

# **Programmation Assembleur X86 IA32**

# ESGI 3

Lien discord :zzQFEe4KnN

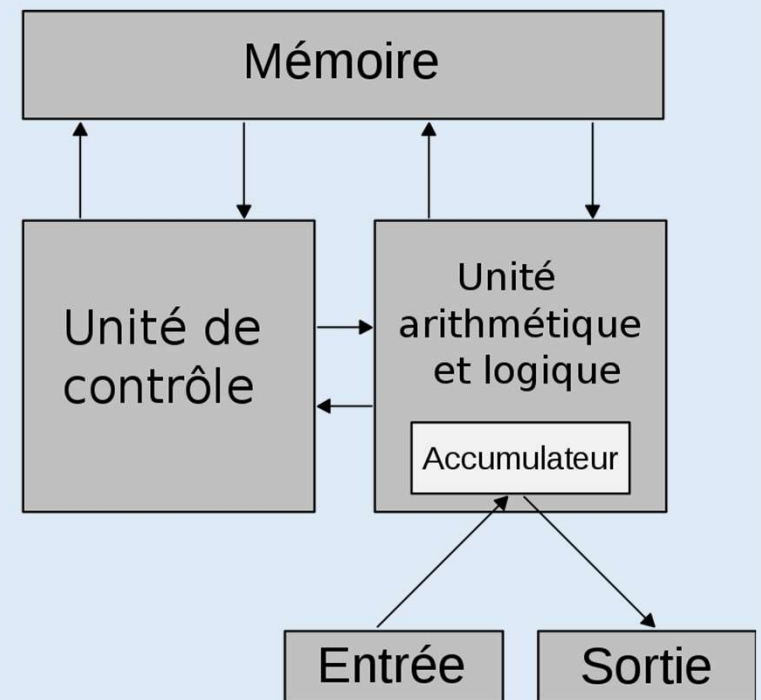
## Plan de la formation

- **Architecture des processeurs**
- **Principes de compilation**
- **Bases de l'assembleur, registres, adressage, pile**
- **Variables, tableaux, branchements, entrées/sorties**
- **Calculs numériques et logiques**
- **Calculs flottants SSE et X87**
- **Fonctions (déclaration, appels, paramètres)**
- **Programmation modulaire**

# Architecture des processeurs

# Architecture des processeurs

- La mémoire (volatile ou permanente) contient les données et les programmes à effectuer
- L'unité de contrôle (CPU ou UCT) réalise les instructions en séquence.
- L'unité de traitement (ALU ou UAL) effectue les opérations de base.
- Les dispositifs d'entrée-sortie communiquent avec le monde extérieur (disques, écran, souris, réseau, USB, etc.)



Architecture de Von Neumann

# Architecture des processeurs

## Types de mémoire

**La mémoire centrale (ou mémoire interne) permettant de mémoriser temporairement les données lors de l'exécution des programmes.**

**La mémoire centrale est réalisée à l'aide de circuits électroniques spécialisés rapides.**

**La mémoire centrale correspond à ce que l'on appelle la mémoire vive.**

# Architecture des processeurs

## Types de mémoire

**La mémoire de masse permet de stocker des informations à long terme, y compris lors de l'arrêt de l'ordinateur.**

**La mémoire de masse correspond aux dispositifs de stockage magnétiques, optiques ou électroniques tels que le disque dur, les CDROM ou DVD-ROM, disques SSD, etc.**

# Architecture des processeurs

## Caractéristiques de la mémoire

- **La capacité** : c'est le nombre total de bits que contient la mémoire. Elle est exprimée en octets.
- **Le format des données** : c'est le nombre de bits que l'on peut mémoriser par case mémoire. Aussi appelé largeur du mot mémorisable (mot de 8, 16, 32 ou 64 bits).
- **Le temps d'accès** : c'est le temps qui s'écoule entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.



# Architecture des processeurs

## Caractéristiques de la mémoire

- **Le temps de cycle** : il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture.
- **Le débit** : c'est le nombre maximum d'informations lues ou écrites par seconde.
- **La non-volatilité** caractérisant l'aptitude d'une mémoire à conserver les données lorsqu'elle n'est plus alimentée électriquement.

# Architecture des processeurs

## Hiérarchie mémoire

Dans l'idéal, on veut :

- Une mémoire extrêmement rapide
- De très grande taille

afin que la mémoire ne limite pas les performances du micro processeur.

Or, le coût d'une mémoire varie comme sa rapidité :

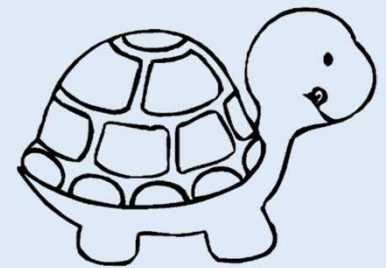
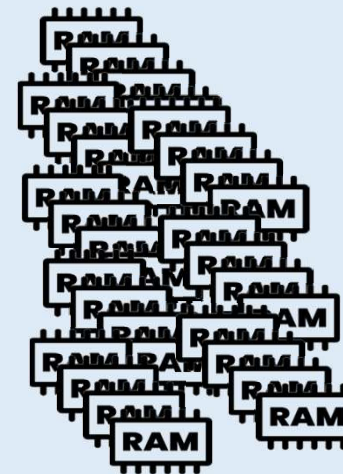
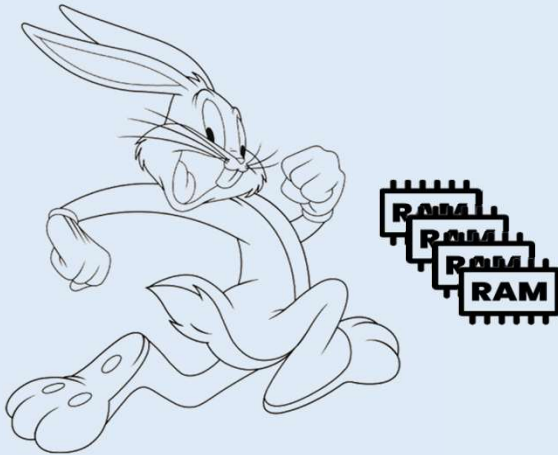
mémoire rapide = mémoire chère

# Architecture des processeurs

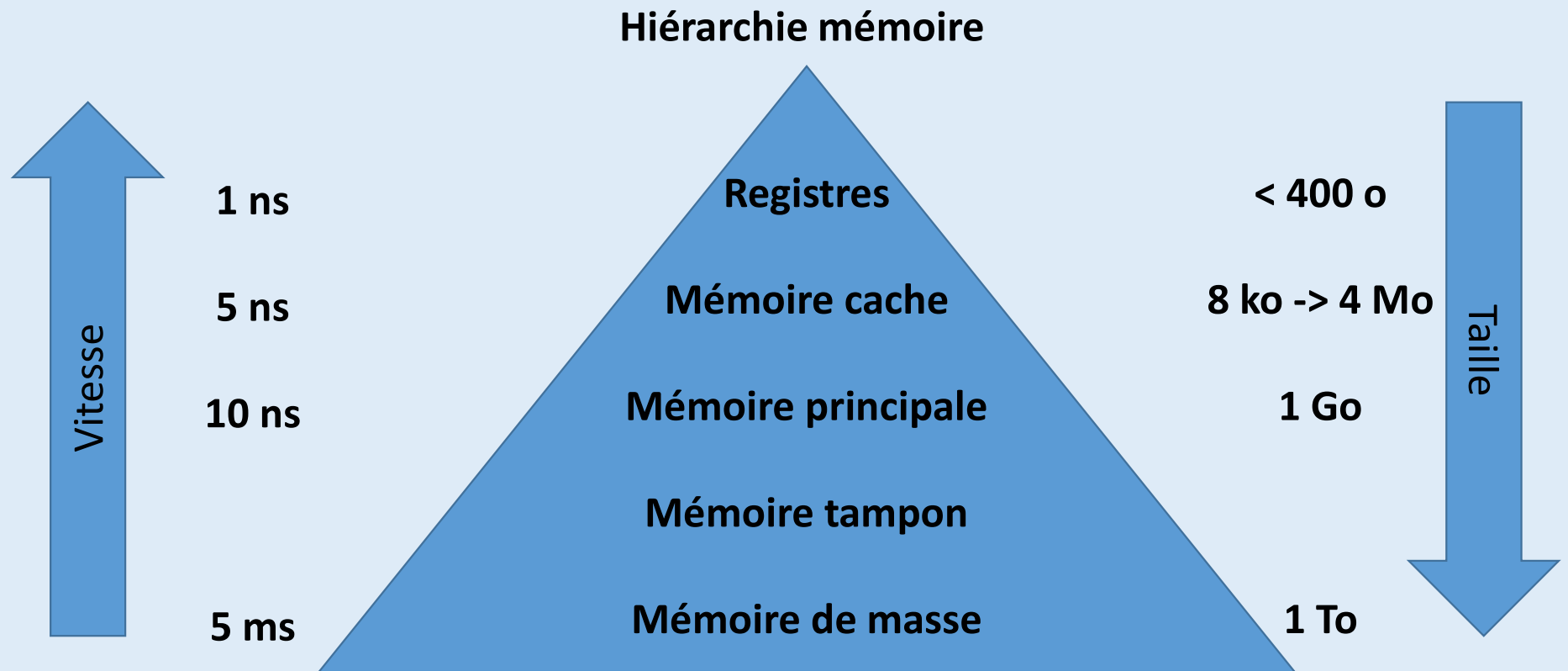
## Hiérarchie mémoire

Compromis coût / performance =

- Peu de mémoires rapides
- Beaucoup de mémoires lentes



# Architecture des processeurs



# Architecture des processeurs

## Hiérarchie mémoire

- **Les registres sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.**
- **La mémoire cache est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.**

# Architecture des processeurs

## Hiérarchie mémoire

- **La mémoire principale est l'organe principal de rangement des informations.**
- **La mémoire tampon sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.**
- **La mémoire de masse est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations.**

# Architecture des processeurs

## Quizz

**Dans un souci de performances, quel type de mémoire va-t-on utiliser en priorité pour les programmes écrits en assembleur ?**

# Architecture des processeurs

## Le CPU

**Le CPU (Central Processing Unit) est un circuit électronique cadencé au rythme d'une horloge interne, délivrant des « tops » à une fréquence donnée.**

**La fréquence d'horloge correspondant au nombre d'impulsions par seconde, s'exprime en Hertz (Hz).**

**La fréquence d'horloge est généralement un multiple de la fréquence du système (FSB, Front-Side Bus), c'est-à-dire un multiple de la fréquence de la carte mère**



# Architecture des processeurs

## Le CPU

A chaque top d'horloge le processeur exécute une action, correspondant à une instruction ou une partie d'instruction.

Le CPI (Cycles Par Instruction) permet de représenter le nombre moyen de cycles d'horloge nécessaire à l'exécution d'une instruction sur un microprocesseur.



La puissance est exprimée en MIPS (Millions d'Instructions Par Seconde)

$$P = \frac{\text{Fréquence(Hz)}}{\text{CPI}}$$

# Architecture des processeurs

## Fonctionnement simplifié du processeur

**Une instruction est l'opération élémentaire que le processeur peut accomplir.**

**Les instructions sont stockées dans la mémoire principale, en vue d'être traitée par le processeur.**

**Une instruction est composée de deux parties :**

- **le code opération, représentant l'action que le processeur doit accomplir ;**
- **le code opérande, définissant les paramètres de l'action.**

# Architecture des processeurs

## Fonctionnement simplifié du processeur

**Le code opérande dépend de l'opération.**

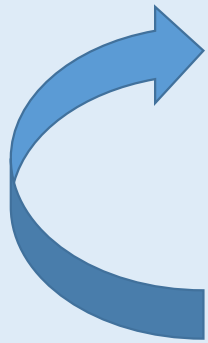
**Il peut s'agir d'une donnée, d'un registre ou bien d'une adresse mémoire.**

**Le nombre d'octets d'une instruction est variable selon le type de donnée (l'ordre de grandeur est de 1 à 4 octets).**

# Architecture des processeurs

## Fonctionnement simplifié du processeur

**En l'absence d'interruption, le processeur réalise en boucle les opérations suivantes :**



- **Lecture de l'instruction située à l'adresse courante (contenue dans le Compteur Ordinal (CO, ou Program Counter PC)**
- **Exécution de l'instruction**
- **Augmentation du CO de la valeur correspondant à la taille de l'instruction et des ses paramètres**

# Architecture des processeurs

**Le CPU est chargé de lire en séquence les instructions contenues dans la mémoire.**

**Chaque instruction est identifiée par un code, suivi par un ou plusieurs arguments.**

**Le CPU lit une instruction, l'exécute, puis passe à la suivante.**

**Ci-contre, le code, pour processeur x86 32 bits, du programme qui permet d'afficher le message « Hello, World » dans la console d'affichage.**

00000000	48656C6C6F2C20576F726C640A00
0000000E	89E5
00000010	83C4FC
00000013	6AF5
00000015	E8(00000000)
0000001A	89C3
0000001C	6A00
0000001E	8D45FC
00000021	50
00000022	6A0E
00000024	68[00000000]
00000029	53
0000002A	E8(00000000)
0000002F	83ECFC
00000032	6A00
00000034	E8(00000000)
00000039	F4

Adresses

Instructions

# Architecture des processeurs

## Quizz

Quelle est la taille de ce programme ?

00000000	48656C6C6F2C20576F726C640A00
0000000E	89E5
00000010	83C4FC
00000013	6AF5
00000015	E8(00000000)
0000001A	89C3
0000001C	6A00
0000001E	8D45FC
00000021	50
00000022	6A0E
00000024	68[00000000]
00000029	53
0000002A	E8(00000000)
0000002F	83ECFC
00000032	6A00
00000034	E8(00000000)
00000039	F4

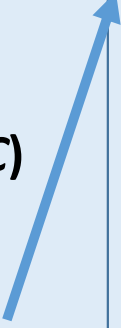
Adresses

Instructions

# Architecture des processeurs

L'instruction à exécuter est repérée par le  
Compteur Ordinal CO (ou *Program Counter : PC*)

Ici, le programme commence à l'adresse  
0000000E, c'est le point d'entrée du programme.



00000000	48656C6C6F2C20576F726C640A00
0000000E	89E5
00000010	83C4FC
00000013	6AF5
00000015	E8(00000000)
0000001A	89C3
0000001C	6A00
0000001E	8D45FC
00000021	50
00000022	6A0E
00000024	68[00000000]
00000029	53
0000002A	E8(00000000)
0000002F	83ECFC
00000032	6A00
00000034	E8(00000000)
00000039	F4

Adresses

Instructions

# Architecture des processeurs

Comme la première instruction est 89  
(« MOV » = affectation) et qu'elle accepte un  
argument (E5 ici), le compteur ordinal va  
augmenter de 2 octets après avoir exécuté  
cette instruction.

Il va alors lire, puis exécuter l'instruction située  
en 00000010.

00000000	48656C6C6F2C20576F726C640A00
0000000E	89E5
00000010	83C4FC
00000013	6AF5
00000015	E8(00000000)
0000001A	89C3
0000001C	6A00
0000001E	8D45FC
00000021	50
00000022	6A0E
00000024	68[00000000]
00000029	53
0000002A	E8(00000000)
0000002F	83ECFC
00000032	6A00
00000034	E8(00000000)
00000039	F4

AdressesInstructions



# Architecture des processeurs

Les instructions les plus courantes permettent de :

- Copier des données d'une zone de la mémoire à une autre
- Effectuer des opérations arithmétiques simples (+, -, \*, /) ou booléennes
- Evaluer des comparaisons entre opérandes
- Réaliser des déplacements dans le programme

00000000 48656C6C6E2C20576F726C640A00

0000000E 89E5

00000010 83C4FC

00000013 6AF5

00000015 E8(00000000)

0000001A 89C3

0000001C 6A00

0000001E 8D45FC

00000021 50

00000022 6A0E

00000024 68[00000000]

00000029 53

0000002A E8(00000000)

0000002F 83ECFC

00000032 6A00

00000034 E8(00000000)

00000039 F4

# Architecture des processeurs

## Exemples d'affectations

- **$r1 \leftarrow \text{CONST}$**

- Écrit CONST dans registre r1
- CONST = constante codée dans le programme

- **$r3 \leftarrow r1 \text{ OP } r2$**

- Lit registres r1 et r2, exécute l'opération OP, écrit le résultat dans r3
- Opérations sur les entiers: add, sub, mul, div, and, or, xor, shift, etc...
- Opérations sur les flottants: fadd, fsub, fmul, fdiv, fsqrt, ...

- **$r2 \leftarrow \text{LOAD } r1$**

- Utilise la valeur contenue dans registre r1 comme **adresse mémoire**
- Lit la valeur stockée en mémoire à cette adresse
- Copie la valeur dans registre r2

# Architecture des processeurs

## Exemples d'instructions de contrôle

- **JNE adresse**

Branchement conditionnel (JNE = Jump if Not Equal)

Saute à l'adresse spécifiée si la dernière comparaison portait sur des valeurs différentes

- **JMP adresse**

Passe directement (Jump) à l'adresse spécifiée. Le Program Counter reçoit la valeur 'adresse'

- **CALL procédure**

Saute à l'adresse de début de la procédure spécifiée

# Architecture des processeurs

## Comparaison RISC / CISC

### CISC (Complex Instruction-Set Computer)

Taille d'instruction variable : Op code + n operandes

Exemple : Instruction (x86) MOV et RCL

Opcode	Mnemonic	Description
8B /r	MOV r/m8,r8	Move r8 to r/m8
89 /r	MOV r/m16,r16	Move r16 to r/m16
89 /r	MOV r/m32,r32	Move r32 to r/m32
8A /r	MOV r8,r/m8	Move r/m8 to r8
8B /r	MOV r16,r/m16	Move r/m16 to r16
8B /r	MOV r32,r/m32	Move r/m32 to r32
8C /r	MOV r/m16,Sreg**	Move segment register to r/m16
8E /r	MOV Sreg,r/m16**	Move r/m16 to segment register
A0	MOV AL,moffs8*	Move byte at (seg.offset) to AL
A1	MOV AX,moffs16*	Move word at (seg.offset) to AX
A1	MOV EAX,moffs32*	Move doubleword at (seg.offset) to EAX
A2	MOV moffs8*,AL	Move AL to (seg.offset)
A3	MOV moffs16*,AX	Move AX to (seg.offset)
A3	MOV moffs32*,EAX	Move EAX to (seg.offset)
B0+ rb	MOV r8,imm8	Move imm8 to r8
B8+ rw	MOV r16,imm16	Move imm16 to r16
B8+ rd	MOV r32,imm32	Move imm32 to r32
C6 /0	MOV r/m8,imm8	Move imm8 to r/m8
C7 /0	MOV r/m16,imm16	Move imm16 to r/m16
C7 /0	MOV r/m32,imm32	Move imm32 to r/m32

Opcode	Mnemonic	Description
D0 /2	RCL r/m8, 1	Rotate 9 bits (CF, r/m8) left once.
D2 /2	RCL r/m8, CL	Rotate 9 bits (CF, r/m8) left CL times.
C0 /2 ib	RCL r/m8, imm8	Rotate 9 bits (CF, r/m8) left imm8 times.
D1 /2	RCL r/m16, 1	Rotate 17 bits (CF, r/m16) left once.
D3 /2	RCL r/m16, CL	Rotate 17 bits (CF, r/m16) left CL times.
C1 /2 ib	RCL r/m16, imm8	Rotate 17 bits (CF, r/m16) left imm8 times.
D1 /2	RCL r/m32, 1	Rotate 33 bits (CF, r/m32) left once.
D3 /2	RCL r/m32, CL	Rotate 33 bits (CF, r/m32) left CL times.
C1 /2 ib	RCL r/m32, imm8	Rotate 33 bits (CF, r/m32) left imm8 times.
D0 /3	RCR r/m8, 1	Rotate 9 bits (CF, r/m8) right once.
D2 /3	RCR r/m8, CL	Rotate 9 bits (CF, r/m8) right CL times.
C0 /3 ib	RCR r/m8, imm8	Rotate 9 bits (CF, r/m8) right imm8 times.
D1 /3	RCR r/m16, 1	Rotate 17 bits (CF, r/m16) right once.
D3 /3	RCR r/m16, CL	Rotate 17 bits (CF, r/m16) right CL times.
C1 /3 ib	RCR r/m16, imm8	Rotate 17 bits (CF, r/m16) right imm8 times.
D1 /3	RCR r/m32, 1	Rotate 33 bits (CF, r/m32) right once.
D3 /3	RCR r/m32, CL	Rotate 33 bits (CF, r/m32) right CL times.
C1 /3 ib	RCR r/m32, imm8	Rotate 33 bits (CF, r/m32) right imm8 times.
D0 /0 ROL r/m8, 1		Rotate 8 bits r/m8 left once.
D2 /0 ROL r/m8, CL		Rotate 8 bits r/m8 left CL times.
C0 /0 ib ROL r/m8, imm8		Rotate 8 bits r/m8 left imm8 times.
D1 /0 ROL r/m16, 1		Rotate 16 bits r/m16 left once.
D3 /0 ROL r/m16, CL		Rotate 16 bits r/m16 left CL times.
C1 /0 ib ROL r/m16, imm8		Rotate 16 bits r/m16 left imm8 times.
D1 /0 ROL r/m32, 1		Rotate 32 bits r/m32 left once.
D3 /0 ROL r/m32, CL		Rotate 32 bits r/m32 left CL times.
C1 /0 ib ROL r/m32, imm8		Rotate 32 bits r/m32 left imm8 times.
D0 /1	ROR r/m8, 1	Rotate 8 bits r/m8 right once.
D2 /1	ROR r/m8, CL	Rotate 8 bits r/m8 right CL times.
C0 /1 ib	ROR r/m8, imm8	Rotate 8 bits r/m8 right imm8 times.
D1 /1	ROR r/m16, 1	Rotate 16 bits r/m16 right once.
D3 /1	ROR r/m16, CL	Rotate 16 bits r/m16 right CL times.
C1 /1 ib	ROR r/m16, imm8	Rotate 16 bits r/m16 right imm8 times.
D1 /1	ROR r/m32, 1	Rotate 32 bits r/m32 right once.
D3 /1	ROR r/m32, CL	Rotate 32 bits r/m32 right CL times.
C1 /1 ib	ROR r/m32, imm8	Rotate 32 bits r/m32 right imm8 times.

# Architecture des processeurs

## Comparaison RISC / CISC

### CISC (Complex Instruction-Set Computer)

**Opérations complexes autorisées : traitement des chaînes de caractères (SCASB), des polynômes ou des complexes, recherche dans une table (XLAT), etc.**

**Modes d'adressage mémoire complexes autorisés**

**Accès mémoire largement utilisé**

**Exemple: IBM 360, VAX, Intel X86**

# Architecture des processeurs

## Comparaison RISC / CISC

**Des études statistiques sur des programmes ont montré les faits suivants.**

- **80 % des programmes n'utilisent que 20 % du jeu d'instructions.**
- **Les instructions les plus utilisées sont :**
  - **les instructions de chargement et de rangement,**
  - **les appels de sous-routines.**
- **Les appels de fonctions sont très gourmands en temps : sauvegarde et restitution du contexte et passage des paramètres et de la valeur de retour.**
- **80 % des variables locales sont des entiers.**
- **90 % des structures complexes sont des variables globales.**
- **La profondeur maximale d'appels imbriqués est en moyenne de 8. Une profondeur plus importante ne se rencontre que dans 1 % des cas.**

# Architecture des processeurs

## Comparaison RISC / CISC

### RISC (Reduced Instruction-Set Computer)

#### Codage uniforme des instructions

Toutes les instructions sont codées avec un même nombre de bits, généralement un mot machine. L'op-code se trouve à la même position pour toutes les instructions. Ceci facilite le décodage des instructions.

#### Registres indifférenciés et nombreux

Tous les registres peuvent être utilisés dans tous les contextes. Il n'y a par exemple pas de registre spécifique pour la pile. Les processeurs séparent cependant les registres pour les valeurs flottantes des autres registres.

# Architecture des processeurs

## Comparaison RISC / CISC

### RISC (Reduced Instruction-Set Computer)

#### Limitation des accès mémoire

**Les seules instructions ayant accès à la mémoire sont les instructions de chargement et de rangement. Toutes les autres instructions opèrent sur les registres. Il en résulte une utilisation intensive des registres.**

#### Nombre réduit de modes d'adressage

**Il n'y pas de mode d'adressage complexe. Les modes d'adresses possibles sont généralement immédiat, direct, indirect et relatifs.**



# Architecture des processeurs

## Comparaison RISC / CISC

### RISC (Reduced Instruction-Set Computer)

#### Nombre réduit de types de données

Types supportés : Entiers de différentes tailles (8, 16, 32 et 64 bits) et nombres flottants en simple et double précision.

Certains processeurs CISC comportent des instructions pour le traitement des chaînes de caractères, des polynômes ou des complexes

Exemple: MIPS, Alpha, PowerPC, Sparc

# Architecture des processeurs

## Jeux d'instructions RISC / CISC

- **CISC**

**Code plus compact, prend moins de place en mémoire**

**Jeu d'instructions plus facile à faire évoluer**

- **Exemple x86: 16  $\Rightarrow$  32  $\Rightarrow$  64 bits, MMX  $\Rightarrow$  SSE  $\Rightarrow$  SSE2  $\Rightarrow$  SSE3  $\Rightarrow$  etc...**

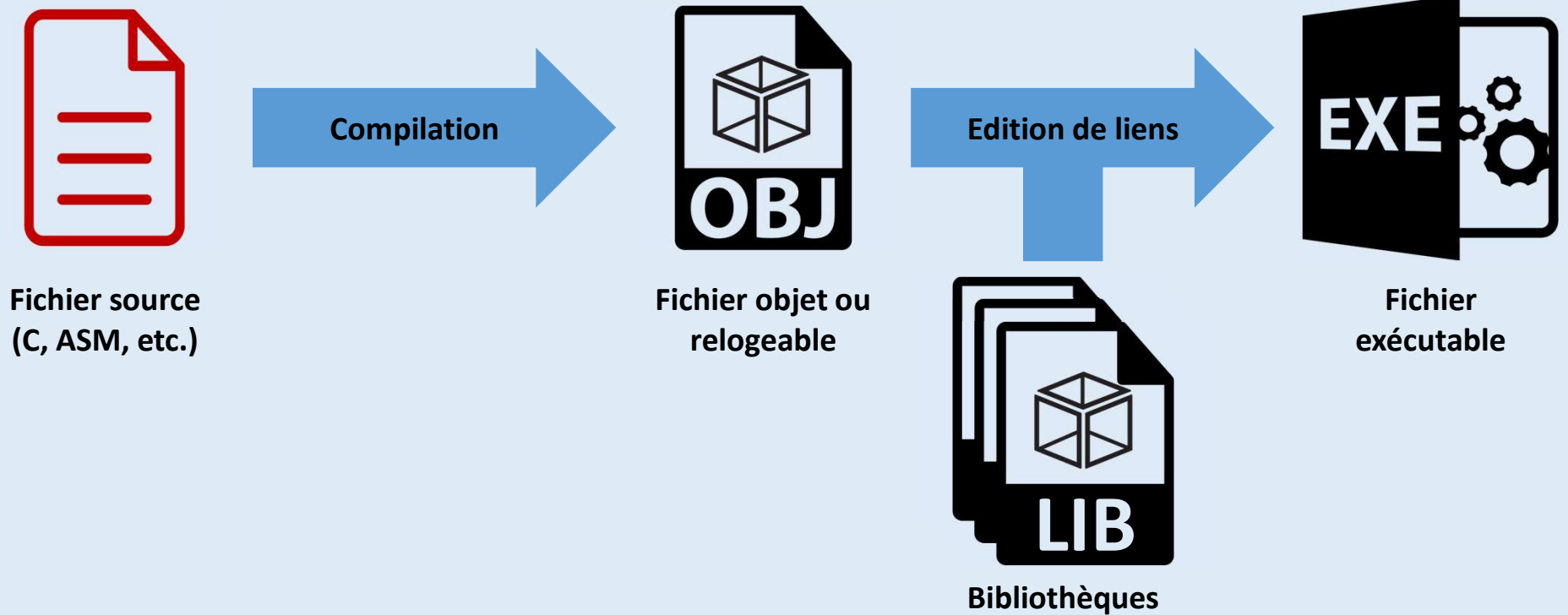
- **RISC**

**Microarchitecture plus simple**

# Principes de compilation

# Principes de compilation

Pour obtenir un programme exécutable à partir d'un code source, au moins deux étapes sont nécessaires



# Principes de compilation

## Installation des outils

Compilateur NASM : <http://www.nasm.us/>

Sur Windows, NASM s'installe avec les droits administrateurs.

Editeur de liens Golang : <http://www.godevtool.com/>



- *Donner les droits d'accès sur les répertoires d'installation*
- *Mettre à jour \$PATH après l'installation des outils*

# Principes de compilation

## Installation des outils

Ligne de commande NASM (crée le fichier .obj)

`nasm.exe -f win32 CodeSource.asm -l Listing.txt`

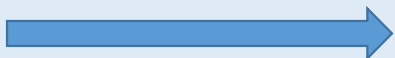
-f win32 = syntaxe  
d'entrée IA32

Nom du fichier  
source

Nom du fichier  
listing

Suggestion de fichier compilation32.bat

`nasm.exe -f win32 %1.asm -l %1.txt`



`C:\Users\Bda> compilation32 MonCode`

# Principes de compilation

## Installation des outils

### Ligne de commande Golink

```
Golink.exe Fichier.obj /console Kernel32.dll User32.dll  
Gdi32.dll msvcrt.dll /entry:Main
```

Fichier à  
transformer en .exe

Crée un fichier à lancer depuis  
l'invite de comande (cmd)

Point d'entrée du  
programme

Liste de bibliothèques à utiliser

### Suggestion de fichier link.bat

```
Golink.exe %1.obj /console Kernel32.dll User32.dll Gdi32.dll  
msvcrt.dll /entry:Main
```

➡ C:\Users\Bda> link MonCode