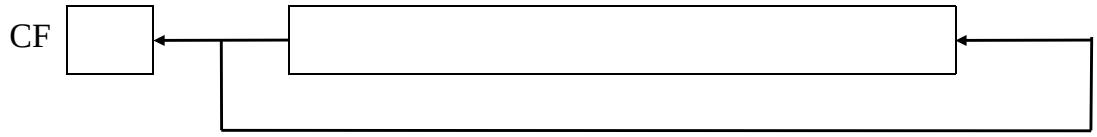


SHR destination,count
Shift Logical Rigth
Décalage logique à droite
Si count = 1, division entière par 2, avec le reste dans CF
Le bit de poids fort est mis à 0.
Le bit éliminé est mis dans CF.

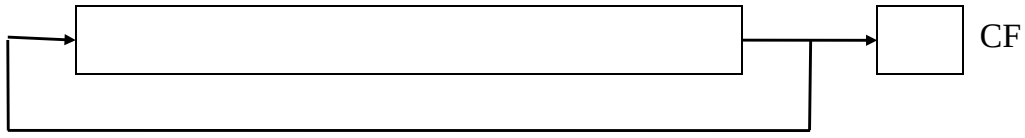


Les opérations de décalage (3)

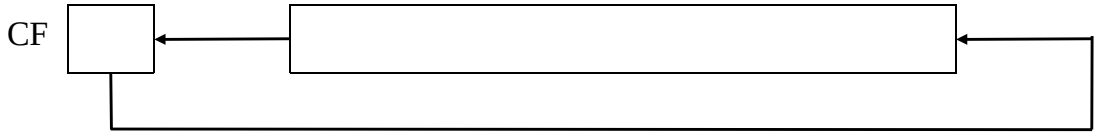
ROL destination,count
Rotate Left : Rotation à gauche
Faire count fois
CF prend la valeur du bit de poids fort
Tous les bits sont décalés de 1 vers la gauche
Le bit de poids faible prend la valeur de CF (de l'ancien bit de poids fort)



ROR destination,count
Rotate Right : Rotation à droite
Faire count fois
CF prend la valeur du bit de poids faible
Tous les bits sont décalés de 1 vers la droite
Le bit de poids fort prend la valeur de CF (de l'ancien bit de poids faible)

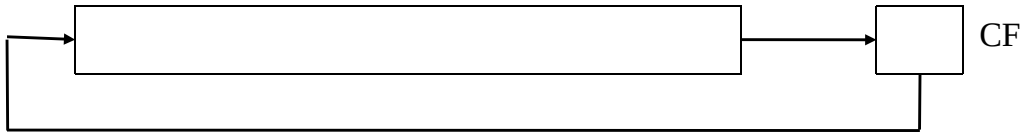


RCL destination,count
Rotate Left through Carry : Rotation à gauche à travers "Carry Flag"
Faire count fois
Tous les bits sont décalés de 1 vers la gauche
Le bit de poids faible prend la valeur de CF
CF prend la valeur du bit de poids fort éliminé



Les opérations de décalage (5)

RCR destination,count
Rotate Right through Carry : Rotation à droite à travers "Carry Flag"
Faire count fois
Tous les bits sont décalés de 1 vers la droite
Le bit de poids fort prend la valeur de CF
CF prend la valeur du bit de poids faible éliminé



Exercices :
1/ Faire l'addition des octets contenus en 40H et 41H, mettre le résultat en 42H.
2/ Multiplier [40H] par 2, mettre le résultat en 41H
3/ Mettre à zéro le quartet de poids fort de [40H], mettre le résultat en 41H
4/ Mettre à zéro l'octet [40H]
5/ Partager [40H] en deux quartets l'un en 41H, l'autre en 42H

Les instructions de contrôle
Les instructions de branchement inconditionnel

JMP destination
Jump
Saut dans le même segment (NEAR) ou dans un autre segment (FAR)

Modification de IP (Saut Intra-segment) ou CS et IP (Saut Inter-segment)

Les instructions de branchement conditionnel

JA ou JNBE CF = 0 et ZF = 0
Jump if Above / if Not Below or Equal
JBE ou JNA CF = 1 ou ZF = 1
Jump if Below or Equal / if Not Above

JAE ou JNC ou JNB CF = 0
Jump if Above or Equal / if Not Carry / if Not Below
JB ou JC ou JNAE CF = 1
Jump if Below / if Carry / if Not Above or equal

Les instructions de branchement conditionnel (2)

JCXZ (CX) = 0
Jump if CX is Zero

JE ou JZ ZF = 1
Jump if Equal / if Zero
JNE ou JNZ ZF = 0
Jump if Not Equal / if Not Zero

JNP ou JPO PF = 0
Jump if No Parity / if Parity Odd
JP ou JPE PF = 1
Jump if Parity / if Parity Even

Les instructions de branchement conditionnel (3)

JG ou JNLE Jump if Greater / if Not Less nor Equal	ZF = 0 et OF = SF
JLE ou JNG Jump if Less or Equal / if Not Greater	ZF = 1 ou OF # SF
JGE ou JNL Jump if Greater or Equal / if Not Less	OF = SF
JL ou JNGE Jump if Less / if Not Greater nor Equal	OF # SF

Les instructions de branchement conditionnel (4)

JNO Jump if No Overflow	OF = 0
JO Jump if Overflow	OF = 1
JNS Jump if No Sign	SF = 0
JS Jump if Sign	SF = 1

Utilisation

Utilisation après un CMP destination,source :

Saut si	Entiers non signés	Entiers signés
destination = source	JE	JE
destination # source	JNE	JNE
destination > source	JA	JG
destination >= source	JAE	JGE
destination < source	JB	JL
destination <= source	JBE	JLE

Les procédures

CALL destination
Appel de procédure dans le même segment (NEAR) ou dans un autre segment (FAR)

Pour un CALL FAR (Inter-segment)
Sauvegarde de CS dans la pile CS = nouvelle valeur
Sauvegarde de IP dans la pile IP = nouvelle valeur

RET
Retour de procédure (NEAR : RET, FAR : RETF)

Restauration à partir de la pile des anciennes valeurs de CS et IP

Exercices

1/ Trouver le plus grand (puis le plus petit) des deux nombres des adresses 40H et 41H, la mettre en 42H

2/ Faire la somme sur 8 bits, des valeurs contenues à partir de l'adresse 42H, le nombre de valeurs à additionner étant en 41H, mettre la somme en 40H

3/ Trouver le nombre de valeurs négatives contenues à partir de l'adresse 42H, le nombre de valeurs étant en 41H, mettre le résultat en 40H

4/ De la même manière, trouver la valeur minimum puis la valeur maximum, puis les deux en même temps

5/ Trouver le nombre de caractères d'une chaîne ASCII terminée par le caractère nul de code ASCII '0'

6/ Faire la concaténation de deux chaînes ASCII terminées par 0 (CH1 et CH2) dans une chaîne RES, mettre le nombre de caractères de la chaîne obtenue dans LG

7/ Convertir un octet NB en une chaîne de caractères numériques RES (123 --> '123')

Les instructions de contrôle du processeur et du registre d'état

NOP
No Opération

CLC
Clear Carry CF = 0

STC
Set Carry CF = 1

CMC
Complement Carry CF = not CF

CLD
Clear Direction Flag DF = 0

STD
Set Direction Flag DF = 1

CLI
Clear Interruption Flag IF = 0

STI
Set Interruption Flag IF = 1

Les interruptions

Une interruption est semblable à un appel de sous-programme mais fait toujours un appel long (segment + offset). L'adresse de branchement n'est pas dans l'instruction, elle est indiquée par le numéro et se trouve dans la table des interruptions dans le segment 0.

On peut donc faire appel aux interruptions indépendamment des adresses mémoires où elles se trouvent.

Les interruptions peuvent être déclenchées par le matériel (clavier, horloge par exemple) ou par un logiciel.

Les interruptions (2)

INT numéro
Appel d'une interruption

•Sauvegarde du registre d'état sur la pile
•Mise à 0 des indicateurs pas à pas (TF) et d'autorisation d'interruption (IF)
•Sauvegarde de CS sur la pile
•Calcul de l'adresse de la table d'interruption (numéro x 4)
•Chargement dans CS du second mot
•Sauvegarde de IP sur la pile
•Chargement du premier mot de la table dans IP
donc Saut à l'adresse CS:IP

Les interruptions (3)

IRET
Retour d'une interruption

Restauration de CS, IP et du registre d'état

Exemples d'interruptions :
- Interruption 0 : Division par 0
- Interruption 10H : Gestion d'écran
- Interruption 16H : Gestion clavier

Codage d'algorithmes

Algorithme

```
Si A = B
Alors C ← 100
Sinon C ← 200
Finsi
```

Assembleur

```
      MOV AX, a
      MOV BX, b
      CMP AX, BX
      JNE sinon
alors:  MOV AX, 100
      MOV c, AX
      JMP finsi
sinon:  MOV AX, 200
      MOV c, AX
finsi:
```

Assembleur optimisé

```
      MOV AX, a
      MOV BX, b
      CMP AX, BX
      JNE sinon
alors:  MOV AX, 100
      JMP finsi
sinon:  MOV AX, 200
finsi:  MOV c, AX
```

Codage d'algorithmes (2)

Algorithme

```
B ← 10
A ← 1
Tantque A ≤ B faire
      A ← A + 1
Fintq
```

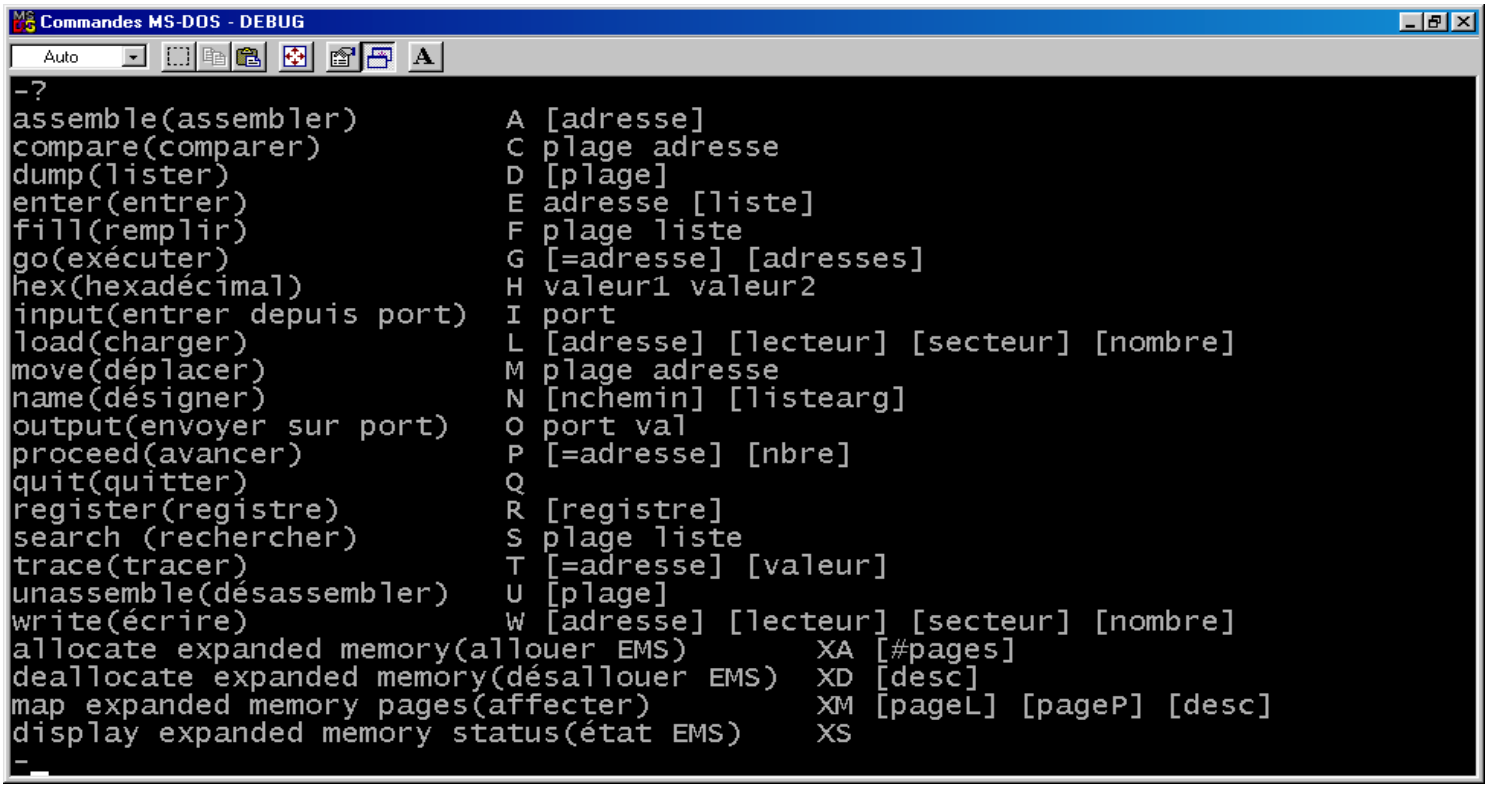
Assembleur

```
      MOV b, 10
      MOV a, 1
      PUSH AX
debtq:  MOV AX, a
      CMP AX, b
      JG fintq
      ADD a, 1
      JMP debtq
fintq:   POP AX
```

Assembleur optimisé

```
      MOV b, 10
      MOV a, 1
      PUSH AX
      PUSH BX
      MOV AX, a
      MOV BX, b
debtq:  CMP AX, BX
      JG fintq
      INC AX
      JMP debtq
fintq:   MOV a, AX
      POP BX
      POP AX
```

Debug



Les principales commandes

assemble(assembler)	A [adresse]
dump(lister)	D [plage]
enter(entrer)	E adresse [liste]
fill(remplir)	F plage liste
go(exécuter)	G [=adresse] [adresses]
proceed(avancer)	P [=adresse] [nbre]
quit(quitter)	Q
register(registre)	R [registre]
search (rechercher)	S plage liste
trace(tracer)	T [=adresse] [valeur]
unassemble(désassembler)	U [plage]

Assembleur

Assembleur / Macro-assembleur

Déclaration de variables
Utilisation d’étiquettes
Utilisation de pseudo-codes, de directives, de macros

Test.asm

```
; Commentaires
; programme de test

STACK        SEGMENT  STACK
         DB  64 DUP('STACK   ')
STACK        ENDS

DSEG     SEGMENT
I        DB  0
A        DW  121
B        DW  1881
result   DW  ?
DSEG     ENDS

CSEG     SEGMENT
         ASSUME CS:CSEG,DS:DSEG,SS:STACK
debut:       MOV AX, DSEG ; étiquette de la 1ere instruction
         MOV DS, AX   ; initialisation de DS

         MOV AX, A
         ADD AX, B
         MOV result, AX   ;

         ; Retour au DOS
         MOV AH, 4CH
         INT 21H
CSEGENDS
    END debut        ; étiquette de la 1ere instruction
```

Procédure

Appel du macro-assembleur ==> **Génération d'un .obj**

```
D:\Gilles\Cours_GT_2002\Architecture\MASM>masm test
The IBM Personal Computer MACRO Assembler
Version 1.00 (C)Copyright IBM Corp 1981

Object filename [TEST.OBJ]:
Source listing  [NUL.LST]:
Cross reference [NUL.CRF]:

Warning Severe
Errors  Errors
0       0
```

Edition de lien ==> **Génération d'un .exe**

```
D:\Gilles\Cours_GT_2002\Architecture\MASM>link test

IBM Personal Computer Linker
Version 2.10 (C)Copyright IBM Corp 1981, 1982, 1983

Run File [TEST.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

D:\Gilles\Cours_GT_2002\Architecture\MASM>
```