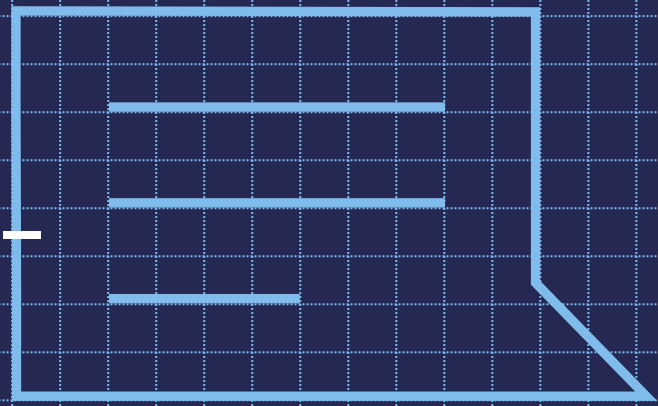
Analyse statique avancée - Assembleur

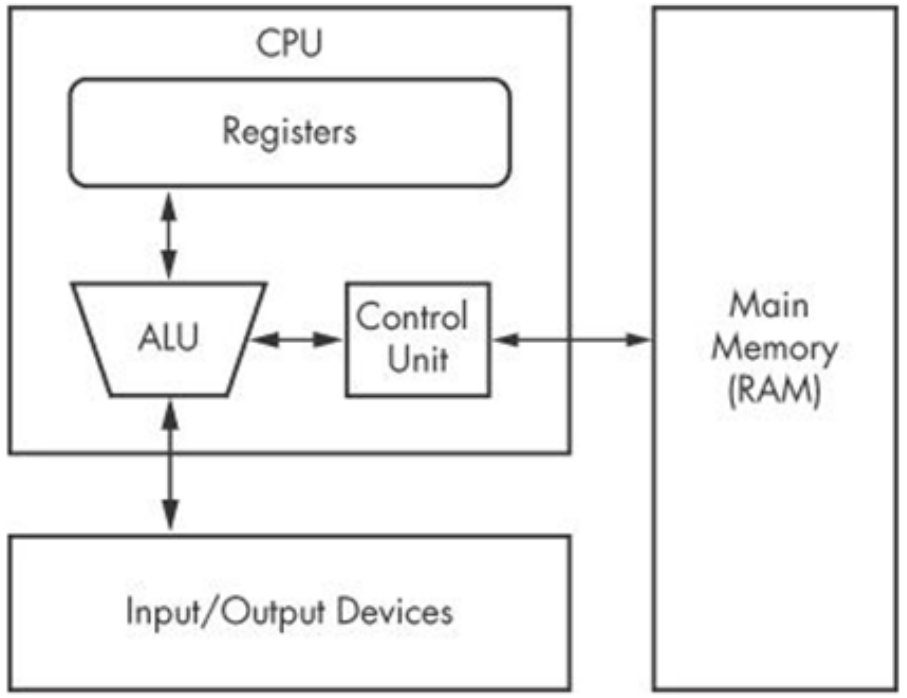


EFREI-PARIS

B. AIT SALEM

Architecture machine de base

- Unité de contrôle (Control unit)
  - Recherche une instruction dont l'adresse se trouve dans le **compteur ordinal** (**program counter**) à partir de la RAM en utilisant un registre appelé **instruction pointer** (le registre **EIP** dans les architecture intel), la décode puis l'envoie à l'UAL pour l'exécuter.
- Registres
  - Mémoire au niveau de la CPU
  - Plus petite, mais plus rapide que la RAM
- UAL (ALU (Arithmetic Logic Unit))
  - Exécute une opération arithmétique et logique et place le résultat dans un registre ou dans la RAM



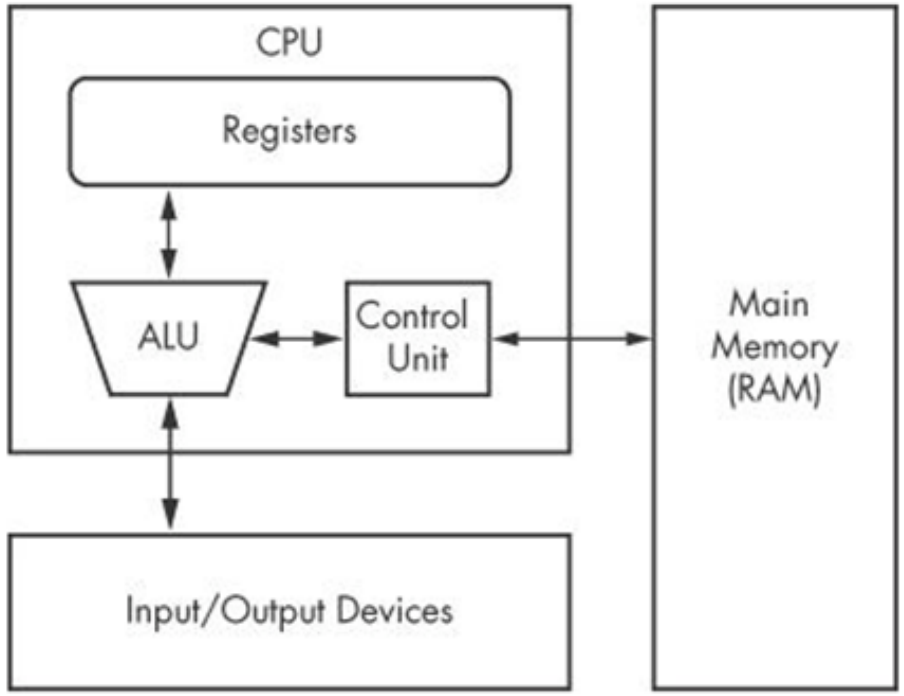
Architecture de Von Neumann

- CPU (**Central Processing Unit**) exécute des instructions machines
- RAM sauvegarde les données et le code
- I/O interface avec les périphériques

Instructions machines

- **Mnemonic** suivi d'un ou plusieurs opérandes "**operands**"

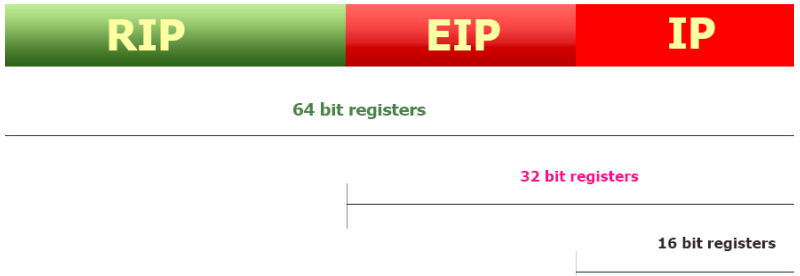
  Exemple: **mov ecx 0x42**
  - Charger dans le registre **ecx** la valeur **42** (hex)
- En langage binaire cette instruction devient:
  - **0xB942000000**
  - **mov ecx** est la représentation de **0xB9** (en hexadecimal)



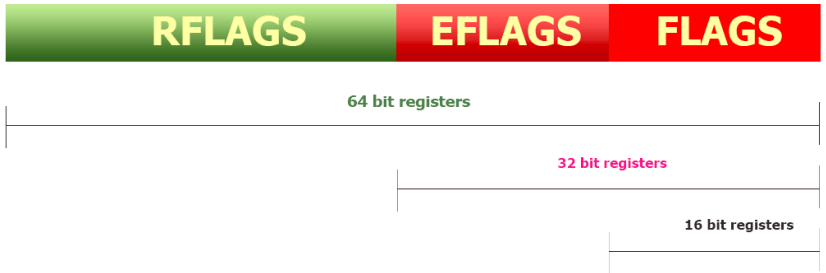
Architecture de Von Neumann

Registres

REGISTER	PURPOSE
ECX	Counter in loops
ESI	Source in string/memory operations
EDI	Destination in string/memory operations
EBP	Base frame pointer
ESP	Stack pointer



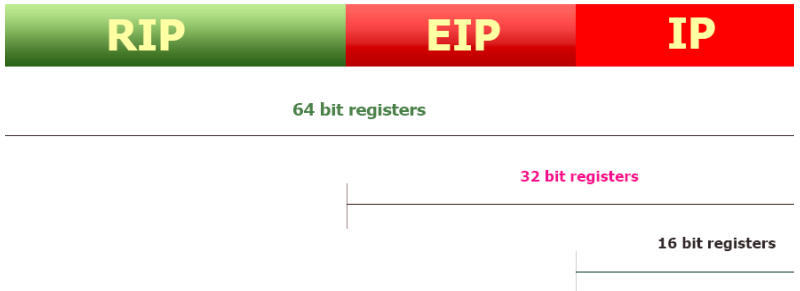
EAX contient souvent la valeur de retour lors d'un appel de fonction
La Multiplication et la division utilisent EAX et EDX

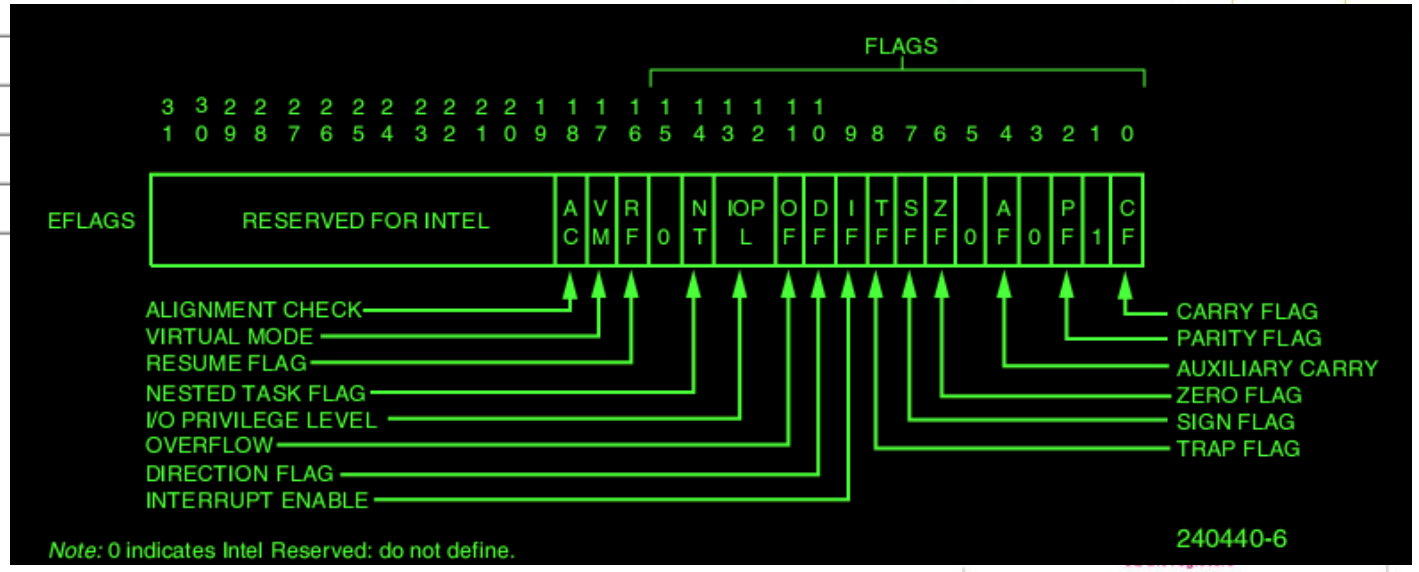


Bytes—8 bits. AL, BL, CL , …
Word—16 bits. AX, BX, CX, …
Double word—32 bits. EAX, EBX, ECX, …
Quad word—64 bits. RAX, RBX, RCX, …

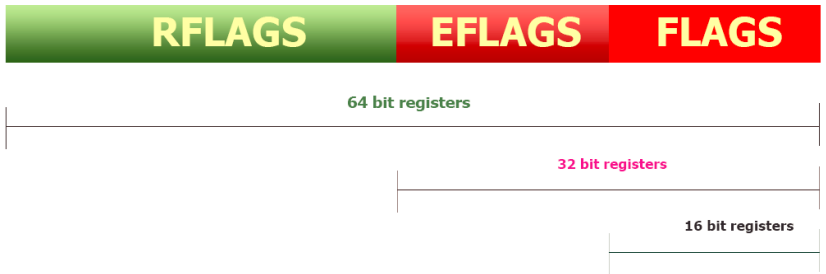
Registres

REGISTER	PURPOSE
ECX	Counter in loops
ESI	Source in string/memory operations
EDI	Destination in string/memory operations
EBP	Base frame pointer
ESP	Stack pointer





EAX contient souvent la valeur de retour lors d'un appel de fonction
La Multiplication et la division utilisent EAX et EDX



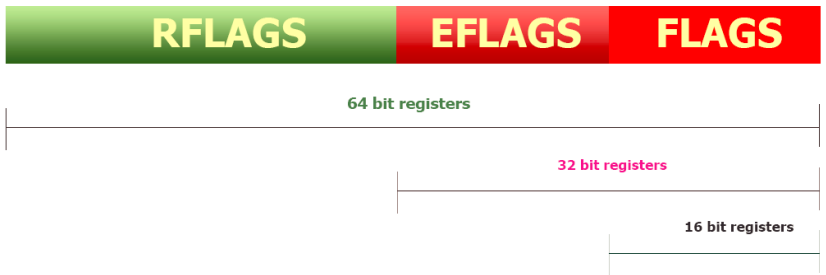
Bytes—8 bits. AL, BL, CL , ...
Word—16 bits. AX, BX, CX, ...
Double word—32 bits. EAX, EBX, ECX, ...
Quad word—64 bits. RAX, RBX, RCX, ...

Registres

- **ZF** Zero flag
  - Activé lorsque le résultat d'une opération est égal à zéro
- **CF** Carry flag
  - Activé lorsque le résultat d'une opération est plus grand ou plus petit que la destination
- **SF** Sign Flag
  - Activé lorsque le résultat est négatif ou lorsque le bit de poids fort est mis à 1 après une opération arithmétique
- **TF** Trap Flag
  - Utilisé pour le débogage. Lorsqu'il est activé, le processeur exécute une seule instruction à la fois.
- **DF** Direction Flag
  - A voir ci-dessous avec l'instruction **rep**

EAX contient souvent la valeur de retour lors d'un appel de fonction

La Multiplication et la division utilisent EAX et EDX



Bytes—8 bits. AL, BL, CL , …
Word—16 bits. AX, BX, CX, …
Double word—32 bits. EAX, EBX, ECX, …
Quad word—64 bits. RAX, RBX, RCX, …

Mouvement de données

Addressing Mode	Description	NASM Examples
Register	Registers hold the data to be manipulated. No memory interaction. Both registers must be the same size.	mov ebx, edx add al, ch
Immediate	The source operand is a numerical value. Decimal is assumed; use **h** for hex.	mov eax, 1234h mov dx, 301
Direct	The first operand is the address of memory to manipulate. It's marked with brackets.	mov bh, 100 mov[4321h], bh
Register Indirect	The first operand is a register in brackets that holds the address to be manipulated.	mov [di], ecx
Based Relative	The effective address to be manipulated is calculated by using **ebx** or **ebp** plus an offset value.	mov edx, 20[ebx]
Indexed Relative	Same as Based Relative, but **edi** and **esi** are used to hold the offset.	mov ecx,20[esi]
Based Indexed-Relative	The effective address is found by combining Based and Indexed Relative modes.	mov ax, [bx][si]+1

Endianness

- **Quelle est la plus petite unité de données adressable ?**

- **Big-Endian**
  - Placer les octets de poids forts en premier
  - 0x42 comme valeur de 64-bits sera sauvegardée en mémoire ainsi **0x00000042**
- **Little-Endian**
  - Placer les octets de poids faibles en premier
  - 0x42 comme valeur de 64-bit sera sauvegardée en mémoire ainsi **0x42000000**
- Les protocoles réseaux utilisent le format big-endian
- Les programmes x86 utilisent le format little-endian

Endianness Exemple d'une adresses IP

- 127.0.0.1 qui vaut **7F 00 00 01** en hex
- Elle est envoyée sur le réseau comme **0x7F000001**
- Elle sera par contre sauvegardée comme **0x0100007F** en mémoire centrale

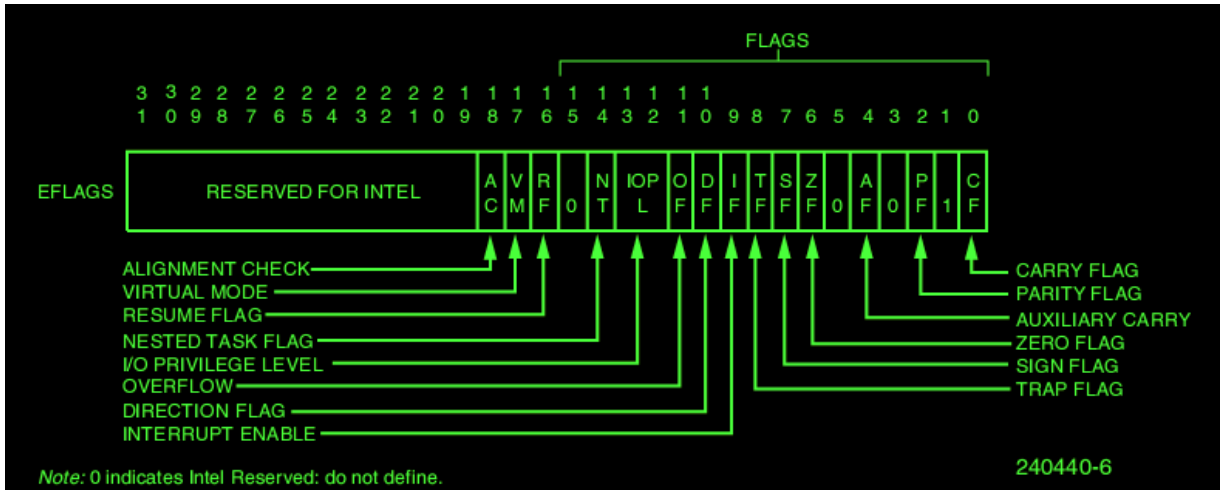
Mov vs lea (Load Effective Address)

- **lea destination, source**
- lea eax, [ebx+8]
  - mettre ebx + 8 dans eax
- **Vs mov**
  - mov eax, [ebx+8]
  - Copie la donnée se trouvant à l'adresse ebx+8 dans eax

mov eax, [ebx+8] → place **0x20** dans eax
Lea eax, [ebx + 8] → place la valeur **0xB30048** dans eax

Instructions Arithmétiques

- **sub** Subtracts
- **add** Adds
- **inc** Increments
- **dec** Decrements
- **mul** Multiplies
- **div** Divides
- **Shr**



→ Ces opérations modifient le registre d'état

→ Le résultat de l'opération est sauvegardé dans le premier opérande

Exemple:
Après une instruction **sub**, le flag **ZF** est mis à **1** si le résultat est égal à zéro et le flag **CF** est positionné à **1** si la destination est plus petite que la valeur soustraite.

NOP

- Ne fait rien et EIP est incrémenté (la CPU passe à l'instruction suivante)
- Son opcode = **0x90**

➢ Permet à un attaquant d'exécuter un code malveillant (shellcode) même en étant imprécis dans l'adresse du saut vers ce code.

Instructions de comparaison

- **test**
  - Compare deux valeurs en réalisant un **AND** des deux valeurs, mais ne modifie pas les deux valeurs
  - **test eax, eax**
    - Active le drapeau Zéro (**ZF**) **si eax est égale à zéro**
- **cmp eax, ebx**
  - Ressemble à l'instruction sub, mais ne modifie pas la valeur des deux opérandes
  - Le drapeau **ZF** est mis à **1 si les deux opérandes sont égaux**

cmp dst, src	ZF	CF
dst = src	1	0
dst < src	0	1
dst > src	0	0

Branchements conditionnels

- **Elle suivent toujours une instruction arithmétique ou souvent une instruction de comparaison**
- **jz loc**
  - Se brancher à l'adresse **loc** si le drapeau Zéro **ZF** est activé (**1**)
- **jnz loc**
  - Se brancher à l'adresse **loc** si le drapeau Zéro **ZF** est désactivé (**0**)

`jz loc`	Jump to specified location if ZF = 1.
`jnz loc`	Jump to specified location if ZF = 0.
`je loc`	Same as `jz`, but commonly used after a `cmp` instruction. Jump will occur if the destination operand equals the source operand.
`jne loc`	Same as `jnz`, but commonly used after a `cmp`. Jump will occur if the destination operand is not equal to the source operand.
`jg loc`	Performs signed comparison jump after a `cmp` if the destination operand is greater than the source operand.
`jge loc`	Performs signed comparison jump after a `cmp` if the destination operand is greater than or equal to the source operand.
`ja loc`	Same as `jg`, but an unsigned comparison is performed.
`jae loc`	Same as `jge`, but an unsigned comparison is performed.
`jl loc`	Performs signed comparison jump after a `cmp` if the destination operand is less than the source operand.
`jle loc`	Performs signed comparison jump after a `cmp` if the destination operand is less than or equal to the source operand.
`jb loc`	Same as `jl`, but an unsigned comparison is performed.
`jbe loc`	Same as `jle`, but an unsigned comparison is performed.
`jo loc`	Jump if the previous instruction set the overflow flag (OF = 1).
`js loc`	Jump if the sign flag is set (SF = 1).
`jecxz loc`	Jump to location if ECX = 0.

Quelques opérateurs

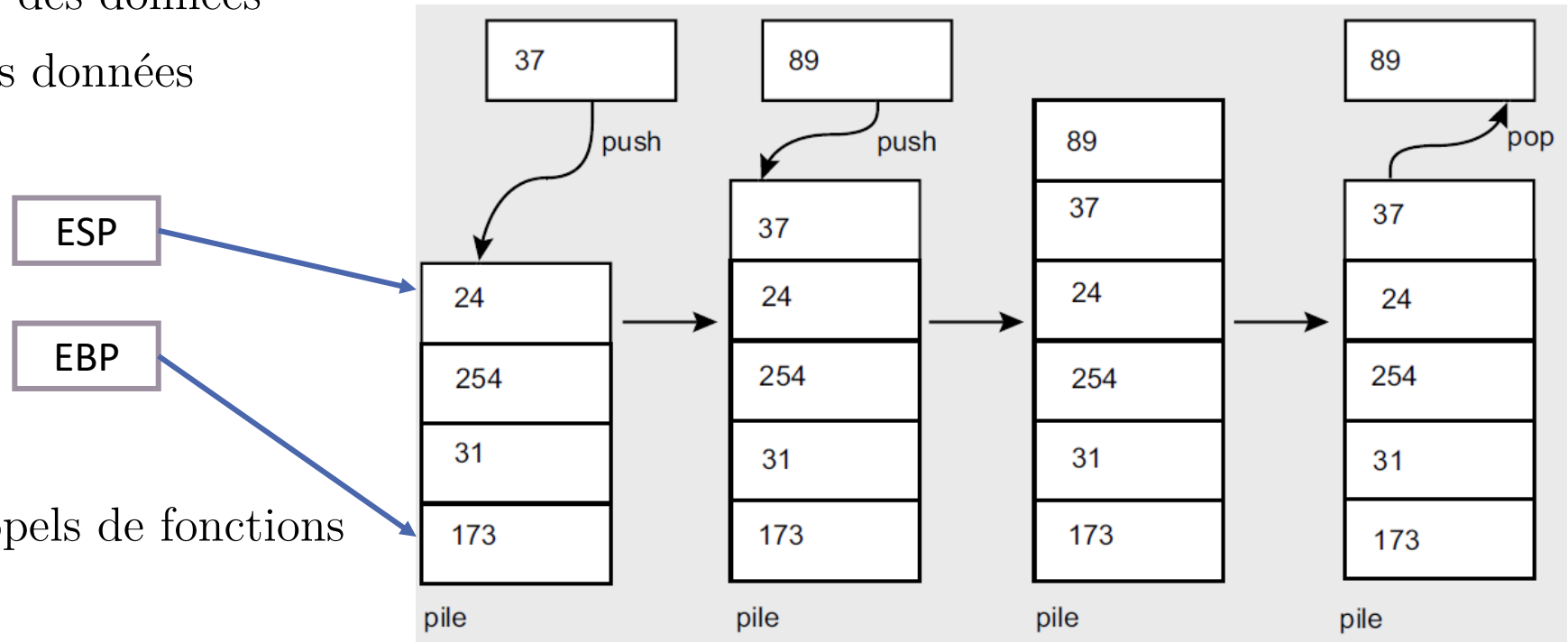
- L'operateur **offset** : nous donne le déplacement d'une variable ou un label par rapport au début de son segment
  - **Mov BX, OFFSET VAR1**
- L'operateur **PTR**: peut être utilisé pour forcer la taille d'un opérande
- **DB, DW, DD**, servent à définir resp. un byte, un word et double word dans le segment data

```
ByteVal    DB    05h, 06h, 08h, 09h, 0Ah, 0Dh, '$'
WordVal    DW    5510h, 6620h, 0A0Dh, '$'

MOV    AX, WORD PTR ByteVal + 2        ;  AX = 0908h
MOV    BL, BYTE PTR WordVal + 4        ;    BL = 0Dh

MOV    BX, OFFSET WordVal              ; BX pointe vers WordVal
INC    [BX]                        ; ambigu – la taille n'est pas spécifiée
INC    BYTE PTR [BX]                   ;  modifie la valeur en mémoire en 11h
INC    WORD PTR [BX]                   ; modifie la valeur en mémoire en 5511h
```

La pile

Pile – Elément clés

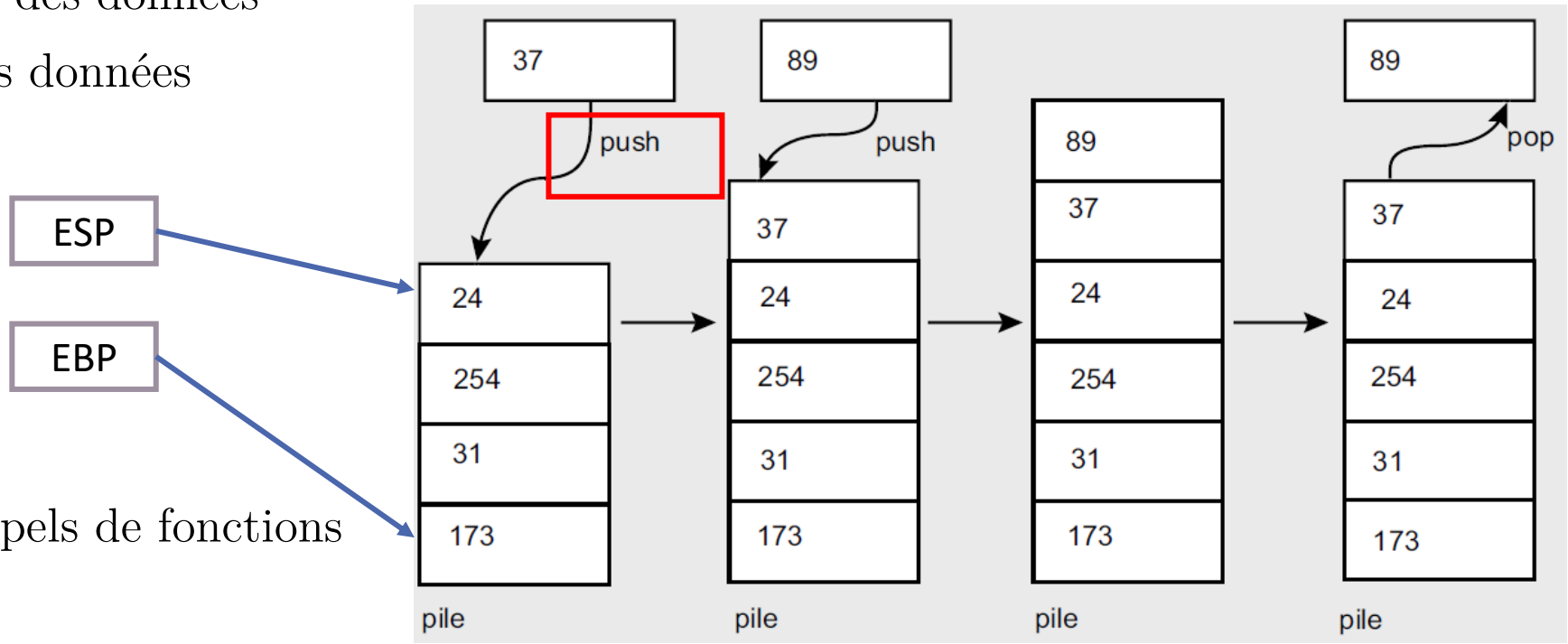
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données



- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

Pile – Elément clés

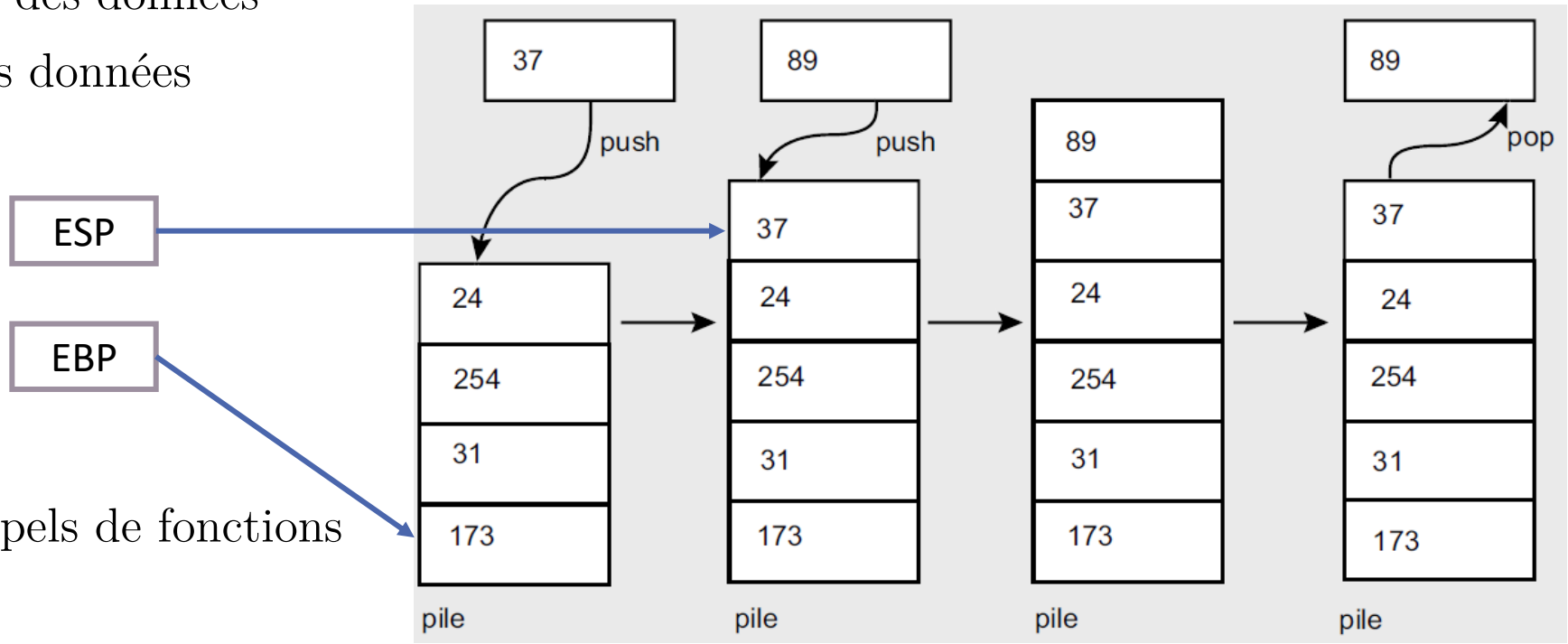
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données



- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

Pile – Elément clés

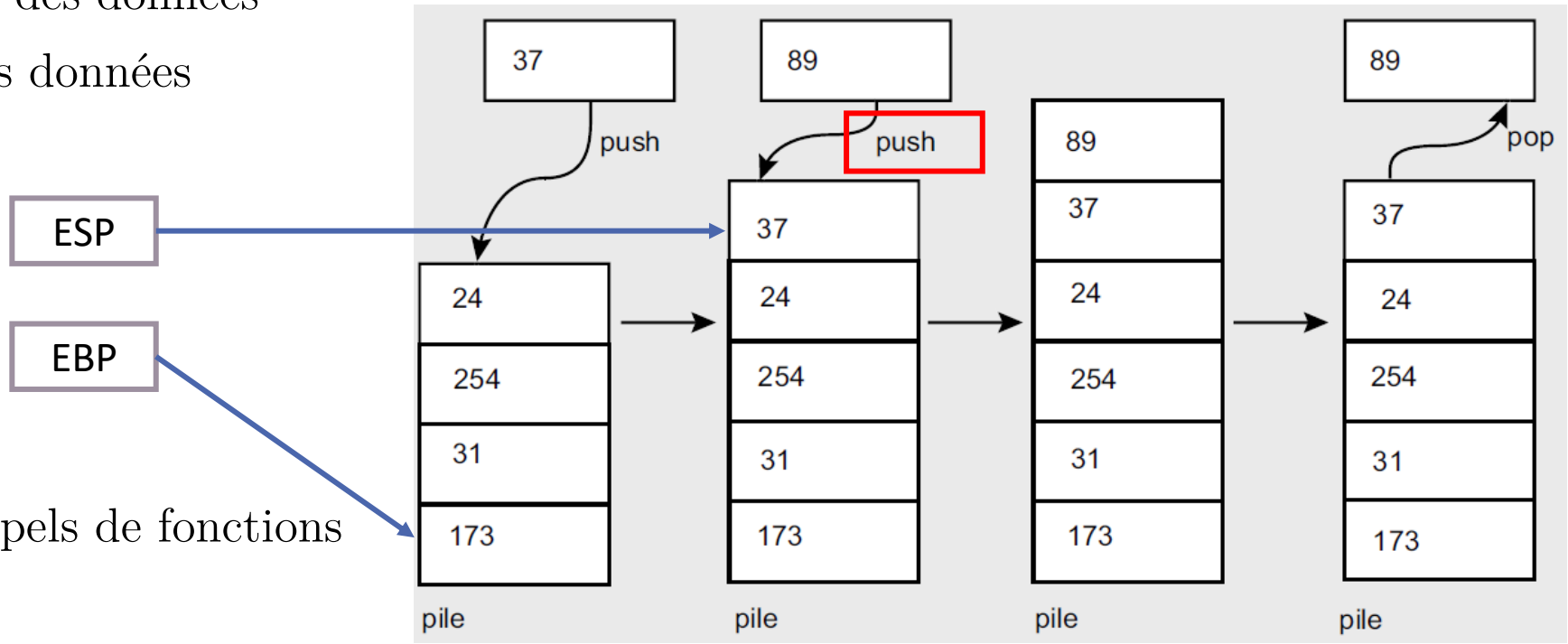
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données



- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

Pile – Elément clés

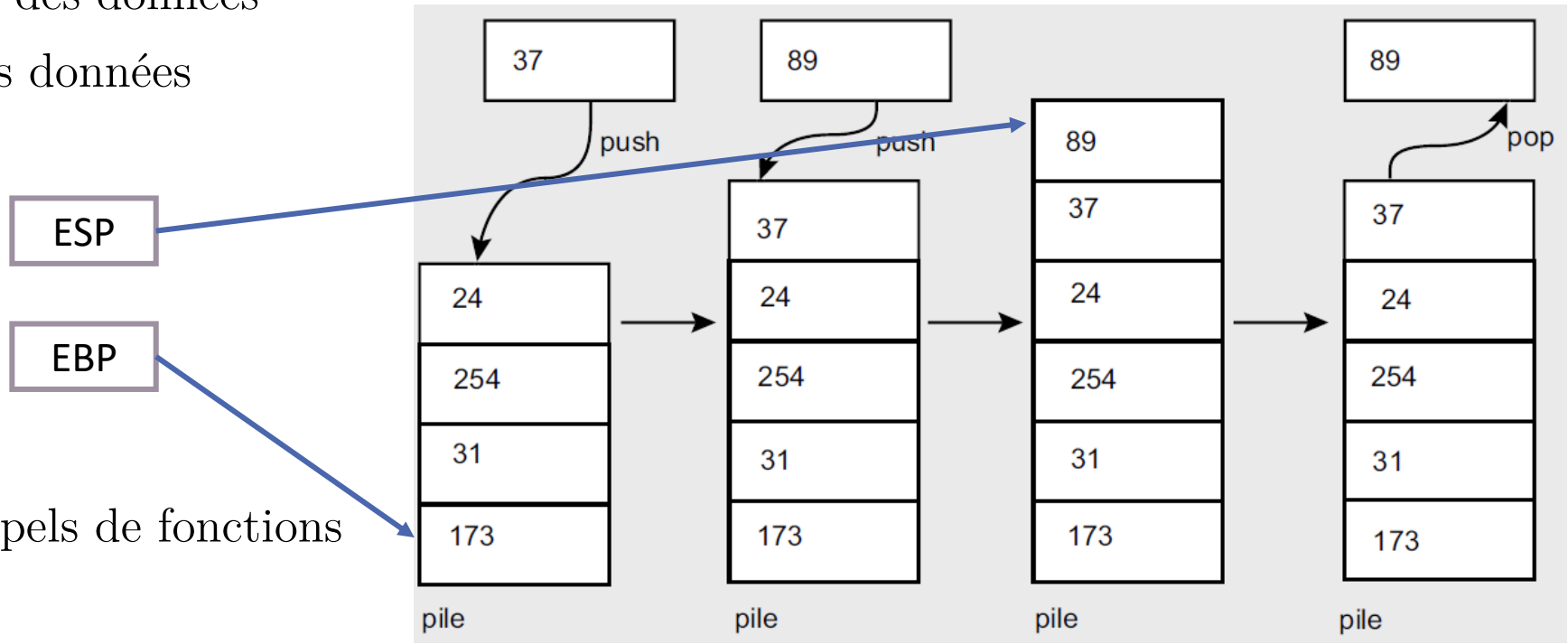
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données



- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

Pile – Elément clés

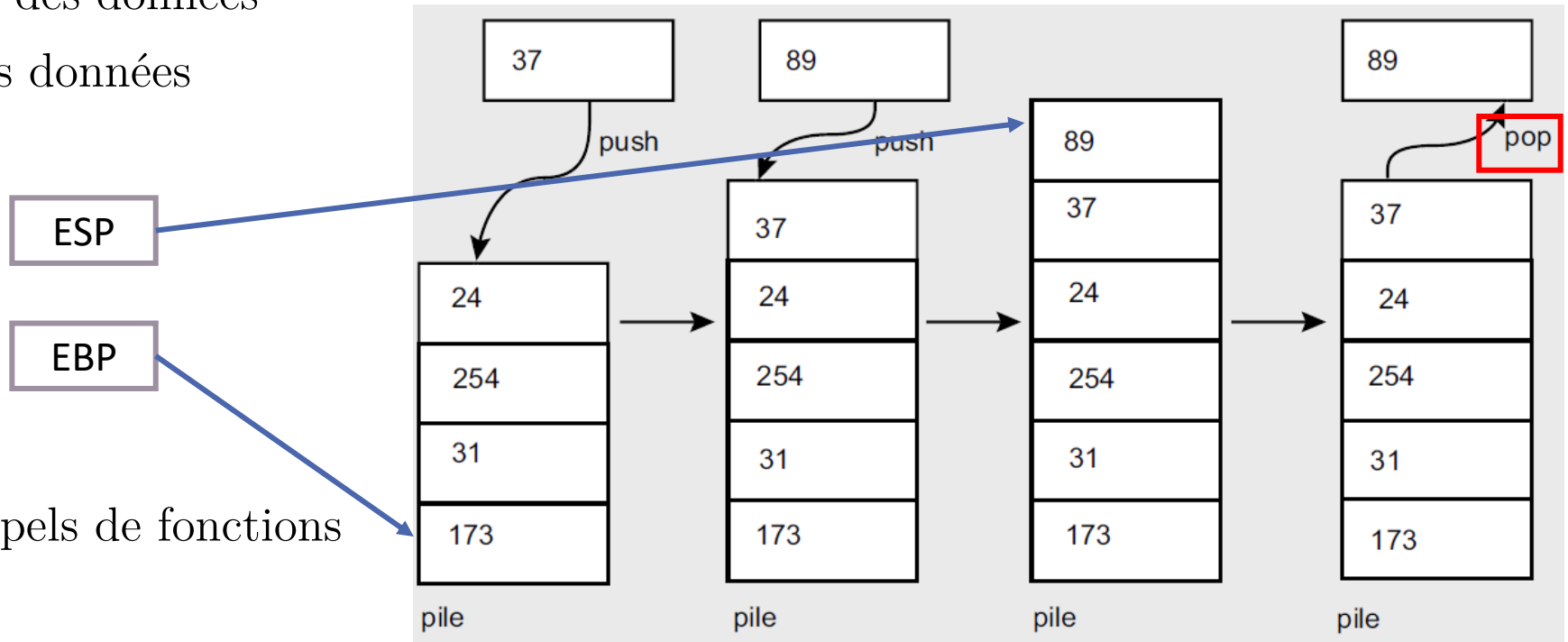
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données



- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

Pile – Elément clés

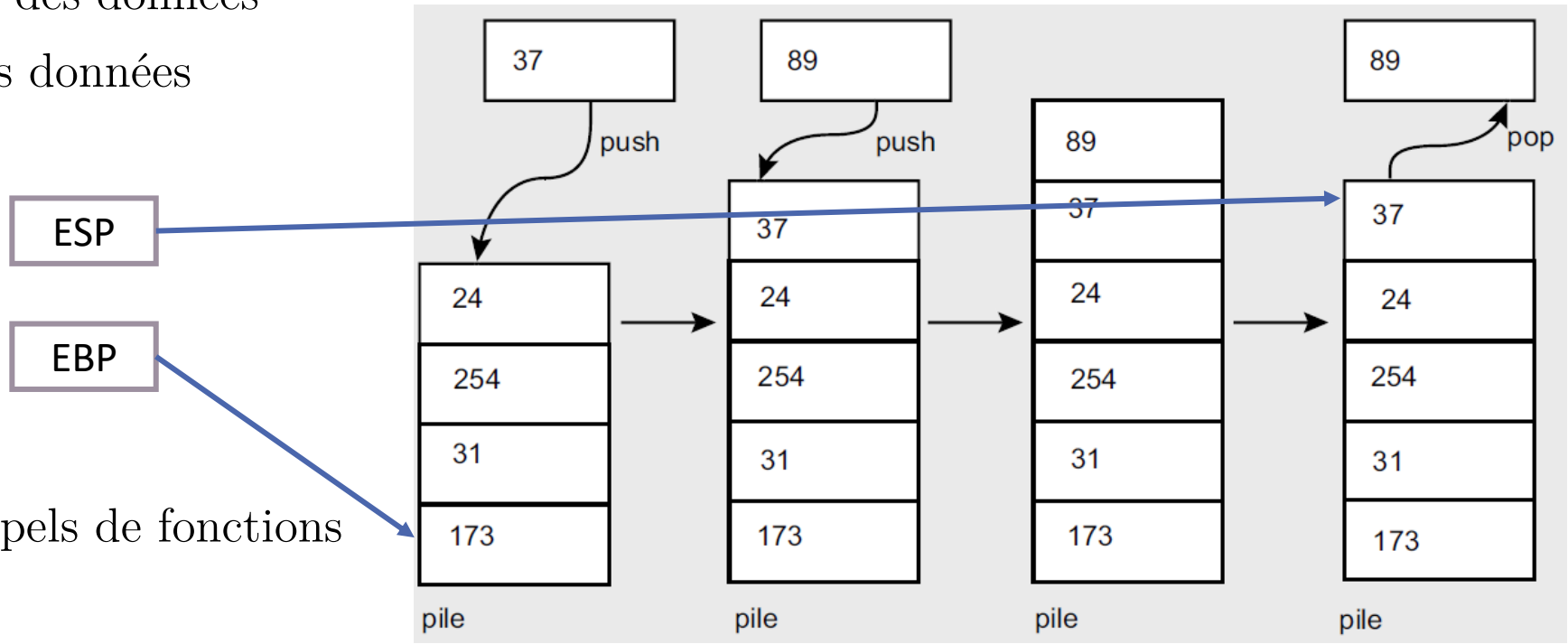
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données



- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

Pile – Elément clés

- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données



- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

Pile – Appel de fonction

A suivre ici

Le résumé Vidéo

Vidéo à regarder

Convention d'appel de fonction

- **cdecl** : les paramètres sont mis dans la pile de la droite vers la gauche et c'est à la fonction appelante de nettoyer la pile à la fin de la fonction appelée. La valeur de retour est sauvegardée dans le registre EAX

```
int test(int x, int y, int z);
int a, b, c, ret;

ret = test(a, b, c);
```

```
push c
push b
push a
call test
add esp, 12
mov ret, eax
```

- **stdcall**: Même fonctionnement que **cdecl** sauf qu'ici c'est à la fonction appelée de nettoyer la pile à la fin de son exécution
- C'est ce qui est utilisé par défaut par l'API Windows

```
int test(int x, int y, int z);
int a, b, c, ret;

ret = test(a, b, c);
```

```
push c
push b
push a
call test
mov ret, eax
```

- **fastcall**: Les premiers paramètres (souvent deux) sont passés à la fonction appelée via les registres (EDX et ECX dans Windows) et le reste obéit aux mêmes règles que la convention cdecl.