

Cours Architecture des Ordinateurs

1ère Année

**IUT de Nice- Côte d'Azur
Département Informatique**

Marie-Agnès PERALDI-FRATI
Maître de Conférences
map@unice.fr

Organisation de ce cours

- **Cours , TD, TP = 25h**
 - 6 séances de cours
 - 3 séances de TD
 - 7 séances de TP
- **Evaluation :**
 - 2 examens de contrôle continu
 - 1 examen final
- **Dates :**
 - Début du cours : 5 septembre
 - Fin du cours 12 Décembre (Examen final):
- **Intervenants :**
 - Marie-Agnès Peraldi-Frati
 - Erol Acundeger
 - Gurvan Huiban

Objectif du cours

- De quoi est composé un ordinateur :
microprocesseur, des mémoires, disque dur ...?
- Quels sont les modèles sous-jacents au
fonctionnement d'une machine ?
- Comment s'exécutent des programmes sur un
ordinateur ?
- Quel est le lien entre le logiciel et le matériel ?
- Comment se fait l'interface avec l'extérieur ?
(fonctionnement des divers périphériques)

Plan

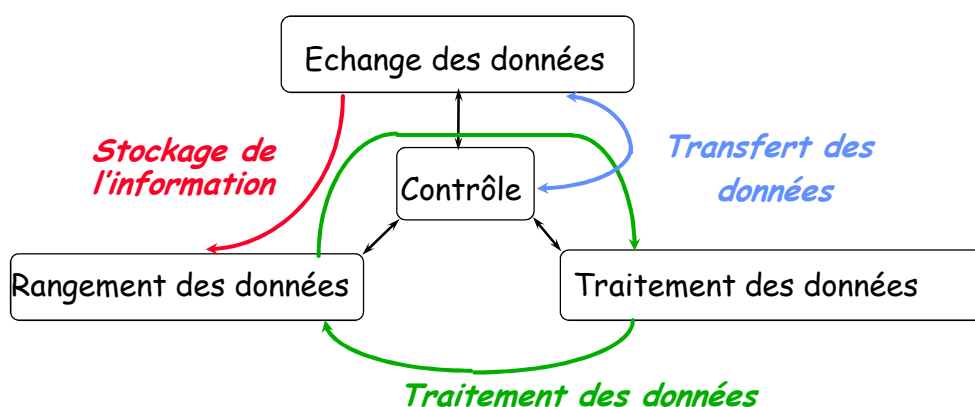
- Introduction
 - Différentes vues d'un ordinateur
 - Evolution et performance des machines
- Représentation de l'information
- Algèbre de Boole
- Circuits séquentiels/Automates
- Architecture type Von Neumann
 - Structure d'interconnexion : bus
 - La mémoire
 - l'unité centrale
- Exemple d'un processeur Intel Pentium
- Couche d'assemblage
 - langage
 - Modes d'adressage
 - Procédures

Introduction

Les différentes « vues » d'un ordinateur

- Vue services
- Vue matérielle
- Vue fonctionnelle

Services rendus par un ordinateur



Introduction

Les différentes « vues » d'un ordinateur

- Vue **services**

- Vue **matérielle**

- Vue **fonctionnelle**

Décomposition matérielle d'un ordinateur

- Un ordinateur est constitué de plusieurs parties :
 - souris
 - écran
 - clavier
 - **unité centrale**
 - lecteur de disquettes ...
- A l'intérieur de l'**unité centrale**
 - une **carte mère**
 - une carte vidéo
 - des disques
- Sur la carte mère
 - un **microprocesseur**
 - de la mémoire (ROM, RAM) ...
- Dans le **microprocesseur la puce**

Constitution d'une Puce

- Une puce est un carré de silicium constitué de millions de transistors
- l'intégration des transistors sur une puce suit la loi de Moore

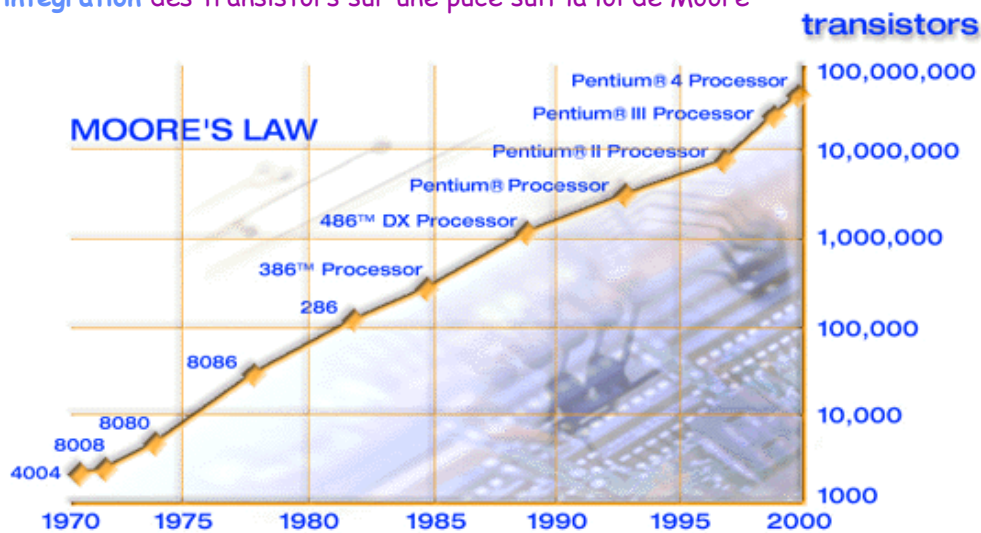


Illustration de la loi de Moore
<http://www.intel.com/research/silicon/mooreslaw.htm>) 9

M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique

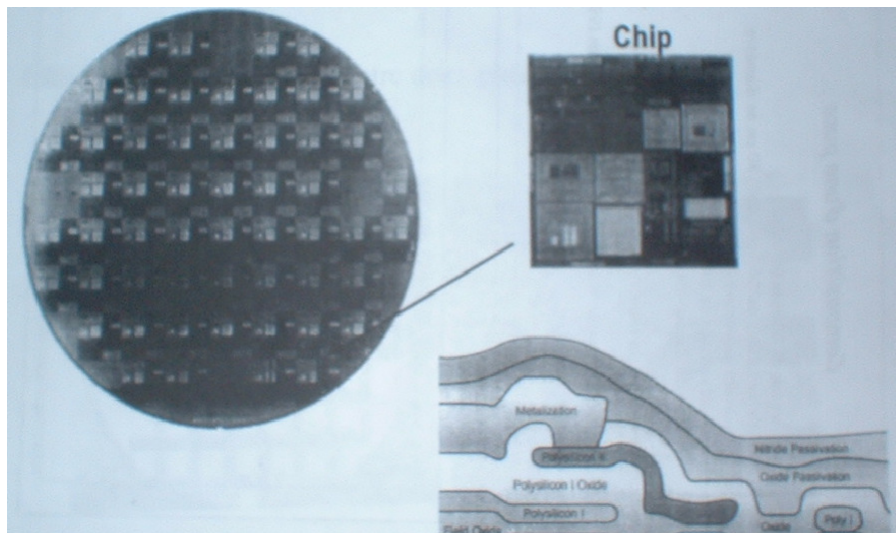


Fabrication d'une puce

- Du **Silicium (SI)** ...du sable... après l'oxygène, le constituant le plus massif et le plus répandu sur Terre.
- Un **barreau de Silicium** fabriqué dans un four à très haute température
- Un **Wafer** (disque de Silicium Dopé) 30 cm de diamètre
- Silicium est un **semi-conducteur** (certaines parties peuvent être plus ou moins conductrices)
- Dépôt de couches de **substrat**
 - (dépôt d'un produit photosensible , zones imprimées par ultraviolet, acide pour le décapage des zones, dépôt d'impuretés pour créer la connectivité)
- Création de **transistors inter-connectés** => plusieurs puces sur un wafer
- Découpage de la puce et tests
- Mise en boîtier et connexion des pattes du support avec les points de contacts de la puce => le **microprocesseur**
- 140 Millions de transistors sur le microprocesseur HP PA850

M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique
 Prix d'un transistor => 1\$ en 1968 0,0000001\$ en 2002 10

Un Wafer



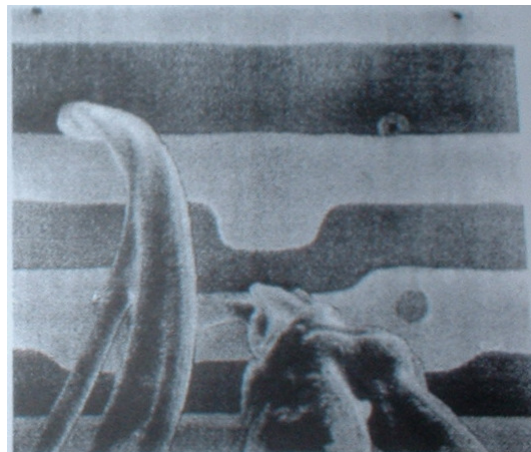
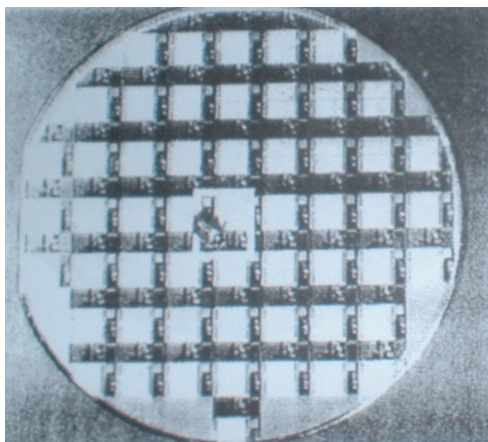
M.-A. Peraldi-Frati-IUT de Nice Dép. Informatique

Source L. Thein Cadence Design Systems, Inc

11

Interconnexions de pistes

- Finesse de gravure de la grille de silicium
- Plus la gravure est fine, plus la **fréquence d'horloge** (exprimée en mégahertz) s'**accroît** et plus la **consommation électrique diminue**
- **Objectif** => moins de 0,08 microns entre deux pistes

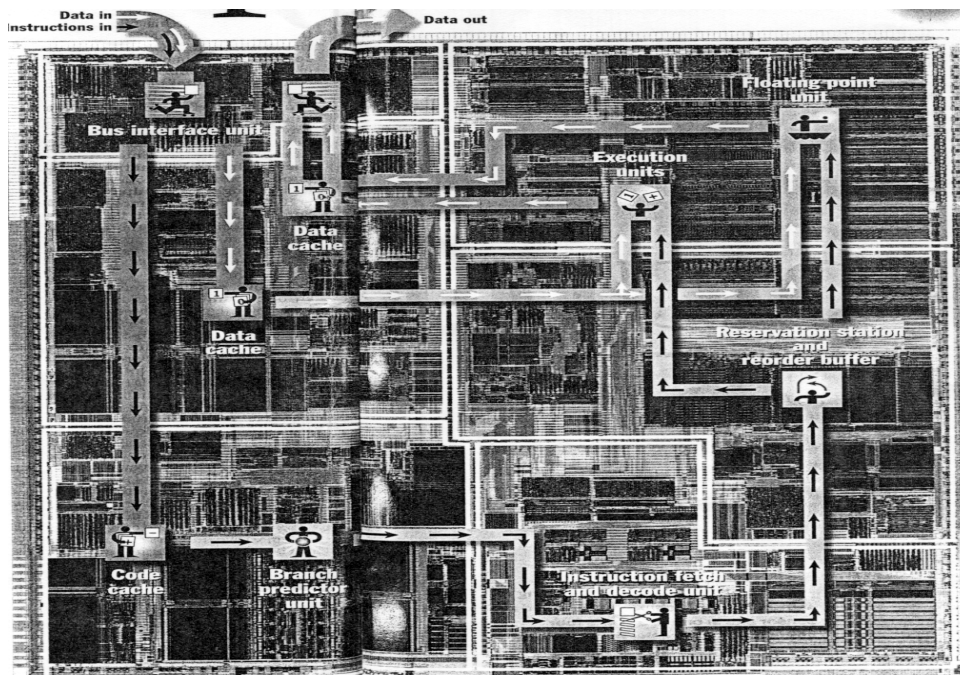


M.-A. Peraldi-Frati-IUT de Nice Dép. Informatique

Source L. Thein Cadence Design Systems, Inc

12

Vue Macroscopique d'une puce



M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique

13

Introduction

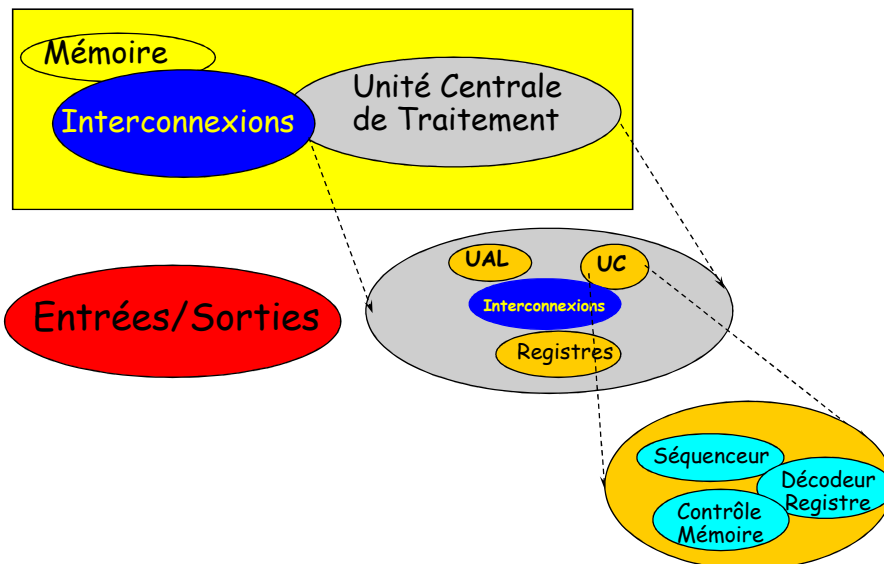
Les différentes « vues » d'un ordinateur

- Vue **services**
- Vue **matérielle**
- Vue **fonctionnelle**

M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique

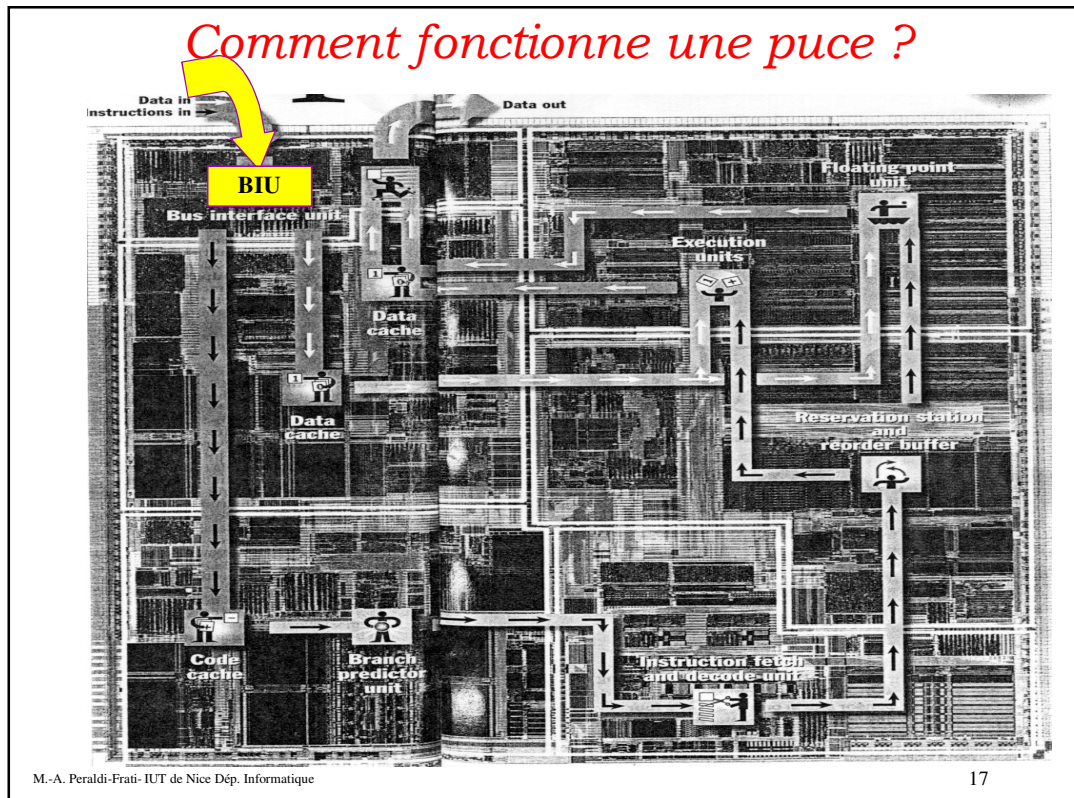
14

Décomposition fonctionnelle d'un ordinateur



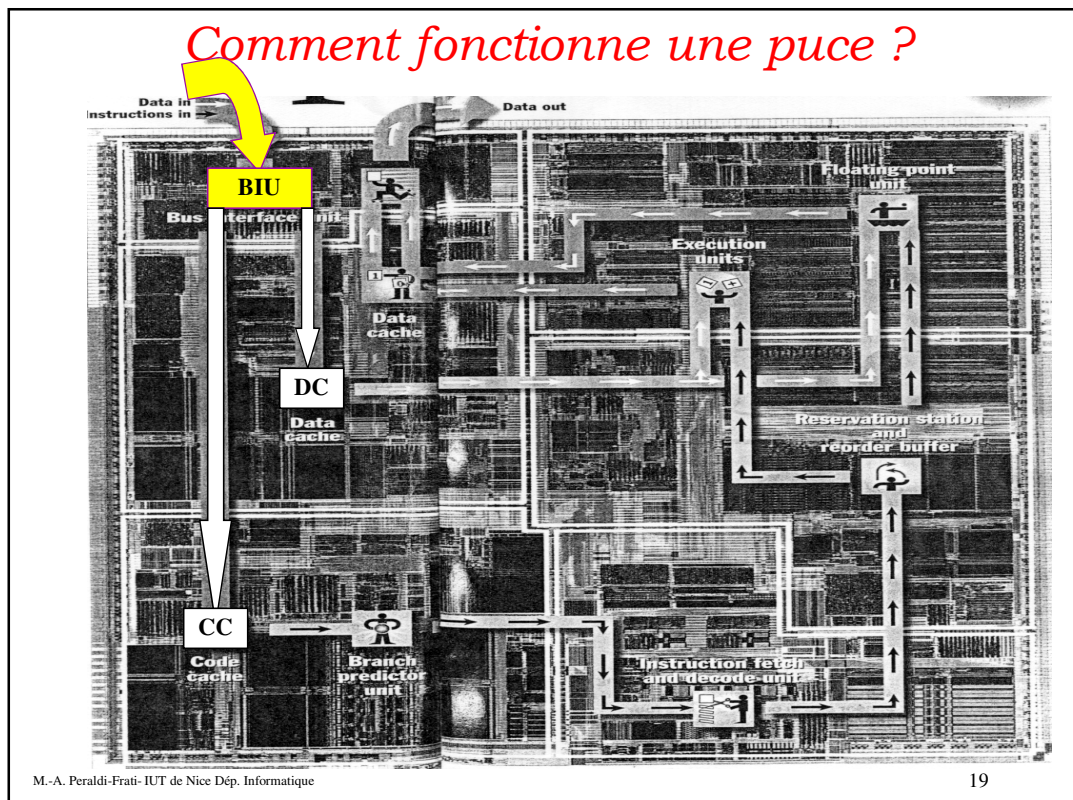
Fonctionnement d'une puce ?

- Une puce => carré de silicium constitué de millions de transistors qui manipulent des instructions et des données
- Une puce de Pentium II exécute 500 millions instructions /seconde
 - Les informations arrivent de la RAM du PC dans la puce par le bus (BIU Bus Interface Unit)



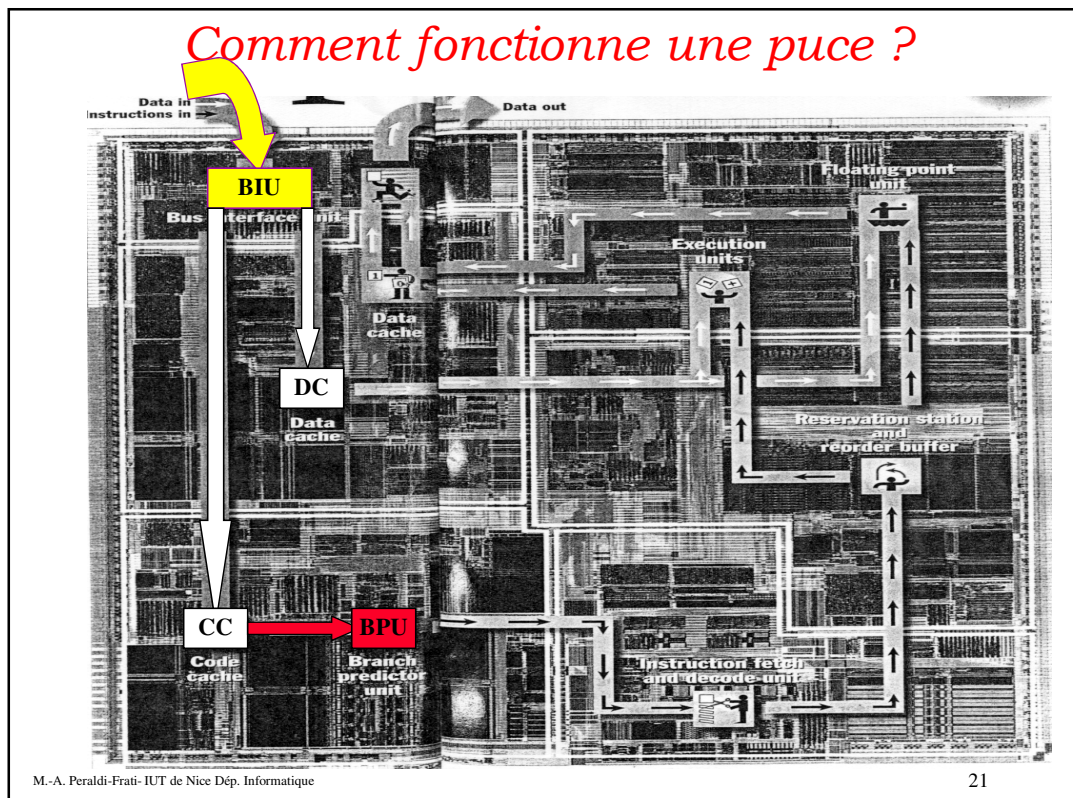
Comment fonctionne une puce ?

- Une puce => carré de silicium constitué de millions de transistors qui manipulent des instructions et des données
- Une puce de Pentium II exécute 500 million instructions /seconde
 - Les informations arrivent de la RAM du PC dans la puce par la bus (BIU Bus Interface Unit)
 - L'information va dans un cache de code ou de donnée (CC,DC)



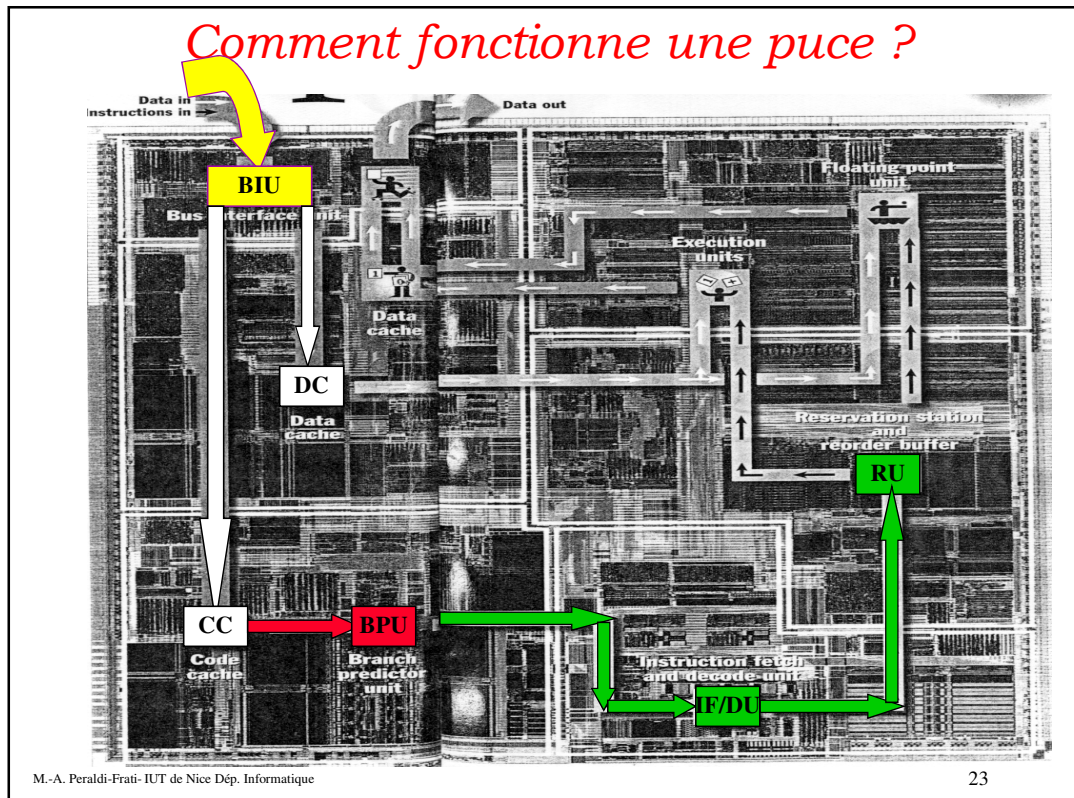
Comment fonctionne une puce ?

- Une puce => carré de silicium constitué de millions de transistors qui manipulent des instructions et des données
- Une puce de Pentium II exécute 500 million instructions /seconde
 - Les informations arrivent de la RAM du PC dans la puce par le bus (BIU Bus Interface Unit)
 - L'information va dans un cache de code ou de donnée (CC,DC)
 - L'unité de prédiction de branchement détermine le chemin optimum pour l'information (BPU)



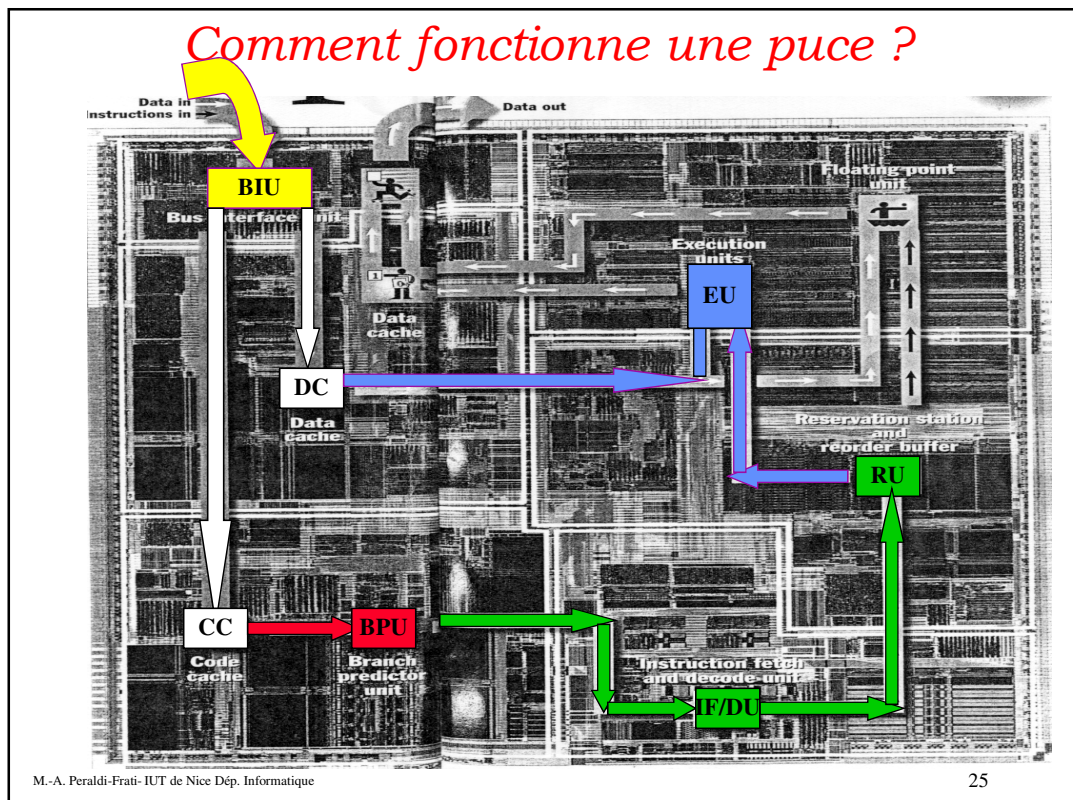
Comment fonctionne une puce ?

- Une puce => carré de silicium constitué de millions de transistors qui manipulent des instructions et des données
- Une puce de Pentium II exécute 500 million instructions /seconde
 - Les informations arrivent de la RAM du PC dans la puce par le bus (*BIU Bus Interface Unit*)
 - L'information va dans un *cache de code ou de donnée (CC,DC)*
 - L'*unité de prédiction* de branchement détermine le chemin optimum pour l'information (BPU)
 - Le *décodeur d'instruction* traduit les instructions en opérations élémentaires (DU)
 - la *station de réservation et* buffer de réordonnancement détermine l'ordre d'exécution le plus efficace (RU)



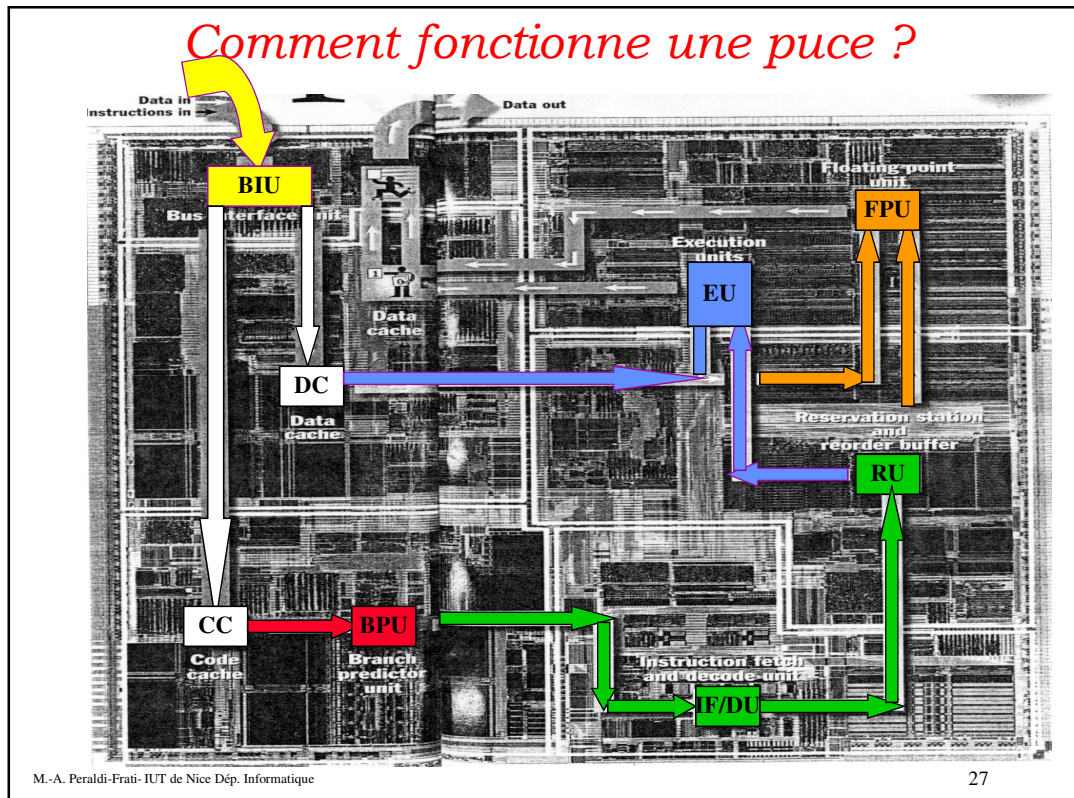
Comment fonctionne une puce ?

- Une puce => carré de silicium constitué de millions de transistors qui manipulent des instructions et des données
- Une puce de Pentium II exécute 500 million instructions /seconde
 - Les informations arrivent de la RAM du PC dans la puce par le bus (BIU Bus Interface Unit)
 - L'information va dans un cache de code ou de donnée (CC,DC)
 - L'unité de prédiction de branchement détermine le chemin optimum pour l'information (BPU)
 - Le décodeur d'instruction traduit les instructions en opération élémentaire (DU)
 - la station de réservation et buffer de réordonnancement détermine l'ordre d'exécution le plus efficace (RU)
 - l'unité d'exécution exécute les opérations et rend les résultats dans le cache de donnée (EU)



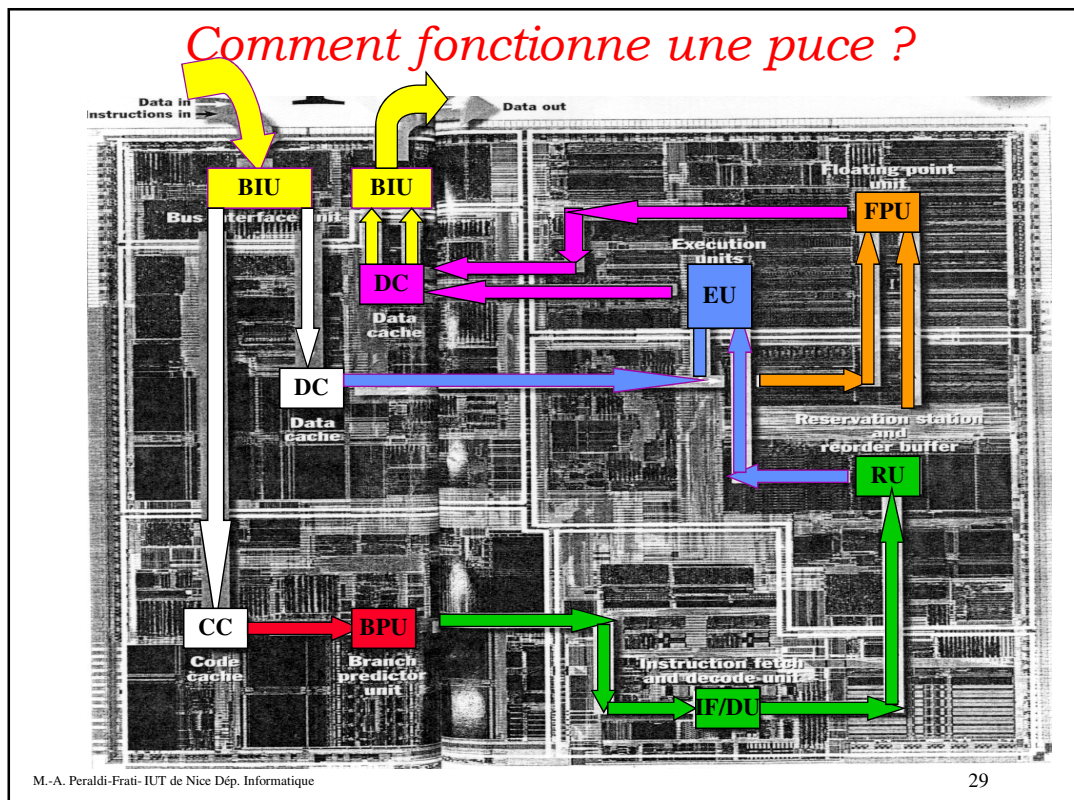
Comment fonctionne une puce ?

- Une puce => carré de silicium constitué de millions de transistors qui manipulent des instructions et des données
- Une puce de Pentium II exécute 500 millions instructions /seconde
 - Les informations arrivent de la RAM du PC dans la puce par le bus (BIU Bus Interface Unit)
 - L'information va dans un cache de code ou de donnée (CC,DC)
 - L'unité de prédiction de branchement détermine le chemin optimum pour l'information (BPU)
 - Le décodeur d'instruction traduit les instructions en opération élémentaire (DU)
 - la station de réservation et buffer de réordonnancement détermine l'ordre d'exécution le plus efficace (RU)
 - l'unité d'exécution exécute les opérations et rend les résultats dans le cache de données (EU)
 - l'unité de virgule flottante (FPU) effectue les opérations arithmétiques



Comment fonctionne une puce ?

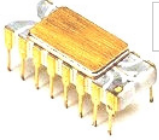
- Une puce => carré de silicium constitué de millions de transistors qui manipulent des instructions et des données
- Une puce de Pentium II exécute 500 million instructions /seconde
 - Les informations arrivent de la RAM du PC dans la puce par le bus (BIU Bus Interface Unit)
 - L'information va dans un cache de code ou de donnée (CC,DC)
 - L'unité de prédiction de branchement détermine le chemin optimum pour l'information (BPU)
 - Le décodeur d'instruction traduit les instructions en opération élémentaire (DU)
 - la station de réservation et buffer de réordonancement détermine l'ordre d'exécution le plus efficace (RU)
 - l'unité d'exécution exécute les opérations et rend les résultats dans le cache de donnée (EU)
 - l'unité de virgule flottante (FPU) effectue les opérations arithmétiques
 - le cache de donnée transfère ses résultats vers l'unité d'interface de bus qui les transmet à la RAM.




Evolution et générations des machines

Génération	Dates	Technologie	Opérations/s
1	1946-57	Tubes à vide	40k
2	1958-64	Transistors	200K
3	1965-71	SSI-MSI	1M
4	1972-77	LSI	10M
5	1978-	VLSI	100M


Exemple de la famille des microprocesseurs **Intel**




4004




8080
Microprocessor
1974



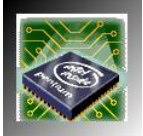
286
Microprocessor
1982




Intel386™
Microprocessor
1985




Intel486™
Microprocessor
1989




Pentium



Pentium II



Pentium III



Pentium 4

M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique 31

Exemple de microprocesseurs

Processeur	Date	nombre de transistors	fréquence quartz	fréquence max (MHz)	bits	taille de la gravure (microns)
4004	nov-71	2300	0,108	0,108	4	?
4040	févr-72	2300	0,747	0,747	4	?
8008	avr-72	3500	0,3	0,3	8	?
8080	avr-74	6000	2	2	8	?
8086	juin-78	29000	5	10	16	?
80286	févr-82	134000	6	12,5	16	?
80386 DX	oct-85	275000	16	33	32	?
80486 DX	avr-89	1200000	25	50	32	?
Pentium P5	mars-93	3100000	60	66	64	1
Pentium	mars-93	3300000	90	120	32	0,6
Pentium pro	oct-95	5500000	150	200	32	0,6
Pentium II	juil-97	7500000	200	450	64	0,35
Pentium III	mars-99	29000000	450	1000	128	0,18
Pentium 4	nov-00	42000000	1400	1500	128	0,13

Evolution des processeurs Motorola

- **6800 en 1974.** Processeur 8 bits développé par Chuck et Charlie Melear.
- **68000 en Septembre 1979.**
 - il contenait 68000 composants. La société avait été retenue au départ, pour équiper le Personal Computer d'IBM mais le choix s'est porté ensuite sur Intel. Le 68000 fût intégré entre autres à des Stations Apollo et Silicon Graphics. On le retrouve bien sûr dans le Macintosh d'Apple.
- **68010 en 1984.** Motorola créa une version optimisée de son 68000. Même si, à vitesse d'horloge égale, peu utilisé...
- **68020 en Juin 1984.** (Macintosh, Commodore Amiga 1200) Processeur 16/32 bits
- **68030 en 1986.** (Macintosh, NeXT Cube, station Unix Hewlett-Packard, Amiga 3000, Atari Falcon 030).
le 68030 est un processeur 16/32 bits qui intègre 300 000 transistors.
- **68040 en 1991** (Macintosh, Stations diverses, Amiga 4000, NeXT).
1,2 millions de transistors, une mémoire cache de 8 Ko et un FPU. Il a aussi la particularité de fonctionner grâce à deux fréquences d'horloge.



Evolution des processeurs Motorola

- **88000: 1988.** Il utilise une technologie RISC (jeu d'instructions réduit) et fonctionnait à sa sortie à la cadence de 20 Mhz (17 Mips) ou 25 Mhz (21 Mips) et 33 Mhz (28 Mips)..
- Le 88000 se compose en fait de deux processeurs: le 88100 (CPU) et le 88200 (MMU et gestion de la mémoire cache).
- L'atout majeur du 88000 était son prix relativement bas par rapport aux concurrents comme le Sparc de Sun ou le R2000.

Apple développe avec IBM et Motorola: le PowerPC ("Power Performance Chip") et utilisant une technologie RISC.

Evolution des Power PC

	601 (32-bit)	603 (32-bit)	604 (32-bit)	740/750 (64-bit)	G4 (64-bit)	G4 rev.3 (64-bit)
Ship Date	1993	1994	1994	1997	1999	2001
Top Speed (MHz)	120	300	350	366	500	867
L1 Cache Instr / Data (Kbyte/Kbyte)	-	16/16	32/32	32/32	32/32	32/32
L2 Cache	-	-	-	256K-1M	256K-1M	256K
L3 Cache	-	-	-	-	-	1M-2M

Dernier né: Le G5 !

Comparaison G4 / G5

Physical specifications

- 58 million transistors
- 130-nanometer, silicon-on-insulator (SOI) process
- Die size: 118 square millimeters

Comparison of PowerPC G4 and PowerPC G5

	PowerPC G4	PowerPC G5
Architecture	32-bit	64-bit
Addressable memory	4 gigabytes	4 terabytes
Maximum clock speed	1.4GHz	2GHz
Frontside bus	Up to 167MHz shared	Up to 1GHz per processor
In-flight instructions	16	215
Floating-point units	One	Two
Integer units	One	Two
Load/Store units	One	Two
L1 data cache	32K	32K
L1 instruction cache	32K	64K
L2 cache	256K	512K
Branch prediction logic	Local	Local/Global/Selector
Process technology	180-nanometer	130-nanometer
Die size	106 square millimeters	118 square millimeters

Performances des machines

- **Constat** : blocs de base des machines sont restés proches de la machine de Von Neumann
- **les technologies ont fortement évoluées**
- **les améliorations des performances résultent des progrès technologiques**
 - Loi de Moore : tous les 3 ans on quadruple le nombre de transistors sur un chip. (Limites atteintes d'ici 30 ans)
 - Augmentation de la taille des mémoires
 - Augmentation de la vitesse des microprocesseurs
- **Progrès non homogènes**
 - Accès aux mémoires toujours limités

Performances des machines

Il faut trouver des solutions toujours + élaborées pour exploiter les progrès techniques.

- **Prévision de branchement**
- **Analyse du flot de données**
- **Exécution spéculative**
- **Amélioration du transfert processeur mémoire**
 - augmentation de la taille des bus
 - amélioration de l'interface des DRAM
 - utilisation des caches
 - augmentation de la bande passante des bus
- **Amélioration des entrées sorties**

Quel est l'avenir ...

- Recherches développées : ordinateurs biochimiques
 - » Passage de l'électronique binaire sur silicium à une électronique biologique chimique moléculaire.
 - » calculateurs biologiques vivants se composent de puces contenant des protéines, des enzymes, des neurones biologiques
- Avantage:
 - » donner des ordres à un système par la pensée, sans aucune action physique
 - » Capacité des réseaux de neurones très supérieure à des calculateurs //
- Inconvénient:
 - » L'inconvénient majeur d'un matériel biologique est sa durée de vie limitée.
 - » Patrimoine biologique => organisme vivant=>apprentissage=> personnalité

Quel est l'avenir ...

- Recherches développées : ordinateurs quantiques
 - » Élément d'information est le QUBITS : mélange de 0 et de 1
 - » Superposition des états
- Avantage:
 - » Multiplication des états => puissance accrue
- Inconvénient:
 - » Superposition quantique est instable et difficile à maintenir ...
 - » A suivre

Les différents types de matériel

- **Ordinateurs Personnels / Systèmes embarqués**
 - Traitement de texte, jeux, poste de travail distant
 - PDA HP, Sony Toshiba
 - Console de jeux
- **Microcontrôleurs Philips, Motorola**
 - Temps réel, contrôle de procédé
- **DSP (Digital Signal Processors)**
 - Texas TMS320C25, Lucent DSP32C, Analogue Devices ADSP2181
 - Traitement du signal audio vidéo
- **Supermini SUN**
 - Serveur de fichiers
- **Mainframe IBM Z series 990**
 - Serveurs, Banque, réservation aérienne
- **SuperComputer Silicon Graphics**
 - Calcul scientifique
 - Traitement d'image



Structure générale d'un ordinateur: Logiciel

- **Software = logiciel , C'est la matière grise de l'ordinateur**
 - **Système d'exploitation:** rôle de Gestion
 - » DOS, Unix, Windows
 - » Ressources hardware
 - » Chargement en mémoire (programmes)
 - » Exécution
 - **Langages**
 - » Langages évolués (C, C++, Java, Pascal, ADA ...)
 - » Langage Machine (Assembleur)

Plan

- Introduction
 - Différentes vues d'un ordinateur
 - Evolution et performance des machines
- **Représentation de l'information**
- Logique, Algèbre de Boole
- Architecture type Von Neumann
 - Structure d'interconnexion : bus
 - La mémoire
 - l'unité centrale
- Exemple d'un processeur Intel Pentium
- Couche d'assemblage
 - langage
 - Modes d'adressage
 - Procédures

Représentation de l'information

- **Information externe**
 - Formats multiples et variés
 - textes, images, sons ...
 - Systèmes d'acquisition des données (micros, capteurs, cartes d'acquisition, scanners)
- **Information interne**
 - Binaire 0101111...
 - Nécessité d'avoir des unités d'échanges : transformation de l'information en binaire

Représentation de l'information

- un système informatique :
 - Ensemble de composants = quelques centaines de milliers de transistors / composants
 - Ils fonctionnent selon deux états logiques notés **0** et **1** (logique binaire)
 - correspond à deux niveaux électriques **0** et + **5 / 3,3 volts**
- Information logique (0,1) représente
 - des chiffres, nombres (entiers, réels)
 - caractères
 - chaîne de caractères
- Autres possibilités de codage (extension du binaire)
 - hexadécimal, Binary Coded Decimal , ASCII ...

Représentation d'un nombre en base b

$$N = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b^1 + a_0 b^0$$

Poids fort (pointing to $a_n b^n$) **Poids faible** (pointing to $a_0 b^0$)

$a_i b^i$ → **Rang de a_i**
↓ ↘
Poids de a_i **base**

$a_i = 0$ ou $1 \Rightarrow$ **bit**

- **Exemple : Base 2**

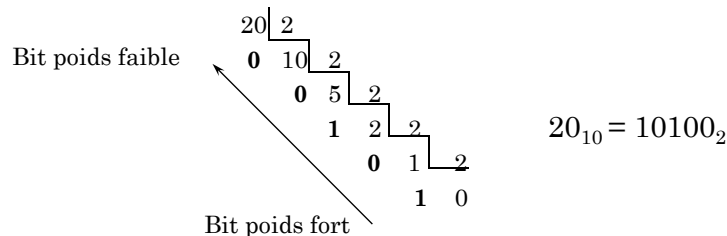
$$10_{10} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

$$10_{10} = 1010_2$$

Indique la base

Passage de la base 10 à la base 2

- **division successives** du nombre par 2 **pour la partie entière**



- **multiplications successives** du nombre par 2 **pour la partie fractionnaire**

$$\begin{array}{rcl} 0,375 & * 2 & = & 0,75 \\ 0,75 & * 2 & = & 1,5 \\ 0,5 & * 2 & = & 1 \end{array} \qquad 0,375_{10} = 0.011_2$$

$$20,375_{10} = 10100.011_2$$

- Remarques

- la représentation en base 2 n'est pas forcément finie (exemple $0,2_{10}$)
- limitation du nombre de bits => *erreur de troncature*

Numération Octale et Hexadécimale

- Octale : 8 symboles 0, 1, 2, 3 ... 7
- Hexadécimale : 16 symboles 0, 1, 2, 3 ... 9, A, B, C, D, E, F
- Passage de la base 10 à la base 8 ou 16
 - divisions successives par 8 ou 16
- Passage de la base 2 à la base 8 ou 16
 - décomposition en groupe de 3 ou 4 bits
 - remplacement de chaque groupe par sa valeur dans la nouvelle base

Exemples : $1011101,011101_2$

$$\text{Base 8} \quad 1 \mid 011 \mid 101,011 \mid 010 \quad = 135,32_8$$

$$\text{Base 16} \quad 101 \mid 1101,0110 \mid 1000 \quad = 5D,68_{16}$$

Représentation des nombres

- Ordinateurs travaillent sur un nombre fixe de **bits**
- La notion d'**octet**
 - un **octet** = 8 bits



- La notion de **mot**
 - un **mot** = 8, 16, 32 64 bits
 - Intel 80386 et 80486 -> 32 bits
 - Pentium 4 -> 128 bits

Exemple en langage C

TYPE	DESCRIPTION	TAILLE
int	entier standard signé	4 octets: - 2^{31} à $2^{31}-1$
unsigned int	entier positif	4 octets: 0 à 2^{32}
short	entier court signé	2 octets: - 2^{15} à $2^{15}-1$
unsigned short	entier court non signé	2 octets: 0 à 2^{16}
char	caractère signé	1 octet : - 2^7 à 2^7-1
unsigned char	caractère non signé	1 octet : 0 à 2^8

Représentation des nombres ...

- Représentation Décimal codé binaire :
Chaque chiffre en base 10 est codé sur 4 bits (un quartet).
- Exemple: $1995_{10} = 0001\ 1001\ 1001\ 0101_{dcb}$
 - Le signe est généralement stocké dans le quartet le moins significatif.
 - Selon les constructeurs
 - + = 0000 et - = 1111
 - + = 1011 et - = 1101 (Correspond aux caractères + et - en ascii)
- Remarques:
 - les représentations de nombres entiers en C_2 se fait sur 16 bits
 - Cette représentation évite les inconvénients mentionnés ci-dessus.

Représentation des données

- Caractères à coder, alphabet + , ! * " %
- Code ASCII (7 bits + 1 bit de parité)
 - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
 - permet 128 combinaisons différentes 2^7
 - utilisation de cette table ...
- Code EBCDIC (8 bits, 256 caractères)
 - **E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode
 - Utilisé principalement par IBM
- Code ANSI
 - **A**merican **N**ational **S**tandard **I**nstitute
 - Utilisés par certains logiciels (Windows)
 - Code ASCII + extensions multilingue alphabets occidentaux

Table des codes Ascii (7 bits Norme ISO 646)

American Standard Code for Information Interchange

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STH	ETH	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	CD2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	spc	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

'G' est codé 47h soit 0100 0111₂

Evolution du code ASCII

- **"iso-latin-1"**, également connu sous le nom de iso-8859-1
 - le huitième bit qui servait pour le contrôle de parité, va être utilisé pour coder plus de caractères.
 - Les codes ASCII de 0 à 7F (127 en décimal) demeurent inchangés,
 - les codes supérieurs (ceux qui ont le bit 7 à 1) représentent quelques symboles supplémentaires, ainsi que les lettres accentuées qui satisfont aux exigences des langues de l'Europe de l'Ouest
- **Unicode**
 - Plus de déclinaison comme dans les codes ISO,
 - Codage sur 2 octets de l'ensemble des signes,
 - Codage contient des informations telles que le sens d'écriture.

Code iso-latin-1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STH	ETH	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	CD2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8	€	□	,	f	"	...	†	‡	^	%	Š	<	œ	□	ž	□
9	□	'	'	"	"	•	-	-	~	™	š	>	œ	□	ž	ÿ
A		ı	¢	£	¥	ı	§	"	©	ª	«	¬	-	@	-	
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ø	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Conclusion sur la représentation de l'information

- Binaire (0, 1)
- Bases 2, 8, 10, 16
 - Passage d'une base à une autre
 - Calculs dans les différentes bases
- Convention de représentation des nombres
 - Classique
 - Complément à 2
 - DCB
- Codage des caractères
 - ASCII, ANSI, EBCDIC (IBM), Code ISO, Unicode

Lien de la représentation de l'information avec l'architecture matérielle

- Information sur la taille des variables 2 octets, 4 octets ...
- Ces informations sont des données
- Transmission et stockage des données doit être efficace
 - Un bus doit être au moins de 32 bits pour les données permet de transférer en un cycle un entier
 - Un bus 64 bits peut transférer deux entiers dans le même temps

Plan

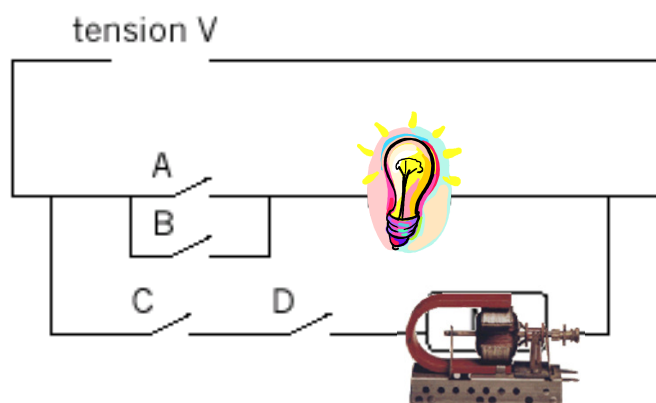
- Introduction
 - Décomposition fonctionnelle/matérielle d'un ordinateur
 - Evolution et performance des machines
- Représentation de l'information
- **Algèbre de Boole**
- Circuits séquentiels/Automates
- Architecture type Von Neumann
 - Structure d'interconnexion
 - La mémoire
 - l'unité centrale
- Exemple d'un processeur Intel Pentium
- Couche d'assemblage
 - langage
 - Modes d'adressage

Logique, Fonctions logiques, Algèbre de Boole

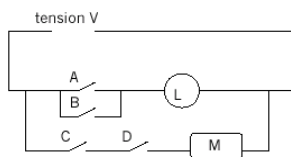
- De nombreuses grandeurs peuvent prendre une infinité de valeurs => vitesse d'une voiture, température ... **Valeurs continues**
- Ces valeurs continues dans les systèmes **analogiques** peuvent être représentées par un nombre fini de **Valeurs discrètes** dans les systèmes **numériques**
- D'autres systèmes travaillent avec des valeurs ne prenant que 2 états : **vrai ou faux**.
 - Interrupteur ouvert ou fermé
 - Lampe allumée ou pas
 - Affirmation vraie ou fausse
 - Grandeurs > ou < à un seuil

Valeurs logiques

Variables logiques : opérateurs de base **ET** et **OU**



Comment exprimer ce problème par des variables logiques ?



Variables logiques : opérateurs de base **ET** et **OU**

- Interrupteurs : Variables A, B, C, D
- Lampe : Variable L
- Moteur : Variable M

Logique Positive : 1=Vrai=Actif 0=Faux=Inactif

OU est noté +

ET est noté x

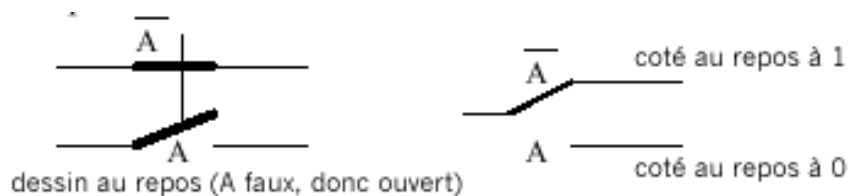
Equations logiques :

$$L = A + B$$

$$M = C \times D = C \cdot D$$

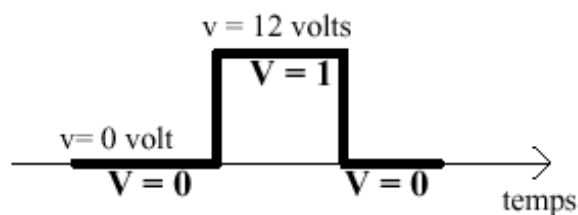
Variables logiques : opérateurs de base **NON**

- Lampe A allumée : $A = 1$
- Lampe A éteinte : $A = 0$ ou $\overline{A} = 1$ (Complément de A.)
- Un interrupteur peut être caractérisé par deux variables A et \overline{A} soit 2 interrupteurs différents pouvant être actionnés en même temps.



Variables logiques et Signal électrique

- Un signal électrique peut prendre deux valeurs de tension:
 - v variant de 0 à 12 Volts
 - V variable logique est associée à v , telle que
 - » $V = 0$ quand la tension est nulle
 - » $V = 1$ quand la tension = 12 Volts



Circuits combinatoires et Séquentiels

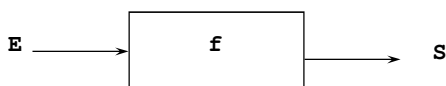
- Élément de l'algèbre de Boole => circuits combinatoires
 - circuits de base des ordinateurs
- Théorie des automates => circuits séquentiels
 - modèle de base pour le fonctionnement des circuits
- Signaux logiques et Analogiques
 - Traitement de l'information
 - » traiter
 - » mémoriser des signaux électriques
 - » transférer

Fonction logique



- Une fonction logique exprime une transformation des entrées d'un circuit pour donner une ou plusieurs sorties.
- **Table de vérité** pour la conception de fonction logique complexe.

Circuits combinatoires



Système combinatoire =>

un ensemble fini d'entrée E

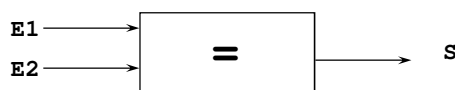
un ensemble fini de sortie S

$$\text{Sortie} = f(\text{Entrée})$$

Exemple :

- portes ET/OU
- Aiguillage d'info (multiplexeur demultiplexeur)

Algèbre de Boole



E1 / E2	0	1
0	1	0
1	0	1

Tables de vérité

Entrées			Sortie
A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

- 3 entrées => $2^3 = 8$ combinaisons possibles
- Ecriture dans l'ordre des entiers naturels
- Fonction logique :

Quand la fonction vaut-elle 1 ?

$$S = \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.\overline{C}$$

Tables de vérité à plusieurs sorties

Entrées			Sorties	
A	B	C	S1	S2
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

- Fonctions logiques :

$$S1 = \overline{A}.B.C + A.\overline{B}.\overline{C}$$

$$S2 = \overline{A}.\overline{B}.\overline{C} + A.B.C$$

*Tables de vérité de
ET, OU et NOT*

ET (AND)

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1

OU (OR)

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NON (NOT)

X	\overline{X}
0	1
1	0

*Tables de vérité de
NON ET, NON OU et OU Exclusif*

NON ET (NAND)

X	Y	$\overline{X.Y}$
0	0	1
0	1	1
1	0	1
1	1	0

NON OU (NOR)

X	Y	$\overline{X+Y}$
0	0	1
0	1	0
1	0	0
1	1	0

OU Exclusif (XOR)

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

$$X \oplus Y = \overline{X}.Y + X.\overline{Y}$$

OU Exclusif détecte la différence de deux variables.

Simplification des fonctions logiques

- Exemple : Deux voyants A et B. On veut déclencher une alarme quand **au moins un des deux** est allumé
- 1ère Solution : $\text{Alarme} = \bar{A}.B + A.\bar{B} + A.B$
- 2ème Solution : $\text{Alarme} = A + B$

2ème Solution plus simple

Simplification des fonctions logiques

- Trois techniques de simplification :
 - Le **raisonnement** comme précédemment
 - L'**algèbre de Boole** : Algèbre des variables binaires et booléennes.
 - La méthode graphique par les **tableaux de Karnaugh**.

Algèbre de Boole

- Commutativité : $X.Y = Y.X$ $X+Y = Y+X$
- Associativité : $X.(Y.Z) = (X.Y).Z$ $X+(Y+Z)=(X+Y)+Z$
- Distributivité : $X.(Y+Z) = X.Y+X.Z$ $X+Y.Z=(X+Y).(X+Z)$

- Identité : $1.X = X$ $0+X = X$
- Nullité : $0.X = 0$ $1+X = 1$
- Idempotence $X.X = X$ $X+X = X$
- Inversion $X.\bar{X} = 0$ $X+\bar{X} = 1$

$$\overline{X+\bar{X}}.Y = X+Y \text{ à démontrer}$$

- Simplifications très utiles :

$$\overline{\bar{X}+X}.Y = \bar{X}+Y \text{ à démontrer}$$

Algèbre de Boole

- Loi de De Morgan :

$$\overline{X+Y} = \bar{X}.\bar{Y}$$
$$\overline{X.Y} = \bar{X}+\bar{Y}$$

- Algèbre du OU Exclusif : $X \oplus Y = \bar{X}.Y + X.\bar{Y}$

$$\overline{X \oplus Y} = \overline{\bar{X}.Y + X.\bar{Y}}$$
$$= \overline{\bar{X}.Y} . \overline{X.\bar{Y}} \text{ à démontrer}$$

Tableau de Karnaugh

- Obtenir l'expression logique la plus simple d'une fonction F
- Trouver des termes communs pour un système afin de limiter le nombre de circuits
- De tenir compte de combinaisons de variables d'entrées jamais utilisées en mettant 0 ou 1 en sortie.
- Pratique pour 4 variables d'entrées, possible pour 5 et 6

Principe : Simplification par adjacence

$$A.B.C + A.B.\overline{C} = A.B(C+\overline{C}) = A.B$$

- En choisissant un code de Gray* pour coder les entrées on retrouve les adjacences sur des cases côte à côte.

*Distance de 1 entre deux mots de code consécutifs

Tableau de Karnaugh

- Tableau à deux variables d'entrées

$$S = \overline{A}.B + AB$$

$$S = B$$

	B	0	1
A			
0		0	1
1		0	1

- Tableau à trois variables d'entrées

$$S = CA + \overline{B}$$

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

Tableau de Karnaugh

- Tableau à quatre variables d'entrées

$$S = C.\bar{A}+D.\bar{C}.\bar{B}$$

		D			
		0	0	1	1
A	B	C			
		0	1	1	0
0	0		1	1	1
0	1		1	1	
1	1				
1	0				1

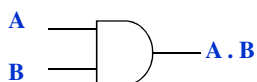
- Etats non utilisés** : états qui existent en théorie mais qui en pratique ne peuvent jamais apparaître pour des raisons physiques ou mécaniques.

$$S = \bar{A}.\bar{D}+B.C$$

		D			
		0	0	1	1
A	B	C			
		0	1	1	0
0	0	1	1		
0	1	1	-	-	-
1	1		1	1	
1	0				77

Circuits logiques de base

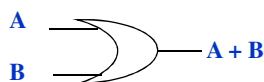
- Circuits standards :
 - Matérialisent les opérateurs de bases
 - 2 3 4 ou même d'avantage d'entrées
 - Fonctionnent sur des signaux électriques
 - Egalement appelés **Portes logiques**



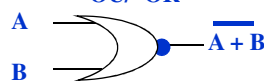
ET/ AND



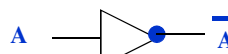
Non ET/ NAND



OU/ OR



Non OU/ NOR



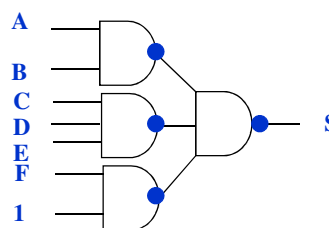
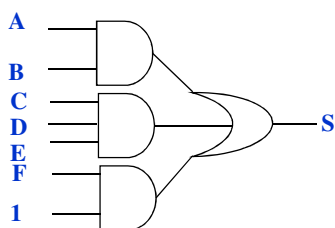
NOT

Circuits logiques de base

- Circuits NAND ou NOR sont plus rapides et moins encombrants qu'un ET ou OU. On utilise dans ce cas De Morgan pour transformer les circuits :

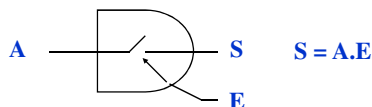
• $S = AB + CDE + F$

$S = \overline{\overline{AB} \cdot \overline{CDE} \cdot \overline{F}}$



Circuits logiques

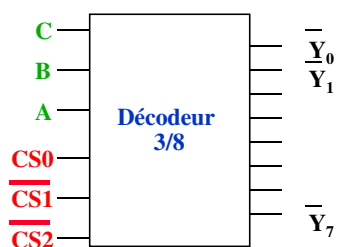
- **Sorties 3 états** : 0 ou 1 (Si E =1) ou ouvert (Si E =0)
- Le signal E est le signal de **validation de la sortie S**



- Les sorties des circuits logiques nommés **mémoires** sont toutes de type 3 états
- Par l'intermédiaire de circuits nommés **BUS** le Microprocesseur peut accéder aux contenus de chaque circuit mémoire

Exemple de circuits combinatoires

- Codeurs décodeurs n vers 2^n :
 - Commander un organe différent parmi 2^n au moyen de n bits.
 - Les lignes Cs_i permettent de valider le circuit. Ici pour que le circuit fonctionne ces lignes doivent être à vrai cad à 100.
 - Ces lignes permettent des montage plus complexes à plusieurs décodeurs.



$$Y_0 = \overline{C.B.A} \quad Y_0 = \overline{C.B.A} . \overline{CS0} . \overline{CS1} . \overline{CS2}$$

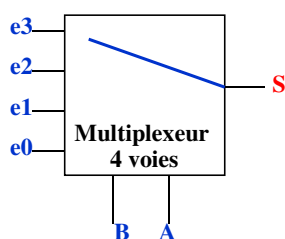
$$Y_1 = \overline{C.B.A}$$

$$\dots$$

$$Y_7 = C.B.A$$

Exemple de circuits combinatoires

- Multiplexeur 4 voies



	B	0	1
A			
0		e0	e2
1		e1	e3

$$S = e0.\overline{B}.\overline{A} + e1.\overline{B}.A + e2.B.\overline{A} + e3.B.A$$

Plan

- Introduction
 - Décomposition fonctionnelle/matérielle d'un ordinateur
 - Evolution et performance des machines
- Représentation de l'information
- Algèbre de Boole
- **Circuits séquentiels/Automates**
- Architecture type Von Neumann
 - Structure d'interconnexion
 - La mémoire
 - l'unité centrale
- Exemple d'un processeur Intel Pentium
- Couche d'assemblage
 - langage
 - Modes d'adressage

Circuits séquentiels

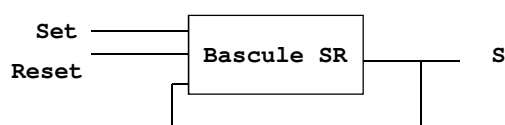
Système séquentiel => $Sortie = f(entrée; \text{état interne})$

- un ensemble fini d'entrées E
- un ensemble fini d'états Q
- une fonction de transitions $G : Q \times E \rightarrow Q$
- un ensemble fini de sorties S
- un état initial Q_0
- une fonction de sortie Mealy $F : Q \times E \rightarrow S$
- Moore $F' : Q \rightarrow S$

Exemple :

- Bascule SR
- Fonction de décalage
- Fonction de comptage...

Théorie des automates



Circuits Séquentiels

- **Fonctionnement :**

Les circuits séquentiels sont conçus à partir de :
 circuits combinatoires (algèbre de Boole)
 + Etat interne

- **Utilisation :**

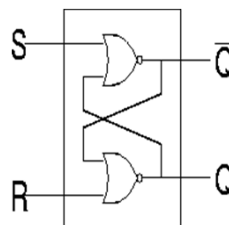
- mémorisation de l'information
- structure de contrôle
- unité de commande
- support théorique pour le développement (algo, langages, techniques de compilation)
- modèle d'expression des systèmes d'état.
- ...

Exemple de circuit séquentiel

- **Bascule RS :**

- 2 entrées : Set et Reset
- 2 variables en sortie Q et \bar{Q}
- La sortie d'une bascule dépend de ses entrées et de son état interne (la valeur antérieure de la sortie).

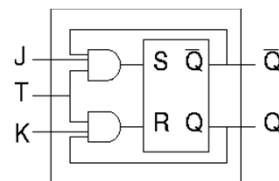
R	S	Qt	Qt+1
0	0	X	X
0	1	X	1
1	0	X	0
1	1	Interdit	Interdit



Exemple de circuit séquentiel

- Bascule JK
- Idem bascule RS
- Indéterminisme levé

J	K	Qt	Qt+1
0	0	X	X
0	1	X	1
1	0	X	0
1	1	X	\bar{X}



Utilisation de ces bascules

- Compteur modulo
- Mémoires

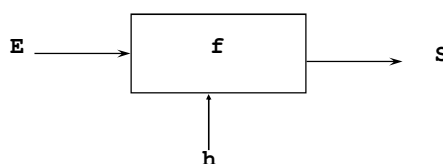
Circuits séquentiels

- Modèles pour la représentation de ces circuits
 - Machine de MOORE
 - » $q = G(q, e)$
 - » $s = F(q)$
 - » la sortie est liée à l'état. La durée de la sortie est égale au temps resté dans l'état
 - Machine de MEALY
 - » $q = G(q, e)$
 - » $s = F(q, e)$
 - » la sortie est liée à la transition. La durée de la sortie est égale au temps que dure l'entrée.
- Modèles asynchrones :
 - le temps n'intervient pas explicitement,
 - problèmes de comportement (stabilité des entrées)

Circuits séquentiels Synchrones

- **Ajout d'une horloge => Circuit séquentiel synchrone**
 - déclenchement des signaux
 - chronologie des actions
 - Exemple : séquenceur.
- Automate d'état fini synchrone

$Sortie = f(entrée; état interne; horloge)$



Circuit séquentiels synchrones

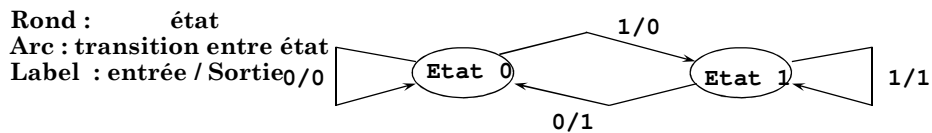
- **Stimulus** : entrée E à l'instant t $E(t)$
- **Instant d'observation** : t, t+1, t+2, ...
- **Fonction de transition** :
 - » **Machine de MEALY**
q à l'instant t+1 $q(t+1) = G[q(t), e(t)]$
s à l'instant t+1 $s(t+1) = F[q(t), e(t)]$
 - » **Machine de MOORE**
q à l'instant t+1 $q(t+1) = G[q(t), e(t)]$
s à l'instant t+1 $s(t+1) = F[q(t)]$

Description d'un circuit séquentiel

- Mémoire d'une position binaire
 - une entrée **e** 0/1
 - deux états **q** 0/1
 - une sortie **s** 0/1
- l'**état** de l'automate $q(t+1)$ dépend de $e(t)$
 $e(t) = 0 \quad \Rightarrow \quad q(t+1) = 0$
 $e(t) = 1 \quad \Rightarrow \quad q(t+1) = 1$
- la **sortie** de l'automate $s(t+1)$ dépend de $q(t)$
 $q(t) = 0 \quad \Rightarrow \quad s(t+1) = 0$
 $q(t) = 1 \quad \Rightarrow \quad s(t+1) = 1$

Description d'un circuit séquentiel

- **Graphe** : Exemple de machine de Mealy



- **Table** :

Ligne : état courant
 Colonne : entrée
 Cellule : état suivant / Sortie

	0	1
Etat 0	Etat0/0	Etat1/0
Etat 1	Etat0/1	Etat1/1

- **Textuel (équations booléennes)**

- si (Etat0 et 0) alors (0,Etat0)
- si (Etat0 et 1) alors (0,Etat1)
- Si (Etat1 et 0) alors (1,Etat0)
- si (Etat1 et 1) alors (1,Etat1)

Mise en œuvre d'un automate en C

```

int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf(" >1\n");}
.....
void main() {
char Entree;
Etat=0; // état initial
do {
    printf(" Etat=%d\n",Etat);
    printf(" Commande==>");
    Commande=getche();
    switch(Etat) {
        case 0 :
            switch (Entree) {
                case '0': Sortie0_0(); Etat=0; break;
                case '1': Sortie0_1(); Etat=1; break;
            }break;
        case 1 :
            switch (Entree) {
                case '0': Sortie1_0(); Etat=0; break;
                case '1': Sortie1_1(); Etat=1; break;
            }
    }
} while (1);
}/* end main */
    
```

Mise en œuvre d'un automate en C

```
int Etat ;  
void Sortie0_0() {printf("0\n");}  
void Sortie1_1() {printf( »1\n");}  
.....
```

Mise en œuvre d'un automate en C

```
int Etat ;  
void Sortie0_0() {printf("0\n");}  
void Sortie1_1() {printf( »1\n");}  
.....  
void main() {  
char Entree;  
Etat=0 ; // état initial  
do {  
    ...
```

```
    }while (Etat !=3|| Etat !=4 );  
}/* end main */
```

Mise en œuvre d'un automate en C

```
int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf( »1\n");}
.....
void main() {
char Entree;
Etat=0 ; // état initial
do {
    printf(" Etat=%d\n",Etat);
    printf(" Commande==>");
    Entree=getche();

}while (1);
}/* end main */
```

Mise en œuvre d'un automate en C

```
int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf( »1\n");}
.....
void main() {
char Entree;
Etat=0 ; // état initial
do {
    printf(" Etat=%d\n",Etat);
    printf(" Commande==>");
    Commande=getche();
    switch(Etat) {
        case 0 :

        case 1 :

    }
}while (1);
}/* end main */
```

Mise en œuvre d'un automate en C

```
int Etat ;
void Sortie0_0() {printf("0\n");}
void Sortie1_1() {printf(" >1\n");}
.....
void main() {
char Entree;
Etat=0 ; // état initial
do {
    printf(" Etat=%d\n",Etat);
    printf(" Commande==>");
    Commande=getche();
    switch(Etat) {
        case 0 :
            switch (Entree) {
                case '0': Sortie0_0(); Etat=0; break;
                case '1': Sortie0_1(); Etat=1; break;
            }break;
        case 1 :
            switch (Entree) {
                case '0': Sortie1_0(); Etat=0; break;
                case '1': Sortie1_1(); Etat=1; break;
            }
    }
}while (1);
}/* end main */
```

M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique

99

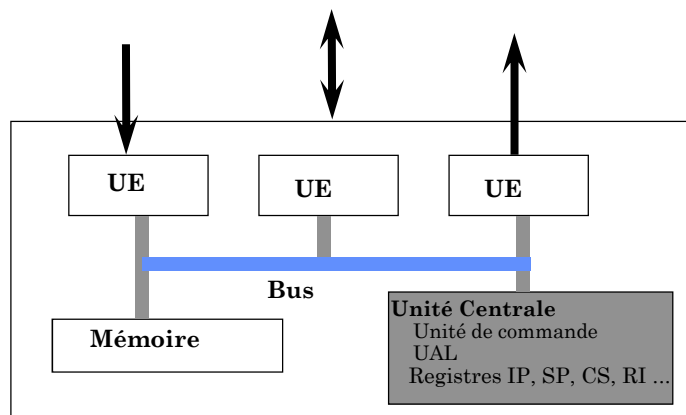
Plan

- Introduction
 - Décomposition fonctionnelle/matérielle d'un ordinateur
 - Evolution et performance des machines
- Représentation de l'information
- Algèbre de Boole
- Circuits séquentiels/Automates
- **Architecture type Von Neumann**
 - Structure d'interconnexion
 - La mémoire
 - l'unité centrale
- Exemple d'un processeur Intel Pentium
- Couche d'assemblage
 - langage
 - Modes d'adressage

M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique

100

Les Bus

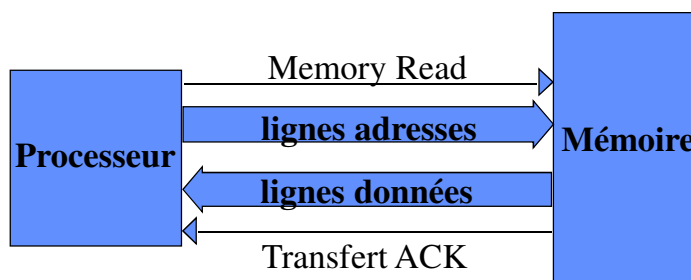


Le bus

- Ensemble de “fils” connectant des unités fonctionnelles au sein d’un ordinateur
- Bus interne CPU \Leftrightarrow cache [300 bits - Pentium pro]
- Bus interne à une machine [*lignes dédiées*]
 - lignes adresses [16, 32, 48 bits]
 - lignes données [8, 16, 32 ou 64 bits]
 - lignes pour signaux de contrôle + logique
- Bus externe [*lignes multiplexées*]
 - nappe + logique
 - Arbitrage : centralisé/décentralisé ; Synchrone/non

Schéma de fonctionnement du bus

- Connexion entre le processeur et la mémoire
 - exemple : lecture d'un mot de la mémoire

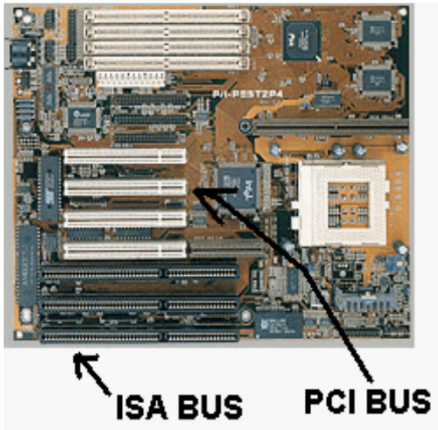


Memory Read : le processeur signale qu'il a placé l'adresse sur la ligne
Transfert ACK : la mémoire répond que les données sont disponibles

Terminologie des bus d'un PC

- Bus local : ISA et/ou PCI
 - Industry Standard Architecture, adressage 16 bits (64 ko), 8 MHz
 - Peripheral Component Interconnect (plus récent), 33 MHz
 - » Vitesses "carte mère" : 66, 75, 83, 100 (133, 200) MHz
- Bus externe
 - IDE : Integrated Drive Electronics
 - » connexion carte mère ↔ contrôleur disque
 - SCSI : Small Computer System Interface
 - » 7/14 périphériques, 8/16 bits, 10 Mb/s.

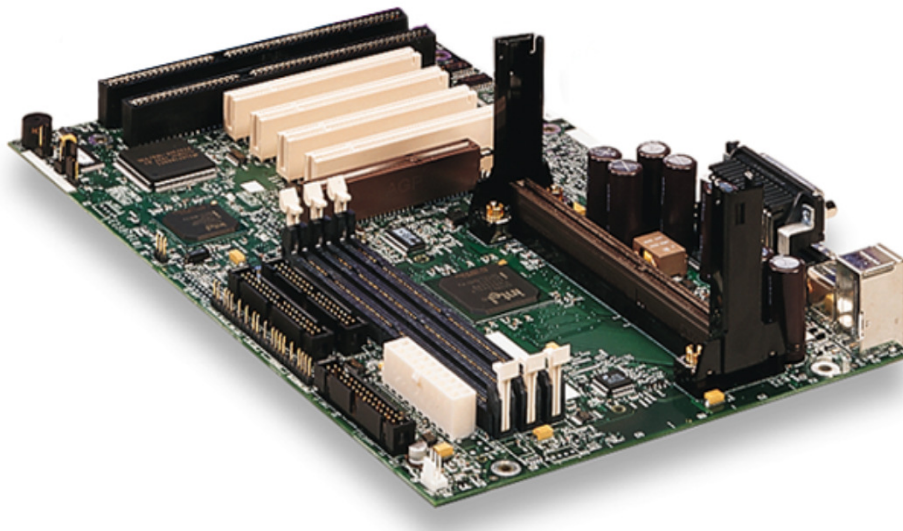
Carte mère et Bus PC



Ex : carte Pentium

- Format standard
 - Classique (AT), ATX
- Supporte :
 - processeur (ou carte fille Slot 1)
 - mémoire (RAM, cache, BIOS)
 - » SIMM, DIMM
 - “chipset” (gestion logique bus)
 - bus ISA et/ou PCI
 - peut inclure un contrôleur SCSI
 - cartes d’extention
 - connecteurs divers périphériques, alimentation

Carte mère Pentium Pro



Structure d'un bus

- Voies de communication entre 2 ou plusieurs équipements
- Médium de communication partagé
 - signal émis par un équipement disponible en réception par équipements connectés
 - Une seule émission à la fois
 - transmissions en //
- Structure de bus
 - 50 à 100 lignes
 - lignes de données
 - lignes d'adresse
 - lignes de contrôle
- Protocole d'échange
 - Envoi d'une donnée
 - Récupération d'une donnée

Caractéristiques d'un bus

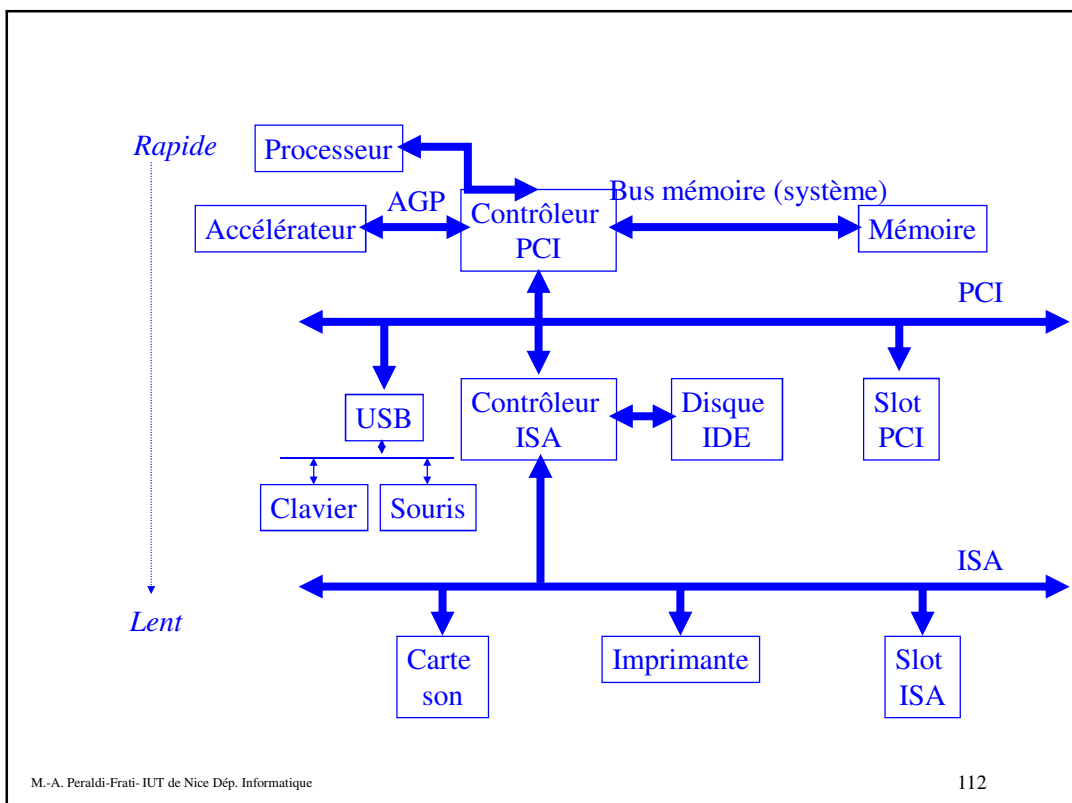
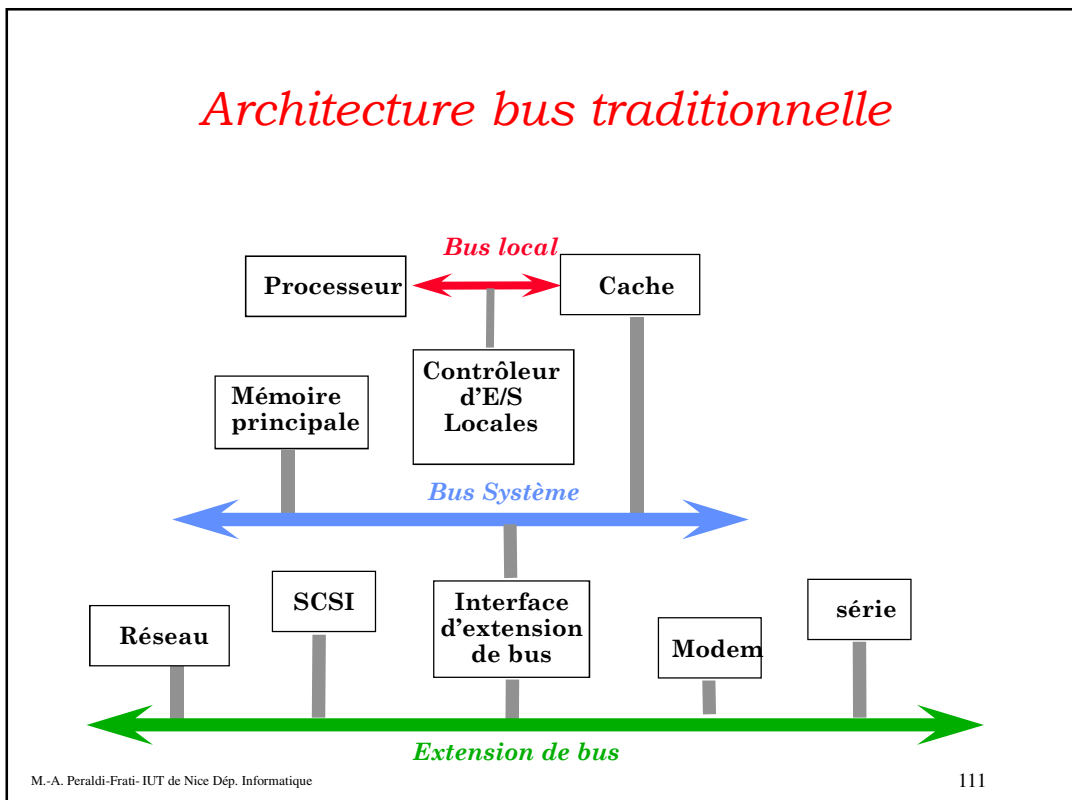
- Type de bus
 - dédié
 - multiplexé
- Méthode d'arbitrage
 - Solution centralisée par *arbitre de bus*
 - Chaque module contient sa logique d'accès : Solution distribuée
- Caractéristiques temporelles
 - Synchrones
 - Asynchrones
- Largeur de bus
 - largeur du bus de données => impact sur performances
 - Largeur du bus d'adresse => capacité d'adressage
- Types de transfert de données
 - multiplexée
 - non multiplexée

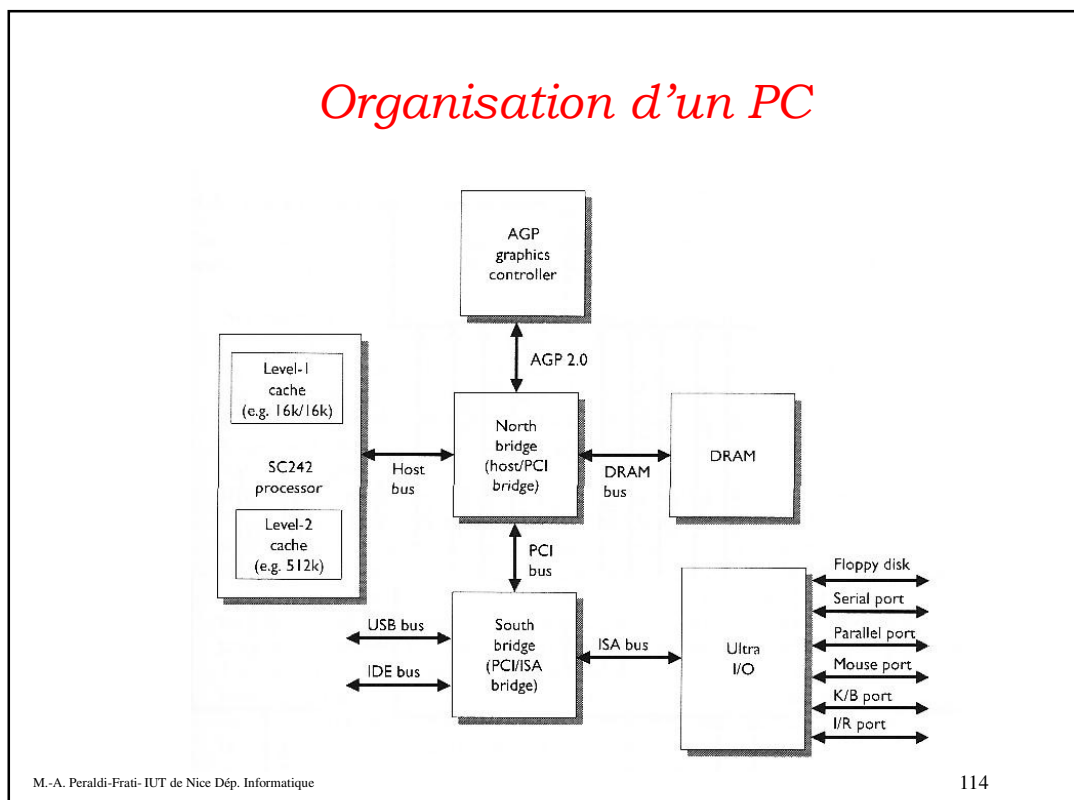
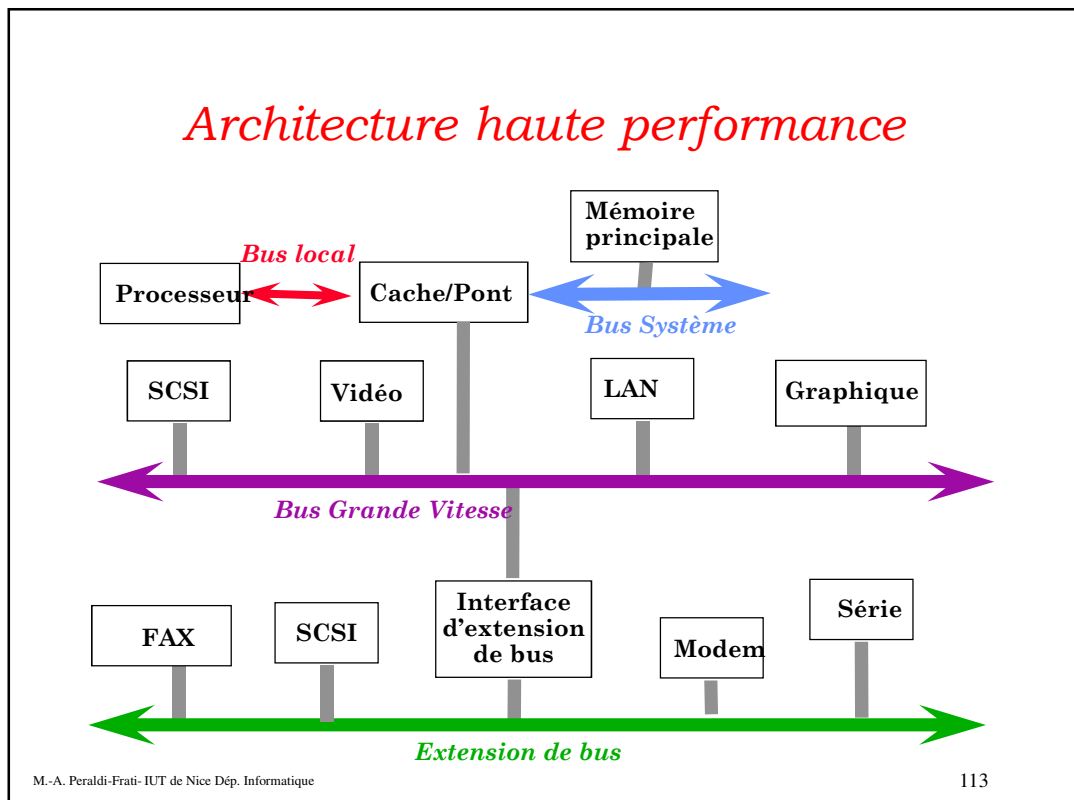
Bus du processeur Pentium III

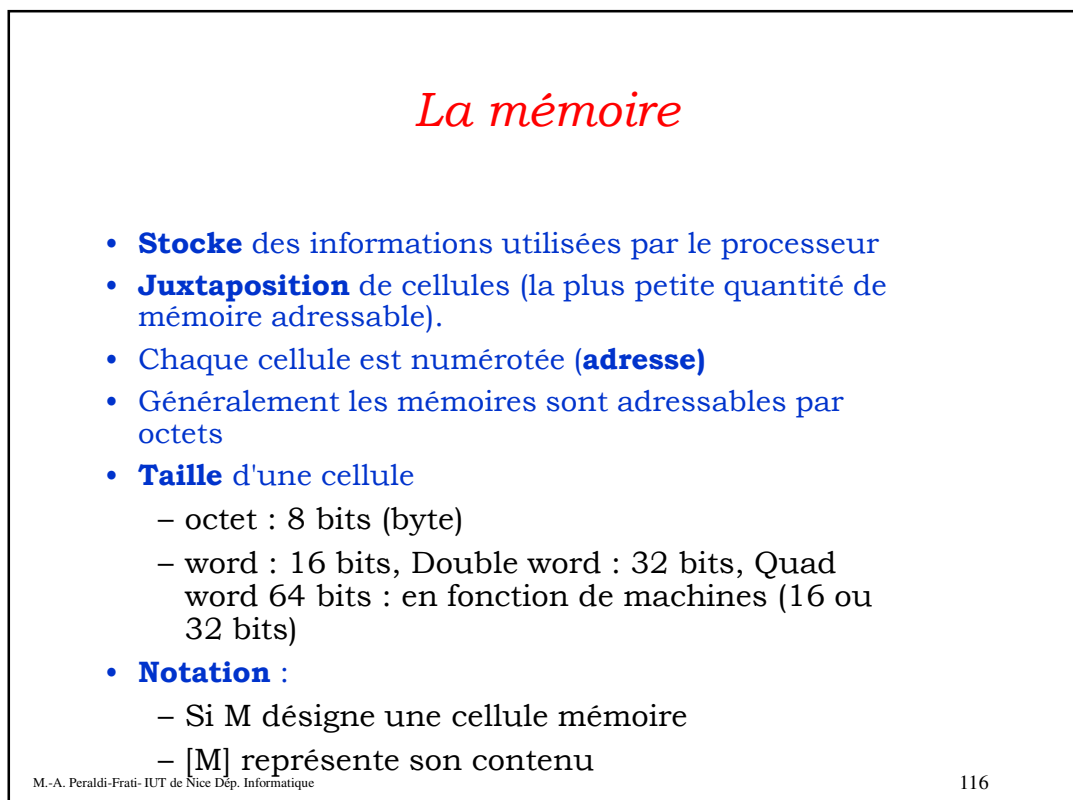
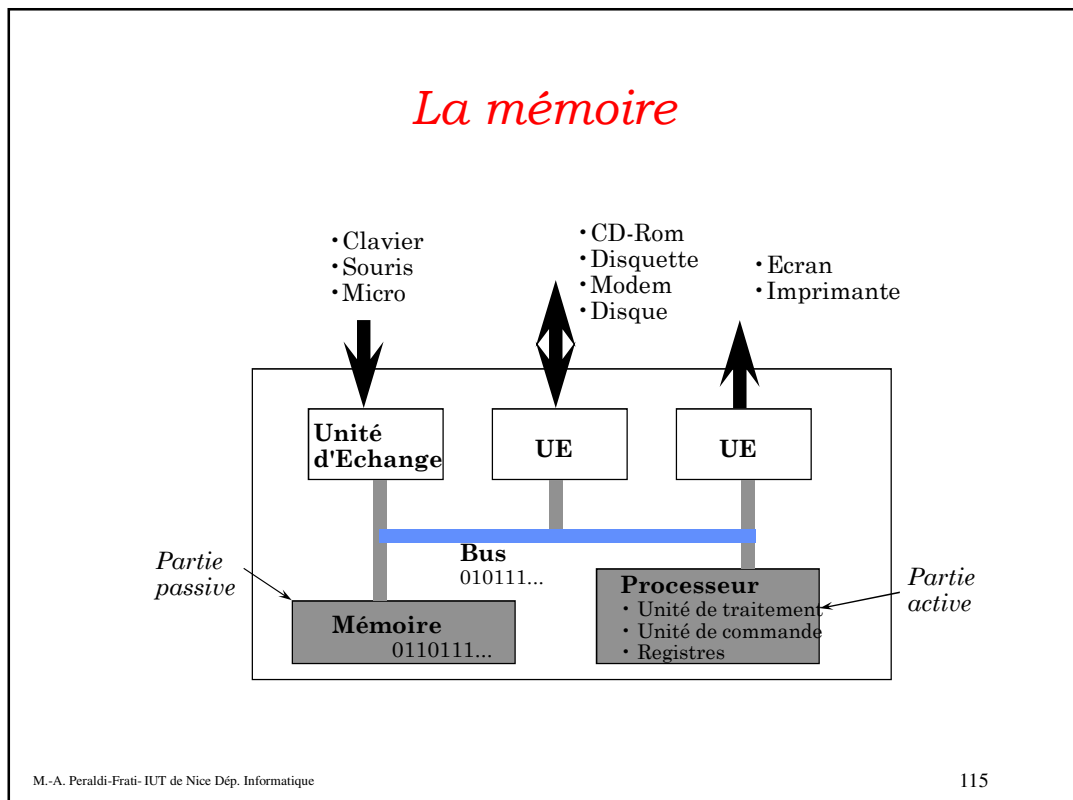
- Bus de données
 - 64 bits (8 chemins de 8 bits)
 - Taux de transfert de 1Go/s à 133 Mhz
 - Espace mémoire accessible de 64 Go
- Bus d'Adresse
 - adresse 32 bits
 - Broche d'adresse haute A31 à A3 et broches de sélection d'octets BE7-BE0
- Vérification de la parité
 - valable sur les bus d'adresse et de donnée. broche DP7 à DP0
- Bus de contrôle
 - Signaux qui déterminent et imposent le cycle du bus du microprocesseur
 - HIT, HITM, HLDA ...

Hiérarchie de bus

- Problème :
 - nombre d'équipements augmente => performance diminuent
 - le bus est un goulet d'étranglement
- Solution :
 - hiérarchie de bus
- Architecture traditionnelle
 - **Bus local** de la mémoire cache vers le processeur
 - Contrôleur de cache connecte bus local/**bus système**
 - **l'extension de bus** relié au bus système par une interface d'extension.
- Architecture haute performance
 - **Bus haute performance** intégré dans le système

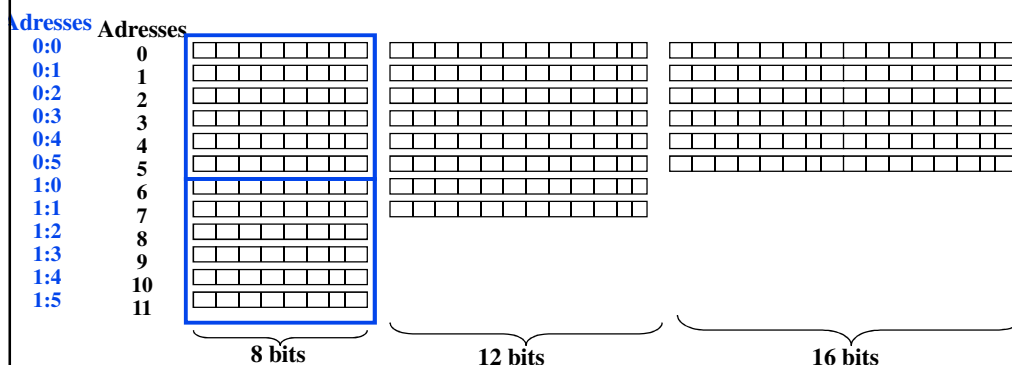






Organisation d'une mémoire

- Mémoire de 12 octets soit 96 bits :
 - Organisation en cellules élémentaires (8bits), 12 bits ou 16 bits :



Caractéristiques des mémoires

- **Capacité** : quantité d'informations qu'elle peut stocker
 - kilo-octet -> Ko = 1024 octets
 - méga-octet -> Mo = 1024 Ko
 - giga-octet -> Go = 1024 Mo
 - téra-octet -> To = 1024 Go
- **Exemple**: Une mémoire centrale de 1 Méga mots de 16 bits
 - 1 x 1024 K-mots
 - 1 x 1024 x1024 mots
 - 1 x 1024 x1024 x 2 octets
 - 1 x 1024 x1024 x 2 x8 bits

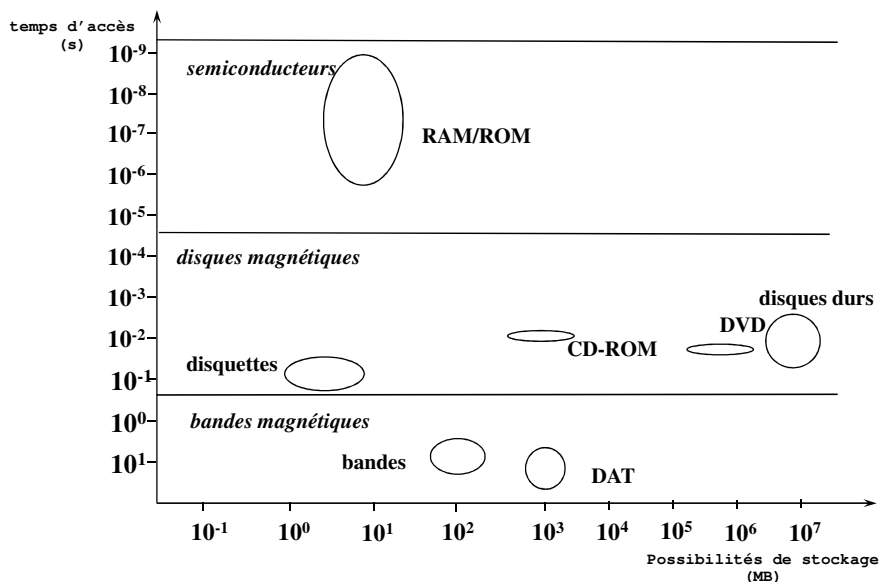
Caractéristiques des mémoires

- **Volatilité**
 - laps de temps durant lequel elle maintient les informations
 - Alimentation de ces mémoires
 - *mémoire de travail de l'ordinateur, mémoire sur support magnétique*
- **Temps d'accès**
 - temps pour accéder à l'information
 - de l'ordre de la nano-seconde ($70 \cdot 10^{-9}$) pour les mémoires actuelles
 - de l'ordre de la milli-seconde pour les supports magnétiques
- **Type d'accès**
 - accès direct à l'information, accès aux **mots-mémoire** par leur **adresse**.
 - accès séquentiel

Caractéristique des mémoires

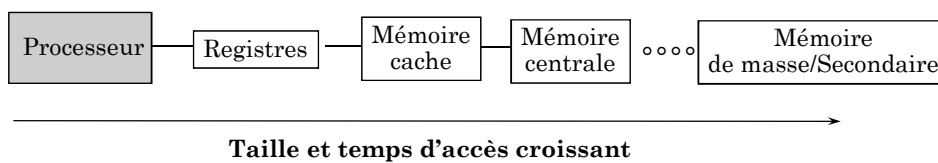
- **mémoire vive: RAM**
 - Mémoire volatile
 - Cycle de lecture/écriture (@ bus d'adresse, V bus de donnée, CS, R/W)
 - Mémoire vive statique -> SRAM
 - Mémoire vive dynamique -> DRAM
 - Mémoire vive dynamique Synchrones -> SDRAM
- **mémoire morte: ROM, PROM, EPROM, EEPROM**
 - A lecture seule,
 - Conserve l'information
 - ROM : information stockée de manière définitive,
 - PROM : Programmable par l'utilisateur
 - EPROM -EEPROM : Programmable + effacement par UV ou électriquement

Différents types de mémoires



Hiérarchie des mémoires

- Machine idéale
 - taille mémoire illimitée
 - temps d'attente nul
- Problème de coût => Hiérarchie de mémoires
- Le processeur va chercher dans les différentes mémoires
- L'optimisation est faite par le gestionnaire de mémoire



Mémoires rapides: registres

- Mémoire **locale et privée** du processeur
- Information temporaire
- Capacité limitée (1 à 128bits)
- Visibles ou invisibles (accessibles par l'utilisateur ou uniquement par le processeur).
- Exemple des processeurs Pentium :
- *Registres Généraux sur 32 bits* : **EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP**
- *Registre pointeur d'instruction* : **EIP**
- *Registre de segment* : **CS, DS, SS, ES, FS, GS**
- *Registre d'état* : **EFLAGS**

Mémoires rapides registres

- **Registres généraux**
 - Accumulateurs pour opérations arithmétiques
 - De taille fixe -> pentium 32 bits, 68000 32 bits
 - **EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP**
 - Adressables sur 16 bits ou sur 8 bits :
 - » Exemple : EAX => AX => AH partie haute (high) AL partie basse (low)
 - Registres relatifs aux zones mémoires programmeur
 - » zone de code => EIP
 - » zone de données => EAX, EBX, ECX, EDX, ESI, EDI
 - » zone de pile => EBP, ESP

Mémoires rapides registres

- Registres de **segment**
 - Registres utilisés pour pointer le début des zones mémoires
 - » CS Code Segment
 - » DS Data Segment
 - » SS Stack Segment
 - » ES, FS, GS Zones mémoires additionnelles
- Registre **Pointeur d'instruction**
 - Contient l'adresse de la prochaine instruction à exécuter

Mémoires rapides registres

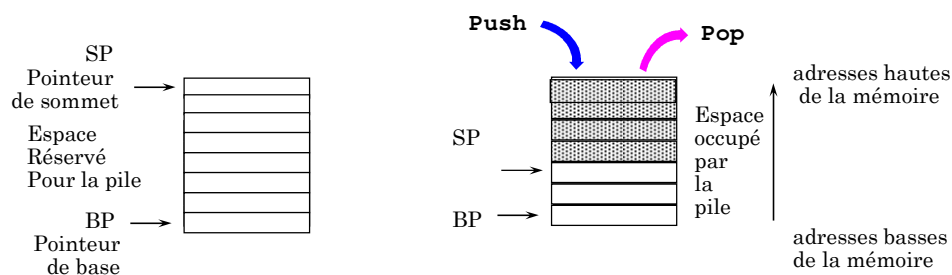
- Registre **d'état**
 - Contrôle et indique des opérations
 - » Indicateurs système
 - » Indicateurs application

31-22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	I	V	V	A	V	R	0	N	I	I	O	D	I	T	S	Z	0	A	0	P	1	C
	D	I	I	C	M	F		T	O	O	F	F	F	F	F	F		F		F	F	F

Exemple : Somme de deux nombres a et b
 a=4294967295 = 0xffffffff
 b=1=0x00000001
 Résultat 0x00000000
 Flags : O=0 D=0 I=1 S=0 **Z=1 A=1 P=1 C=1**

Mémoires rapides registres

- Registres de gestion de la pile
 - zone mémoire spécialisée
 - mémorise les adresses d'appel et retour de sous programmes



- On **réserve** par exemple 256 octets pour la pile
- **Au départ Pile vide** => SP=00FF et BP= 0000
- LIFO (Last In First Out)
- Supposons des éléments de taille 2 octets
- **ajouter un élément** Push(X) $SP \leftarrow SP - 2$
- **retirer un élément** X := Pop $SP \leftarrow SP + 2$

La mémoire cache

- Mémoire située sur la carte mère => accès direct au processeur
- Idée principale du cache: diminuer le temps d'accès à certaines données en les ramenant de la mémoire centrale dans le cache.
- Gestion du cache se fait par microprogramme

La mémoire cache

- Le préchargeur d'instruction : un petit cache d'anticipation
- Un **cache externe**
 - réduction des accès à la mémoire principale à des accès sans états d'attente
 - les accès mémoires sont encore limités par la vitesse du bus
- Un **cache interne** unifié instructions /données
 - Cache sur la puce du processeur
 - Occupation du bus
 - Compétition interne instructions/données
- Des caches séparés pour les données et les instructions

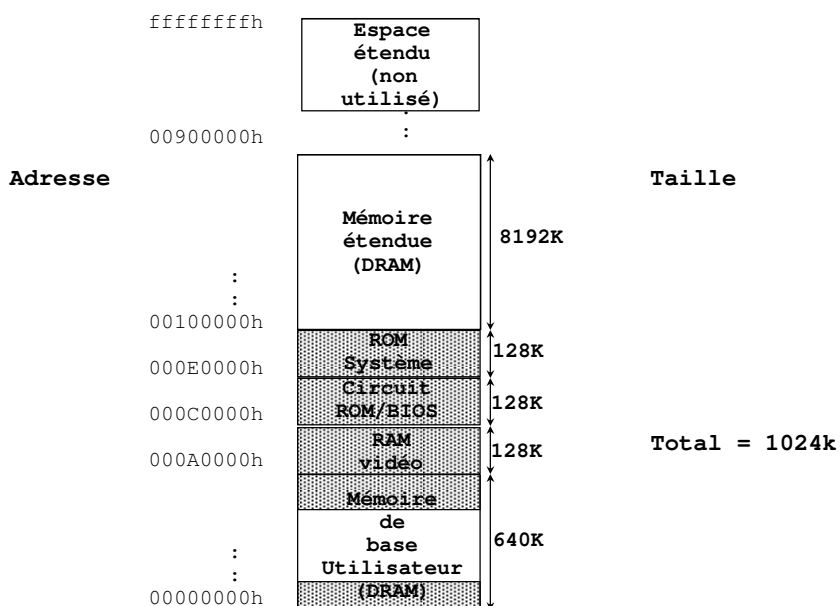
La mémoire centrale

- Elle mémorise les programmes utilisateurs, le système d'exploitation
- Sa taille
 - les 16,32, 64.. Mega d'un PC
- la mémoire centrale est située sur une carte mémoire
- Elle a une organisation standard
- Elle communique avec le processeur (carte mère) via des bus (de données, d'adresse)
- Représentation d'une variable en mémoire
 - Sur les intels LITTLE ENDIAN
 - Sur les motorola BIG ENDIAN

Organisation de la mémoire centrale

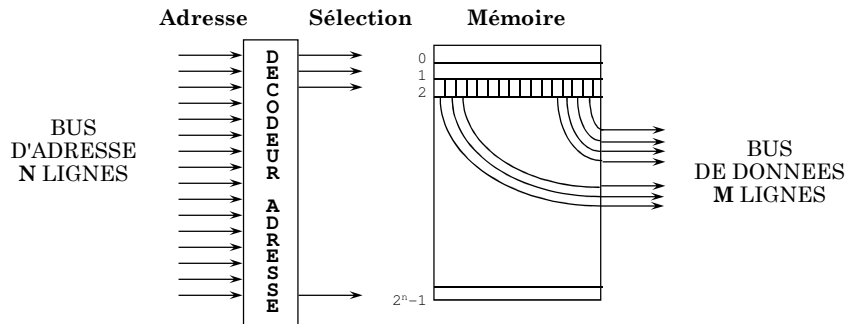
- Nécessaire d'implanter dans un microprocesseur
 - espace pour le **système**
 - espace pour l'**utilisateur**
 - espace pour les **entrées / sorties**
- Taille suffisante pour ranger les programmes
- Gestion sûre pour une intégrité des données
- Séparation entre les différents espaces (utilisateur, système)

Exemple de configuration de mémoire



Mécanisme d'accès à la mémoire

L'accès à une information -> positionnement de l'adresse sur les n lignes de bus



Exemple: N = 8, Adresse = 00000010
253 sélections possibles, Sélection = 2
M = 16 -> bus de données reçoit des valeurs sur 16 bits

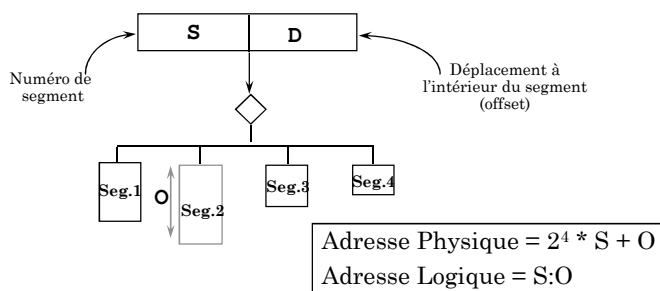
Mécanisme d'adressage

Trois types d'adresses pour le microprocesseur

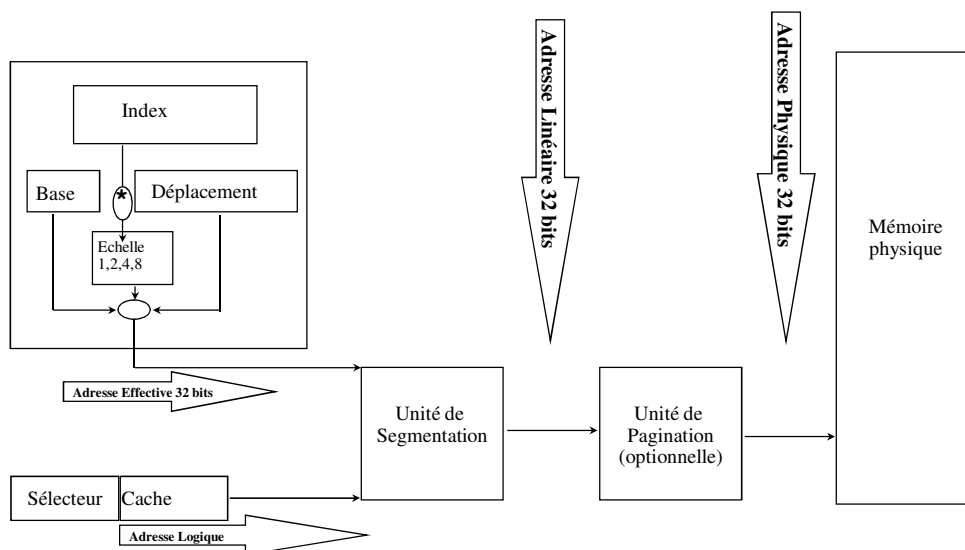
- **L'adresse logique** registre de segment plus offset
- **L'adresse linéaire** sur 32 bits générée par l'unité de segmentation
- **L'adresse physique** qui est sur 32 bits
 - si pagination active : adresse linéaire est transformée en adresse physique par l'unité de pagination
 - sinon elles sont égales.

La segmentation

- **Adresse physique** : correspond à un emplacement physique (adresse sur 20 bits sur le 8086)
- **Adresse logique** (effective) la mémoire n'est pas linéaire -> espace logique d'adressage. (adresse sur 16 bits sur le 8086)
- **Exemple:** mémoire segmentée du 80x86



Mécanisme d'adressage



Modèle d'organisation de la mémoire

- **Modèle mémoire linéaire**
 - valable pour OS qui nécessitent une vitesse performante
 - protection moindre
 - limité dans le cas des applications de plus de 4Go de mémoire
- **Modèle de mémoire segmentée**
 - utilise les registres de segment et quatre niveaux de protection
 - les OS peuvent se protéger des applications
 - le processeur gère la mémoire et vérifie les protections

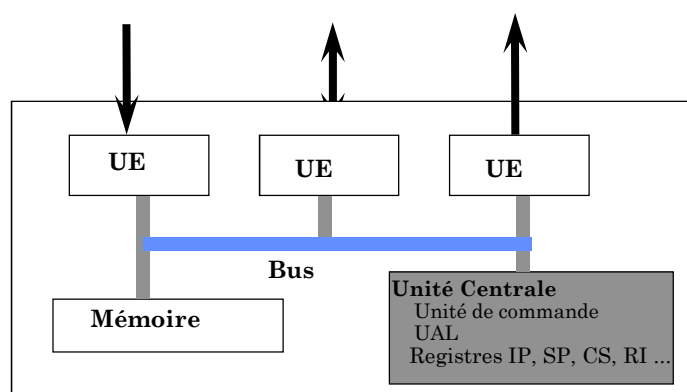
La mémoire virtuelle

- Utilisation de la mémoire secondaire «comme si» c'était de la mémoire centrale
- Disponible à partir des 80386 et 68030
- Gestion se fait au niveau du système d'exploitation
- Système de gestion de mémoire virtuelle très sophistiqué
 - **Pagination :**
 - » mono dimensionnelle
 - » mappage de l'espace d'adressage sur la mémoire physique.
 - » Les zones mémoires lues sur la mémoire secondaire sont *les pages de 4Ko*
 - **Segmentation :**
 - » possibilité d'avoir plusieurs espaces d'adressage virtuels
 - » chaque espace est indépendant : *les segments*
 - » les segments sont de taille variable (limités à 64k)
 - » Sur les 80X86 la segmentation a été utilisée pour résoudre le problème des adresses sur 20 bits

La pagination

- Idée vient de la séparation entre espace d'adressage et emplacement mémoire
- Une page est une zone de mémoire de taille fixe (4k)
- Déplacement des pages de la mémoire secondaire vers la mémoire centrale -> *Pagination*
- La *Pagination*
 - se fait au fur et à mesure des besoins
 - Pb du défaut de page
 - La pagination est transparente à l'utilisateur
- Réalisation de la *pagination*
 - à la charge du système d'exploitation

L'unité centrale de traitement (UC)



L'unité Centrale

- **Son Rôle**

- Exécute les programmes en mémoire centrale
- Cycle de chargement, décodage et exécution des instructions
- Stocke les résultats intermédiaires dans des registres

- **Sa Composition**

- Registres : mémorisation temporaire
- Unité Arithmétique et Logique : effectue les opérations
- Unité de commande : orchestre les opérations
- Bus : Communication des résultats
- Jeu d'Instructions : opérations

L'UC: Instructions

- **Une instruction**

- Opération élémentaire
- Information sur :
 - » ce que l'instruction fait (add, sub, mov ..)
 - » sur quelles données (AL, AX, 10h ...)

- **les instructions sont spécifiques à un microprocesseur**

- 680XX, 80X86, SPARC ...
- Exemple
 - » Pentium 220 instructions.
 - » Techno MMX +57 instructions

- **Transformation d'une instruction**

- » D'un langage évolué vers le langage machine (Compilateur)
- » D'un langage bas-niveau vers le langage machine (Assembleur)

L'UC: Instructions ...

- **Codage d'une instruction**

Zone Opération	Zone adresse
----------------	--------------

- *Zone Opération*
 - » Code Opération -> Codage unique
 - » jeu d'instruction limité (2 octets->
- *Zone d'adresse*
 - » adresse(s) de la donnée
 - » Adressage implicite -> utilisation d'un accumulateur
 - » **Exemple** : B0 01 -> MOV AL , 01

L'UC: l'unité de commande

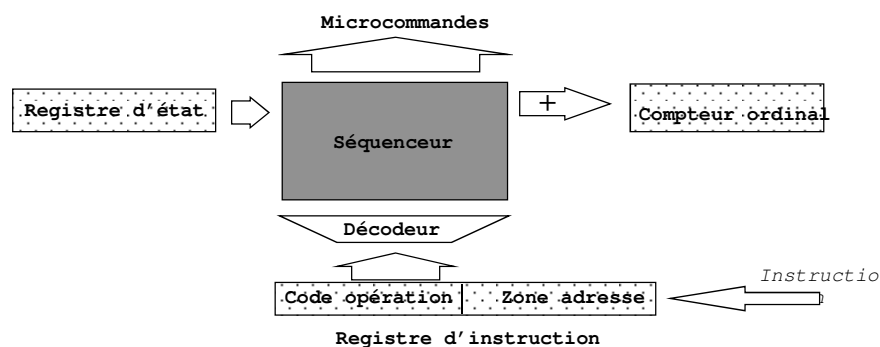
- Phase de **recherche de l'instruction**

- programme en mémoire centrale
- lancement du programme *Compteur Ordinal* <- @ 1ère instruction (OS)
- le séquenceur va générer les microcommandes pour placer l'instruction dans le Registre d'Instruction
- microcommande pour incrémenter le compteur ordinal
Compteur Ordinal <- @ prochaine instruction

- Phase de **traitement de l'instruction**

- Le décodeur analyse la zone opération de l'instruction
- Le séquenceur exécute les microcommandes correspondantes
- Exemple: addition
MOV AL, [0200]
ADD AL, [0300]
MOV [0400], AL

Schéma simplifié d'une unité de commandes

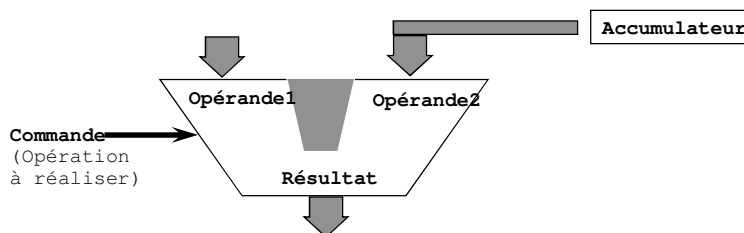


L'UC: l'Unité de commande

- *Registre d'instruction:*
 - Contient l'instruction courante
- *Séquenceur:*
 - Emet les microcommandes suivant un ordre (séquencement) .
 - Rythmé par une horloge interne du système, Cycle machine (horloge)
- *Registre d'état*
 - Donne l'état de l'opération dernièrement effectuée (retenue, parité)
 - Indicateurs
 - Très utile au séquenceur
- *Compteur Ordinal (program counter)*
 - Registre IP
 - Contient l'adresse mémoire de la prochaine instruction à exécuter
 - Incrémenté à chaque chargement d'une instruction en mémoire

L'UC: L'unité arithmétique et logique

- Composée de circuits logiques
- Réalise des opérations
 - arithmétiques: + - * ~%
 - logiques ~ + . comparaison décalage
- Les opérations à effectuer et les opérandes lui sont indiquées par l'Unité de commande.



L'UC: L'UAL...

- UAL et registre d'état
 - **Exemple:** Overflow, Zéro, Carry

`CMP AL, 0`

↑
Instruction de comparaison } Si AL = 0
en assembleur 8086 } alors Z=1
sinon Z=0
fini

- Flag Z du registre d'état

- **NB:** Une UAL fait un nombre limité d'opérations-> fonctions manquantes à programmer

Plan

- **Introduction**
 - Décomposition fonctionnelle/matérielle d'un ordinateur
 - Evolution et performance des machines
- **Représentation de l'information**
- **Architecture type Von Neumann**
 - Structure d'interconnexion
 - La mémoire
 - l'unité centrale
- **Exemple d'un processeur Intel Pentium**
- **Couche d'assemblage**
 - langage
 - Modes d'adressage
 - Procédures

Exemple de processeur

- **Processeur Intel Pentium**
 - performance doublée voir multipliée par 7 par rapport au 486 DX2 66
 - compatibilité binaire avec les 486, 386, 286 ...
- **Origine de ces performances ?**
 - unité de calcul entier super-scalaire
 - unité de calcul flottant super-scalaire
 - cache
- **Processeur avec registre et adressage 32 bits.**
- **Possibilité de connexion à un bus de donnée 64 bits**
- **Possibilité de fonctionnement en multi-processeur.**

Exemple de processeur

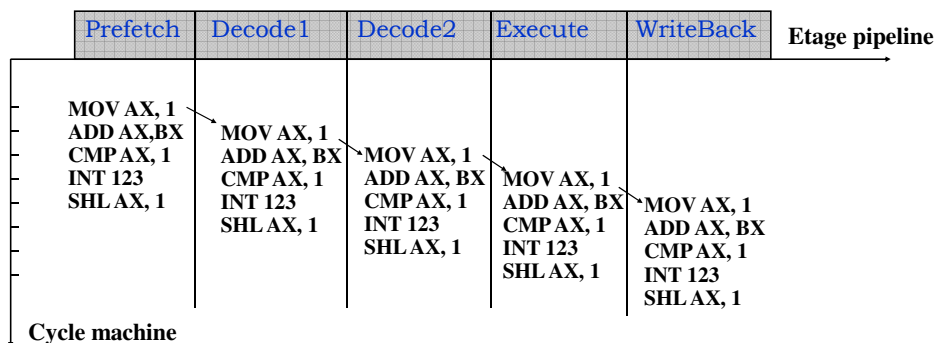
- Exécution à l'intérieur du pipeline
 - mécanisme déjà implémenté sur un 486 : cinq phases
 - *Prefetch* : recherche de l'instruction en mémoire
 - *Décode 1 et Décode 2* : détermine le type d'action à exécuter et éventuellement les opérandes
 - *Exécute* : mise en œuvre de l'instruction avec accès mémoire
 - *Write back* : met-à-jour les registres du processeur
- Sur un pentium 2 pipelines en parallèle
 - doublement des performances ?
 - problème des dépendances entre instructions.
 - problème des branchements : interruption de l'exécution de l'instruction
 - compilateurs pour Pentium optimisent l'utilisation des 2 pipelines

Fonctionnement d'un Pipeline

Programme en mémoire

```

MOV  AX, 1
ADD  AX, BX
CMP  AX, 1
INT  123
SHL  AX, 1
    
```



Plan

- [Introduction](#)
 - Décomposition fonctionnelle/matérielle d'un ordinateur
 - Evolution et performance des machines
- [Représentation de l'information](#)
- [Architecture type Von Neumann](#)
 - Structure d'interconnexion
 - La mémoire
 - l'unité centrale
- [Exemple d'un processeur Intel Pentium](#)
- [Couche d'assemblage](#)
 - Langage
 - Modes d'adressage
 - Procédures

Jeu d'instruction processeurs Intel x86

[Instructions de mouvement](#)

★ **MOV Dest, Source**

Réalise : Dest := Source

Expl: MOV CL, AL
MOV AX, 12

★ **XCHG D1, D2**

Réalise : D1:= D2

D2:= D1

Expl: XCHG BX, CX

Jeu d'instruction processeurs Intel x86

- Instructions arithmétiques
- 4 opérations de base qui prennent en compte les dépassements de capacité
 - Multiplication signée et non signée

★ IMUL Opérande → signée
MUL Opérande → non signée

Réalise : 1er opérande (sur 8 bits) est AL, le résultat est dans AX
1er opérande (sur 16 bits) est AX, le résultat est dans DX,AX

Expl: MOV AX, 1178H
MOV BX, 320H
MUL BX → résultat DX=0036, AX=9706

Jeu d'instruction processeurs Intel x86

- Incrémentation et décrémentation

★ INC Registre
DEC Registre
Réalise : Registre := Registre +/- 1

- Addition et Soustraction avec ou sans retenue

★ ADD Registre, Valeur
ADC Registre, valeur
Réalise : Registre := Registre + valeur

★ SUB Registre, Valeur
SBB Registre, valeur
Réalise : Registre := Registre - valeur

- **NB:** Pour les opération arithmétiques plusieurs modes d'adressage sont possibles.

Jeu d'instruction processeurs Intel x86

- [Instructions logiques](#)
- Opèrent sur les registres et la mémoire

★ OPERATION Dest, Source

OR	OU logique
XOR	OU Exclusif
AND	ET logique
NOT	Complément à 1
NEG	Complément à 2
TEST	Et logique sans modification de la destination

Expl: OR AX, BX
 XOR AX, 1426H \longrightarrow Si AX=FE64h AX:=EA42h
 NEG COMPTE

Jeu d'instruction processeurs Intel x86

- [Décalages](#)
- Multiplication ou Division par puissance de 2
- Isolent des bits dans un mot
 - Décalage
 - » Arithmétique: Conserve le signe
 - » logique: Pas de signe
 - A droite: Division par 2, injecte des zéros par la droite
 - A gauche : Multiplication par 2

★ SXX Opérande, nbre de bits

Jeu d'instruction processeurs Intel x86

- **Exemple** de décalage à droite

– *Division non Signée* SHR

7:2 -> 0111

Décalage -> 0011 -> 1

$7 = 2 \times 3 + 1$



– *Division signée* SAR

-7:2 -> 1001

Décalage -> 0100 -> 1

Signe réinjecté -> 1100

$-7 = -4 \times 2 + 1$



Jeu d'instruction processeurs Intel x86

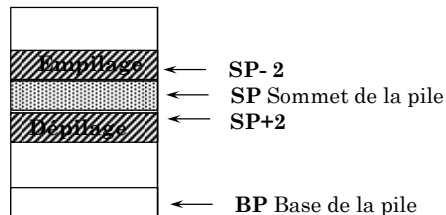
- Comparaisons
- L'instruction **CMP** compare deux opérandes et met à jour les drapeaux qui sont ensuite utilisés pour des sauts conditionnels

★ CMP Oper1, Oper2

- CMP réalise la soustraction des 2 opérandes sans les modifier.
- L'exploitation du résultat se fait en signé ou pas
- **Exemple:** CMP AX, BX

Jeu d'instruction processeurs Intel x86

- Instructions de gestion de la pile
- Pile organisée en mots (2 octets)



- L'adressage se fait par **BP** (offset % à SS)
- **PUSH** empile
- **POP** dépile

Jeu d'instruction processeurs Intel x86

- Boucle de programme
- Code assembleur
 - ranger dans CX la valeur N
 - exécuter les instructions
 - décrémenter [CX]
 - Continuer tant que [CX] ≠ 0

- **Exemple** : `MOV CX, 10` !Initialisation du compteur de boucle

Répéter :

! Séquence d'instructions

Loop Répéter

- Initialisation du compteur de boucle

Jeu d'instruction processeurs Intel x86

- Variantes de LOOP
- LOOP *Boucle si CX<>0*
- LOOPNE *Boucle si ZF =0 et CX<>0*
- LOOPE *Boucle si ZF =1 et CX<>0*

Jeu d'instruction processeurs Intel x86

- **Exemple:** d'utilisation des boucles

Recherche du caractère EOT dans une chaîne

```
mov SI, offset Zone      ; début de Zone
mov CX, 80                ; 80 caractères max
mov AL, EOT              ; caractère à rechercher
Rechercher                ; étiquette
    cmp AL, [SI]         ; comparaison
    inc SI                ; caractère suivant
loopne Rechercher        ; boucle si CX<>0 ET ZF=0
```

- **NB:** Une étiquette permet de faire un saut dans le programme sans connaître l'adresse de l'instruction ou l'on veut aller

Jeu d'instruction processeurs Intel x86

- Instruction de saut
- Saut permet d'accéder à une instruction particulière dans le programme
- Saut inconditionnel
- Adresse du saut = étiquette

★ JMP XXX

- **Exemple:** JMP La-bas

la-bas :

Jeu d'instruction processeurs Intel x86

- Sauts directs ou pas
- Exemple : JMP la-bas ; *direct*

faire le saut JMP BX ; *BX contient l'adresse où doit se*

Jeu d'instruction processeurs Intel x86

- Sauts Conditionnels dans un programme
- Les programmes s'exécutent instruction par instruction
- Certaines permettent de sauter un bout de code
- la "hauteur" du saut est limitée à l'intérieur du segment
- **Exemple:** JC XXXX ; Saut conditionnel à XXXX si C=1
- XXXX:
- **NB:** Si on veut réaliser un saut > il faut coder l'instruction contraire et utiliser un saut inconditionnel long

```

        JNC YYYY
        JMP FAR XXXX
YYYY:   ; exécution du code si C=0
XXXX :   ; exécution du code si C=1
    
```

Jeu d'instruction processeurs Intel x86

- Sauts Conditionnels
- Ils sont utilisés pour prendre une décision en fonction des drapeaux du registre d'état

Sémantique	Signé	Non Signé
=	JEQ	JEQ
>	JG	JA
<	JL	JB
>=	JGE	JAE
<=	JLE	JBE
<>	JNE	JNE
Carry	JC/JNC	Saut si retenue ou pas
Overflow	JO/JNO	Saut si débordement ou pas
Signe	JS/JNS	Saut si signe =1 ou pas

Modes d'adressage

- Permet la manipulation des différents structures de données
- Permet l'accès des données en mémoire.
 - Des variables par leur **valeur**
 - Accès aux données via des **pointeurs**
 - Des éléments de **tableaux**
 - Accès aux éléments de **structures**
- Ces données sont les **opérandes** pour des instructions
 - Arithmétique ...
 - Déplacement ...

Modes d'Adressage

- Les opérandes peuvent être dans :
 - des **registres**
 - une **zone mémoire**: on utilise alors l'adresse effective qui peut être une combinaison de 3 éléments
 - » la base
 - » l'Index
 - » le Déplacement
- Cela donne des adressages
 - IMMEDIAT
 - DIRECT
 - INDIRECT
 - BASE
 - INDEXE

Modes d'Adressage

- Adressage **immédiat**

- Le champs d'adresse contient la valeur de l'opérande ou son adresse physique,
- Mode utile pour manier les constantes,
- **Exemple :**

CMP AX, 12H



Modes d'Adressage

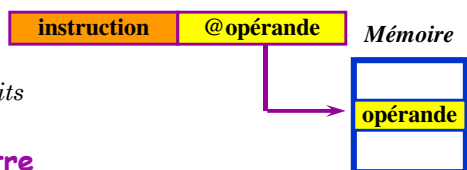
- Adressage **direct**

- l'adresse est l'adresse effective ou l'offset de la variable par rapport au segment de base. L'offset peut être dans

- » un registre, une variable

- » **Exemple :**

```
MOV EAX, TAUX ; 32 bits
ADD AL, VAL ; 8 bits
```

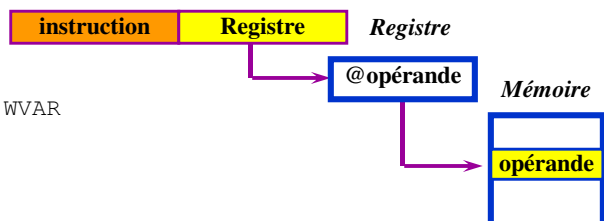


- Adressage indirect ou par registre

- L'offset de la variable est contenu dans un registre de base ou d'index.

- **Exemple :**

```
MOV BX, offset WVAR
MOV AX, [BX]
```

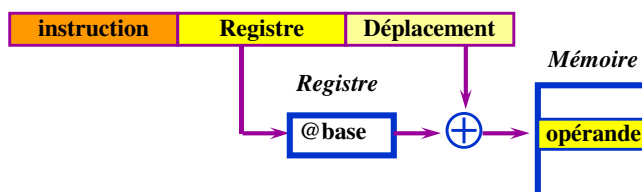


- **NB :** Le type de la donnée doit être accordé avec le registre utilisé

Modes d'Adressage

- Adressage **basé**

- Similaire à l'adressage indirect par registre sauf qu'un déplacement est ajouté à la base
- adresse effective = adresse base + déplacement

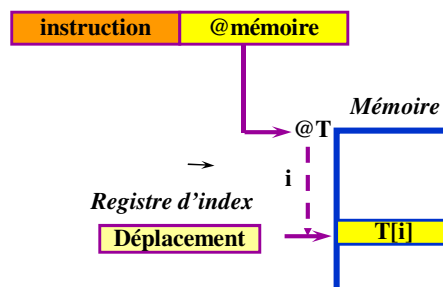


- **Exemple** : `MOV EBX, OFFSET DVAR`
`MOV EAX, [EBX+4]`

Modes d'Adressage

- Adressage **indexé**

- On utilise un registre d'index SI ou DI plus un déplacement
- L'adressage est noté par des crochets
- Utile pour le parcours de tableaux



- **Exemple** : `MOV SI,0`
`MOV AX, T[SI]`

Mode d'adressage

- Adressage **Basé indexé**
 - utilise le contenu d'un registre de base, le contenu d'un registre d'index et un déplacement optionnel.
 - Combinaison des deux modes précédents
 - **Exemple :**

```
XOR EAX,EAX
MOV EBX, OFFSET DVAR0
MOV ESI, 10
ADD EAX, [EBX+ESI+4]
```

Exemple d'Adressage

- Soit une matrice 4 colonnes X 3 lignes contenant des mots
- On veut accéder à une case de cette matrice
 $[case] = Offset MAT + (NBCOL * Lig + Col) * 2$
- On utilise un adressage basé indexé

Exemple d'adressage

; Déclaration des variables

```
NBCOL    EQU    4
NBLIG    EQU    3
Lig      DW     ?           ; ligne à atteindre
Col      DW     ?           ; colonne à atteindre
MAT      DW     NBLIG*NBCOL DUP(?) ; Déclaration de la matrice
```

; Accès à la case (Lig, Col)

```
MOV      BX, Offset MAT
MOV      AL, Lig
MOV      CL, NBCOL
MUL      CL           ; AX <- Lig*NBCOL
MOV      SI, AX
MOV      AX, Col
ADD      SI, AX       ; SI <- (Lig*NBCOL + Col)
SHL      SI, 1        ; 2 octets par case => multi. par 2
MOV      AX, [BX][SI] ; Contenu de la case
```

Programmation en assembleur 8086

- Structure de programme en assembleur
- Un programme doit contenir
 - une zone de **CODE** \Rightarrow segment de code
 - une zone de **DONNEES** \Rightarrow segment de données
 - une zone de **PILE** \Rightarrow segment de pile
 - + une zone Extra-segment pour les chaînes et les tableaux
- Zones à définir dans un programme

Programmation en assembleur 8086

- Déclaration des segments pseudo-instruction **SEGMENT**
- **Exemple:**

```
DONNEES SEGMENT WORD
        ; Déclaration des données
DONNEES SEGMENT ENDS

CODE SEGMENT WORD
        ; Instructions
CODE SEGMENT ENDS
```

Programmation en assembleur 8086

- Gestion des registres de segment
 - Ils sont initialisés par le programme
 - Seuls DS et SS sont explicitement maniés
 - CS est automatiquement mis-à-jour lors du chargement du programme en mémoire par l'OS

- Initialisation de **DS**

```
MOV     AX, nom_segment_donnee
MOV     DS, AX
```

- Initialisation de **SS** et **SP**. Pour la pile il faut :

- Initialiser SS avec l'adresse de segment de pile

```
MOV     AX, nom_segment_pile
MOV     SS, AX
```

- Initialiser SP avec le sommet de la pile

```
LEA    SP, TOS ; TOS est déclaré comme l'élément sommet de
la pile
```

Programmation en assembleur 8086

- Pseudo-Directive ASSUME
- Le langage d'assemblage exige que l'accès à une variable fasse mention du registre employé
- **Exemple:** MOV AX, DS:CNT
- ASSUME permet de rendre visible par un bout de code toutes les variables contenues dans un segment
- **Exemple:** ASSUME DS: GLOBAL
 MOV AX, CNT

Programmation en assembleur 8086

- Définition des données
- Définition de variables en assembleur
 - nom
 - type
 - valeur
- **Exemple :** de type
 - DB 1 octet
 - DW 2 octets = 1 mot
 - DD 4 octets = 2 mots

Programmation en assembleur 8086

• Variable et constantes

- CONS EQU 10 ; Constante =10
- OCTET DB 1 ; Variable=1
- MOT DW ? ; Variable non initialisée
- POINTEUR DD ? ; Variable qui pointe sur une @ (32 bits)

• Vecteurs Tableaux

- VO DB 50 DUP(?) ; 50 octets non initialises
- VM DW 10 DUP(0) ; 10 mots a zéro
- VPT DD 5 DUP(?) ; Table de 5 pointeurs

• Message

- MESS DB 'Bonjour' ; Chaîne de caractères
- CONST DW 7,6,12h,11B ; tableau de 4 mots constants

Programmation en assembleur 8086

• Structures

Les structures permettent de regrouper des données de types différents

Définition :

Module **STRUC**

```
Nom          DB    10 DUP(?)
H_Cours      DD    ?
H_TD         DD    ?
Responsable  DB    10 DUP(?)
```

Module **ENDS**

Initialisation :

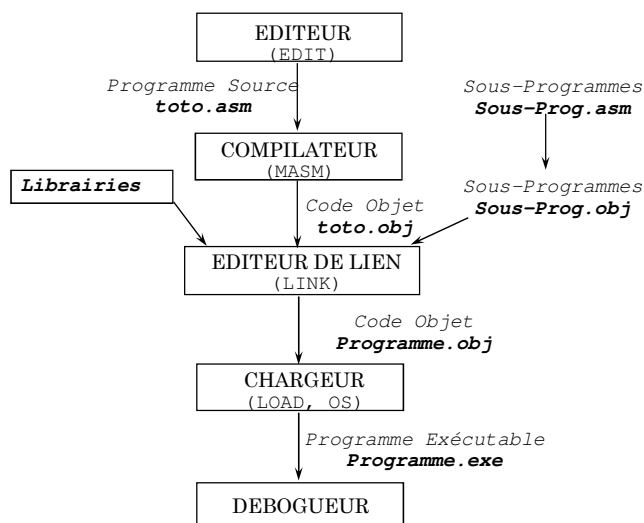
```
Archi1  Module  <'Archi', 15, 15, 'map' >
```

Modèle de programme assembleur

```

;TD8b.asm
; Appel de procédure extra segment
; Affichage du résultat contenu dans AX
PILE          SEGMENT STACK
                DB      256 DUP(?)
PILE          ENDS
DONNEES      SEGMENT
    facteur    DW      4
    puissance DW      2
DONNEES      ENDS
CODE1        SEGMENT
    ASSUME    CS:CODE1, DS:DONNEES
                EXTRN prbyte : FAR
    EX08b    PROC    FAR
                PUSH    DS
                MOV     AX,0
                PUSH    AX
                MOV     AX,DONNEES
                MOV     DS,AX
                CALL   FAR PTR Puissance2
                ...
    EX08b    ENDP
CODE1        ENDS
                END EX08b
    
```

Développement d'un programme



L'assembleur MASM

- MASM : Programme qui traduit du code source (Assembleur) en code objet (code Machine)
- Traduction en plusieurs passes:
 - Analyse lexicale (Repère les mots clefs : loop, while, ; ...)
 - Analyse syntaxique (Détece les erreurs de syntaxe)
 - » Construction de la table des symboles
 - » Résolution des références avant et traduction en code objet
 - » **Exemple de référence avant :**

```
JMP     INST_PLUS_LOIN
.....
.....
INST_PLUS_LOIN : .....

```
 - » Résolution des références extérieures
 - Analyse sémantique
 - Analyse logique

L'assembleur MASM

- *Lexicale :*
 - Fautes d'orthographe dans les identificateurs
 - caractères interdits
- *Syntaxique :*
 - Défaut de parenthèse
 - absence de séparateurs
- *Sémantique :*
 - Identificateur non déclaré
 - incompatibilité entre opérateurs et opérandes
- *Logique :*
 - erreurs arithmétiques
 - erreurs de programmation (non détectées par le compilateur)

L'assembleur MASM

- **1ère Phase** : *Construction de la table des symboles*

- Identification des références externes, Construction de la table des références externes,

EXTRN fonction : FAR

- Lecture du programme source et identification des symboles (nom variables étiquettes)

- » leur adresse d'implantation mémoire

- » la place qu'ils occupent

- » **Exemple :**

```
SIX      EQU      6
SAUT :
        MOV      SIX, R0
        . . . . .
        JE       SAUT
```

Symbole	Valeur	Etat
SIX	6	Const
SAUT	2375	Etiquette

L'assembleur MASM

- **2ème Phase** : *Traduction de l'assembleur en code machine*

- Noms symboliques -> adresses
- Modes d'adressage -> adresses
- Instructions -> code opération
- Détection des erreurs de syntaxe
- Génération du code objet
- Listing d'assemblage

- » numéro de ligne

- » numéro d'instruction

- » texte source

- » représentation hexa du source

Optimisation du code objet généré

- **Compilateur tient compte**
 - des ressources de la machine
 - des redondances dans les programmes
 - **Exemple** : d'optimisation de code

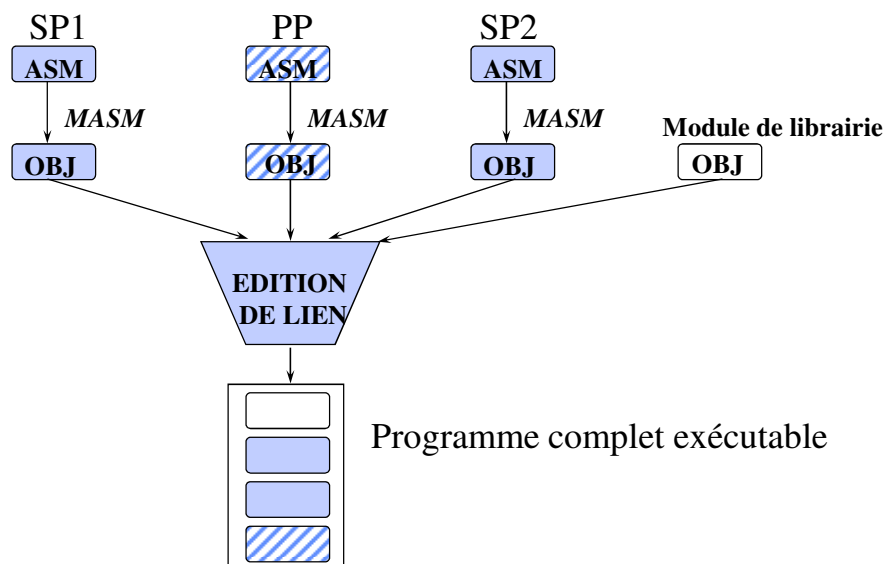
```
Tmp1 := 3.1414 x100
Tmp2 := RealtoInteger(Tmp1)           Tmp3 := Id3 +
    314
Tmp3 := Id1 + Tmp2   Id3 := Tmp3 X Id2
Tmp4 := Tmp3 x Id2
Id3 := Tmp4
```

- Cas des *machines vectorielles* : parallélisme important dans les calculs
- Cas des *machines RISC* : nombre de registre important

L'édition de liens : LINK

- **Définition** : l'éditeur de lien
 - permet de combiner plusieurs fichiers objets en un seul
 - résout les références extérieures
 - gère les adresses à l'intérieur de chaque fichier objet
 - » chaque fichier objet => adresse de départ = 0
 - » après le link, 1 seul programme objet qui démarre à 0

Exemple d'édition de lien d'un programme type

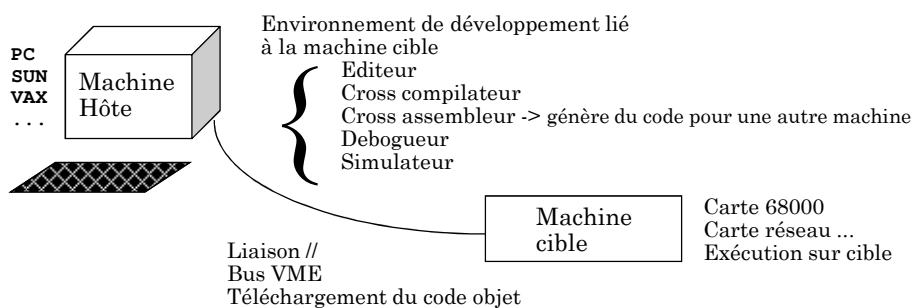


Le Chargeur

- Charge le code objet en mémoire centrale pour être exécuté
- **Remarque** : Dans les ordinateurs avec un seul programme en mémoire centrale, adresse préfixée -> *Chargeur absolu*
- Dans les systèmes multiprogrammés on peut décider au dernier moment de l'endroit où va être implanter le programme.
- **Exemple** : Programme relogeable:
 - on veut charger un programme à l'adresse 2000
 - » le première instruction est en 2000
 - » on ajoute 2000 à toutes les autres adresses
 - Problème: Le chargeur doit modifier les adresses de certaines opérantes. L'assembleur indique si une instruction est relogeable ou pas

Cross-Compilation

- Développement de programmes sur une **machine hôte**
 - édition
 - compilation
 - édition de liens
- Chargement de l'exécutable sur une **machine cible**
- Configuration classique



195

Exemple de cross compilation

- Hôte *PC*
- Cible *RTX2000*
- Liaison // avec carte d'entrée-sortie

- Hôte *SUN4*
- Cible *Carte 68020*
- liaison *bus VME avec Racks*

- Hôte *PC*
- Cible *Carte Réseau FIP*
- liaison //

Sous-programmes & Macros

- Améliorer la structuration des programmes
 - **grouper** une série d'instruction
 - **réutiliser** cette série d'instructions
 - **partager** ces instructions entre plusieurs programmes
- Définition d'un *Sous-Programme* :
 - C'est une séquence d'instructions
 - Un sous-programme possède un nom unique
 - Appel à l'intérieur d'un programme
 - engendre un accès à la pile (mémorisation de l'@ de retour)
 - branchement se fait lors de l'appel => *Overhead*
- SP => Optimisation de la place mémoire
 - stocké dans une seule zone mémoire
 - adresse de stockage déterminée par l'assemblage

Sous-programmes & Macros

- Définition d'une MACRO :
 - C'est une séquence d'instruction,
 - Une macro est recopiée dans le programme appelant lors de l'assemblage
 - => *pas d'économie de mémoire*
 - => *pas d'accès à la pile*
 - => *pas d'overhead*
 - Problème : utilisation peut-être "dangereuse"

Sous-programmes & Macros

- Déclaration en assembleur

- **MACRO**

- » `Nom_Macro MACRO`

- *Code macro*

- » `Nom_Macro ENDM`

- NB : MACRO et ENDM sont des directives
 - Les MACROS ne sont autorisées que par les macro-assembleurs

- **PROCEDURES**

- » `Nom_Procédure PROC NEAR / FAR`

- *Code proc*

- **RET**

- » `Nom_Procédure ENDP` *Intra segment* *Extra segment*

← *Retour de la procédure*

Sous-programmes & Macros

- Appel en assembleur

- **MACRO** : il suffit d'appeler le nom de la macro

- `Main PROC FAR`

- `Nom_Macro`

- `Main ENDP`

- **Sous-Programme** : Appel Intra-Segment

- `Main PROC FAR`

- `Call near ptr Nom_Procédure`

- `Main ENDP`

- **Sous-Programme** : Appel Extra-Segment

- `Main PROC FAR`

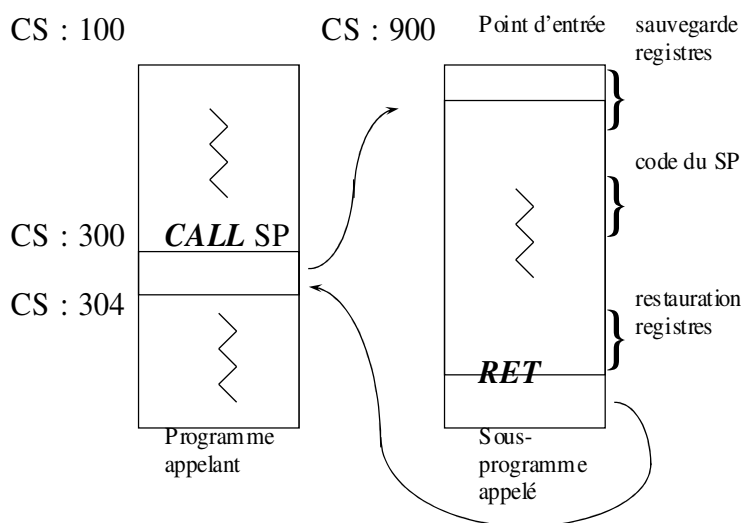
- `Call far ptr Nom_Procédure`

- `Main ENDP`

Effet des appels sur le microprocesseur

- **MACRO** : A l'assemblage, le code des macros est recopié dans le programme appelant
 - pas de sauvegarde de contexte
 - efficacité accrue
 - mais taille de code augmente
- **SOUS-PROGRAMME** : un appel à un sous-programme revient à effectuer un branchement à une adresse qui contient son code.
- **Problème** : Il faut sauvegarder l'état du microprocesseur pour pouvoir revenir dans le programme principal

Effet des appels sur le microprocesseur



Effet des appels sur le microprocesseur

- **Point d'entrée** : adresse de la première instruction exécutée dans le S-P
- **L'appel d'un sous-programme** : lors d'un appel on doit sauvegarder l'adresse de retour du SP c-à-d
 - CS : lorsque le S-P est extra segment
 - IP : dans tous les cas
 - **NB**: Les sauvegardes se font sur la pile.
- **Instructions de sauvegarde** : si le S-P utilise des registres du processeur, il est important de sauvegarder ces registres sur la pile au début du SP.
- **Corps du programme** : le S-P réalise des opérations sur les registres du processeur. Le S-P utilise les ressources de la machine.

Effet des appels sur le microprocesseur

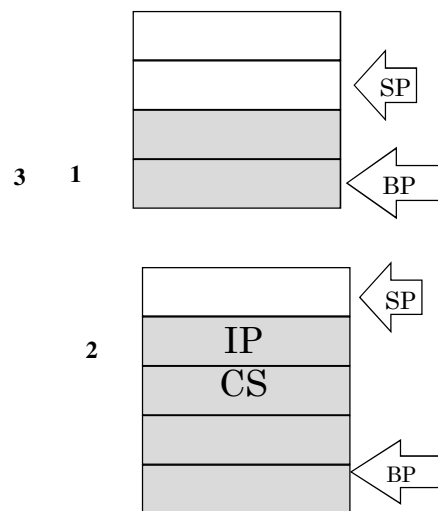
- **Instruction de restauration** : avant de revenir dans le programme principal, le SP "remet en état" les registres.
- **Instruction RET** : Très importante !!!
Cette instruction dépile dans IP et dans CS;
son but est de positionner dans ces registres l'adresse de l'instruction de retour.
 - NB: Il doit y avoir symétrie entre CALL et RET, sauvegarde et restauration.
 - Après un retour de SP : **PILE VIDE**

Exemple de modification de la pile

```

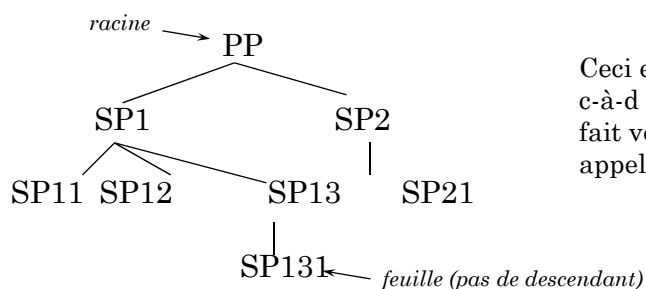
main proc far
  1  ~
  2  call SP1
  3  ~
main endp

SP1  proc near
      ~
      ret
SP1  endp
    
```



Arbre des appels

- On peut résumer par un graphe la structure des appels de SP.



Ceci est un arbre
 c-à-d le retour se
 fait vers le SP
 appelant.

- Nombre d'appels = profondeur de l'arbre
- La taille de la pile dépend de la profondeur de l'arbre et des sauvegardes.

Passage de paramètres

- **Paramètres** : variables dont le nom est connu (P.formel) mais dont la valeur (P.effectif) n'est connue:
 - qu'à la compilation (macro),
 - à l'exécution (sous-programme)
- **Classification des paramètres** :
 - par valeur
 - par référence
- Les paramètres peuvent être en entrée, en sortie ou en entrée-sortie

Passage de paramètres

- **Passage par valeur** : recopie de la valeur à transmettre dans une zone mémoire connue du sous-programme (S-P):
 - ++ Le S-P. travaille sur une copie de la variable => la variable du Programme principal (PP) ne varie pas (protection)
 - Recopie => si la variable est un tableau de 10000 éléments il faut les recopier un à un. Problèmes de place mémoire et d'overhead.
- **Passage par référence** : transmission de l'adresse de la variable :
 - ++ plus rapide
 - + le S-P travaille sur les variables du PP => ces variables ont été modifiées par le S-P.
 - mauvaise solution si on veut protéger les variables

Passage de paramètres

- **Exemple1** : Passage par valeur

Prog. Principal

```

    •••
    MOV  AX, 4 } Paramètres
    MOV  BX, 5 }
    call Somme → Appel
    •••
    -> Résultat dans AX
    
```

Sous-Prog.

```

    Somme PROC NEAR
            ADD  AX, BX
            RET
    Somme ENDP
    
```

- **Exemple2** : Passage par référence

Prog. Principal

```

    •••
    MOV  CX, 10 ; taille de TAB
    LEA  BX, TAB ; BX← @TAB
    call Affiche
    •••
    
```

Sous-Prog.

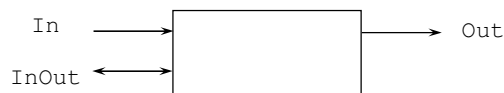
```

    Affiche PROC NEAR
            MOV  DI, 0
    boucle :
            MOV  AL, [BX+DI]
            call COUT
            INC  DI
    LOOP  boucle
            RET
    Affiche ENDP
    
```

Types de paramètres

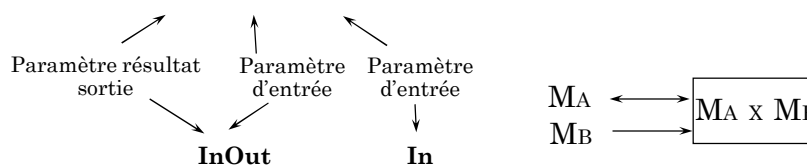
- **Entrée, Sortie, Entrée/Sortie**

– sont liés à la modification ou pas des paramètres par le S-P appelé. C'est aussi **l'interface** du sous-programme



- **Exemple** : si on veut calculer le produit de deux matrices

$$M_A = M_A \times M_B$$



Mécanismes de transmission des paramètres

- **Par registre**

- 1 registre par paramètre à passer
- beaucoup de soin dans les conventions

Prog. Principal

```
•••  
MOV AX, 5  
call plus2  
•••
```

Sous-Prog.

```
Plus2 PROC NEAR  
        ADD AX, 2  
        RET  
Plus2 ENDP
```

- *Avantages :*

- accès rapide aux registres
- possibilité de réentrance dans les S.-P.

- *Inconvénients*

- nombre de registre limité
- le S.-P. a besoin de registres pour s'exécuter => sauvegarde des registres en cours d'exécution

Mécanismes de transmission des paramètres

- **Par la pile :**

- les paramètres sont empilés. On y accède par adressage basé par BP.
- Cette méthode demande de préciser l'ordre de passage des arguments

- *Avantages :*

- on peut ranger plusieurs arguments
- la limitation est donnée par la taille de la pile

- *Inconvénient :*

- performances moindre en vitesse d'exécution

Mécanismes de transmission des paramètres

• **Exemple :**

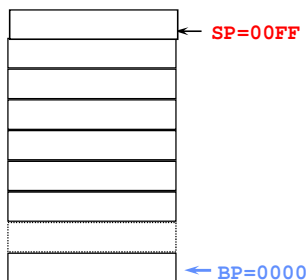
- on utilise un sous-programme *Somme* qui calcule la somme d'une zone d'octets et qui met le résultat dans une variable *Résultat*.



- Le *programme appelant* passe les paramètres sur la pile
 - » l'adresse de la zone
 - » la taille de la zone
 - » l'adresse de la variable résultat
- Le *programme appelé* récupère les paramètres sur la pile en utilisant BP

Programme Principal

...



Sous-Programme

```

Somme PROC FAR
;Sauvegarde de BP
    PUSH BP
;Récupère le sommet de la pile
    MOV BP, SP
;Paramètres
    MOV SI, [BP+10] ;Zone
    MOV CX, [BP+8] ;Taille
SommeZone:
    ADD DX, [SI]
    INC SI
    LOOP SommeZone
;Résultat
    MOV SI, [BP+6]
    MOV SI, DX
;Retour au Prog.Principal
    POP BP
    RET
Somme ENDP
  
```


Programme Principal	Sous-Programme
<pre> ••• 1 LEA AX, Zone PUSH AX </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>
<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">1</div> <div style="border: 1px solid black; padding: 5px; width: 150px;"> <div style="background-color: yellow; padding: 2px;">@Zone</div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> </div> <div style="margin-left: 10px;"> <p>← SP=00FF</p> <p>← SP=00FD</p> <p style="color: blue;">← BP=0000</p> </div> </div>	
M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique	215

Programme Principal	Sous-Programme
<pre> ••• 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>
<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">1</div> <div style="border: 1px solid black; padding: 5px; width: 150px;"> <div style="background-color: yellow; padding: 2px;">@Zone</div> <div style="background-color: yellow; padding: 2px;">Taille</div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> <div style="padding: 2px;"> </div> </div> <div style="margin-left: 10px;"> <p>← SP=00FD</p> <p>← SP=00FB</p> <p style="color: blue;">← BP=0000</p> </div> </div>	
M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique	216

Programme Principal																													
<pre> ... 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX </pre>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 20px;">1</td><td style="width: 100px;">@Zone</td><td></td></tr> <tr><td>2</td><td>Taille</td><td>← SP=00FB</td></tr> <tr><td>3</td><td>@Résultat</td><td>← SP=00F9</td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </table>	1	@Zone		2	Taille	← SP=00FB	3	@Résultat	← SP=00F9																			<pre> Somme PROC FAR ;Sauvegarde de BP PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>
1	@Zone																												
2	Taille	← SP=00FB																											
3	@Résultat	← SP=00F9																											
	← BP=0000																												
<p><small>M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique 217</small></p>																													

Programme Principal																													
<pre> ... 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 4 CALL far ptr Somme ... </pre>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 20px;">1</td><td style="width: 100px;">@Zone</td><td></td></tr> <tr><td>2</td><td>Taille</td><td></td></tr> <tr><td>3</td><td>@Résultat</td><td>← SP=00F9</td></tr> <tr><td>4</td><td>CS (retour)</td><td></td></tr> <tr><td> </td><td>IP (retour)</td><td>← SP=00F5</td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </table>	1	@Zone		2	Taille		3	@Résultat	← SP=00F9	4	CS (retour)			IP (retour)	← SP=00F5													<pre> Somme PROC FAR ;Sauvegarde de BP PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>
1	@Zone																												
2	Taille																												
3	@Résultat	← SP=00F9																											
4	CS (retour)																												
	IP (retour)	← SP=00F5																											
	← BP=0000																												
<p><small>M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique 218</small></p>																													

Programme Principal	Sous-Programme																								
<pre> *** 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 3 CALL far ptr Somme *** </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP 5 PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>																								
<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="width: 20px; text-align: center;">1</td><td style="background-color: #ffffcc;">@Zone</td><td></td></tr> <tr><td style="text-align: center;">2</td><td style="background-color: #ffffcc;">Taille</td><td></td></tr> <tr><td style="text-align: center;">3</td><td style="background-color: #ffffcc;">@Résultat</td><td></td></tr> <tr><td style="text-align: center;">4</td><td style="background-color: #ffffcc;">CS (retour)</td><td></td></tr> <tr><td style="text-align: center;">5</td><td style="background-color: #ffffcc;">IP (retour)</td><td>← SP=00F5</td></tr> <tr><td></td><td style="background-color: #ffffcc;">BP=0000</td><td>← SP=00F3 ← BP=00F3</td></tr> <tr><td></td><td style="background-color: #ffffcc;"> </td><td></td></tr> <tr><td></td><td style="background-color: #ffffcc;"> </td><td>← BP=0000</td></tr> </table>	1	@Zone		2	Taille		3	@Résultat		4	CS (retour)		5	IP (retour)	← SP=00F5		BP=0000	← SP=00F3 ← BP=00F3						← BP=0000	
1	@Zone																								
2	Taille																								
3	@Résultat																								
4	CS (retour)																								
5	IP (retour)	← SP=00F5																							
	BP=0000	← SP=00F3 ← BP=00F3																							
		← BP=0000																							

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 219

Programme Principal	Sous-Programme																								
<pre> *** 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 3 CALL far ptr Somme *** </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP 5 PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>																								
<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="width: 20px; text-align: center;">1</td><td style="background-color: #ffffcc;">@Zone</td><td>← BP+10</td></tr> <tr><td style="text-align: center;">2</td><td style="background-color: #ffffcc;">Taille</td><td></td></tr> <tr><td style="text-align: center;">3</td><td style="background-color: #ffffcc;">@Résultat</td><td></td></tr> <tr><td style="text-align: center;">4</td><td style="background-color: #ffffcc;">CS (retour)</td><td></td></tr> <tr><td style="text-align: center;">5</td><td style="background-color: #ffffcc;">IP (retour)</td><td></td></tr> <tr><td></td><td style="background-color: #ffffcc;">BP</td><td>← SP=00F3 ← BP=00F3</td></tr> <tr><td></td><td style="background-color: #ffffcc;"> </td><td></td></tr> <tr><td></td><td style="background-color: #ffffcc;"> </td><td></td></tr> </table>	1	@Zone	← BP+10	2	Taille		3	@Résultat		4	CS (retour)		5	IP (retour)			BP	← SP=00F3 ← BP=00F3							
1	@Zone	← BP+10																							
2	Taille																								
3	@Résultat																								
4	CS (retour)																								
5	IP (retour)																								
	BP	← SP=00F3 ← BP=00F3																							

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 220

Programme Principal	Sous-Programme																								
<pre> *** 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 3 CALL far ptr Somme *** </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP 5 PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>																								
<table border="0" style="width: 100%;"> <tr> <td style="width: 20px; text-align: right;">1</td> <td style="border: 1px solid black; padding: 2px;">@Zone</td> <td style="width: 100px;"></td> <td style="text-align: right;">← BP+10</td> </tr> <tr> <td style="text-align: right;">2</td> <td style="border: 1px solid black; padding: 2px;">Taille</td> <td></td> <td style="text-align: right;">← BP+8</td> </tr> <tr> <td style="text-align: right;">3</td> <td style="border: 1px solid black; padding: 2px;">@Résultat</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">4</td> <td style="border: 1px solid black; padding: 2px;">CS (retour)</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">5</td> <td style="border: 1px solid black; padding: 2px;">IP (retour)</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">5</td> <td style="border: 1px solid black; padding: 2px;">BP</td> <td style="text-align: right;">← SP=00F3</td> <td style="text-align: right;">← BP=00F3</td> </tr> </table>	1	@Zone		← BP+10	2	Taille		← BP+8	3	@Résultat			4	CS (retour)			5	IP (retour)			5	BP	← SP=00F3	← BP=00F3	
1	@Zone		← BP+10																						
2	Taille		← BP+8																						
3	@Résultat																								
4	CS (retour)																								
5	IP (retour)																								
5	BP	← SP=00F3	← BP=00F3																						

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 221

Programme Principal	Sous-Programme																								
<pre> *** 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 3 CALL far ptr Somme *** </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP 5 PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV DX, 0 MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI, 2 LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>																								
<table border="0" style="width: 100%;"> <tr> <td style="width: 20px; text-align: right;">1</td> <td style="border: 1px solid black; padding: 2px;">@Zone</td> <td style="width: 100px;"></td> <td style="text-align: right;">← BP+10</td> </tr> <tr> <td style="text-align: right;">2</td> <td style="border: 1px solid black; padding: 2px;">Taille</td> <td></td> <td style="text-align: right;">← BP+8</td> </tr> <tr> <td style="text-align: right;">3</td> <td style="border: 1px solid black; padding: 2px;">@Résultat</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">4</td> <td style="border: 1px solid black; padding: 2px;">CS (retour)</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">5</td> <td style="border: 1px solid black; padding: 2px;">IP (retour)</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">5</td> <td style="border: 1px solid black; padding: 2px;">BP</td> <td style="text-align: right;">← SP=00F3</td> <td style="text-align: right;">← BP=00F3</td> </tr> </table>	1	@Zone		← BP+10	2	Taille		← BP+8	3	@Résultat			4	CS (retour)			5	IP (retour)			5	BP	← SP=00F3	← BP=00F3	
1	@Zone		← BP+10																						
2	Taille		← BP+8																						
3	@Résultat																								
4	CS (retour)																								
5	IP (retour)																								
5	BP	← SP=00F3	← BP=00F3																						

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 222

Programme Principal	Sous-Programme																								
<pre> ... 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 3 CALL far ptr Somme ... </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP 5 PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>																								
<table border="0" style="width: 100%;"> <tr> <td style="width: 20px; text-align: right;">1</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">@Zone</td> <td style="width: 20px;"></td> <td style="text-align: right;">← BP+10</td> </tr> <tr> <td style="text-align: right;">2</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">Taille</td> <td></td> <td style="text-align: right;">← BP+8</td> </tr> <tr> <td style="text-align: right;">3</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">@Résultat</td> <td></td> <td style="text-align: right;">← BP+6</td> </tr> <tr> <td style="text-align: right;">4</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">CS (retour)</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">5</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">IP (retour)</td> <td></td> <td></td> </tr> <tr> <td></td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">BP</td> <td style="text-align: right;">← SP=00F3</td> <td style="text-align: right;">← BP=00F3</td> </tr> </table>	1	@Zone		← BP+10	2	Taille		← BP+8	3	@Résultat		← BP+6	4	CS (retour)			5	IP (retour)				BP	← SP=00F3	← BP=00F3	
1	@Zone		← BP+10																						
2	Taille		← BP+8																						
3	@Résultat		← BP+6																						
4	CS (retour)																								
5	IP (retour)																								
	BP	← SP=00F3	← BP=00F3																						

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 223

Programme Principal	Sous-Programme																												
<pre> ... 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 3 CALL far ptr Somme ... </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP 5 PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV [SI], DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>																												
<table border="0" style="width: 100%;"> <tr> <td style="width: 20px; text-align: right;">1</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">@Zone</td> <td style="width: 20px;"></td> <td style="text-align: right;">← BP+10</td> </tr> <tr> <td style="text-align: right;">2</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">Taille</td> <td></td> <td style="text-align: right;">← BP+8</td> </tr> <tr> <td style="text-align: right;">3</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">@Résultat</td> <td></td> <td style="text-align: right;">← BP+6</td> </tr> <tr> <td style="text-align: right;">4</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">CS (retour)</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">5</td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">IP (retour)</td> <td style="text-align: right;">← SP=00F5</td> <td></td> </tr> <tr> <td></td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;">BP</td> <td style="text-align: right;">← SP=00F3</td> <td style="text-align: right;">← BP=00F3</td> </tr> <tr> <td></td> <td style="border: 1px solid black; background-color: #ffffcc; padding: 2px;"></td> <td></td> <td style="text-align: right;">← BP=0000</td> </tr> </table>	1	@Zone		← BP+10	2	Taille		← BP+8	3	@Résultat		← BP+6	4	CS (retour)			5	IP (retour)	← SP=00F5			BP	← SP=00F3	← BP=00F3				← BP=0000	
1	@Zone		← BP+10																										
2	Taille		← BP+8																										
3	@Résultat		← BP+6																										
4	CS (retour)																												
5	IP (retour)	← SP=00F5																											
	BP	← SP=00F3	← BP=00F3																										
			← BP=0000																										

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 224

Programme Principal	Sous-Programme																																
<pre> *** 1 LEA AX, Zone PUSH AX 2 MOV CX, LENGHT Zone PUSH CX 3 LEA AX, Résultat PUSH AX 3 CALL far ptr Somme *** </pre>	<pre> Somme PROC FAR ;Sauvegarde de BP 5 PUSH BP ;Récupère le sommet de la pile MOV BP, SP ;Paramètres MOV SI, [BP+10] ;Zone MOV CX, [BP+8] ;Taille SommeZone: ADD DX, [SI] INC SI LOOP SommeZone ;Résultat MOV SI, [BP+6] MOV SI, DX ;Retour au Prog.Principal POP BP RET Somme ENDP </pre>																																
<table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20px; text-align: right;">1</td> <td style="border: 1px solid black; padding: 2px;">@Zone</td> <td style="width: 100px;"></td> <td style="text-align: right;">← BP+10</td> </tr> <tr> <td style="text-align: right;">2</td> <td style="border: 1px solid black; padding: 2px;">Taille</td> <td></td> <td style="text-align: right;">← BP+8</td> </tr> <tr> <td style="text-align: right;">3</td> <td style="border: 1px solid black; padding: 2px;">@Résultat</td> <td style="text-align: right;">← SP=00F7</td> <td style="text-align: right;">← BP+6</td> </tr> <tr> <td style="text-align: right;">4</td> <td style="border: 1px solid black; padding: 2px;">CS (retour)</td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">5</td> <td style="border: 1px solid black; padding: 2px;">IP (retour)</td> <td style="text-align: right;">← SP=00F5</td> <td></td> </tr> <tr> <td></td> <td style="border: 1px solid black; padding: 2px;">BP</td> <td></td> <td></td> </tr> <tr> <td></td> <td style="border: 1px solid black; padding: 2px;"> </td> <td></td> <td></td> </tr> <tr> <td></td> <td style="border: 1px solid black; padding: 2px;"> </td> <td></td> <td style="text-align: right;">← BP=0000</td> </tr> </table>	1	@Zone		← BP+10	2	Taille		← BP+8	3	@Résultat	← SP=00F7	← BP+6	4	CS (retour)			5	IP (retour)	← SP=00F5			BP										← BP=0000	
1	@Zone		← BP+10																														
2	Taille		← BP+8																														
3	@Résultat	← SP=00F7	← BP+6																														
4	CS (retour)																																
5	IP (retour)	← SP=00F5																															
	BP																																
			← BP=0000																														

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 225

Conclusion sur cet exemple

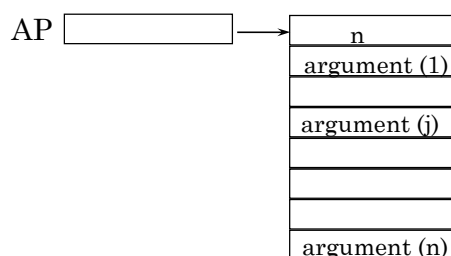
- ⌚ On utilise BP pour adresser la pile car il n'est pas possible d'utiliser SP
- On sauve BP car le sous programme peut avoir lui même été appelé par un autre S.-P.
- ⌚ RETn à deux effets
 - retour classique
 - ajoute 'n' au pointeur de pile ce qui a pour effet de restaurer SP

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 226

Mécanismes de transmission des paramètres

- **Par registre spécialisés :**

- Structure de donnée spéciale qui contient tous les paramètres.
- Ce mécanisme est réalisé par logiciel
- il nécessite l'utilisation d'un pointeur d'argument qui contient l'adresse de base de la structure



Mécanismes de transmission des paramètres

- **Exemple :**

- on peut définir une structure en assembleur qui contient les arguments

```
Struct_Arg STRUCT
    Nbarg dw    2
    arg1  dw    offset chaîne
    arg2  dw    10
Struct_Arg ENDS
```

Mécanismes de transmission des paramètres

Par liste d'argument située dans le code:

- *Avantages :*

- un programme peut appeler plusieurs S.-P. utilisant tous la même liste d'arguments
- on peut écrire des programmes réentrants

- *NB :*

Si on veut appeler un S.-P. avec d'autres paramètres, on ajoute dynamiquement d'autres arguments à la liste

Si plusieurs S.-P. traitent la même chaîne de caractères , la liste d'argument

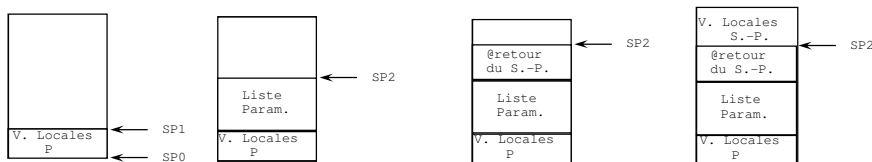
- chaîne
- taille
- Nb arguments

Mécanismes de transmission des paramètres

• Passage par variable locale

- variables qui sont chargées sur la pile au début du S.-P. et qui disparaissent à sa terminaison
- C'est la pile locale attribuée au S.-P.

• Exemple : Gestion des variables locales



1- Démarrage de P avec variables locales

2- P prépare les paramètres sur la pile

3- Appel du programme S.-P. par P

4- Le S.-P. utilise la pile pour des variables locales

Passage de paramètres à une macro

- **Passage par registre**
 - la macro manipule les registres du micro
- **Passage par valeur**
 - déclaration de la macro avec paramètres

Exemple :

```
ADDITION MACRO terme1, terme2, somme
    MOV    AX, terme1
    ADD    AX, terme2
    MOV    somme, AX
ADDITION ENDM
```

```
Appel:  ADDITION #16, R1, R2
```

Plan (suite)

- **Mécanismes d'Interruption**
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259
- **les Périphériques :**
 - Liaison série,
 - Liaison parallèle,
 - Imprimante
- **Exemple de processeurs embarqués**
 - Robots lego
 - Microcontrôleur Beck

Cours Architecture des Ordinateurs

1ère Année

Semestre 2

IUT de Nice- Côte d'Azur
Département Informatique

Marie-Agnès PERALDI-FRATI
Maître de Conférences
map@unice.fr

Organisation de ce cours

- **Cours , TD, TP = 25h**
 - 6 séances de cours
 - 3 séances de TD
 - 7 séances de TP
- **Evaluation :**
 - 2 examens de contrôle continu
 - 1 examen final
- **Dates :**
 - Début du cours : 30 janvier
 - Fin du cours 15 Mai (Examen final):
- **Intervenants :**
 - Erol Acundeger
 - Heikel Batnini
 - Marie-Agnès Peraldi-Frati

Plan

- Mécanismes d'Interruption
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259
- les Périphériques :
 - Liaison série,
 - Liaison parallèle,
 - Imprimante
- Exemple de processeurs embarqués
 - Robots lego
 - Microcontrôleur Beck

Mécanisme d'interruption

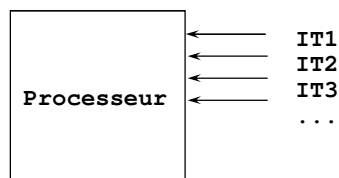
- Principe :
 - interrompre un programme en cours pour traiter une tâche plus urgente
 - prise en compte d'événements asynchrones
- Objectif :
 - Détecter un événement imprévu
alarme, coupure d'alimentation ...
 - Sans avoir à faire une scrutation permanente
analogie avec une sonnerie de téléphone
 - Pour exécuter un sous programme appelé sous-programme d'interruption.

Mécanisme d'Interruption

- IT externe et matérielle
 - provoquée par un périphérique (clavier, port ES, imprimante ...)
 - permet de gérer les conflits d'accès au processeur
- IT externe logicielle
 - IT est générée par un programme. L'instruction assembleur INT
- IT interne trap ou exception
 - IT est générée par le processeur lui même.
Division/0, overflow
- Priorités des interruptions
 - hiérarchisation des IT: classement par ordre de priorités.
 - Priorité IT interne > Priorité IT matérielles > Priorité IT logicielles

Reconnaissance des interruptions

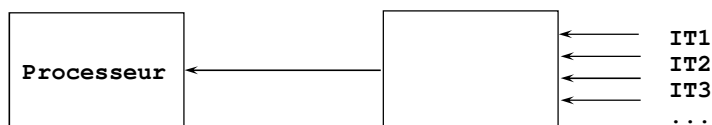
- Différents moyens physique pour déterminer la source d'une IT
- Interruptions **multi-niveaux**:
 - Chaque équipement est relié à une entrée d'IT particulière sur le micro.



- » **Avantage:** solution techniquement simple
- » **Inconvénients:** coûteuse en broches d'entrée du processeur, pas très portable

Reconnaissance des interruptions

- Interruption **ligne unique** :



- » **Avantage** : une seule ligne d'IT sur le processeur
- » **Inconvénient** : scrutation des périphériques pour déterminer le générateur de l'interruption

Reconnaissance des interruptions

- Interruption **vectorisée** :

- 1 signal de demande
- un identificateur qui permet le branchement direct sur le Sous programme d'IT
- le vecteur est déposé sur le bus de donnée
- il est fourni par un composant appelé Contrôleur d'IT
 - » **Avantages** : le microprocesseur reconnaît de suite le périphérique qui a déclenché l'IT
 - » **Inconvénient** : il est nécessaire de gérer des priorités (dépôts simultanés de 2 vecteurs sur le bus)

Détection d'une interruption

- **Interruptions matérielles:**

- Détection sur une ligne du processeur
- Ligne active => déroutement du programme pour traiter l'IT
- le microprocesseur termine l'instruction en cours avant de traiter l'IT
- événement asynchrone

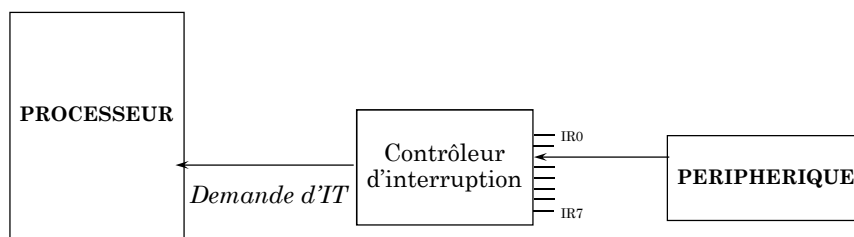
- **Interruptions logicielles**

- invoquée par un processus à un moment précis de son exécution (instruction INT)
- événement synchrone
- peut être assimilé à un appel de sous programme

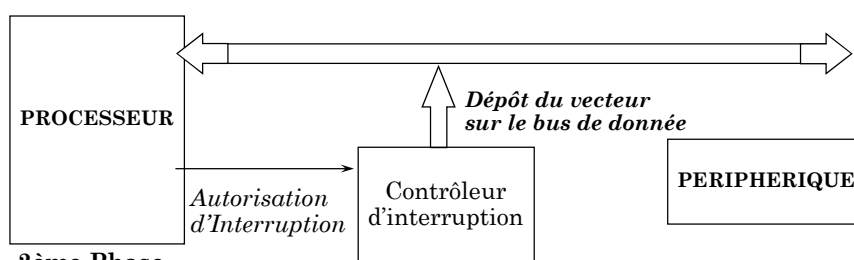
Traitement d'une interruption

- Réception par l'UC d'une demande d'IT interne ou externe
- Acceptation ou rejet par l'UC de cette demande
- Fin de l'instruction en cours
- Sauvegarde de l'état du système
- Forçage du compteur ordinal qui prend l'adresse de la première instruction du SP associé à cette IT
- le SP une fois terminé provoque la restauration des registres et du micro.

Traitement d'une interruption

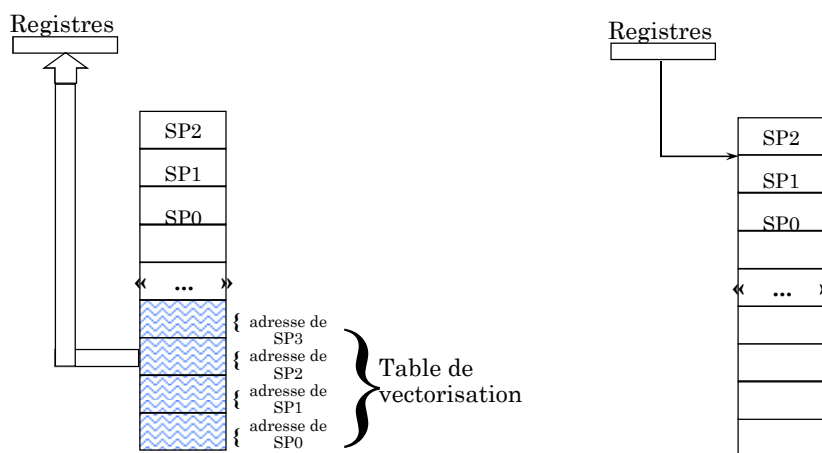


1ère Phase



2ème Phase

Traitement d'une interruption



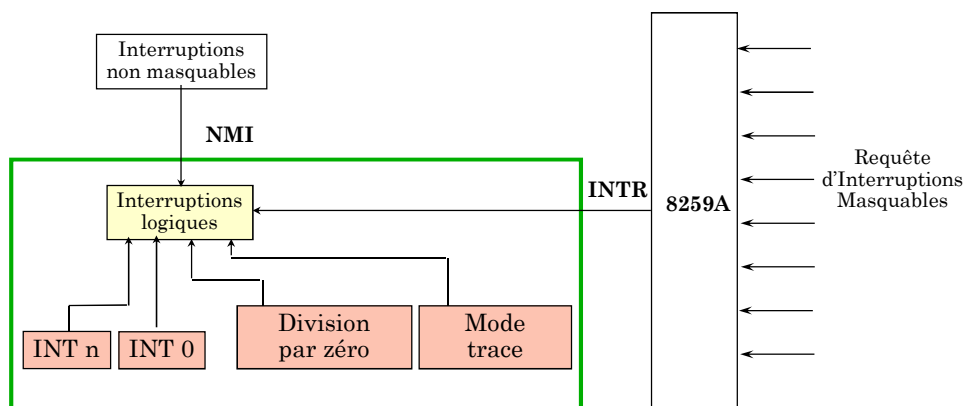
3ème Phase

4ème Phase

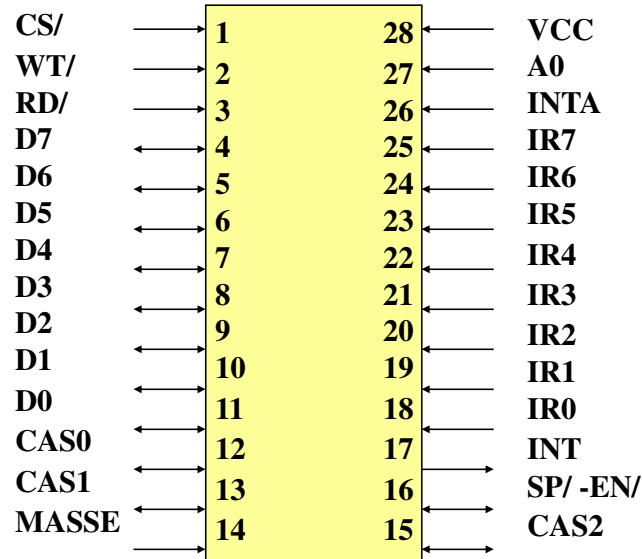
Les interruptions externes matérielles

- Exemple de la famille des processeurs i86
- le processeur comporte 2 broches susceptibles de recevoir des IT
 - NMI (No Masquable Interrupt)
 - » interruption non masquable
 - » défaillance d'alimentation, sortie de boucle infinie,
 - INTR : (Interruptions externes masquables)
 - » générée par un contrôleur d'interruptions lui même connecté aux circuits susceptibles de générer l'IT
 - » le PIC (programmable Interrupt Controller) 8259A
 - » Quand INTR est actif l'état du flag IF du registre d'état conditionne la réponse du CPU
 - » IF = 1 => IT non masquées
 - » IF = 0 => IT masquées

Les différentes sources d'interruptions



Le contrôleur d'interruption 8259A



Le contrôleur d'interruption 8259A

- 8 niveaux d'IT (IR0 -IR7) gèrent les périphériques
- INT et INTA : demande et accusé de réception de l'interruption
- D0 à D7 : Bus de donnée
- Chip Select: Sélection du PIC afin qu'il puisse être accédé
- A0, WT et RD : Connexion au bus d'adresse pour lecture et écriture sur le PIC
- CAS0-1-2 : Multiplication des IT.
- SP-EN: PIC en mode maître ou esclave

Le contrôleur d'interruption 8259A

Découpage par fonctions :

- logique de contrôle => connexion au micro-processeur
- Buffer de données => connexion au bus de données
- logique de lecture /écriture
- Cascade - Comparateur : Gestion de PIC en cascade
- registre des IT en service **ISR**
- registre de demande d'IT **IRR**
- résolveur de priorité lorsque plusieurs PIC sont en cascade
- Registre du masque d'IT **IMR**, mémorise les IT interdites

Le contrôleur d'interruption 8259A

Traitement d'une interruption:

- Demande du périphérique IRQ0-7
- Réception par le PIC + positionnement IRR
- Evaluation de la demande (Priorité)
- PIC informe le μ C => INT
- Le μ C prend connaissance du flag IF + contexte => INTA
- Réception de INTA par le PIC
- Positionnement de ISR et IRR
- PIC => bus de donnée le type de l'IT
- Le μ C déduit son traitement => Table des vecteurs à l'indice $4 * N^{\circ}IT$
- Branchement du sous programme
- Reprise de la tâche interrompue

Traitement d'une interruption:

- a) Type = 14h
 b) Type * 4 = 50h -> Offset dans la Table des vecteurs d'IT
 c) Table des vecteurs d'IT

00054	XX
00053	20
00052	00
00051	34
00050	56
0004F	XX
...	...
00002	XX
00001	XX
00000	XX

} ISR Adresse
 = 2000:3456
 =23456

- d) Flags --> pile

CS:IP--> pile
 0 --> IF
 2000:3456--> CS:IP

e) Procédure ISR

23456 PUSH AX
 PUSH BX

23874 IRET

- f) Pile --> CS:IP
 Pile --> Flags

Les interruptions logicielles

IT logicielles peuvent être provoquées

• internes

- flag OF=1 indique un overflow => interruption de type 4 est générée par l'instruction spéciale INTO.
- résultat d'une division est de taille supérieur à s destination => interruption de type 0 est déclenchée
- le flag TF a été mis à 1 => le CPU génère une IT de type 1 après chaque instruction ce qui permet de faire du pas à pas

• externes

- appel de l'instruction INT <numéro de l'IT>

Les interruptions logicielles

Les interruptions logicielles

- Fonctions du DOS
- Fonctions du BIOS
- Routines à programmer

Interruptions logicielles

- Appel aux fonctions du BIOS (Basic Input Output System)
 - Interface normalisée (OS)
 - Compatible PC
 - Fonctions élémentaires
- **Exemple :**
 - Accès écran
 - Accès imprimante
 - Consultation date du système
 - ...

Interruptions logicielles

- **Fonction vidéo du Bios**
 - Une fonction BIOS => une routine d'IT
 - une ISR = ensemble de fonctions lié à un périphérique
 - **Exemple:** interruption 10h => Routine vidéo
 - » Fonction 00h: sélection du mode vidéo monochrome couleur
 - ...
 - » Fonction 01h : définition de la forme du curseur
 - » ...défilement des pages ...
 - **Appel :** Fonction 0Ah de l'IT 10h (Affiche un caractère à l'écran sur le curseur)

```
MOV AH, 0Ah
MOV AL, 'x'
INT 10h
```

Interruptions logicielles

INTERRUPTION 1AH : ROUTINE DE GESTION DE LA DATE ET DE L'HEURE DU SYSTEME

Fonction 00h : Lecture de l'heure système (unité BIOS)

Entrée :

AH 00h

Sortie :

AL 1 si l'horloge a dépassé 24 heures depuis la dernière invocation de la routine, 0 dans le cas contraire

CX Mot de poids fort du compteur

DX Mot de poids faible du compteur

L'heure gérée par le bios est incrémentée à chaque top d'horloge. La valeur est exprimée en nombre de tops d'horloge depuis minuit.

Interruptions logicielles

- Appel aux fonctions du DOS (Disk Operating System)
 - Application Program Interface
 - DOS -> 200 fonctions
 - Fonctions regroupés dans L'INT 21h, fonctions multipléxées
 - AH contient le numéro de la fonction

- Exemple :

- Création d'un fichier,
- Affichage de caractères à l'écran
- ...

- Appel : Ecriture de "IUT de Nice" sur écran

```
DS:DX contient l'adresse de la donnée
MOV DX, 0h
MOV AH, 09h ; fonction d'affichage d'une chaîne à l'écran
INT 21h
```

Interruption Logicielles

- Interruptions déclenchées par un programme
- Fonctions d'interruptions accessibles
 - Fonctions disponibles 60-67
 - Adresses des vecteurs 180 à 19F
 - Adresse des procédures contenues dans ces vecteurs

Plan

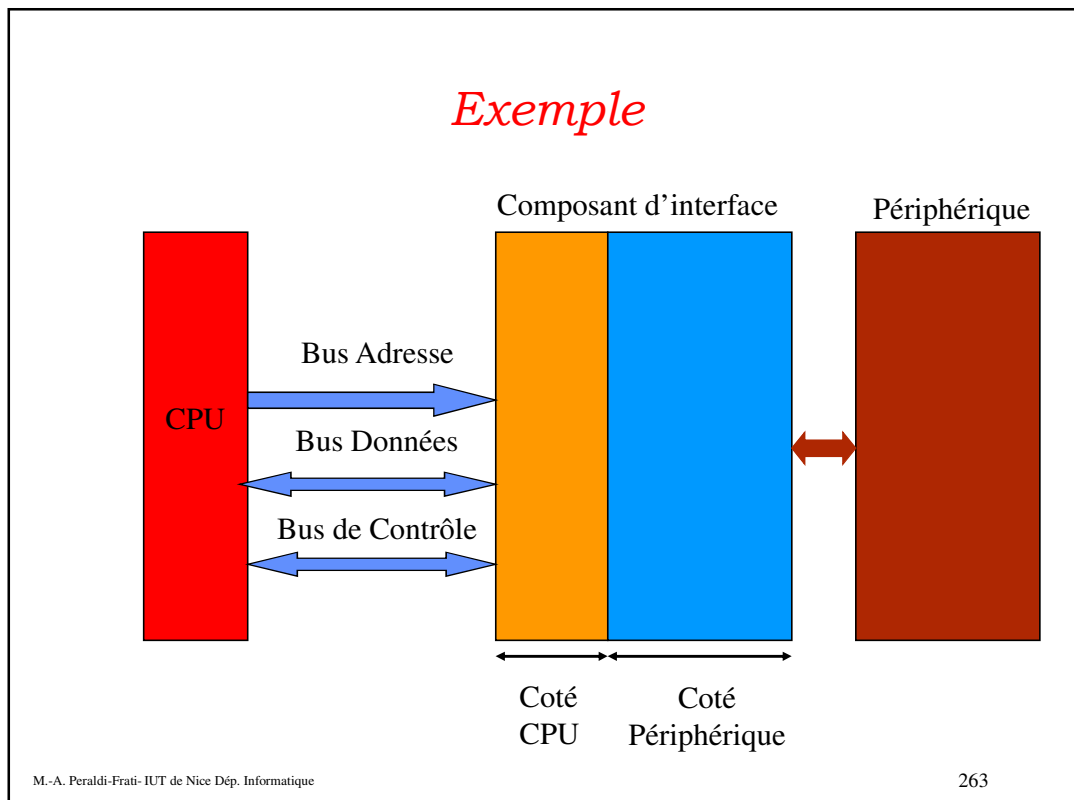
- Mécanismes d'Interruption
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259
- les Périphériques d'Entrées Sorties
 - Liaison série,
 - Liaison parallèle,
 - Imprimante

Les entrées/sorties

- Comment le CPU réalise les transactions d'entrée sortie
 - **Stratégie** qui détermine quelles données sont transférées
 - Le **circuit d'interface** qui réalise le transfert hors et dans le PC
 - Les **composants d'E/S** qui convertissent les données dans les deux sens.
- **Exemple:**
 - Un ordinateur connecté à un clavier et à un écran télé

Exemple

- Programme data transfert
 - Stratégie* de transmission des données vers le périphérique via
 - » son port de sortie (pour écran) ou
 - » son port d'entrée (pour clavier)
 - *Port* est un pont entre le périphérique et le PC
 - » Met en forme les données (série <->Parallèle)
 - » Ajoute des bits de synchronisation
 - *Connexion* se fait par une « twisted pair » (2 fils)
 - » Données internes transmises au port sont //
 - » Données transmises par le port au périphérique se fait en série



Handshaking et Buffering

- Les transferts de données d'E/S sont de 2 types :
- **Boucle ouverte** : les données sont transférées et leur réception correcte est supposée après un certain délai
 - » Exemple du service de mail
 - » Généralement suffisant
- **Boucle fermée** : il y a **accusé de** réception des messages (acknowledgement)
 - » Exemple de procédure d'approche d'un aéroport

Transfert en boucle ouverte

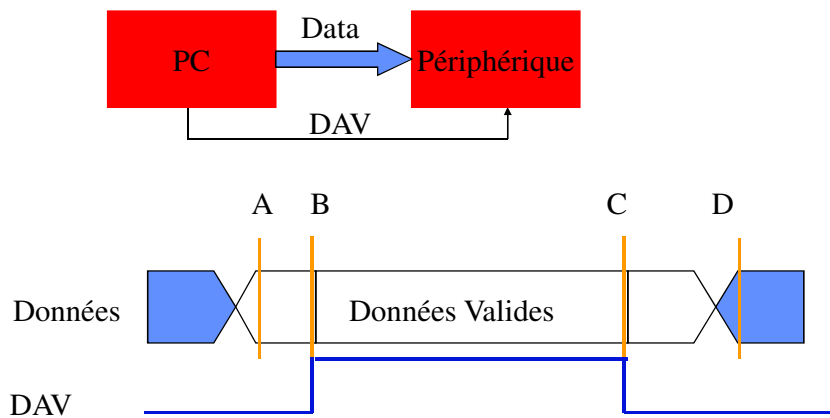


Diagramme temporel

- A : données valides
- B: le PC signale au périphérique la disponibilité des données
- C: le PC signale au périphérique que les données ne sont plus valides
- D: Données invalides

Transfert en boucle fermée

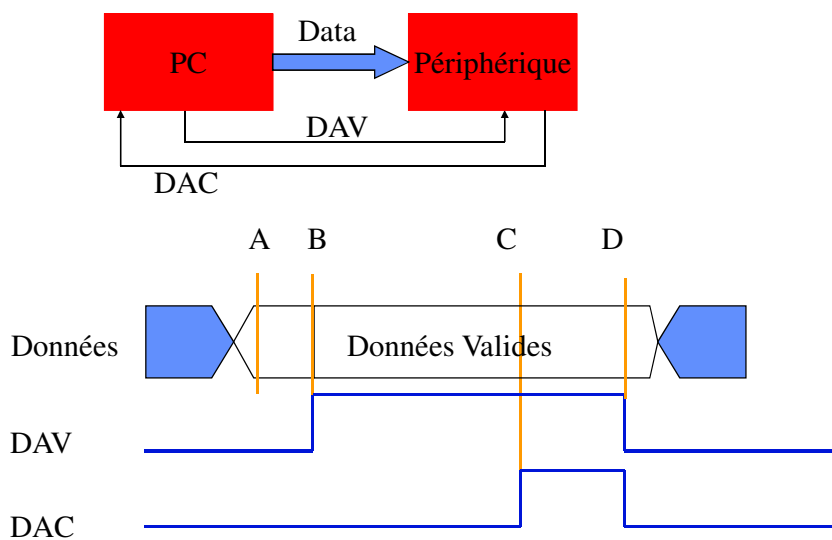
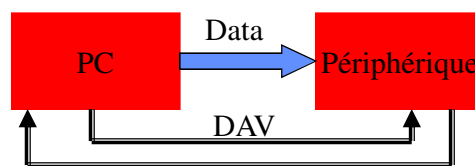
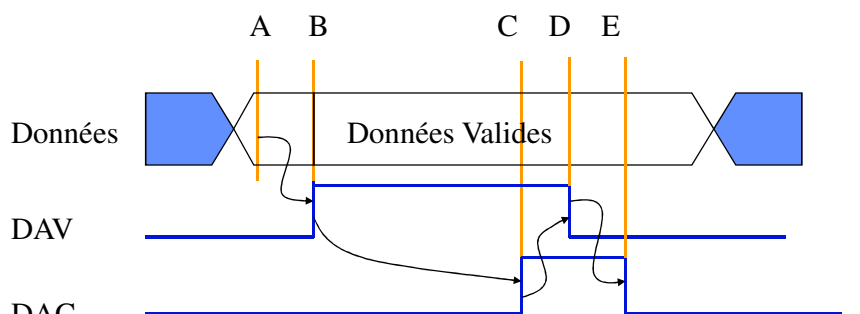


Diagramme temporel : *Handshaking*

Transfert en boucle fermée



- Handshaking :
 - utile pour les périphériques lents
 - Permet de valider la réception des données
- Fully interlocked data transfert
 - Accusé de réception de DAC lui même



Transfert en boucle fermée

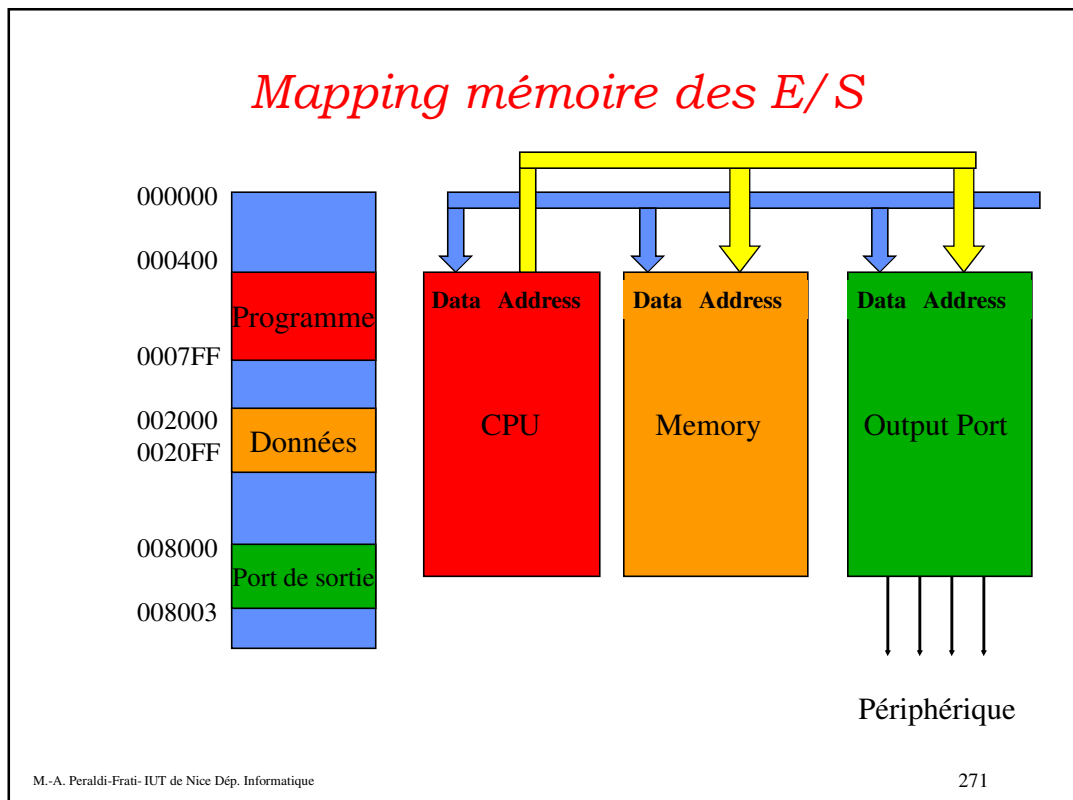
- Problème de défaut d'équipement qui ne renverrait pas le DAC
 - Timers intégrés
 - Opération interrompue

Bufferisation des E/S

- Temps de traitement des interfaces ?
 - Rapide => problème de maintien des données
 - Lent => perte de temps à accéder aux données
 - Exemple : problème du temps moyen d'attente chez les médecins
- Buffer d'interface => FIFO
 - Détermination de la taille de la FIFO
- « Buffering »: stockage des données avant leur utilisation

Programmation des E/S

- Les E/S sont initiées par programme
 - Instructions spécifiques
 - » OUT <port>
 - E/S mappées en mémoire
 - » Les entrées sorties sont considérées comme des extensions mémoire
 - » Une partie de l'espace mémoire est réservée aux entrées sorties



Mapping mémoire des E/S

- Le port est connecté au périphérique
- Les données sont transmises en écrivant dans la mémoire à partir de 008000.
- Exemple d'un programme qui écrit les 128 caractères contenus dans la zone de données.
- La taille de la mémoire du port de sortie est < à 128 octets
- **Problème** : Si le périphérique n'est pas suffisamment rapide, les données seront écrasées.
- **Solution** : demander au périphérique si il est prêt à recevoir des données avant envoi (Handshaking).

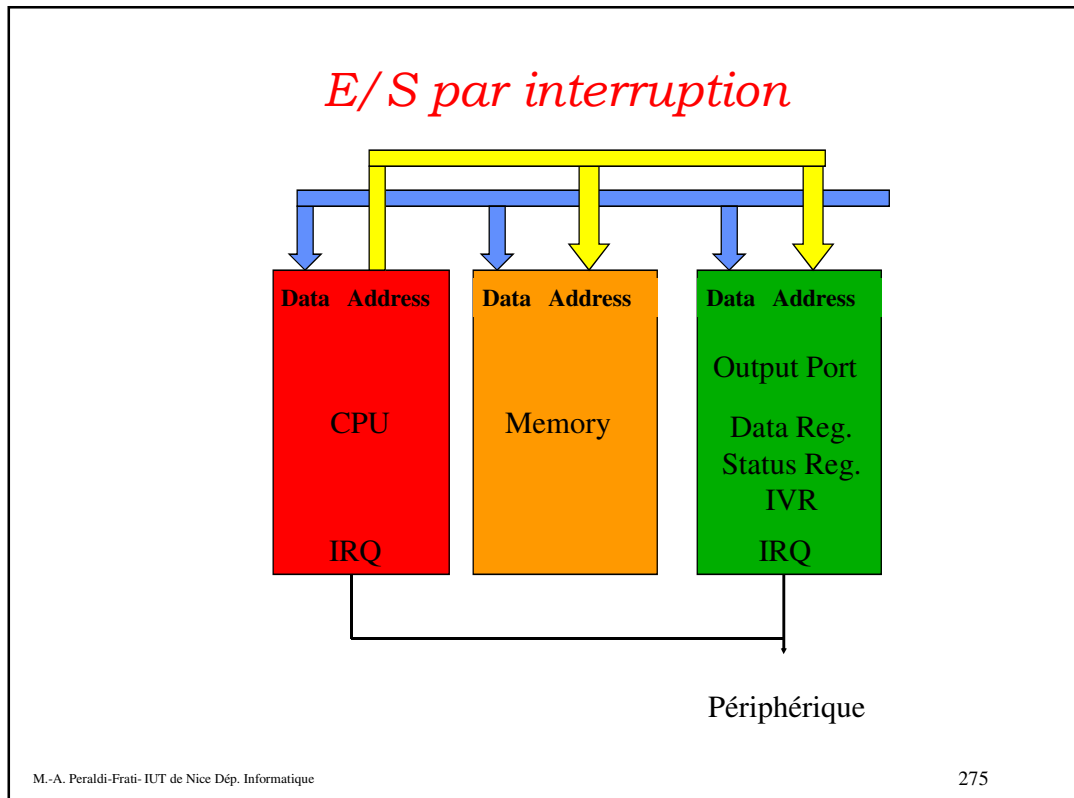
Exemple de handshaking pour la RS232

```
do {  
    do {  
        /* Attente du canal prêt pour transmettre */  
        retour = EtatCom(&port,0X40);  
    } while (retour !=0);  
    retour = EmetCar(&port,&Chaine[i]);  
  
    if (retour !=0)  
        printf ("\nErreur de transmission !!!!");  
    else  
        i=i+1;  
} while (Chaine[i-1] != '\0');
```

Exemple de handshaking pour la RS232

- Do ... while : Pooling de l'état

```
do {  
    /* Attente du canal prêt pour transmettre */  
    retour = EtatCom(&port,0X40);  
} while (retour !=0);
```
- Pooling :
 - Interrogation du périphérique
 - Inefficace en terme de gestion du temps pour du multitâche
- Solution plus efficace :
 - Gestion par interruption

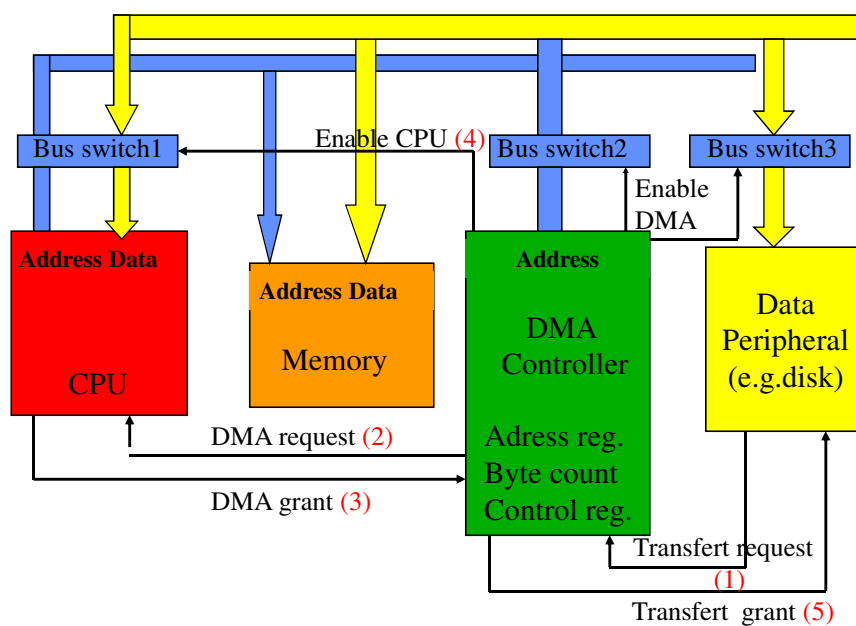


- E/S par interruption*
- Mécanisme d'interruption déjà vu en cours
 - Mécanisme plus efficace
- M.-A. Peraldi-Frati- IUT de Nice Dép. Informatique 276

Direct Memory Access

- Données transmises de la mémoire du PC vers le périphérique sans passer par les registres du CPU
- Le CPU donne l'ordre de transfert de blocs mémoire
- Le composant d'interface DMAC
 - Réalise le transfert
 - Signale la disponibilité des données
 - Peut utiliser les bus de données et d'adresse le temps des transferts

DMA (suite)



Flût de contrôle d'un DMA

- Un périphérique veut que soit fait une opération d'ES
 1. il demande au DMA une requête de transfert
 2. le DMAC envoie une requête au CPU pour l'utilisation des bus
 3. un DMA grant revient qui l'autorise à utiliser les bus
 4. Le cycle DMA peut se faire
 - » switch 1 est ouvert accès au CPU inhibé
 - » switch 2 et 3 sont fermés
 5. Le DMAC envoie un transfert grant vers le périphérique qui peut maintenant écrire ou lire a mémoire
 6. Le DMAC possède des registres qui maintiennent la dernière adresse mémoire lue ainsi que la taille des données transférées.

Interface série RS232

- Transfert bit à bit des données sur une ligne simple
- Utilisée pour transmettre sur quelques mètres
- Interfaçage du CPU à la liaison série se fait par une UART (Universal Asynchronous Receiver Transmitter)
- A l'origine, tous les compatibles PC possèdent 2 ports séries: COM1 et COM2. L'un d'entre-eux se présente sous la forme d'une prise DB9 mâle et le deuxième, sous la forme d'une DB25 mâle.

Brochage des connecteurs

Broche DB9	Broche DB25	Nom	
1	8	DCD	←
2	3	RX	←
3	2	TX	→
4	20	DTR	→
5	7	GND	←
6	6	DSR	←
7	4	RTS	→
8	5	CTS	←
9	22	RI	←

Attributs des signaux

- **DCD (Data Carrier Detect)**: cette ligne est une entrée active haute. Elle signale à l'ordinateur qu'une liaison a été établie avec un correspondant.
- **RX (Receive Data)**: cette ligne est une entrée. C'est ici que transitent les informations du correspondant vers l'ordinateur.
- **TX (Transmit Data)**: cette ligne est une sortie. Les données de l'ordinateur vers le correspondant sont véhiculées par son intermédiaire.
- **DTR (Data Terminal Ready)**: cette ligne est une sortie active haute. Elle permet à l'ordinateur de signaler au correspondant que le port série a été libéré et qu'il peut être utilisé s'il le souhaite.
- **GND (GrouND)**: c'est la masse.

Attributs des signaux

- **DSR (Data Set Ready)**. Cette ligne est une entrée active haute. Elle permet au correspondant de signaler qu'une donnée est prête.
- **RTS (Request To Send)**: cette ligne est une sortie active haute. Elle indique au correspondant que l'ordinateur veut lui transmettre des données.
- **CTS (Clear To Send)**: cette ligne est une entrée active haute. Elle indique à l'ordinateur que le correspondant est prêt à recevoir des données.
- **RI (Ring Indicator)**: cette ligne est une entrée active haute. Elle permet à l'ordinateur de savoir qu'un correspondant veut initier une communication avec lui.

Fonctionnement liaison Série

- **La communication série nécessite trois fils au minimum:**
 - une masse pour référencer les signaux,
 - un fil émetteur
 - un fil récepteur.
 - liaison série est full-duplex, c'est à dire que l'on peut émettre et recevoir en même temps;

Fonctionnement liaison série

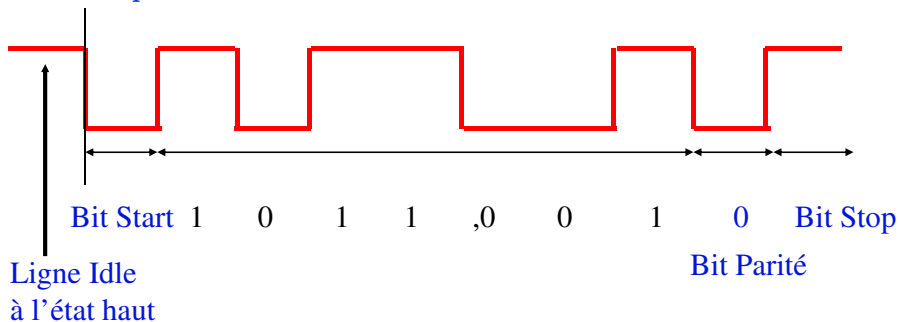
- la liaison série est totalement asynchrone.
- Aucune horloge n'est transmise.
 - Il faut donc se mettre d'accord sur la vitesse de transfert des bits
 - rajouter des bits de synchronisation.
- Vitesse de transmission :
 - de 300 à 9600 bauds
 - vitesse doit être fixée par les deux parties communicantes

Format des trames

- Longueur du mot : La transmission des données peut se faire suivant plusieurs formats (7 ou 8 bits)
- Contrôle de parité
 - pouvant être gérée comme paire ou impaire
- Bit de start :
 - une trame commence par 1 bit de *start* ("0" logique)
 - permet de re-synchroniser le récepteur
- Bit de stop :
 - signale la fin de la transmission.
 - il peut y avoir 1, 1.5, ou 2 bits de stop (« 1 » logique).

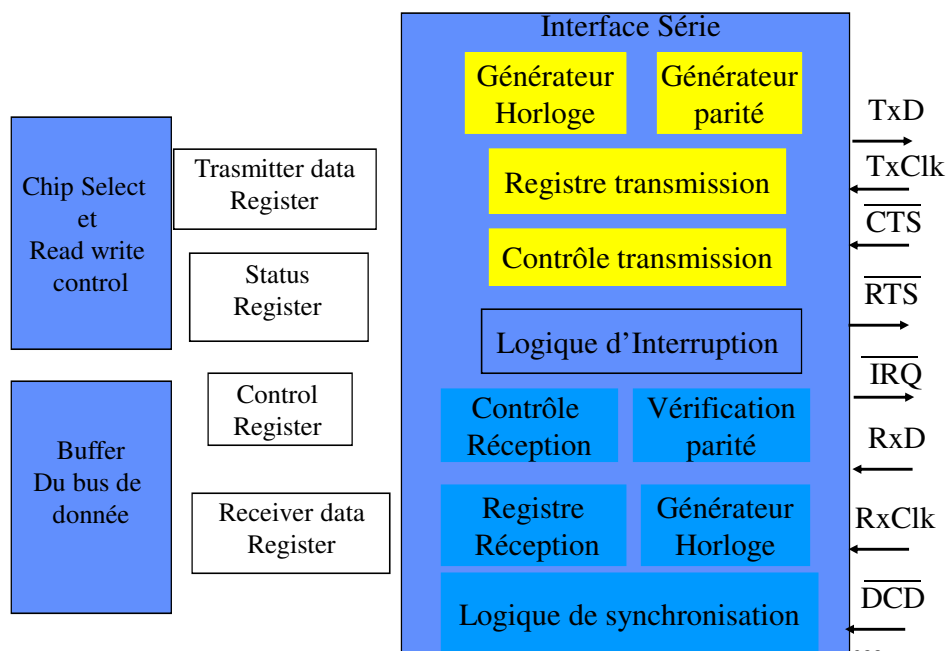
Format des trames: exemple

- Exemple : Transmission du caractère M 100 1101



Durée d'un bit dépend du taux de transfert en bit/seconde:
 Avec un taux de transfert de 9600 bauds le temps de transmission
 = $1/9600 = 0,1042\text{ms}$

Fonctionnement de l'UART



Fonctionnement de l'UART

- Registres internes :
 - 4 registres internes : TDR, RDR, CR, SR
 - Seuls 2 sont accessibles par le CPU en écriture

Les protocoles de transmission

- On ne peut réussir une transmission qu'à partir du moment où l'émetteur et le récepteur se sont entendu sur la vitesse, le nombre de bit de stop, la longueur du mot et la parité. A ce niveau là, savoir à quel voltage correspond un état haut n'a aucune importance.
- D'une manière générale, la parité est toujours présente car elle permet de détecter la plus grande partie des erreurs de transmission.

Programmation du port série

- L'accès aux registres contrôlant les ports série se fait par l'intermédiaire de l'interruption DOS 14h. A cette IT correspond 4 fonctions permettant de configurer et de contrôler l'interface série,
- Fonction 0x00: Initialisation de l'interface série
- Fonction 0x01: Emission de caractères
- Fonction 0x02: Réception de caractères
- Fonction 0x03: Tester état du port série

Fonction 0x00: Initialisation de l'interface série

Cette fonction permet de fixer le protocole de transmission.

Entrée: AH = 0x00
DX = Numéro de l'interface série
0x00 = COM1
0x01 = COM2
AL = Paramètres de configuration
Bits 0-1: longueur du mot
10 = 7 bits
11 = 8 bits
Bit 2: nombre de bits de stop
0 = 1 bit de stop
1 = 2 bits de stop
Bit 3-4: contrôle de parité
00 = aucun
01 = impair
11 = pair
Bit 5-7: vitesse de transmission
000 = 110 bauds
001 = 150 bauds
010 = 300 bauds
011 = 600 bauds
100 = 1200 bauds
101 = 2400 bauds
110 = 4800 bauds
111 = 9600 bauds

Sortie: AH = Etat de l'interface série
Bit 0: données prêtes
Bit 1: données effacées
Bit 2: erreur de parité
Bit 3: violation de protocole
Bit 4: interruption détectée
Bit 5: transmission Hold Register vide
Bit 6: transmission Shift Register vide
Bit 7: time out (le périphérique ne répond pas)

AL = Etat du modem
Bit 0: (delta) modem prêt à émettre
Bit 1: (delta) modem activé
Bit 2: (delta) sonnerie
Bit 3: (delta) liaison établie
Bit 4: modem prêt à émettre
Bit 5: modem activé
Bit 6: sonnerie
Bit 7: liaison établie

Exemple de programmation

le réglage du protocole à 1200 bauds, 7 bits et parité paire.

```
struct BYTEREGS {  
    /* Registres vus comme 2 registres de 8 bits */  
    unsigned char al, ah, bl, bh, cl, ch, dl, dh;  
};  
Union REGS {  
    struct BYTEREGS h;  
    struct WORDREGS x;  
} pregs;  
/* fonction 00h: réglage du protocole */  
pregs.h.ah = 0x00;  
/* 7 bits, parité paire, 1200 bauds */  
pregs.h.al = 0x9A;  
/* COM1 */  
pregs.x.dx = COM1;  
int86(0x14, &pregs, &pregs); /* IT DOS 14 */
```


- Initialisation de la liaison série
- Boucle
 - Traitements sur caractères saisis :
 - » Tant que caractère différent de '\n' et longueur chaîne < 70 caractères
 - Saisie du caractère au clavier
 - Affichage du caractère à l'écran
 - Stockage dans une chaîne
 - » Si caractère == '\n'
 - Envoi de la chaîne par la liaison série
 - Si erreur d'envoi arrêter l'envoi
 - Traitements sur caractères reçus
 - » Lire sur la liaison série
 - » Si erreur de réception arrêter réception
 - » Tant que caractère différent de '\n'
 - Affichage à l'écran du caractère
- Fin de boucle

Interface parallèle

- Le port parallèle des PC et compatibles se présente sous la forme d'une prise DB25 femelle,
- Ce port a été pensé pour communiquer avec une imprimante, ses signaux sont très liés à ce périphérique

Brochage des connecteurs

Broche DB25	Nom
1	/STROBE
2 --9	D0 – D7
10	/ACK
11	BUSY
12	PE
13	SELECT
14	/AUTOFEED
15	/ERROR
16	/INIT
17	/SELECT IN
18-25	MASSE

Attributs des signaux

- **STROBE**: cette ligne active basse indique à l'imprimante que des données sont présentes sur les lignes D0 à D7 et qu'il faut les prendre en compte.
- **D0 à D7**: c'est le bus de données qui véhicule la valeur du caractère à imprimer. On ne peut écrire sur ce port, à moins d'avoir un port parallèle étendu (c'est le cas pour les ports de type ECP/EPP).
- **ACK**: l'imprimante met à 0 cette ligne pour indiquer à l'ordinateur qu'elle a bien reçu le caractère transmis et que ce dernier peut continuer la transmission.

Attributs des signaux

- **BUSY**: cette ligne est mise à 0 par l'imprimante lorsque son buffer de réception est plein. L'ordinateur doit attendre que cette ligne revienne à 1 pour recommencer à émettre.
- **PE**: signifie " paper error ".
- **SELECT**: cette ligne indique à l'ordinateur si l'imprimante est "on line" ou "off line".
- **AUTOFEED**: lorsque ce signal est à 1, l'imprimante doit effectuer un saut de ligne à chaque caractère "return" reçu. En effet, certaines imprimantes se contentent d'effectuer un simple retour du chariot en présence de ce caractère.

Attributs des signaux

- **ERROR**: indique à l'ordinateur que l'imprimante a détecté une erreur.
- **INIT**: l'ordinateur peut effectuer une initialisation de l'imprimante par l'intermédiaire de cette ligne.
- **SELECT IN**: l'ordinateur peut mettre l'imprimante hors ligne par l'intermédiaire de ce signal.
- **MASSE**: c'est la masse du PC.

Performances

- Les ports plus récents, de type EPP (pour Enhanced Parallel Port, développé par Xircom, Zenith et Intel), permettent d'atteindre un débit de 2Mo/s
- il permet néanmoins la réception de périphériques tels que des lecteurs de CD-ROM, disques durs, Zip
- En plus d'un débit supérieur, les ports EPP sont bidirectionnels.

Programmation

- *Se fait au travers de ports d'entrée/sortie, ports d'E/S, adresse d'E/S,*
- **Voici une liste de quelques adresses de base courantes:**
 - 060h - clavier
 - 170h/376h - contrôleur IDE secondaire
 - 1F0h/3F6h - contrôleur IDE primaire
 - 220h - carte son
 - 300h - carte réseau
 - 330h - carte adaptatrice SCSI
 - 3F2h - contrôleur de lecteur de disquettes
 - 3F8h - COM1
 - 3E8h - COM2
 - 3E8h - COM3
 - 2E8h - COM4
 - **378h - LPT1**
 - **278h - LPT2**

Programmation

- trois registres seulement sont nécessaires au contrôle total des signaux.
 - **Lignes de données (378h)** : écriture/lecture
 - **Etat de l'imprimante (379h)** : lecture
 - **Commande de l'imprimante (37Ah)** : écriture

Plan

- **Mécanismes d'Interruption**
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259
- **les Périphériques d'Entrées Sorties**
 - Liaison série,
 - Liaison parallèle,
 - Imprimante

Plan

- **Mécanismes d'Interruption**
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259
- **les Périphériques d'Entrées Sorties**
 - Liaison série,
 - Liaison parallèle,
- **Système Multitâches**
 - Structure, fonction d'un OS multitâche
 - Gestion des tâches
 - Relations entre tâches
 - Mécanismes d'exclusion mutuelle
 - Synchronisation
 - Communication
- **Exemples de systèmes multitâches**
 - Exécutif temps-réel RTC

Système multitâche

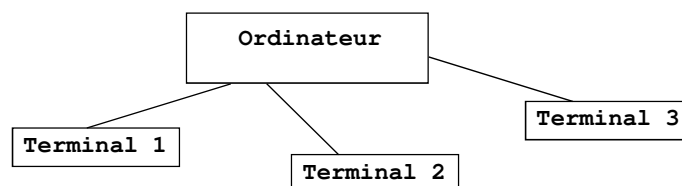
- **Systèmes classiques : multitâches au niveau de l'OS**
 - parallélisme réglé par le système d'exploitation
 - transparent à l'utilisateur
 - destiné à optimiser les ressources du processeur
- **Systèmes temps réel : programmation multi-tâches**
 - décomposition en tâches par l'utilisateur
 - fonction des contraintes de temps
 - priorité déterminée
 - séparation entre programme de commande (1er plan) et programmes peu urgents (tâche de fond)

Systeme multitache

- Point de vue de l'utilisateur :
 - nature des interactions
- Point de vue des traitements
 - prise en compte des traitements par l'ordinateur
 - impact sur les performances
- Modes de Traitement
 - séquentiel
 - en temps partagé
 - Multi-tâches – multi-utilisateurs

Systeme multitache

- Motivation
 - servir équitablement différents utilisateurs
 - Accès au CPU par plusieurs utilisateurs
 - Comment partager le CPU ?



- le système doit
 - » lire les commandes tapées au clavier
 - » afficher les résultats à l'écran

Premier type de partage Traitement séquentiel

- on ne donne l'accès qu'à un seul utilisateur

Lire1	Traiter1	Imprimer1	Lire2	Traiter2	...
-------	----------	-----------	-------	----------	-----

=> délais d'exécution importants

=> mauvaise utilisation des ressources

- Traitement séquentiel très rarement utilisé
- Si un utilisateur oublie de taper sur le clavier => les autres doivent patienter ...

Processus séquentiel

S_occuper_terminal(int a){

- lire une commande du terminal a
- exécuter la commande
- afficher le résultat de la commande sur le terminal a

Le séquentiel s'exprime alors de la façon suivante :

Système_séquentiel(){

```
while (true){  
    S_occuper_terminal(1);  
    S_occuper_terminal(2);  
    S_occuper_terminal(3);  
}
```


Processus séquentiel

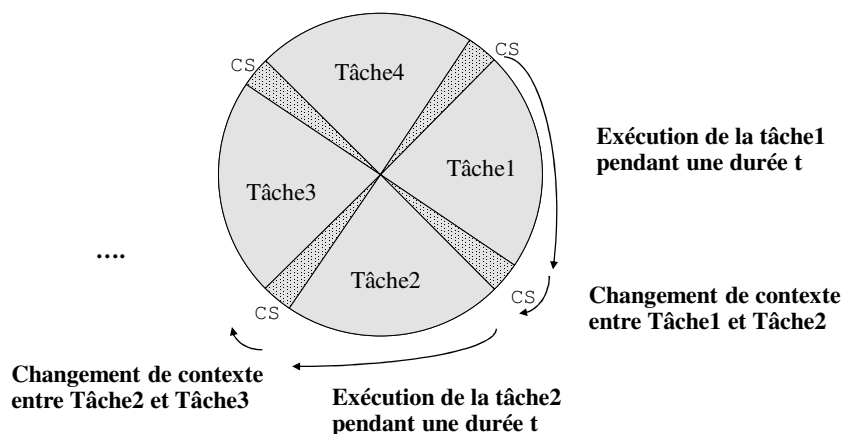
- Cette **solution n'est pas satisfaisante** car:
 - elle exprime les entrées sorties en mode boucle d'attente
 - » pas très efficace en terme d'utilisation des ressources
 - » pendant qu'un caractère est attendu sur T1 on pourrait afficher sur T2
 - ne tient pas compte des temps inégaux des commandes. Partage non équitable du processeur
 - ne tient pas compte du temps d'exécution de chaque terminal

- **Autre solution pour remédier à cela:**

Traitement en Temps partagé

Traitement en temps partagé

- Allocation du processeur par tranches de temps fixes
- Système multi-utilisateurs/multi-tâches



Processus en temps partagé

- Diminution du délai d'exécution pour les petits travaux
- Ne convient pas aux travaux lourds
- Exemple ...
- **Autre solution pour remédier à cela:**
 - Entrées /sorties en mode interruption
 - Modification de l'expression
 - » Programmation en mode interruption
 - » La programmation n'est plus séquentielle
 - Transformation des programmes ...

Traitements multi-tâches multi-utilisateurs

- Plusieurs tâches et utilisateurs
- Exécution contrôlée par un exécutif en fonction
 - de priorité
 - d'événements extérieurs
 - de contraintes de temps
- L'exécutif reprend le contrôle à l'occasion
 - de trappes
 - d'interruptions

Multitâche

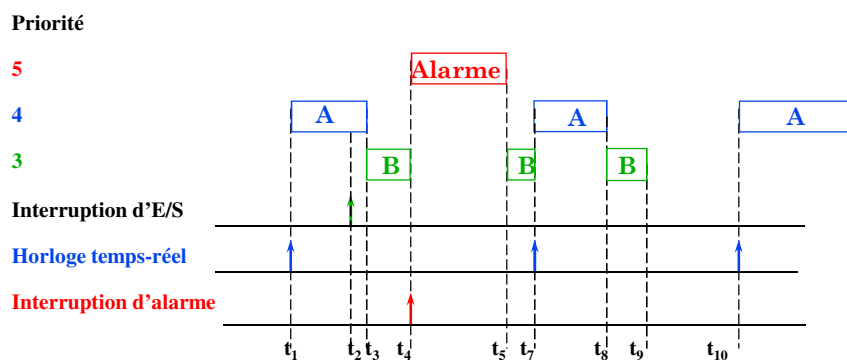
```
Process_terminal1(){  
    while(true){  
        s'occuper_terminal(1);  
    }  
}  
Process_terminal2(){  
    while(true){  
        s'occuper_terminal(2);  
    }  
}  
Process_terminal3(){  
    while(true){  
        s'occuper_terminal(3);  
    }  
}
```

Multitâche

- Chaque processus n'a plus qu'une partie du problème initial à résoudre
- un problème non séquentiel est ramené à plusieurs sous problèmes séquentiels
- Ceci est rendu possible grâce à la notion de **noyau**
- Avec un seul processeur un seul processus s'exécute à la fois
- le noyau exécute à tour de rôle chacun pour donner l'impression que chaque processus dispose de son propre processeur

Traitement multi-tâche multi-utilisateurs

- Exemple d'exécution multi-tâche
 - l'exécutif achève la tâche la plus urgente
 - valeur des priorités => priorité croissante
 - traitement des événements urgents à l'instant désiré

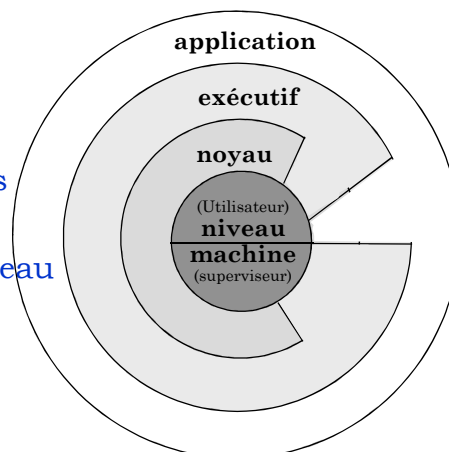


Systeme temps-réel

- Système destiné à l'automatisation
- Couplage étroit avec le monde extérieur
- Contraintes sévères de délai d'exécution (milliseconde)
- Tâches cycliques
 - réglage de moteurs
 - échantillonnage de capteurs
- Traitements asynchrones
 - réponse à des événements externes
 - gestion d'alarme
- Systèmes qui fonctionnent sans opérateurs (en ligne)

Structuration d'un système d'exploitation

- Il faut assurer un maximum d'indépendance entre les différentes parties du système
- structuration en couches ou en «pelures d'oignons»
 - **processeur**
 - **noyau**
 - **exécutif**
- regroupement des fonctions en famille
- chaque groupe est à un niveau d'abstraction
- règle de visibilité



Structuration d'un système d'exploitation

- Exécution de nombreuses tâches en parallèle
- Problèmes de
 - partage de ressource
 - communication
 - synchronisation
- Solution
 - créer des routines spécialisées communes aux tâches

=> **noyau temps-réel**

Structuration d'un système d'exploitation

Le noyau

- Donne à l'exécutif l'accès à une machine virtuelle
- Réside en mémoire centrale
- Assure
 - la gestion des tâches
 - l'allocation du processeur
 - la gestion de la mémoire
 - la gestion des interruptions et des trappes
 - la synchronisation et l'exclusion mutuelle
 - la communication entre tâches
 - la communication avec l'extérieur

Pseudo parallélisme et quasi parallélisme

Partage d'un processeur par plusieurs processus =>

- **Pseudo-parallélisme :**
 - à tout moment un processus est en exécution
 - la commutation de processus se fait à l'insu des processus
 - c'est le schéma utilisé pour un langage temps réel
- **Quasi-parallélisme :**
 - à tout instant un processus est en exécution
 - la commutation de processus se fait à la demande du processus actif par une instruction particulière
 - utilisé dans le contexte de la simulation

Pseudo-parallélisme : Rôle du noyau

- Noyau d'un OS = ensemble de procédures
- Un noyau est un allocateur de processeur
- Le noyau gère la commutation de processus
- Structures de données gérées par le noyau:
 - *Descripteur de processus*
 - » ensemble des variables propres au processus
 - » priorité et état
 - » contenu des registres du processus
- Ces informations constituent le *contexte d'exécution*
 - ceci permet aux procédures d'être réentrantes
 - possibilité d'exécuter une procédure simultanément par plusieurs processus

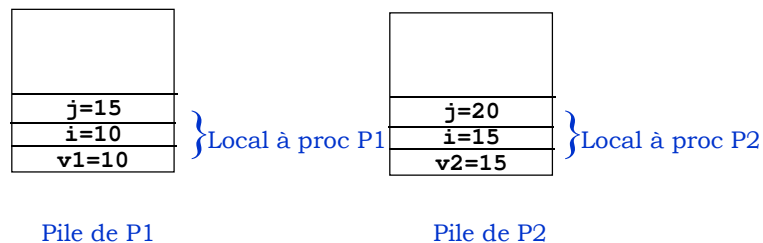
Pseudo-parallélisme: Rôle du noyau

- Exemple de procédure réentrante

```
processP1(){  
    int v1;  
    v1=10;  
    procédure_réentrante(v1)  
}  
processP2(){  
    int v2;  
    v2=15;  
    procédure_réentrante(v2)  
}  
  
procédure_réentrante(int i){  
    int j;  
    j=i+5;  
}
```

Pseudo-parallélisme: Rôle du noyau

- Etat de la pile des processus P1 et P2



- Dans les descripteurs de processus chaque descripteur contient un pointeur vers la pile

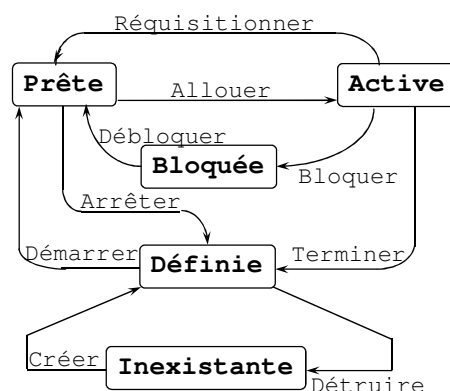
Rôle du noyau : Descripteur de tâches

- Information essentielles concernant la tâche
- Pointeur-suivant* : rôle de chaînage entre tâches de même priorité
- Contexte* : sauvegarde des informations (pile, registres, ...)
- Etat* de la tâche (prête, active, bloquée...)
- Supervision* : priorité, droit d'accès ...

Pointeur-suivant
Contexte
Etat
Supervision

Etat d'une tâche

- Création d'une tâche => Nom + descripteur
- **Définie**
 - connue du superviseur
 - descripteur spécifié
- **Prête**
 - en attente du processeur
- **Active**
 - en cours d'exécution
- **Bloquée**
 - en attente d'une ressource



Plan

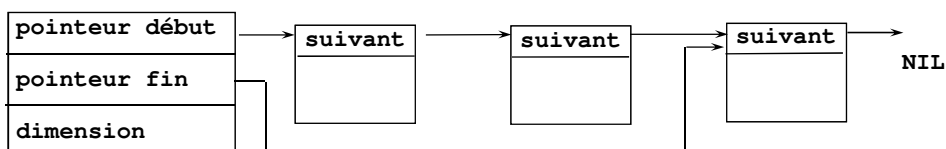
- **Mécanismes d'Interruption**
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259
- **les Périphériques d'Entrées Sorties**
 - Liaison série,
 - Liaison parallèle,
- **Système Multitâches**
 - Structure, fonction d'un OS multitâche
 - Gestion des tâches
 - Relations entre tâches
 - Mécanismes d'exclusion mutuelle
 - Synchronisation
 - Communication
- **Exemples de systèmes multitâches**

Gestion des tâches

- Passage de l'état prête à active se fait par *l'allocateur du processeur*.
 - à la suite d'un blocage
 - à la fin de l'exécution d'une tâche
- Stratégie d'allocation *variable*
 - à chaque interruption ou trappe
- la gestion s'organise en associant pour chaque état une ou plusieurs FIFO.

Gestion des tâches

- Pour chaque état, on organise une file d'attente à partir des informations contenues dans leur descripteur
- Les files sont gérées selon une discipline FIFO

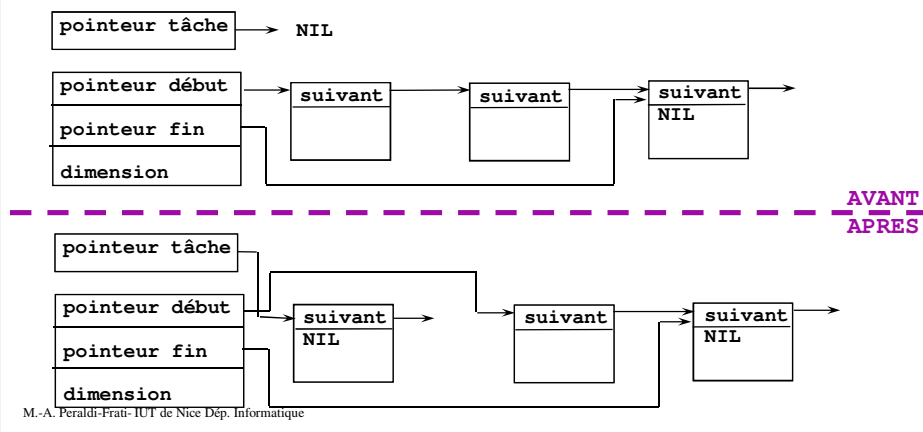


Gestion des tâches

- Extraction d'une tâche en tête de file

```

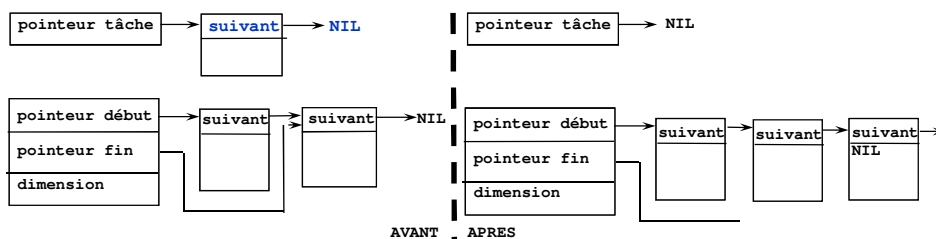
pointeur-tache := pointeur_debut
pointeur_debut := pointeur_suivant(premier)
pointeur_suivant(premier) := NIL
dimension := dimension - 1
    
```



M.-A. Peraldi-Frati-IUT de Nice Dép. Informatique

331

Gestion des tâches



- Insertion en fin de file

```

SI      pointeur-tache != NIL
ALORS  pointeur_suivant(dernier) := pointeur_tache
        pointeur_suivant(tâche à insérer) := NIL
        pointeur_fin := pointeur_tache
        pointeur_tache := NIL
        dimension := dimension + 1
    
```

M.-A. Peraldi-Frati-IUT de Nice Dép. Informatique

332

Commutation de processus

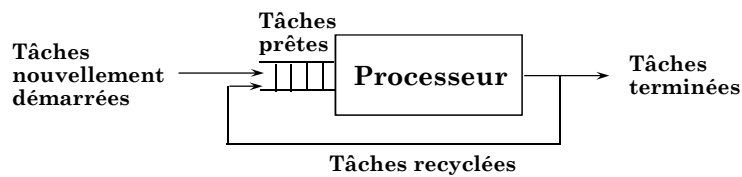
- L'instant de commutation dépend du périphérique horloge qui produit les interruptions
- A chaque interruptions => **Commutation de processus**
 - sauvegarde registres processus interrompu sur pile
 - sauvegarde SP dans descripteur du processus
 - mise à jour des descripteurs
 - chargement SP avec la valeur contenue dans le processus de tête de liste
 - restitution des registres sauvegardés sur la pile

Commutation de processus

- **Question :**
 - est il indispensable que le noyau utilise les interruptions de l'horloge?
 - Ne peut-il pas se contenter des interruptions des autres périphériques?
- **Système à temps partagé cela est essentiel**
 - “computer bound process”
- **Système purement temps réel :**
 - de part les applications
 - aucun processus ne monopolise le processeur
 - les commutations sur horloges ne sont pas implémentées
 - “input/output bound process”

Allocation du processeur

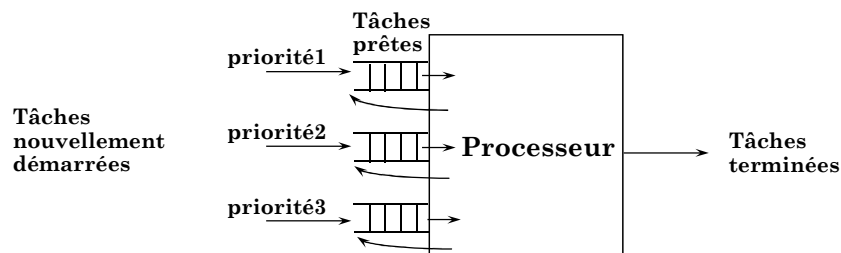
- Algorithme du tourniquet (Round-Robin scheduling)
 - files des tâches prêtes
 - chaque tâche reçoit le processeur
 - » durant une durée fixe
 - » jusqu'à ce qu'elle se bloque
 - les tâches suspendues retournent en fin de file



=> **Traitement interactif multi-utilisateurs**

Allocation du processeur

- Allocation par priorité
 - priorité matérielle, priorité logicielle
 - Round robin pour un même niveau de priorité



Gestion du processeur

- le processeur doit traiter
 - les tâches synchrone
 - les tâches de fond
 - les tâches asynchrones
- les tâches asynchrones correspondent à des opérations à effectuer à des instants imprévisibles :
 - déclenchées par des interruptions
 - qui provoquent la réallocation du processeur
 - => le temps de service doit être court
 - priorité
 - » inverse du temps de service
 - » plus élevée pour les périphériques lents

Gestion de tâches temps-réel

- Connaissance précise du temps
 - Détermination de l'heure d'apparition d'un événement
 - Scrutation périodique des capteurs et des actionneurs
 - Mise en attente de tâches durant un temps de garde

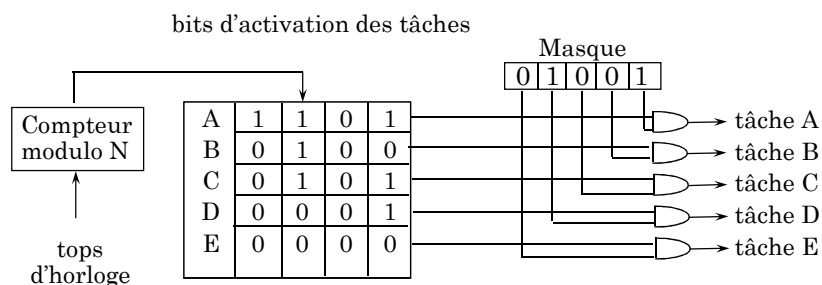
1ère solution : autant de temporisateurs que nécessaire
=> trop de sources d'interruptions

Gestion globale du temps

- Fait appel à une horloge temps-réel
 - interruptions régulières
- Tous les temps sont des multiples du temps de base
- Echéancier
 - tâche spéciale du noyau
- les descripteurs des tâches synchrones et des tâches retardées sont placés dans une file d'attente
 - allocation organisée à partir d'un tableau
 - organisée par ordre croissant de délai

Gestion des tâches synchrones

- Fait appel à un tableau d'activation

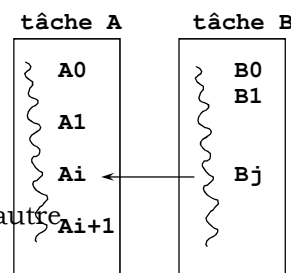


Plan

- Mécanismes d'Interruption
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259 les Périphériques : écrans, imprimante, drivers ...
- Circuits séquentiels/Automates
- Système Multitâches
 - Structure, fonction d'un OS multitâche
 - Gestion des tâches
 - Relations entre tâches
 - Mécanismes d'exclusion mutuelle
 - Synchronisation
 - Communication
- Exemples de systèmes multitâches
 - Exécutif temps-réel RTC

Relations entre tâches

- Les tâches sont parallèles et quasi indépendantes !!!mais!!!
 - indépendance est partielle
 - elles interagissent en certains points de leur exécution
- **Communication**
- **Synchronisation**
 - le déroulement d'une tâche en un point dépend d'un événement produit par une autre
- **Signalisation**



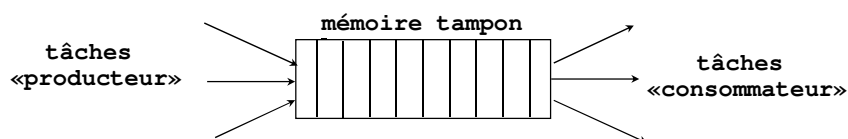
Relations entre tâches

• Exclusion mutuelle

- partage d'une ressource critique utilisable par une seule tâche à la fois
- exemple de l'accès multiple à une imprimante

• Coordination

- Exemple du modèle Producteurs / Consommateurs

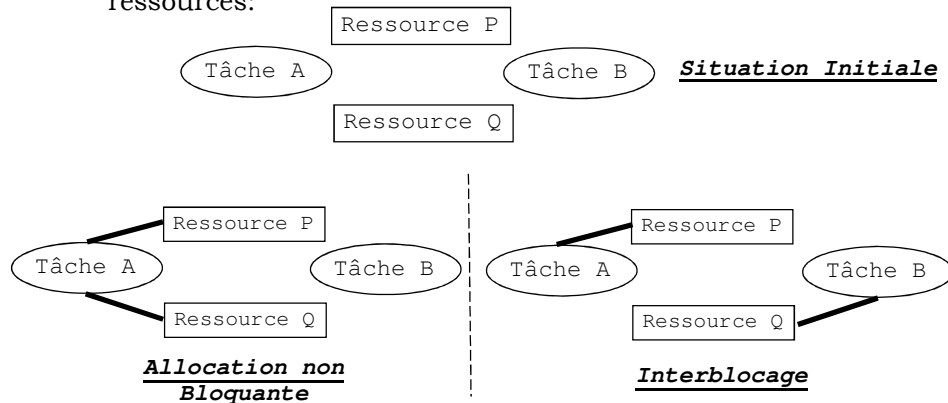


- éviter de retirer un message d'une file vide
- de déposer un message dans une file pleine

Relation entre tâches

• Blocage fatal

- Arrêt de l'évolution des tâches par suite d'un manque de ressources:



Relation entre tâches

- ♦ **Blocage fatal**

- Prévention
- détection
- guérison

!!! Ce sont des pannes logicielles

- ♦ **Résolution des problèmes liés aux relations entre tâches**

- solutions ad hoc
- mécanismes élémentaires

Plan

- **Mécanismes d'Interruption**

- Détection
- Traitement
- Différents types d'interruptions : logicielles / matérielles
- Contrôleur d'interruption 8259 les Périphériques : écrans, imprimante, drivers ...

- **Circuits séquentiels/Automates**

- **Système Multitâches**

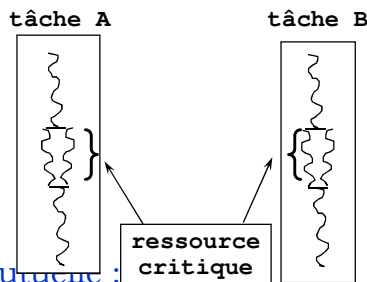
- Structure, fonction d'un OS multitâche
- Gestion des tâches
- Relations entre tâches
- Mécanismes d'exclusion mutuelle
- Synchronisation
- Communication

- **Exemples de systèmes multitâches**

- Exécutif temps-réel RTC

Mécanisme d'exclusion mutuelle

- Ressource critique matérialisée par une **séquence critique**



- on peut assurer l'exclusion mutuelle :
 - en masquant les IT
 - par un verrou
 - par un sémaphore

Mécanisme d'exclusion mutuelle

- **Verrou** : V est une variable binaire appelée verrou
 - V=0 droit d'entrée
 - V=1 interdiction d'entrée
- Le verrou est consulté et modifié avant l'entrée dans la section critique

Exemple :

```
verrou B ...  
E1: TAS  B  
E2:     BMI  E1  
E3 : 1ère instruction de la section critique  
...  
...  
CLR  B
```

Résout l'exclusion mutuelle par attente active

Mécanisme d'exclusion mutuelle

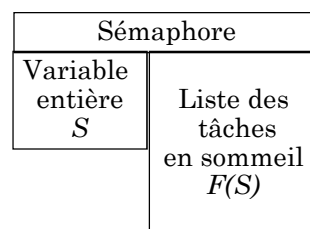
• Sémaphore :

- évite l'attente active
- tâches en attente sont mises en sommeil
- Sémaphore traite l'exclusion mutuelle avec accès multiples simultanés.
- Structure de données + primitives

ATTENDRE et **SIGNALER**

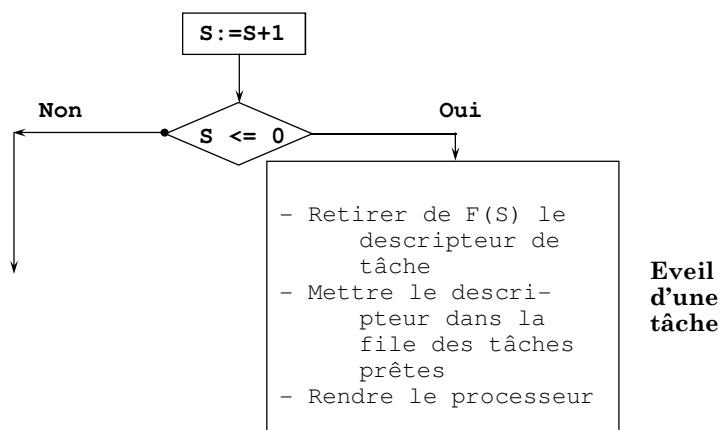
$P(S)$

$V(S)$



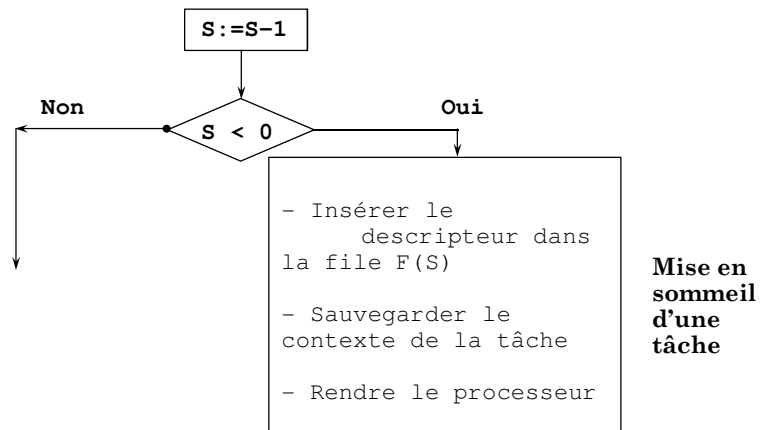
Mécanisme d'exclusion mutuelle

• Primitive $V(S)$: Signaler



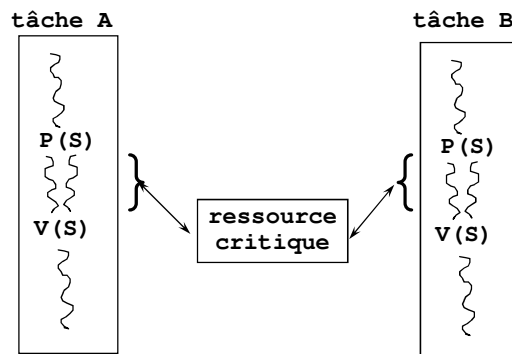
Mécanisme d'exclusion mutuelle

- Primitive P(S) : Attendre



Mécanisme d'exclusion mutuelle

- Utilisation d'un sémaphore



Plan

- Mécanismes d'Interruption
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259 les Périphériques : écrans, imprimante, drivers ...
- Circuits séquentiels/Automates
- Système Multitâches
 - Structure, fonction d'un OS multitâche
 - Gestion des tâches
 - Relations entre tâches
 - Mécanismes d'exclusion mutuelle
 - Synchronisation
 - Communication
- Exemples de systèmes multitâches
 - Exécutif temps-réel RTC

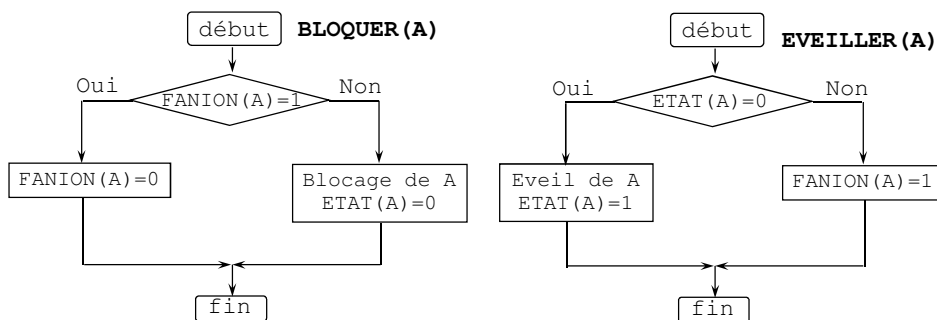
Synchronisation des tâches

- Plusieurs tâches interdépendantes
- Elles doivent être capable d'interagir pour synchroniser leur déroulement
- Deux types d'actions:
 - la tâche A doit se bloquer en attente d'un signal venant de la tâche B
 - la tâche B doit pouvoir éveiller la tâche A en lui transmettant de l'info.

Synchronisation des tâches

• Synchronisation directe

- une tâche agit directement sur une autre tâche (elle connaît le nom de la tâche).
- Utile pour stopper une tâche qui boucle indéfiniment
- Deux primitives



Synchronisation des tâches

• Synchronisation indirecte

- La tâche n'est plus explicitement nommée
- fonctionne par l'intermédiaire d'objets qui peuvent être
 - » des **variables booléennes**
 - » Notion d'événement
 - » positionnement à 1 de la variable pour la tâche synchronisante
SIGNAL(événement)
 - » la tâche synchronisée se bloque par une instruction ATTENTE(événement) tant que la variable est égale à 0

Synchronisation des tâches

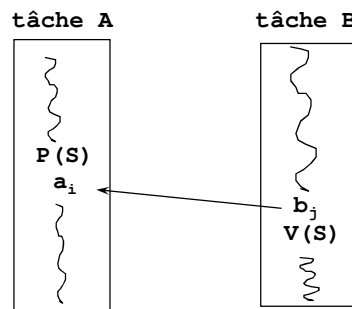
- Synchronisation indirecte

- » par sémaphores

- Exemple :

a_i doit se faire
après b_j

Sémaphore
initialisé avec $S=0$



Autres types de synchronisations

- Synchronisation sur conditions complexes
- ET, OU de sémaphores
- **Exemple 1** : Condition de synchronisation sur un OU
l'exécution de a_i est subordonnée à la réalisation d'au moins une opération $V(S)$ correspondant aux tâches B ou C.
- **Exemple 2**: condition de synchronisation sur un ET

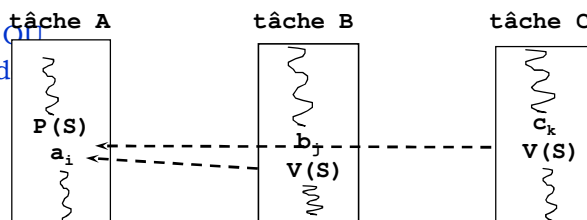
!!! Problèmes potentiels d'interblocages

Autres types de synchronisations

- Exemple 1 :

Synchronisation sur S

Initial: $S=0$, $F(S) = \text{vide}$

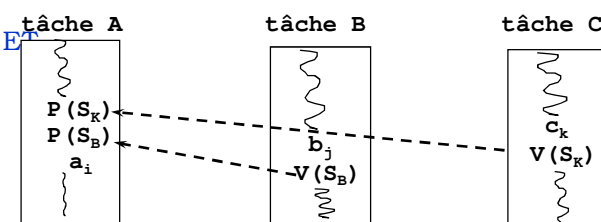


- Exemple 2 :

Synchronisation sur S_B

Initial : $S_B = 0$, $S_K = 0$

$F(S_B)$, $F(S_K)$ vides

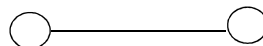


Plan

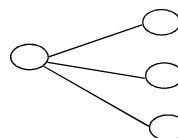
- Mécanismes d'Interruption
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259 les Périphériques : écrans, imprimante drivers ...
- Circuits séquentiels/Automates
- Système Multitâches
 - Structure, fonction d'un OS multitâche
 - Gestion des tâches
 - Relations entre tâches
 - Mécanismes d'exclusion mutuelle
 - Synchronisation
 - Communication
- Exemples de systèmes multitâches
 - Exécutif temps-réel RTC

Communication entre tâches

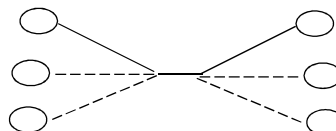
- Echanges d'informations sous forme de messages
- Configuration de type
 - point à point



- diffusion



- multipoint symétrique



Communication entre tâches Boîte aux lettres

- ♦ **Boîte aux lettres**
 - échange indirecte de messages
 - mémoire tampon => file d'attente
 - modèle producteur consommateur
 - tampon de capacité maximum N messages
 - ♦ **Problème**
 - éviter que plusieurs tâches prennent ou déposent simultanément des messages
 - Cesser de retirer des messages quand la boîte est vide
 - Cesser d'en mettre quand la boîte est pleine
- => **protection par sémaphores**

Communication entre tâches Boîte aux lettres

- Primitives atomiques

Deposer (message, boite-aux-lettres)

Retirer (message, boite-aux-lettres)

- Primitives protégées par deux sémaphores

– *positions* : nombre de positions libres

– *messages* : nombre de messages dans le tampon

Communication entre tâches Boîte aux lettres

- Exemple d'implémentation du modèle producteur-consommateur

par sémaphore et BAL :

Initialisation:

Positions :=N

Messages :=0

F(Positions) et F(Messages) vides

ENVOYER :

P(Positions)

DEPOSER(message, boite_aux_lettres)

V(Messages)

RECEVOIR :

P(Messages)

RETIRER(message, boite_aux_lettres)

V(Positions)

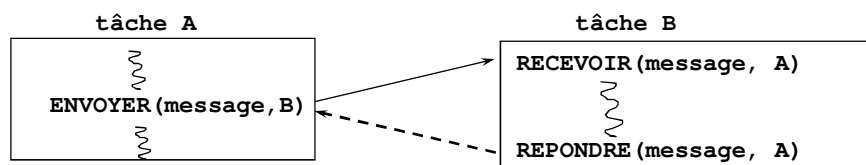
Communication entre tâches Rendez-vous

- Synchronisation rigoureuse entre l'émetteur et le récepteur
- Méthode du *rendez-vous* :
 - Les tâches s'arrêtent en un point convenu pour s'échanger leurs messages
- Primitives:
 - ENVOYER(message, destinataire)
 - RECEVOIR(message, emetteur)
 - REPONDRE(message, emetteur)

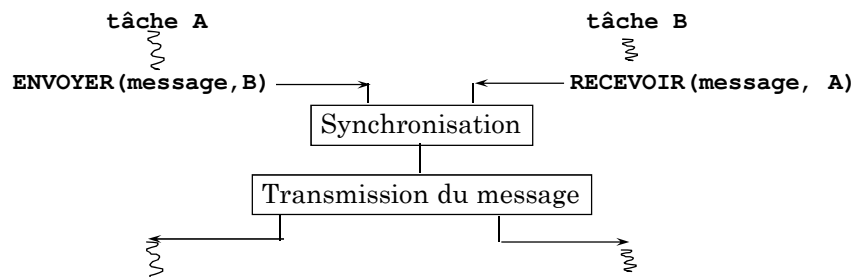
Emetteur et destinataire sont respectivement le nom de la tâche qui envoie et la tâche qui réceptionne le message

Communication entre tâches Rendez-vous

- Echange de message par rendez-vous



- Principe du rendez-vous



Plan

- **Mécanismes d'Interruption**
 - Détection
 - Traitement
 - Différents types d'interruptions : logicielles / matérielles
 - Contrôleur d'interruption 8259 les Périphériques : écrans, imprimantes , drivers ...
- **Circuits séquentiels/Automates**
- **Système Multitâches**
 - Structure, fonction d'un OS multitâche
 - Gestion des tâches
 - Relations entre tâches
 - Mécanismes d'exclusion mutuelle
 - Synchronisation
 - Communication
- **Exemple de systèmes multitâches**

Systemes d'exploitation multitâche

- **Multitâche existe depuis fort longtemps dans l'industrie**
 - OS multi tâche => OS2, Unix, Solaris, Win95, NT
 - OS temps -réel => Irmx86, RTC
- **Deux exemples :**
 - multitâches TR : RTC
 - multitâche non TR : Windows NT

Exemple d'un exécutif temps réel : STR

- Ensemble de primitives de gestion de tâches,
- Primitives sont indivisibles
- Le code d'initialisation du module noyau a pour rôle de créer une première tâche appelé *ProcessusIdle* en lui affectant la priorité maximum *InitNoyau()*
- Primitives de création et de gestion de tâches
- Primitives de synchronisation et communication
- Prise en compte du temps partagé

Exemple d'un noyau temps réel : STR

- Gestion des tâches

```
CreateTask(IdTache, PrioTache, Tache)
StartTask(IdTache);
StopTask(IdTache);
Desactive();
TaskPriority(Idtache);
TaskState(IdTache);
CurrentTask();
ChangePriority(IdTache, PrioTache);
RTClockInit(Ticks);
DelayTask(NbTicks);
EnterRegion();
LeaveRegion();
```

Exemple d'un noyau STR

- **Communications**

```
InitMailBox(IdBL, DimBl);  
Send(IdBL, Message);  
Receive (IdBL, &Message, Time-out);  
TestReceive(IdBl, &Message);
```

- **Exclusion Mutuelle**

```
InitSemaphore(IdSem, Compte);  
P(IdSem, Time-Out);  
V(IdSem);  
TestP(IdSem);
```

- **Synchronisation**

```
SignalEvent (IdTache, IdEvt);  
WaitEvents'Evs Attendus, Time-Out, &CopieEvtSig);  
ClearEvents(Liste Evs);
```

Exemple d'un noyau STR

- **Primitives Vidéo**

```
F1 = OuvreFen(X, Y, X+L, Y+H, CoulFond, CoulAff,  
«Titre»);  
EffaceFen(F1);  
FermeFen(F1);  
SelectFen(F1);  
LireCar(F1, ON/OFF, &Car);  
EcrCar(F1, Car);  
EcrVar («Texte %d», Var);
```

Solution du problème des philosophes

```
void PoseFourchettes( int iPhilosophe )
{ /* Protéger l'accès au tableau nEtat */
  WaitForSingleObject( hMutexModifierEtat, INFINITE );
  /* Mettre à jour l'état */
  nEtat[ iPhilosophe ] = PENSIF;
  TracePhilosophe( &TC, iPhilosophe, TRUE );
  /* Donner l'accès du sémaphore aux deux voisins */
  Test( VOISIN_GAUCHE( iPhilosophe ) );
  Test( VOISIN_DROITE( iPhilosophe ) );
  /* Libérer l'accès à nEtat */
  ReleaseMutex( hMutexModifierEtat );
}
```

Les périphériques

- les mémoires secondaires
 - Bandes magnétiques
 - Disques magnétiques
 - Disques optiques (CD ROM)
- les terminaux de visualisation
 - terminaux graphiques
- les modems
- les souris
- les imprimantes
 - matricielle à aiguilles
 - laser

Périphériques: les mémoires secondaires.

<i>Support</i>	<i>Ecriture</i>	<i>Accès</i>	<i>Utilisation</i>
Bande Magnétique	Magnétisation locale	Séquentiel	Archivage
Disque magnétique	Tête de lecture écriture flottante	Rotationnel	Stockage et accès rapide
Disque souple	Tête de L/E en contact avec le support	Rotationnel	Support de distrib. logiciel
Disque Optique	Mesure de la réflectivité	Rotationnel	CDROM Info. Permanente

Périphériques: les écrans

- **Ecrans alphanumériques**
 - Zone mémoire contenant des caractères
 - Caractère codé sur 2 octets : attribut + code Ascii
 - Mémoire de 4 k pour un écran alpha couleur

- **Ecrans Graphiques**
 - Zone mémoire de *pixels*
 - Organisation en 480x640 ou 800x600 ou 1024x1024 pixels
 - Ecran *bit map* (juxtaposition de plans mémoire)
 - Taille mémoire augmente (écran 16 couleurs de taille 1024x1024 = 1Mega)
 - Temps d'affichage lent => processeurs doivent être rapides.

Périphériques: les modems

- Connexions distantes via le réseau téléphonique.
- Transmission d'un signal sinusoïdal
 - modulation d'un signal de base (porteuse)
 - *amplitude* : signal fort (1) ou faible (0)
 - *fréquence* : signal sonore aigu (1) ou grave (0). moins sensible aux bruits
 - *de phase* : phase modifiée quand la donnée passe de 0 à 1 (vice versa)
- Vitesse de modulation : baud
 - nombre d'état de modulation par seconde.
 - plusieurs bits transmis par état
- Transmission synchrone ou asynchrone
 - horloge ou bits de synchronisation
- Transmission simplex, semi duplex ou full duplex
 - modes d'exploitation de la ligne

Périphériques: les souris

- Souris
 - mécanique : déplacement de roues => calcul de la position
 - optique : déplacement d'une LED sur une grille,
 - optomécaniques : Déplacement de roues + photodétecteur
- Codage des informations
 - 3 octets par 100 ms
 - pooling des informations
 - Etat, x, y

Périphériques: les imprimantes

- **Imprimantes matricielles**
 - aiguilles activées par un électro-aimant
 - impression par point
- **Imprimantes laser**
 - photosensibilisation
 - poudre électrostatique => toner
 - Processeur interne /externe
 - Langage Postscript

Evolution des architectures : du CISC au RISC

- **CISC Complex Instruction Set Computer**
 - jeu d'instruction complexe
- **RISC Reduced Instruction Set Computer**
 - jeu d'instruction réduit
 - multiplication des unités d'exécution
 - usage intensif des caches
 - branchements prédictifs ...

Caractéristiques des processeurs CISC

- Jeu d'instruction complexe => décodeur d'instructions
 - ++ programme d'application plus courts
 - ++ moins de place mémoire
 - -- temps d'exécution variables d'une instruction à une autre
 - -- la vitesse du processeur doit s'aligner sur les temps max
- Microcode
 - contient l'ensemble des micro-instructions
 - reçoit les instructions du décodeur et les transforme en opération élémentaires
 - surface de silicium augmentée par le microcode
 - problème des défauts de surface de silicium

Suppression du microcode avec le RISC

- RISC développé par IBM à partir de 1975
 - Concept freiné à ses débuts par le prix des memoires (code + important)
 - travaux parallèles sur les compilateurs
- Jeu d'instruction réduit => suppression du microcode
- Instructions complexes décomposées => rôle du compilateur
- Instructions complexes maintenues sont cablées
- temps d'exécution constant pour chaque instruction
- taille des instructions constante
- adressage simplifié
- multiplication des registres internes

Quelques processeur RISC

- Alpha de Digital
- Pentium 66 et +
- Power PC de Apple IBM et motorola
- Sparc de Sun

un exemple de RISC : PowerPC

- architecture superscalaire : plusieurs instructions différentes peuvent être exécutées simultanément par des unités indépendantes.
- multiplication des unités d'exécution : unité de branchement, unité en virgule fixe, unité en virgule flottante
- fonctionnement de registres à registres
 - registres d'usage général
 - registres spécialisés
- des instructions de longueur fixe
- hiérarchie de mémoires
- fonctionnement en multi-processeurs
- du RISC vers les VLIW ?

Evolution des architectures

- 2 voies d'évolution
 - accroissement des performances
 - miniaturisation des composants
- **Accroissement des performances**
 - augmentation de la vitesse d'exécution
 - » composants
 - » vitesse de propagation
 - » dimension physique
 - » optimisation des connexions
 - » unité de calcul rapide
 - » registres spécialisés (pipeline)
 - » organiser les traitements (//)

Evolution des architectures

- **Accroissement des performances**
 - Parallélisation des programmes
 - » ++ Utilisation max des unités de traitement de la machine
 - » complexité de la machine augmente
 - » les systèmes hautement parallèles ne peuvent être exploités à 100 %
- **Miniaturisation des composants**
 - développement de micro-plaquettes (chip)
 - 1970 1 cpu sur une seule puce
 - + coprocesseur arithmétique
 - Actuellement : on est arrivé aux limites d'intégration
 - » recherche de nouveaux matériaux
 - » physique des composants

Evaluation des performances

- Pour l'utilisateur :

Performance = Temps d'exécution

- Difficile à évaluer car elle dépend de :
 - la vitesse d'exécution du microprocesseur
 - Temps de réponse de la mémoire
 - Entrée / sortie
- Programmes de tests *Benchmarks*
 - valables sur toutes les machines
 - problème de la représentativité des programmes

Evaluation de performances

- **SPEC** System Performance Evolution Coopérative
 - consortium de constructeurs
 - standards communs de test
 - exécution sur différentes plates-formes
 - => SPECMARK
 - De là viennent les unités de mesure
 - » **MIPS** : Million Instructions Per Second
 - » **MFlops** : Million of Floating Point Opération Per Second
- **Problème:**
 - le calcul des Mips dépend du constructeur
 - les jeux d'instructions étant différents => résultats non comparables entre eux.
- **NB:**
 - Ces mesures sont des indicateurs théoriques de performance
 - Choix d'une machine : benchmarks avec programmes types qu'on va utiliser.

Evaluation de performances

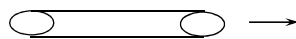
- **Systèmes temps-réel :**
 - systèmes critiques
 - dépassement de temps => DANGER
 - **Exemple :**
 - » Détection d'alarme dans une centrale nucléaire
 - » Détection d'un décrochage sur un A 310
- **Pour ces systèmes**
 - Evaluation doit être précise
 - benchmarks
 - analyse statique des programmes
 - simulation
- **Problème ouvert :**
 - activité de recherche importante
 - quête d'une approche systématique pour ces évaluations

Technique d'optimisation pour les supers ordinateurs

- **Pipelining :** s'inspire du travail à la chaîne. Dans la même unité de temps plusieurs tâches sont exécutées

E D C B A

Re Rd Rc Rb Ra



Opération O

Gestion des périphériques : drivers

- **fonctions :**
 - réalise une interface entre les niveaux matériel et logiciel
- **réalisation :**
 - écrit en assembleur sous DOS
 - inclut dans le code des OS anciens
 - à partir du DOS 2.0 possibilité d'inclure des drivers
- **Composition d'un driver**
 - information d'état sur le driver
 - série de routines : fonctions du driver
- **Intégration d'un driver**
- **2 Types de drivers : caractères ou blocs**

Structure d'un driver

- **Structure de l'en-tête**

Adresse	Contenu	Type
+00h	Adresse du prochain driver	1 PTR
+04h	Attribut du périphérique	1 Word
+06h	Offset de la routine de stratégie	1 Word
+08h	Offset de la routine d'interruption	1 Word
+0Ah	Nom du driver pour les drivers de caractères	8 Bytes

Structure d'un driver.

- **13 fonctions** numérotées de 0 à 12
- Fonctions **obligatoires** même si inutilisées : bit Termine =1
- Certaines sont propres aux drivers de blocs ou de caractère
- Elles tirent leurs arguments du bloc de données
- Elles rendent leur résultats dans ce même bloc
- Principales fonctions:
 - 00h Initialisation du driver
 - 03h Lecture directe
 - 04h Lecture
 - ...

Les Bus

- Liens de communications entre l'Unité centrale et l'extérieur
- Trois types d'informations sont véhiculées par le bus
 - **les données:** mémoire <-> UC
 - » instructions ou données
 - » 8, 16, 32, 64 bits
 - » bidirectionnel
 - **les adresses**
 - » 20 lignes d'adresse
 - » adresse instruction à charger dans registre d'instruction
 - » donnée à charger dans un registre ou sur une entrée de l'UAL
 - **Les commandes**
 - » véhicule les microcommandes générées par le séquenceur

Différents types de bus

- **Synchrone**
 - ligne d'horloge (fréquence de 5 à 50 Mhz)
 - période d'horloge = cycle du bus
 - période constante
 - Problème de l'hétérogénéité des vitesses des circuits : on se base sur le circuit le plus lent
- **Asynchrone**
 - pas d'horloge
 - cycle variable
 - adapté aux circuits hétérogènes.
 - le déclenchement des actions sur le bus se fait par des signaux de synchronisation.
 - maître /esclave qui se synchronisent.
- **Bus synchrone le plus utilisé**

Les Bus

- **Principaux type de bus**
 - Bus PC, PC/AT largement orienté INTEL
 - Bus VME : hautes performances, usage industriel
- **Bus PC/AT**
 - Bus synchrone : cycle de bus de 70ns => bande passante de 1,2Mo /s
 - A donné lieu au standard EISA qui correspond à une extension vers un 32 bits
- **Bus VME**
 - Versa Module Eurocard
 - Bus asynchrone (pas d'horloge)
 - Description fonctionnelle complète pour augmenter l'interopérabilité
 - Fiabilité augmentée par connecteur et Racks blindés
 - Lignes indépendantes (données, adresse, interruption et commande)

Avant: Architecture d'une machine informatique

Maintenant: Architecture d'un système sur puce (# 2 cm²)

Exemple: Téléphone portable 3G

Informatique / Electronique ← technologie dépendantes

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 397

Contribution de l'électronique à l'automobile -un exemple-

Sécurité	Confort, aide à la conduite	Performance	Protection de l'environnement
<ul style="list-style-type: none"> • Freinage urgence • ABS • DYC • ASC • Calibrage Xenon • Vision nocturne • Airbag • Pré-tensionneur ceintures • Anti-collision • Anti-attraction/voil • Commandes essuie-glace • Démarreur carte à puce • Pression pneus • Détecteur de pluie 	<ul style="list-style-type: none"> • Climatisation • Direction électrique • Vitres électrochromes • Navigation • Affichage multimédia • Calculateur de maintenance • Localiseur 	<ul style="list-style-type: none"> • Injection directe essence • Injecteur pompe diesel • Moteur hybride • Turbo géométrie variable • Contrôle numérique moteur • Multiplexage • Suspension active • Alternateur • Transmission continue 	<ul style="list-style-type: none"> • Bureau embarqué • Salon embarqué

2000: 20% 2010: 40%

Pourcentage de l'électronique dans le coût direct de fabrication d'un véhicule (307)

A. Dubois - Sénat 1991 - 21 / 1 / 01 Source: ST

M.-A. Peraldi-Frati - IUT de Nice Dép. Informatique 398