

# De l'ordinateur au processus : rôle d'un système

---

---



Laurent Réveillère

Université de Bordeaux

[laurent.reveillere@u-bordeaux.fr](mailto:laurent.reveillere@u-bordeaux.fr)  
<http://www.reveillere.fr/>

*D'après le cours d'introduction aux systèmes d'exploitation de Télécom SudParis*



## Présentation du cours

---

---

- ❑ Contexte du cours :
  - Introduire notre objet d'étude : les systèmes d'exploitation
  
- ❑ Objectifs :
  - Comprendre ce qu'est un ordinateur
  - Comprendre ce que sont un logiciel et un programme
  - Comprendre ce qu'est un système d'exploitation
  - Comprendre ce qu'est un processus
  
- ❑ Notions abordées :
  - Ordinateur, mémoire, processeur, périphérique, système d'exploitation, processus, communication, programme, logiciel

# I. Qu'est ce qu'un ordinateur ?

---

---



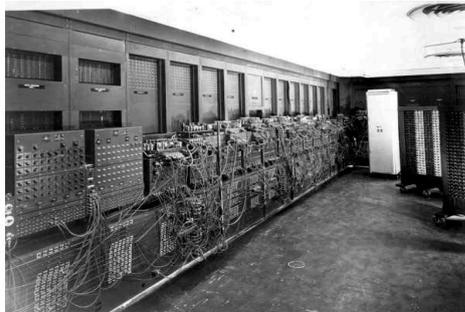
IF 110  
Introduction aux Systèmes d'Exploitation

## Définition d'un ordinateur

---

---

- ❑ Machine électronique capable d'exécuter des instructions effectuant des opérations sur des nombres



- 1946 : ENIAC*
- *calculateur à tubes*
  - *30 tonnes, 72m<sup>2</sup>*
  - *357 mult/s*
  - *35 div/s*

*En panne la moitié du temps*



## Définition d'un ordinateur

- ❑ Machine électronique capable d'exécuter des instructions effectuant des opérations sur des nombres



*Janv 1948 : SSEC (premier ordinateur chez IBM) avec une capacité mémoire de 150 nombres*

IF110

5



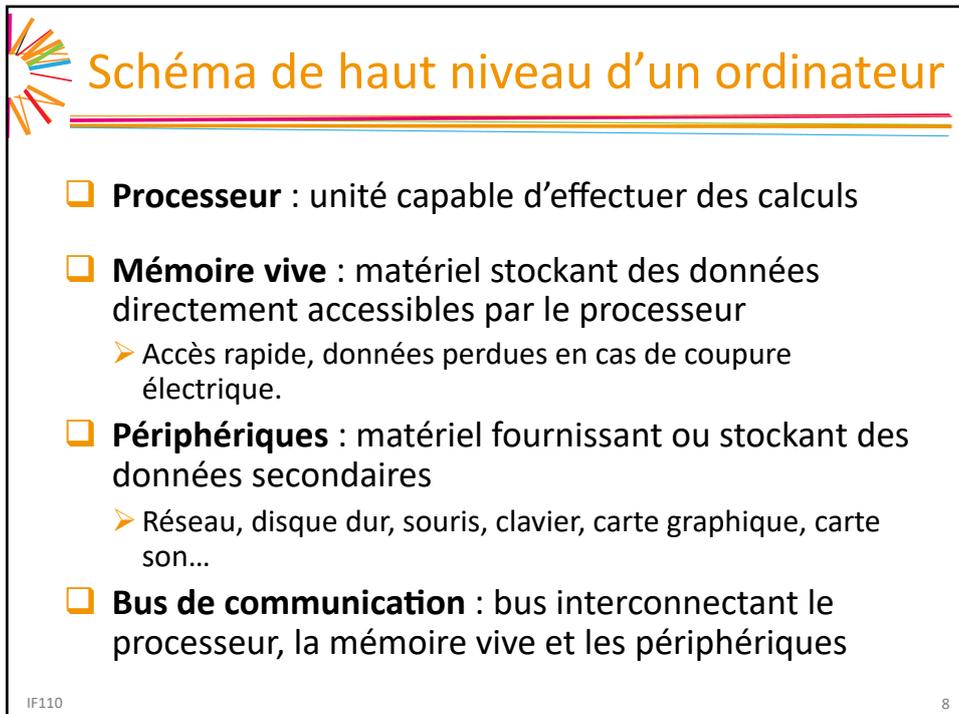
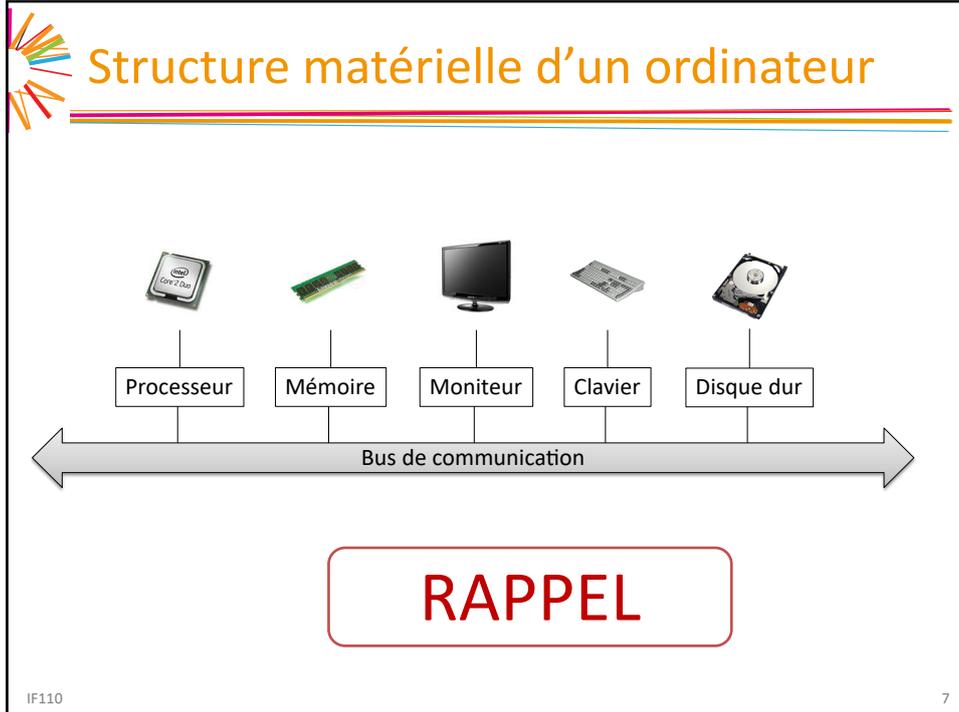
## Définition d'un ordinateur

- ❑ Machine électronique capable d'exécuter des instructions effectuant des opérations sur des nombres



IF110

6





## Qu'est ce que la mémoire vive ?

- ❑ Mémoire vive : ensemble de cases numérotées contenant des octets
- ❑ Une case contient un octet (*byte* en anglais) = regroupe 8 bits
- ❑ Bit : valeur valant 0 ou 1
  - 0 : bit non chargé ("courant ne passe pas")
  - 1 : bit chargé ("courant passe")
- ❑ Un octet permet de représenter  $2^8 = 256$  valeurs

Case 0	0110 0001b
Case 1	0101 1001b
Case 2	0110 0001b
Case 3	1111 0000b
	⋮
Case 800	1100 1011b

IF110

9



## Représentation des nombres

- ❑ Notation décimale : un chiffre peut prendre 10 valeurs de 0 à 9

$$276 = 2 \cdot 10^2 + 7 \cdot 10^1 + 6 \cdot 10^0$$

- ❑ Notation binaire : un chiffre peut prendre 2 valeurs de 0 à 1

$$1101b = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13$$

- ❑ Notation hexadécimale : un chiffre peut prendre 16 valeurs de 0 à f

$$0x276 = 2 \cdot 16^2 + 7 \cdot 16^1 + 6 \cdot 16^0 = 630$$

$$0xb6 = 11 \cdot 16^1 + 6 \cdot 16^0 = 182$$

IF110

10



## Hexadécimal en informatique

- Avec 4 bits, on encode 16 valeurs, soit 1 chiffre hexadécimal

Case 0	1110 0001b
Case 1	0101 1001b
Case 2	0110 0001b
Case 3	1111 0000b
	⋮
Case 800	1100 1011b

IF110

11



## Hexadécimal en informatique

- Avec 4 bits, on encode 16 valeurs, soit 1 chiffre hexadécimal

- L'hexadécimal est donc plus concis pour représenter les valeurs des octets

- Un octet est représenté par 2 chiffres hexadécimaux

Case 0	0xe1
Case 1	0x59
Case 2	0x61
Case 3	0xf0
	⋮
Case 800	0xc3

IF110

12



## Que représentent les octets ?

□ Une série d'octets peut représenter :

- Un entier naturel
- Un entier relatif
- Une suite de caractères
- Une valeur de vérité (vrai/ faux)
- Un nombre flottant
- Un nombre complexe
- Une instruction machine

Case 0	0xe1
Case 1	0x59
Case 2	0x61
Case 3	0xf0
	⋮
Case 800	0xc3

- Ou tout autre ensemble énumérable

IF110

13



## Représentation des entiers

□ Les octets sont regroupés pour former des valeurs entières

(souvent par 1, 2, 4 ou 8 octets)

- Peut être vu comme un naturel (dans N)

$$0xe159 = 14 \cdot 16^3 + 1 \cdot 16^2 + 5 \cdot 16 + 9$$

⇒ 0xe159 représente 57689

Le nombre  
0xe159

0xe1
0x59
0x61
0xf0

IF110

14



## Représentation des relatifs

### ❑ Codages possibles

- Bit de signe
- Complément à un
- Complément à deux

- Peut être vu comme un relatif (dans Z)
- ➔ représente -7 847

Le nombre  
-7 847

0xe1
0x59
0x61
0xf0



## Représentation des réels

### ❑ Codage d'un nombre entier naturel en base B

➤  $(N)_B = a_n a_{n-1} a_{n-2} \dots a_1 a_0$

### ❑ Pour coder un nombre réel non signé

- On rajoute une partie fractionnaire après une virgule

$(N)_B = a_n a_{n-1} a_{n-2} \dots a_1 a_0 , b_1 b_2 \dots b_{m-1} b_m$

### ❑ Valeur en décimal

- calcul du polynôme

$a_n B^n + a_{n-1} B^{n-1} + \dots + a_1 B + a_0 + b_1 B^{-1} + \dots + b_{m-1} B^{-(m-1)} + b_m B^{-m}$



## Conversion reel base B en décimal

### □ Exemples

- $123,45 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$
- $(101,101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$   
 $= 4 + 1 + 0,5 + 0,125$   
 $= 5,625$
- $(AB,4E)_{16} = 10 \times 16^1 + 11 \times 16^0 + 4 \times 16^{-1} + 14 \times 16^{-2}$   
 $= 160 + 11 + 4 \times 0,0625 + 14 \times 0,00390625$   
 $= 171,3046875$

IF110

17



## Conversion réel décimal en base B

### □ Conversion d'un nombre décimal réel en base B

- Pour la partie entière
  - » Utiliser la méthode de la division entière comme pour les entiers
- Pour la partie fractionnaire
  - » Multiplier la partie fractionnaire par B
  - » Noter la partie entière obtenue
  - » Recommencer cette opération avec la partie fractionnaire du résultat et ainsi de suite
  - » Arrêter quand la partie fractionnaire est nulle
    - Ou quand la précision souhaitée est atteinte
    - Car on ne peut pas toujours obtenir une conversion en un nombre fini de chiffres pour la partie fractionnaire
  - » La partie fractionnaire dans la base B est la concaténation des parties entières obtenues dans l'ordre de leur calcul

IF110

18



## Conversion réel décimal en base B

### ❑ Exemple : conversion de 12,6875 en binaire

➤ Conversion de 12 : donne  $(1100)_2$

➤ Conversion de 0,6875

$$\begin{array}{rcl} 0,6875 \times 2 = 1,375 & = & \underline{1} + 0,375 \\ 0,375 \times 2 = 0,75 & = & \underline{0} + 0,75 \\ 0,75 \times 2 = 1,5 & = & \underline{1} + 0,5 \\ 0,5 \times 2 = 1 & = & \underline{1} + 0 \end{array} \quad \downarrow$$

➤  $(12,6875)_{10} = (1100,1011)_2$

### ❑ Exemple : conversion de 171,3046875 en hexadécimal

➤ Conversion de 171 : donne  $(AB)_{16}$

➤ Conversion de 0,3046875

$$\begin{array}{rcl} 0,3046875 \times 16 = 4,875 & = & \underline{4} + 0,875 \\ 0,875 \times 16 = 14,0 & = & \underline{14} + 0 \end{array} \quad \downarrow$$

➤  $(171,3046875)_{10} = (AB,4E)_{16}$

IF110

19



## Nombre reels et précision

### ❑ Exemple : conversion de 25,3 en binaire

➤ Conversion de 25 : donne  $(11001)_2$

➤ Conversion de 0,3

... on boucle!

### ❑ Précédentes conversions vues

➤ Peut toujours convertir avec un nombre fini de chiffres

➤ Pas le cas pour la conversion de la partie fractionnaire

» On remplit alors les chiffres disponibles selon la **précision**

IF110

20



## Nombres réels en virgule flottante

### □ Principe et intérêts

- Avoir une virgule flottante et une précision limitée
- Ne coder que des chiffres significatifs
- $N = +/- M \times B^E$ 
  - » N = nombre codé
  - » M = mantisse : nombre de X chiffres de la base B
  - » E = exposant : nombre de Y chiffres de la base B
  - » +/- = codage du signe : positif ou négatif
- Le nombre est présenté sous forme normalisée pour déterminer la mantisse et l'exposant
- Pas de chiffre avant la virgule :  $0,XXXXX * B^E$

IF110

21



## Exemple de codage de réels

### □ Exemple : 1234,5 en base 10

- On normalise pour n'avoir que des chiffres après la virgule  
 $(1234,5)_{10} = 0,12345 \times 10^4$
- Mantisse codée = 12345, exposant = 4, signe = +

### □ Standard IEEE 754 : codage binaire de réels en virgule flottante

- Précision simple : 32 bits
  - » 1 bit de signe, 8 bits exposant, 23 bits mantisse
- Précision double : 64 bits
  - » 1 bit de signe, 11 bits exposant, 52 bits mantisse
- Précision étendue : sur 80 bits
  - » 1 bit de signe, 15 bits exposant, 64 bits mantisse

IF110

22



## Représentation des valeurs de vérités

- ❑ Booléen : valeur pouvant valoir vrai ou faux
- ❑ Peut être stocké sur 1 bit, 1 octet, 2 octets, 4 octets, 8 octets...
- ❑ Convention :
  - 0 vaut faux
  - Toute autre valeur vaut vrai

Une valeur vraie

0xe1
0x59
0x61
0xf0
⋮
0xc3

IF110

23



## Représentation des caractères

- ❑ Un octet peut être vu comme un caractère
- ❑ Table ascii

0x1f	←	0x41	A	0x61	a
0x20	·	0x42	B	0x62	b
...	...	0x43	C	0x63	c
0x30	0	0x44	D	0x64	d
0x31	1	0x45	E		e
0x32	2	0x46	F	0x66	f
0x33	3	0x47	G	0x67	g
...	...	...	...	...	...

Ceci est un 'a'

0xe1
0x59
0x61
0xf0
⋮
0xc3

IF110

24



## Fonctionnement d'un processeur

- ❑ Un processeur exécute des instructions qui peuvent
  - Effectuer des calculs
  - Accéder à la mémoire
  - Accéder aux autres périphériques
  - Sélectionner l'instruction suivante à exécuter (saut)
  
- ❑ Le processeur identifie une instruction par un numéro  
(Par exemple : 1 = additionne, 2 = soustrait, etc.)

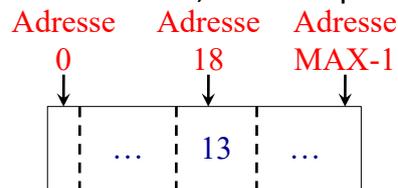
IF110

25



## Fonctionnement d'un ordinateur

- ❑ Mémoire : tableau d'octets, indexé à partir de 0



- ❑ Processeur : possède des variables internes appelées registres
  - PC (*Program Counter*) : adresse de l'instruction suivante  
*Si PC == 18, alors l'instruction suivante à exécuter est l'instruction 13*
  - Autres : registres sur lesquels le processeur effectue les calculs

IF110

26



## Fonctionnement d'un ordinateur

- ❑ À chaque cycle d'horloge, le processeur :
  - Charge l'instruction à l'adresse PC à partir de la mémoire
  - Place le PC sur l'instruction qui suit
  - Sélectionne le circuit à activer en fonction du numéro d'instruction
  - Exécute l'instruction
  
- ❑ Quelques exemples d'instructions
  - 0x10 0x4000 ⇒ charge l'octet à l'adresse 0x4000 dans le registre nommé R0 (lit une variable)
  - 0x12 0x89 ⇒ ajoute 0x89 à PC (saut)
  - 0x14 0x20 ⇒ ajoute 0x20 au registre R0 (calcul)
  - 0x17 0x70 0x12 ⇒ envoie 0x70 au périphérique 0x12

IF110

27



## Mémoire

- ❑ Mémoire
  - Dispositif capable d'enregistrer, de conserver et de restituer des informations
  - Informations binaires pour un ordinateur
- ❑ On classe les mémoires selon
  - Caractéristiques : capacité, débit ...
  - Type d'accès : séquentiel, direct ...

IF110

28



## Organisation de l'information

- ❑ Unité de base : bit
  - Le plus petit élément de stockage
- ❑ Octet (ou *byte*) : groupe de 8 bits
- ❑ Mot mémoire
  - Groupement d'octets (8, 16, 32, 64 ...)
  - Unité d'information adressable en mémoire
- ❑ Adresse
  - Valeur numérique référant un élément de mémoire
- ❑ Capacité ou taille
  - Nombre d'octets que peut contenir la mémoire
    - » 512 Ko, 8 Mo, 16 Go, 2To ...

IF110

29



## Caractéristiques des mémoires

- ❑ Temps d'accès
  - Temps entre le lancement d'une opération de lecture/écriture et son accomplissement
- ❑ Débit
  - Nombre d'informations lues ou écrites par seconde
  - Exemple: 2 Go/s
- ❑ Volatilité
  - Conservation ou disparition de l'information dans la mémoire hors alimentation électrique de la mémoire

IF110

30



## Méthodes d'accès

- ❑ Accès séquentiel
  - Parcourir de toutes les informations précédentes
  - Exemple: bande magnétique
- ❑ Accès direct
  - Chaque information a une adresse propre accessible
  - Exemple: mémoire centrale
- ❑ Accès semi-séquentiel
  - Mixte des deux précédentes
  - Exemple disque dur magnétique
    - » Accès direct au cylindre
    - » Accès séquentiel au secteur sur un cylindre

IF110

31



## Type de mémoire

- ❑ Mémoires non volatiles : ROM (Read Only Memory)  
dites mémoires mortes
  - Leur contenu est fixe (ou presque ...)
  - Conservé en permanence
- ❑ Mémoires volatiles : RAM (Random Access Memory)  
dites mémoires vives
  - Leur contenu est modifiable
  - Perte des informations hors alimentation électrique
  - Random : à prendre dans le sens « accès sans contraintes »  
(et non pas aléatoire)

IF110

32



## Mémoires non volatiles (ROM)

- ❑ ROM
  - « Câblage en dur » de l'information
  - Premier type de mémoire morte, on a gardé son nom pour toute cette famille
- ❑ PROM : mémoire programmable une seule fois
- ❑ EPROM : mémoire reprogrammable (via des ultraviolets)
- ❑ EEPROM : mémoire reprogrammable (électriquement)
  - Exemple : BIOS d'un ordinateur

IF110

33



## Mémoires volatiles (RAM)

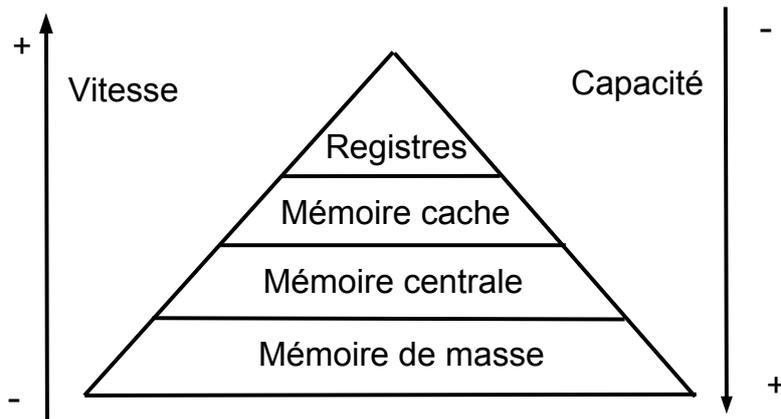
- ❑ DRAM : Dynamic RAM
  - Dynamique : nécessite un rafraîchissement périodique de l'information
  - Peu coûteuse
- ❑ SRAM : Static RAM
  - Statique : ne nécessite pas de rafraîchissement
  - Beaucoup plus rapide que la DRAM
  - Mais beaucoup plus chère

IF110

34



## Hiérarchie mémoire



IF110

35



## Registres

- ❑ Intégrés dans le CPU
- ❑ Servent à gérer l'exécution des opérations
  - Spécialisés, ex : adresse de la prochaine instruction en mémoire
  - Généraux : résultats intermédiaires des calculs
- ❑ Peu nombreux et de petites taille
  - Jusqu'à quelques dizaines/centaines
  - Quelques octets chacun
- ❑ Très rapides (vitesse du CPU)

IF110

36



## Mémoire cache

### ❑ Mémoire SRAM

- Intégrée dans le processeur et cadencée à la même fréquence
- Très chère, petite taille

### ❑ Objectifs

- Intermédiaire entre le processeur et la mémoire centrale
- Accélérer les temps d'accès à la mémoire centrale
- Éviter les accès inutiles en mémoire centrale (lectures multiples)

IF110

37



## Mémoire centrale et de masse

### ❑ Mémoire centrale

- Mémoire volatile de type DRAM à accès direct
- Capacité jusqu'à plusieurs dizaines de Go
- Vitesse relativement lente

### ❑ Mémoire de masse

- Mémoire non volatile
- Capacité de plusieurs To
- Vitesse très lente

IF110

38



## Fonctionnement des processeurs

- ❑ Unités de calculs
  - Un processeur intègre plusieurs unités de chaque type
- ❑ UAL (ALU : Arithmetic and Logic Unit)
  - Calculs logiques et arithmétiques sur les entiers
  - Calculs simples sur les entiers
  - Calculs logiques (comparaison, OR, NOT, ...)
- ❑ FPU (Floating Point Unit)
  - Calculs sur des flottants
  - Fonctions mathématiques avancées : sqrt, sin ...
- ❑ Unité multimédia
  - Calculs vectoriels
  - Exécution parallèle d'une même instruction sur plusieurs données

IF110

39



## Exécution d'une commande

- ❑ 2 cycles se succèdent
- ❑ Cycle de recherche de l'instruction
  - Lecture en mémoire de l'instruction à exécuter
  - Décodage de son contenu
- ❑ Cycle d'exécution de l'instruction
  - Lecture en mémoire des registres pour envoyer les opérandes à l'unité de calcul ou d'accès en mémoire
  - Exécution du calcul ou de l'accès mémoire
  - Enregistrement du potentiel résultat en mémoire ou dans un registre

IF110

40



## Unité de commande : horloge

- ❑ Horloge
  - Définit le cycle de base : cycle machine (clock cycle)
  - Utilisée pour synchroniser chaque étape des cycles de recherche et d'exécution
- ❑ Temps d'exécution d'un cycle de recherche ou d'exécution
  - Prend un certain nombre de cycles de base
- ❑ Cycle CPU = temps d'exécution minimal d'une instruction (recherche + exécution)

IF110

41



## Registres

- ❑ Registre = mots mémoire internes au processeur
- ❑ Registres de fonctionnement
  - Compteur Ordinal (CO), Registre Instruction (RI), ...
  - Accumulateur
  - Registres internes à l'UAL
  - Stockent les opérandes et le résultat d'un calcul
- ❑ Registres généraux
  - Registres accessibles par le programme
  - Servent à stocker
    - » Des valeurs souvent utilisées
    - » Des résultats intermédiaires

IF110

42



## Bus internes au CPU

- ❑ Au cœur du CPU, utilisation de bus
  - Pour envoi de données entre les éléments
    - » Contrôleur mémoire, unités de calculs, registres ...
  - Pour envoi d'adresse
    - » Lecture/écriture en mémoire ou dans un registre ...
  - Pour commander, coordonner les éléments

IF110

43



## Architecture X bits

- ❑ Processeurs souvent différenciés selon leur architecture : 16, 32 ou 64 bits
- ❑ Historiquement : taille des registres (8, 16 bits...)
  - Mais dans processeurs récents : registres de toute taille (16, 32, 64, 80 ou 128 bits)
  - Selon que l'on manipule des entiers, des adresses, des flottants, des vecteurs ...
- ❑ Norme de fait de nos jours
  - Taille des registres généraux
    - » Un processeur 64 bits a des registres généraux de 64 bits

IF110

44



## Architecture X bits

---

---

### ❑ Conséquences

- Unités de calcul entier doivent gérer des nombres de même taille que les registres généraux
- Bus internes doivent avoir aussi cette même taille

### ❑ Registre général peut contenir une adresse mémoire

- Définit alors aussi la taille maximale de mémoire adressable par le processeur
  - » 32 bits :  $2^{32}$  octets = 4 Go
  - » 64 bits :  $2^{64}$  octets = 18 Millions de To
- En pratique : codage d'adresses sur moins de bits
  - » AMD Athlon 64 : 48 bits = 256 To théorique mais 1 To en pratique

IF110

45

## *Optimisation et augmentation des performances des processeurs*

---

---



IF 110

Introduction aux Systèmes d'Exploitation



## Fréquence

---

---

- ❑ Temps d'exécution d'une instruction
  - Cycle CPU
  - Augmentation de la fréquence de fonctionnement
- ❑ Avantages
  - Plus d'instructions exécutées en moins de temps
- ❑ Problèmes technologiques et physiques
  - Dissipation thermique
  - Temps de propagation des signaux
- ❑ Solutions
  - Finesse de gravure des transistors

IF110

47



## Mémoire cache

---

---

- ❑ Débit soutenu en lecture d'instructions et de données
  - Plusieurs niveaux de caches de taille raisonnable
  - Algorithmes de chargement prédictifs

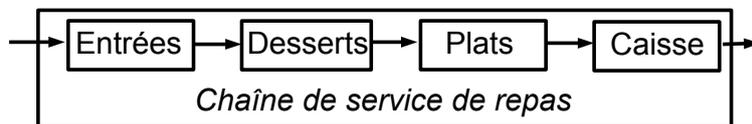
IF110

48



## Pipeline

- ❑ Principe du pipeline par l'exemple Restaurant Universitaire
- ❑ On passe, dans l'ordre devant 4 éléments
  - Un présentoir pour les entrées
  - Un présentoir pour les desserts
  - Un présentoir pour les plats de résistance
  - Une caisse



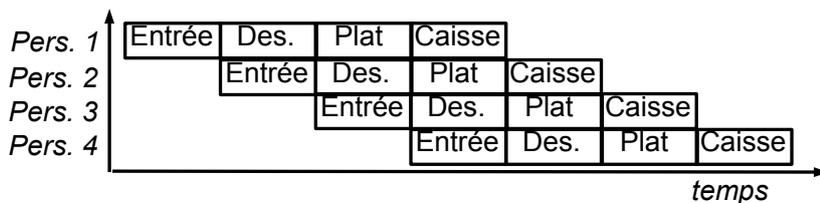
IF110

49



## Pipeline

- ❑ 2 modes d'utilisation pour se servir un repas
  - Une seule personne à la fois dans toute la chaîne de service
  - Plusieurs personnes à la fois, en décalé



IF110

50



## Pipeline

- ❑ Intérêts du deuxième mode
  - Plusieurs personnes se servent en même temps
  - Gain de temps : plus de personnes passent pendant une même durée
  - Meilleure gestion des éléments : toujours utilisés
- ❑ Inconvénients du deuxième mode
  - Plus difficile de faire « demi-tour » dans la chaîne d'éléments
  - Nécessite des synchronisations supplémentaires et des éléments dont le parcours prend un temps proche pour une bonne optimisation

IF110

51



## Pipeline dans un processeur

- ❑ Une opération comporte plusieurs sous-opérations
  - Pipeline pour exécution de ces sous-opérations
  - Une sous-opération utilise une sous-unité du processeur qui n'est pas utilisée par d'autres sous-opérations (si possible...)
- ❑ Exemple de pipeline simple (fictif)
  - LE : Lecture de l'instruction en mémoire
  - DE : Décodage de l'instruction
  - CH : Chargement des registres sources dans l'unité de calcul
  - EX : Exécution du calcul
  - ENR : Enregistrement du résultat dans le registre destination

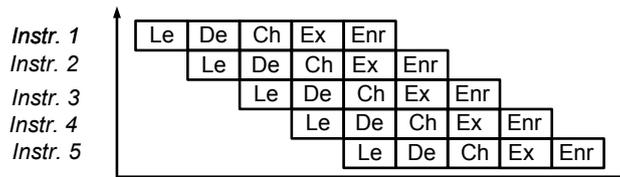
IF110

52



## Pipeline processeur fictif

- ❑ Exécutions décalées de plusieurs instructions en même temps



- ❑ Gain important en utilisant le pipeline
  - Sans : exécution séquentielle de 2 instructions en 10 cycles
  - Avec : exécution parallèle de 5 instructions en 9 cycles
  - Gain théorique car nombreux problèmes en pratique

IF110

53



## Profondeur de pipeline

- ❑ En pratique
  - Actuellement autour de 15 étages
  - Plus d'instructions en cours d'exécution simultanée
  - Permet une montée en fréquence du processeur
    - » Plus d'unités plus petites
    - » Temps de propagation plus courts entre les unités
    - » Raccourcir le temps de réalisation d'un cycle
- ❑ Problème des grandes profondeurs
  - Aléas

IF110

54



## Pipeline et aléas

- ❑ Aléas
  - Problèmes rencontrés lors de l'exécution d'instructions par le pipeline
- ❑ Aléas structurels
  - Des sous-unités du CPU doivent être utilisées simultanément par plusieurs étages du pipeline
- ❑ Aléas de données
  - Une instruction de calcul en cours d'exécution dépend d'une valeur non encore calculée
- ❑ Aléas de contrôle
  - L'instruction suivante dépend du résultat d'une instruction pas encore connu (test)

IF110

55

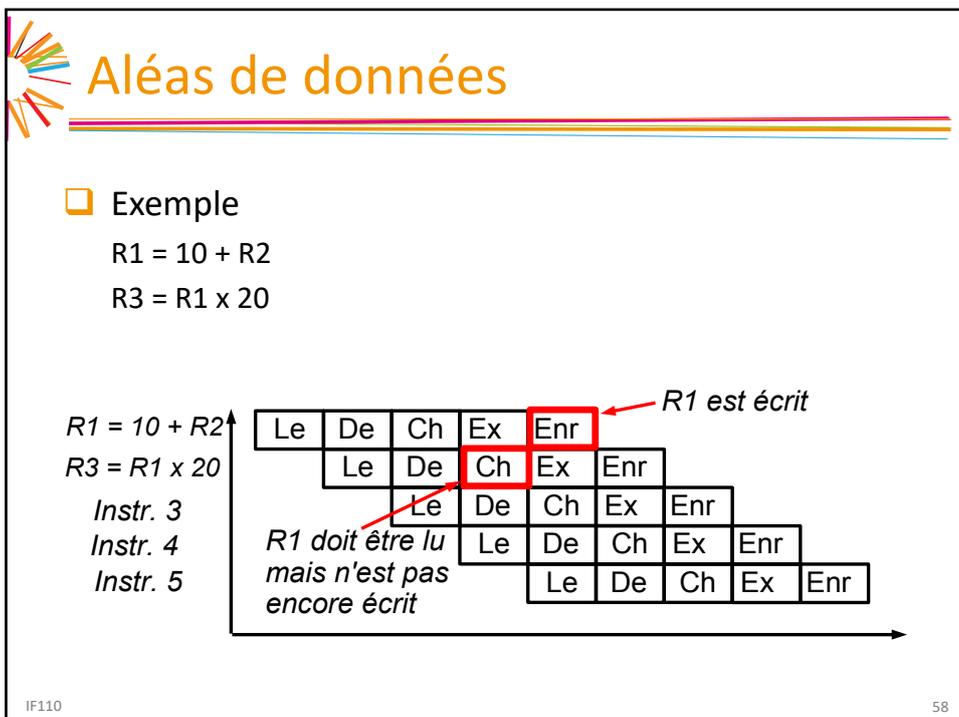
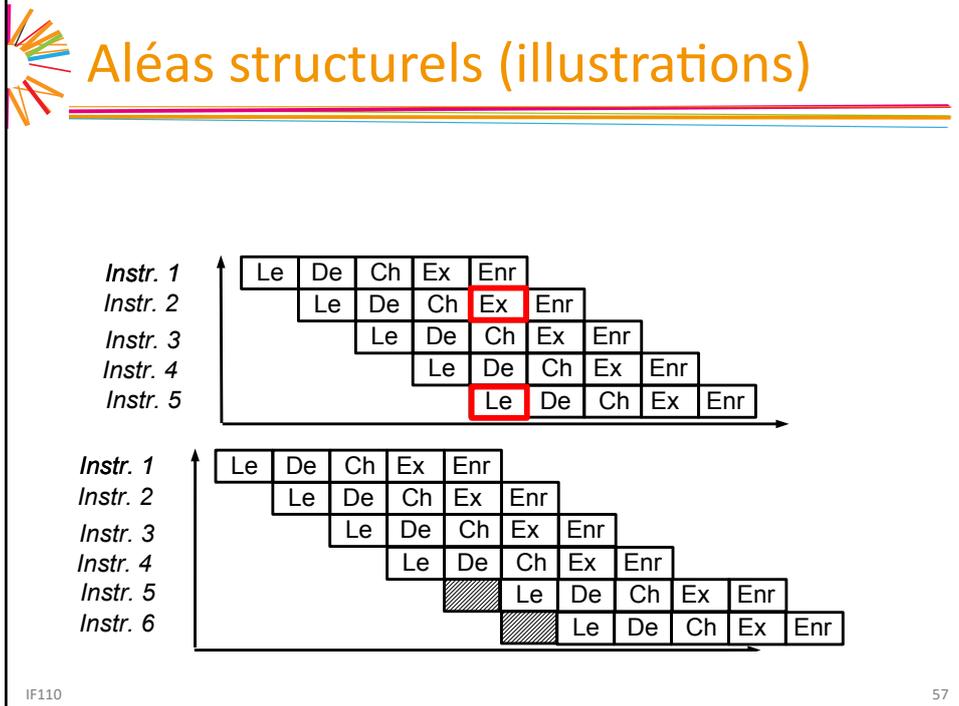


## Aléas structurels

- ❑ Exemple
  - Accès à la mémoire dans les étapes
    - » LE : lecture de l'instruction suivante en mémoire
    - » EX dans le cas d'une opération de lecture/écriture en mémoire
  - Utilise une même sous-unité (accès mémoire) du processeur
- ❑ Solutions
  - Attendre pour une instruction que l'unité soit disponible
  - Dupliquer dans le processeur ces sous-unités
    - » Découper le cache L1 en deux parties : parties données et partie instructions

IF110

56



## Aléas de données - solutions

- Suspension du pipeline**
  - Deuxième instruction suspendue tant que R1 n'est pas écrit

$R1 = 10 + R2$   
 $R3 = R1 \times 20$

- Court-circuit du pipeline**
  - Réinjecter la valeur de R1 dans l'UAL sans attendre son écriture au niveau du banc de registre

$R1 = 10 + R2$   
 $R3 = R1 \times 20$

IF110 59

## Aléas de données – solutions

- Réordonnancement des instructions**
  - Logiciel : fait par le compilateur
  - Matériel : fait par le processeur en interne

$R1 = 10 + R2$   
 $R3 = R1 * 20$   
 $R4 = 2 * R5$   
 $R6 = 10 + R5$

$R1 = 10 + R2$   
 $R4 = 2 * R5$   
 $R6 = 10 + R5$   
 $R3 = R1 * 20$

$R1 = 10 + R2$   
 $R4 = 2 * R5$   
 $R6 = 10 + R5$   
 $R3 = R1 * 20$

IF110 60



## Aléas de contrôle

### ❑ Exemple des branchements

If ( $R1 > 30$ )

then  $R3 = 10 + R1$

else  $R3 = 20 + R1$

### ❑ Solution avec attente



IF110

61



## Aléas de contrôle

### ❑ Prédiction de branchement

- Indispensable pour efficacité du pipeline
- À l'aide de tables statistiques dynamiques
- Prédit le résultat d'un test
- On commence ensuite l'instruction suivante prédite

### ❑ Problème si prédiction erronée

- On a commencé des calculs inutiles
- Vidage du pipeline pour reprendre dans un état correct
  - » Très coûteux en temps
  - » Très pénalisant pour des pipelines profonds

IF110

62



## Prédiction de branchement

- ❑ Principes de la prediction
  - Mémoriser les adresses des branches du programme et regarder celles qui sont souvent atteintes
- ❑ Exemple

```
1    R0 = R2 - 3
2    if R1 = 0 jump suite
3    R3 = 2 x R1
4    R4 = R3 + R1
    suite:
5    R3 = 0
```
- ❑ Deux branches : adresses 3 et 5
- ❑ Prédiction : lors du saut conditionnel à l'adresse 2, on prendra la branche la plus souvent atteinte

IF110

63



## Parallélisation

- ❑ Système du pipeline
  - Parallélisation d'exécution d'instructions
- ❑ Approches complémentaires
- ❑ Ajouter des unités (approche superscalaire)
  - Unités de calculs, de commandes ou même pipeline complet
- ❑ Mettre plusieurs processeurs sur la même puce
  - Approche « multi-core »

IF110

64



## Approche superscalaire

- ❑ Intérêts de l'ajout d'unités
  - Peut faire plusieurs calculs/traitements en même temps
- ❑ Problèmes de l'ajout d'unité
  - Nécessite plus de synchronisation entre unités
  - Sous-utilisation de certaines unités
  - Coût important en terme de transistors et de place
- ❑ Au final
  - Rarement plus de 2 ou 3 unités d'un type

IF110

65



## Approche multi-core

- ❑ Plusieurs cœurs de processeurs sur la même puce
  - Couramment entre 2 et 8
  - Possiblement jusqu'à 32 ou 64 cœurs
- ❑ Utilité : faire du multi-processeur mais avec un seul
  - Possibilité du multi-processeur sur des cartes mères ayant un seul support physique de processeur
- ❑ Mise en œuvre et problèmes
  - Bus de communication entre cores
  - Mise en commun d'un niveau de cache (L3)
- ❑ Problème
  - Cohérence des données entre les caches et la mémoire centrale

IF110

66



## 2 grands types de jeux d'instructions

### ❑ CISC : Complex Instruction Set Computing

- Exemples : processeurs Intel et AMD : familles x86
- Fonctionne en modèle mémoire (3,3) généralement
  - » 3 opérandes, toutes accessibles directement en mémoire
  - » Exemple : `ADD @A, @B, @C`

### ❑ RISC : Reduced Instruction Set Computing

- Exemples : Oracle Sparc et IBM PowerPC
- Fonctionne en modèle mémoire (3,0) généralement
  - » Aucun accès direct en mémoire pour une opération de calcul
  - » Tout passe par des registres
  - » Exemple: `LOAD R1, @B`  
`LOAD R2, @C`  
`ADD R3, R1, R2`  
`STORE R3, @A`

IF110

67



## Jeux d'instructions

### ❑ Taille de codage des instructions

- CISC : taille variable
  - » Selon le nombre d'adresses mémoires codées dans l'instruction
- RISC : taille fixe
  - » 32 bits par exemple

### ❑ Parallélisation facilitée en RISC

- On sait que tous les 32 bits (par exemple) il y a une instruction
- Peut commencer à lire une nouvelle instruction sans avoir fini de décoder la précédente

### ❑ Parallélisation plus compliquée en CISC

- Doit décoder les premiers 32 bits pour savoir si l'instruction est complète ou si on doit lire les 32 bits suivants pour l'avoir en entier
- Durée d'exécution de l'instruction variable (selon nombre accès mémoire, registres ...)

IF110

68



## Processeurs modernes

---

---

- ❑ Fonctionnent en RISC en interne
  - Pour un processeur avec un jeu d'instruction CISC
  - Traduction interne des instructions CISC en micro-opérations de type RISC
    - » Exemple des processeurs AMD et Intel X86
  - Avantage de la traduction
    - » Réordonnement des micro-instructions

IF110

69



## Fonctionnement d'un ordinateur

---

---

Et c'est tout!

Un ordinateur ne sait rien faire de mieux que des  
calculs

IF110

70



## Ce qu'il faut retenir

---

---

- ❑ Une machine est constituée d'un processeur, d'une mémoire vive et de périphériques, le tout interconnecté par un bus
- ❑ Un processeur exécute de façon séquentielle des instructions qui se trouvent en mémoire
- ❑ Chaque instruction est identifiée par un numéro, elle peut
  - Effectuer une opération sur des variables internes (registres)
  - Lire ou écrire en mémoire ses registres
  - Accéder à un périphérique
  - Modifier la prochaine instruction à effectuer (saut)

IF110

71



---

---

## II. Logiciels et programmes



## Ordinateur vu par l'utilisateur

- ❑ L'utilisateur installe des **logiciels**  
Microsoft office, Chrome, Mario
  
- ❑ Logiciel = ensemble de fichiers
  - Fichiers ressources : images, vidéos, musiques...
  - Fichiers programmes : fichier contenant des données et des instructions destinées à être exécutées par un ordinateur
  
- ❑ *In fine*, l'utilisateur lance l'exécution de **programmes**  
Excel, Word, Chrome, Mario

IF110

73

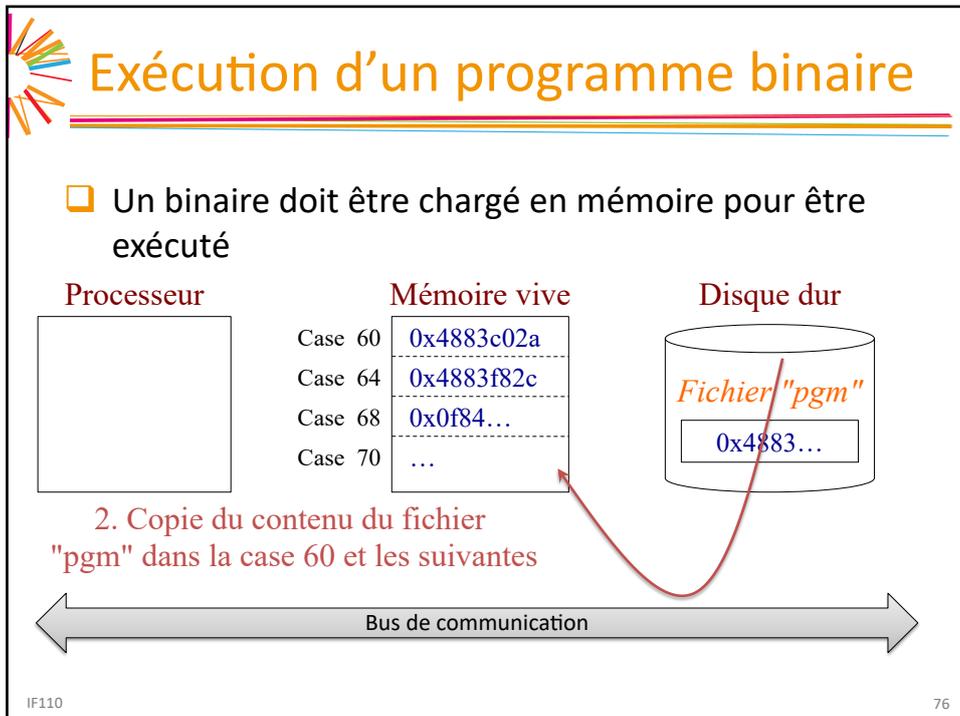
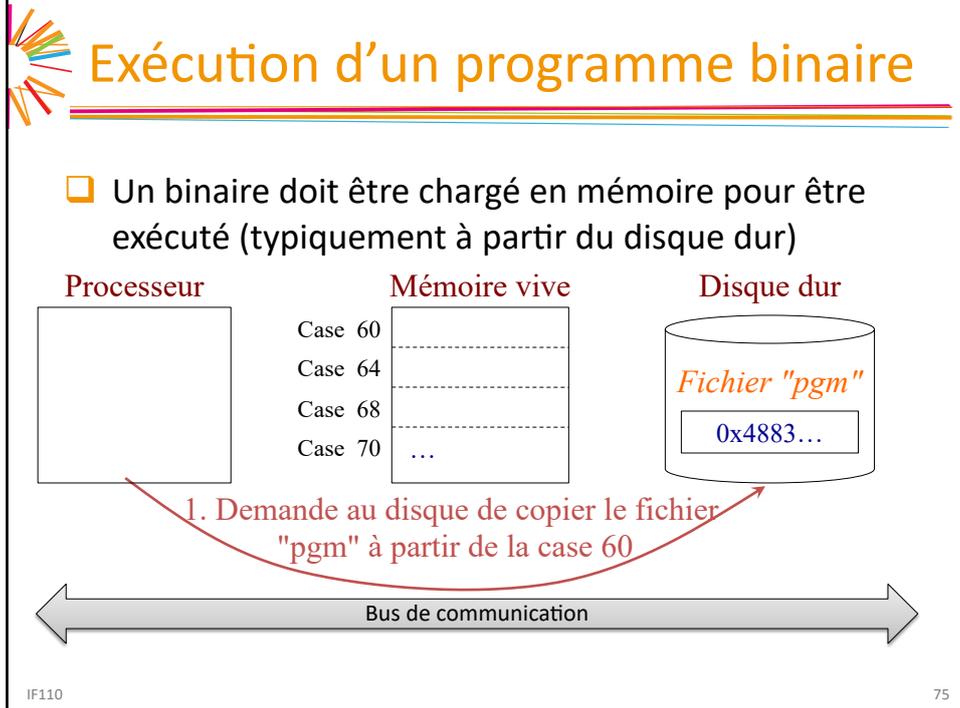


## Qu'est ce qu'un programme ?

- ❑ **Programme binaire** =  
Ensemble d'instructions exécutables par le processeur + des données manipulées par ces instructions
  
- ❑ **Programme source** =  
Ensemble d'opérations abstraites décrivant les actions à effectuer + des données manipulées par ces opérations

IF110

74



## Exécution d'un programme binaire

- Exécution d'un programme à partir de la mémoire centrale

3. L'exécution du programme peut alors démarrer

IF110 77

## Exécution d'un programme source

- Solution 1 : après une traduction vers un programme binaire

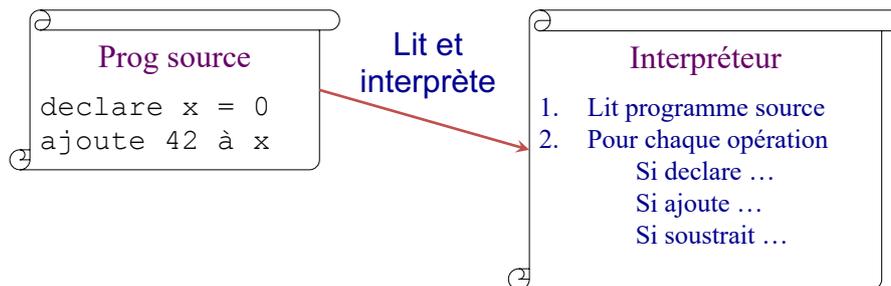
En informatique le traducteur s'appelle un **compilateur**

IF110 78



## Exécution d'un programme source

- ❑ Solution 2 : en le faisant interpréter par un autre programme (appelé interpréteur)



IF110

79



## Quelques exemples de programmes

- ❑ Word, Excel ou Chrome sont des programmes binaires
- ❑ En général, dans un logiciel de jeux
  - Le jeu lui-même est un programme binaire
  - Capable d'interpréter les *mods* qui, eux, sont directement des programmes sources (*mod = extension du jeu*)
- ❑ Les applications Android sont
  - Interprétées avant Android KitKat (version 4.4)
  - Compilées dès qu'elles sont installées depuis Android KitKat
- ❑ Les pages Web dynamiques sont interprétées

IF110

80



---

# Processus et système



---

## Du programme au processus

- ❑ Un **processus** est un programme en cours d'exécution
  - Contient bien sûr les opérations du programme
  - Mais aussi son état à **un instant donné**
    - » Données en mémoire manipulées par le programme
    - » Valeurs des registres du processeur
    - » État des périphériques (fichiers ouverts, connexions réseaux...)

IF110 82



## Gestion des processus

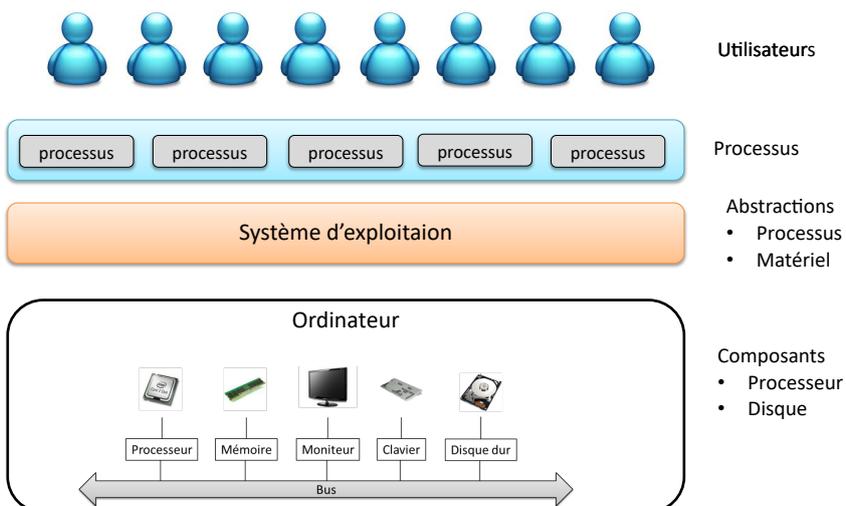
- ❑ Le **système d'exploitation** est un logiciel particulier qui gère les processus  
(Le système est le seul programme qu'on n'appelle pas processus quand il s'exécute)
  
- ❑ Rôle du système d'exploitation
  - Démarrer des processus  
(en chargeant le programme binaire ou l'interpréteur adéquat)
  - Arrêter des processus
  - Offrir une vision de haut niveau du matériel aux processus
  - Offrir des mécanismes de communication inter-processus (*IPC*)

IF110

83



## Architecture globale à l'exécution



IF110

84





## Notions clés du cours

---

- ❑ Un ordinateur
  - Est composé de : mémoire, processeur, périphérique et bus
  - Un processeur exécute des instructions se trouvant en mémoire
- ❑ Un logiciel contient des fichiers
  - Ressources (images, sons, textures...)
  - Programmes (source et/ou binaire)
- ❑ Un programme est une suite d'opérations + des données
- ❑ Un processus est un programme en cours d'exécution
  - Opérations + état à un instant donné
- ❑ Le système gère les processus et abstrait le matériel