

***MI01 – Automne 2011***

# **Systèmes à microprocesseurs : programmation**

Stéphane Bonnet

Poste : 52 56

Courriel : [stephane.bonnet@utc.fr](mailto:stephane.bonnet@utc.fr)

# Plan général A11

- Définitions, concepts et principes généraux
- Architecture IA-32 (*Intel Architecture 32-bit*)
- Programmation IA-32
- Principes de gestion de la mémoire
- Programmation SIMD : MultiMedia eXtensions

# Sommaire

1. Systèmes à microprocesseur
2. La mémoire
3. Le processeur
4. Notre cas d'étude : la famille x86

# Sommaire

## 1. Systèmes à microprocesseur

1.1. Solutions programmées

1.2. Architectures

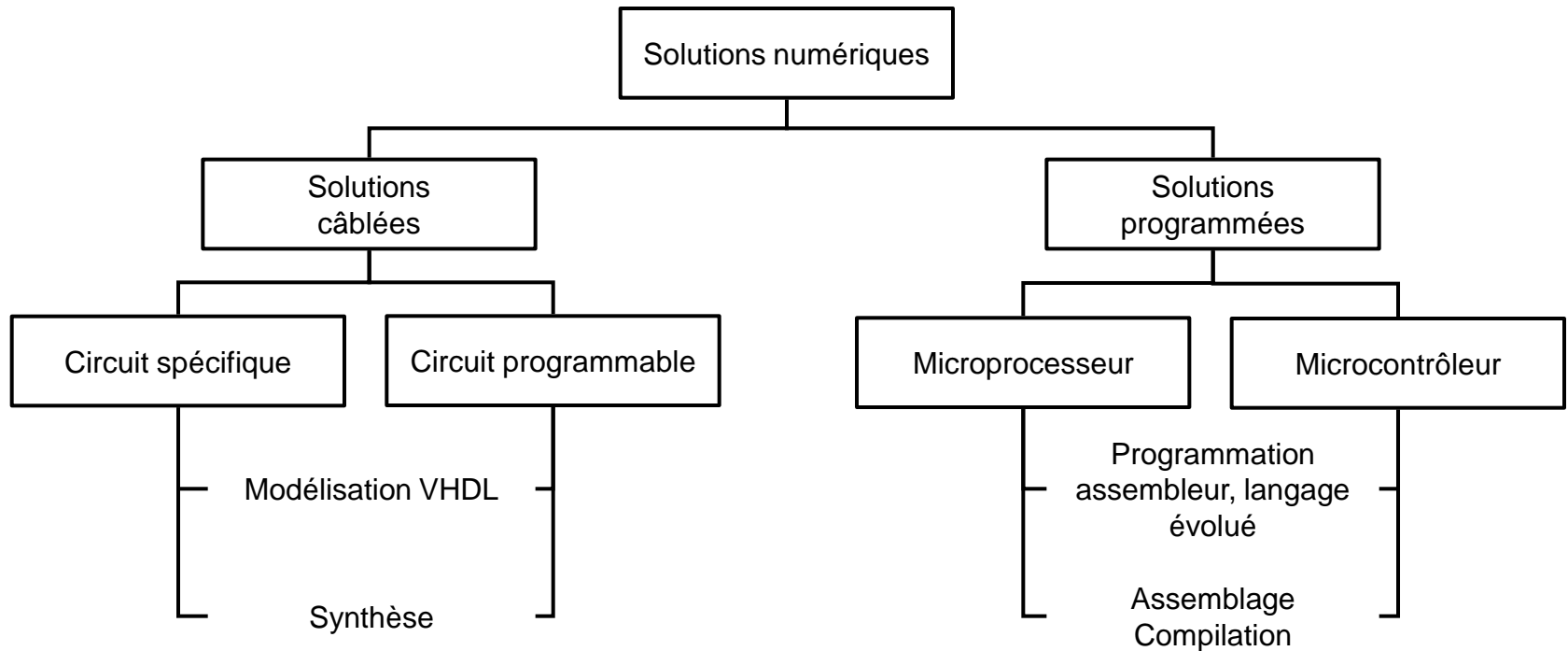
1.3. Composants d'un système

## 2. La mémoire

## 3. Le processeur

## 4. Notre cas d'étude : la famille x86

# 1.1. Solutions programmées



+ performance

- souplesse

=> Solutions spécifiques

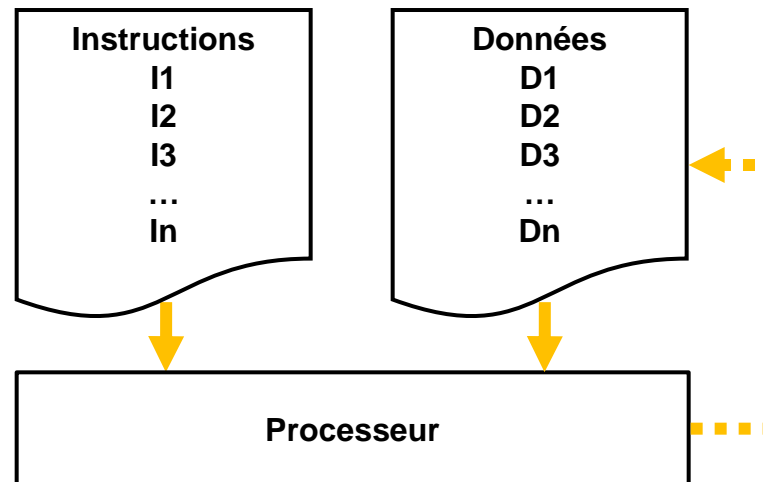
- performance

+ souplesse

=> Solutions d'usage  
général

# 1.1. Solutions programmées

- Un programme
  - Suite d'opérations élémentaires séquentielles (algorithmes)
  - Jeu de données
    - Constantes (jamais modifiées)
    - Variables (Modifiées par les opérations)



# 1.1. Solutions programmées

## Un programme...

- En C...

```
int sum = 0;
int tab[4] = {12,42,18,23};

int main(void)
{
    int i;
    for(i = 0; i < 4; i++)
        sum = sum + tab[i];
}
```

- En assembleur (IA32)...

```
.data
    sum dd 0
    tab dd 12,42,18,23

.code
main:
    mov ecx, 0
    mov eax, [sum]
@loop:
    add eax, [tab + ecx * 4]
    inc ecx
    cmp ecx, 4
    jne @loop
    mov [sum], eax
```

# 1.1. Solutions programmées

- Chaque instruction élémentaire est composée de:
  - Une opération à réaliser
  - Des opérandes (données) sur lesquelles réaliser l'opération (opérandes sources)
  - Un opérande spécifiant l'emplacement où doit être stocké le résultat (opérande destination).

Exemple :  $Z \leftarrow X + Y$

- X et Y sont les opérandes source
- Z est l'opérande destination
- + est l'opération



# 1.1. Solutions programmées

- Les opérandes peuvent être localisés
  - Dans la mémoire de données de la machine (mémoire centrale)
  - Dans des registres du processeur
- Les opérandes peuvent être
  - Explicites, ils sont identifiés dans l'instruction
  - Implicites, l'instruction agit toujours sur des opérandes situés au même endroit.

# 1.1. Solutions programmées

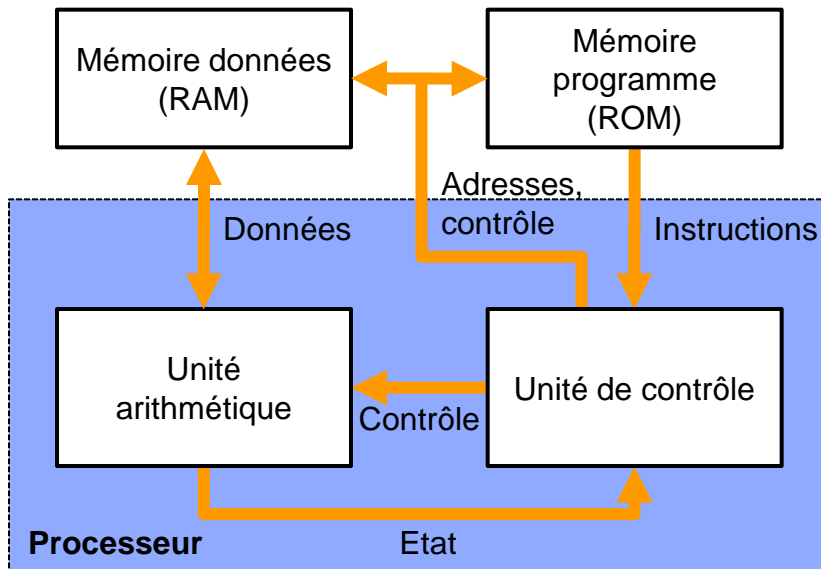
- Instructions et données sont stockées en mémoire, le processeur effectuant sans cesse les opérations suivantes :
  - Recherche de l'instruction à exécuter
  - Décodage de l'instruction
  - Chargement éventuel de données depuis la mémoire
  - Exécution et mise à jour de son état
  - Stockage éventuel des résultats en mémoire.

# 1.2. Architecture d'un calculateur

- Deux architectures dominantes
  - Modèle HARVARD
  - Modèle VON NEUMANN

# 1.2. Architecture

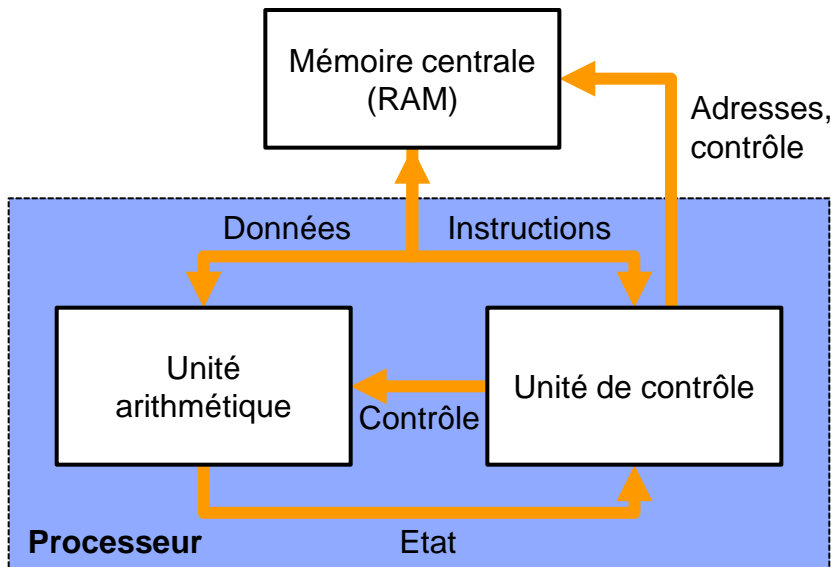
## Modèle HARVARD



- Mémoires séparées
  - Instructions
  - Données
- Structure complexe
- Microcontrôleurs
  - Microchip PIC
  - Atmel AVR...

# 1.2. Architecture

## Modèle VON NEUMANN (PRINCETON)

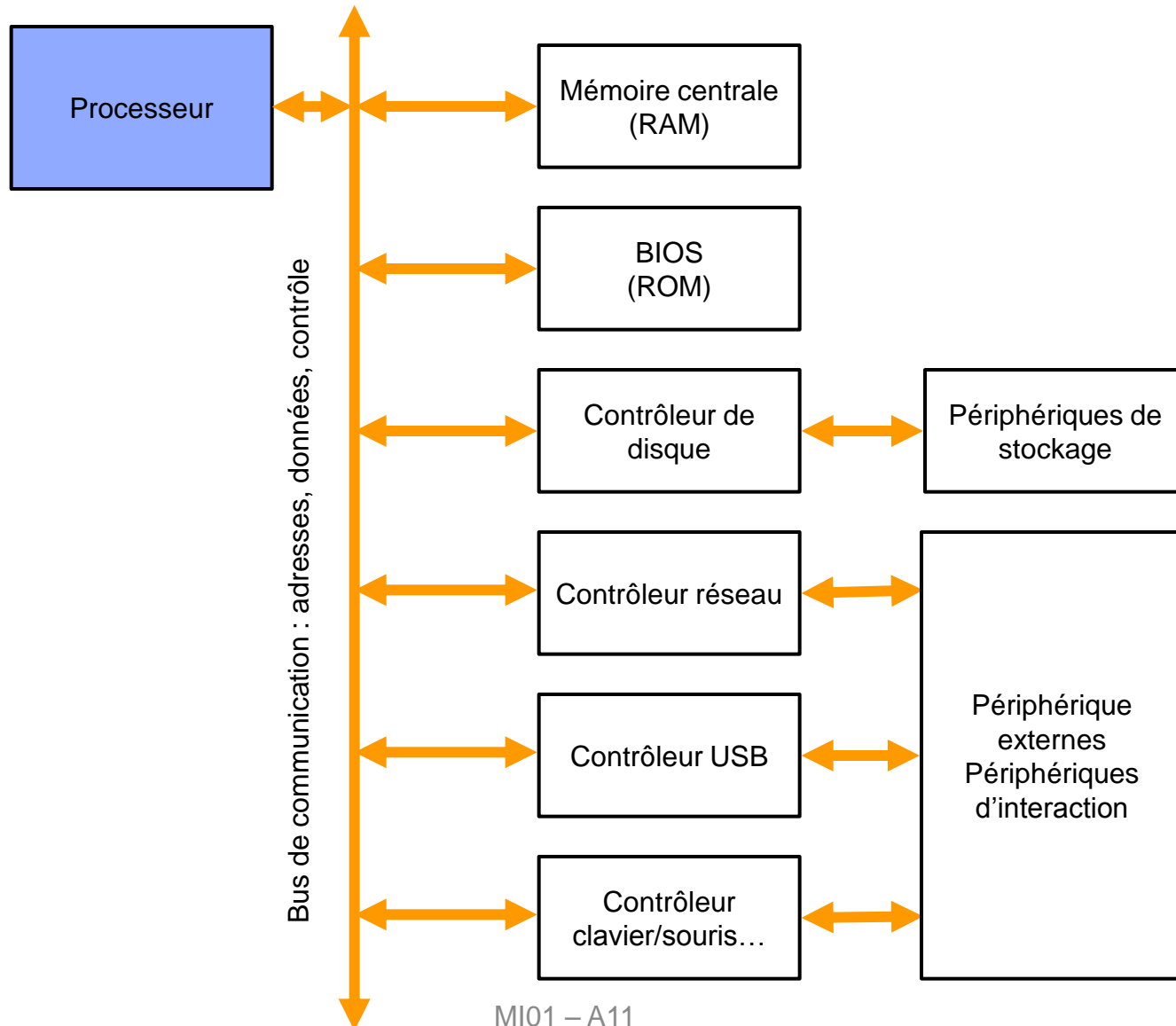


- Mémoire unique
  - Instructions
  - Données
- Structure plus simple
- Microprocesseurs

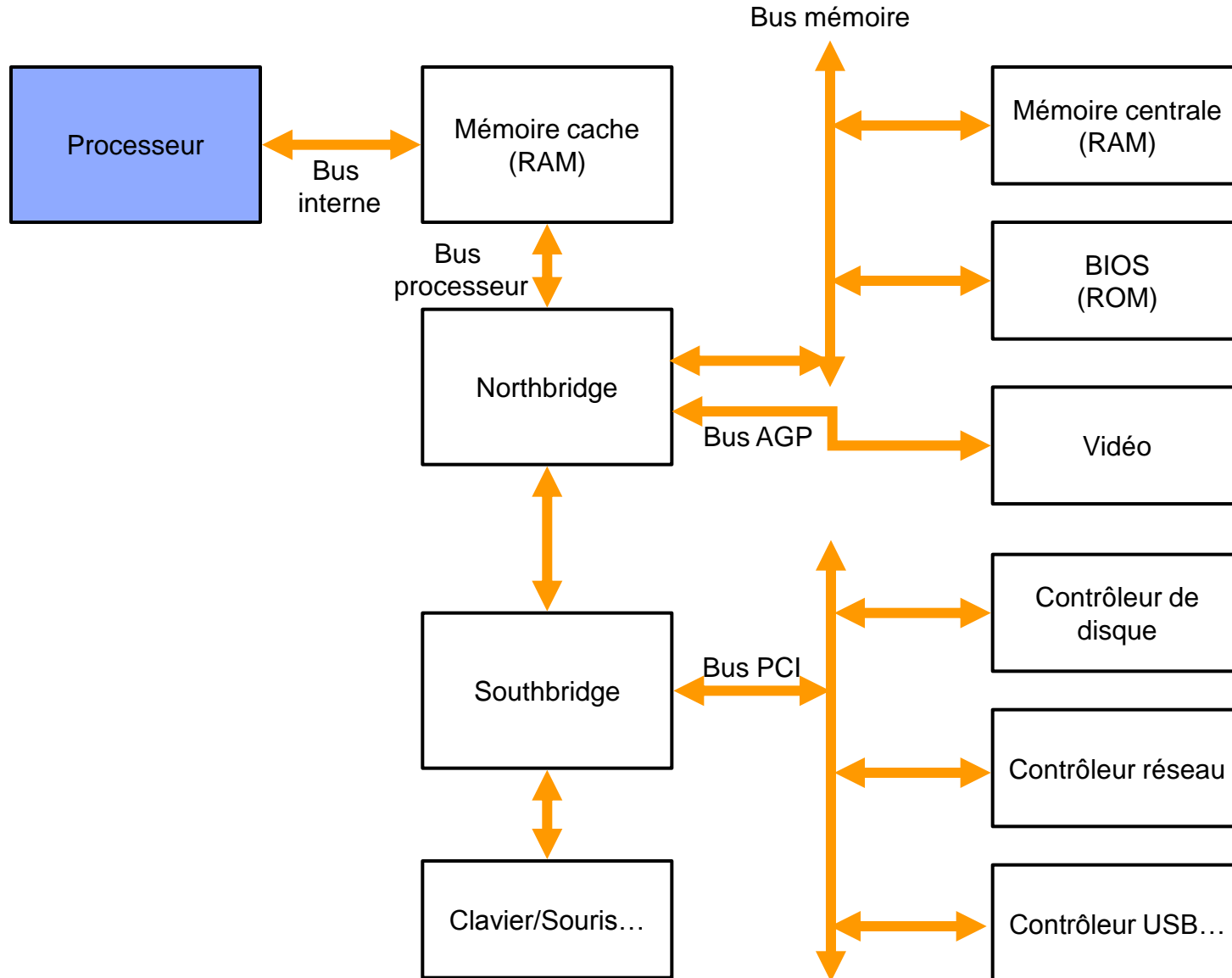
# 1.3. Système complet

- Regroupe trois composants essentiels assurant
  - La fonction **mémorisation**
  - La fonction **exécution**
  - La fonction **communication**
- Son organisation logique (du point de vue d'un programme) est rarement identique à son organisation physique (du point du vue du matériel)

# 1.3. Système complet (vue logique)



# 1.3. Système complet (vue physique)





# Sommaire

1. Systèmes à microprocesseur
- 2. La mémoire**
  - 2.1 Hiérarchie mémoire
  - 2.2 Organisation logique
3. Le processeur
4. Notre cas d'étude : la famille x86 d'Intel

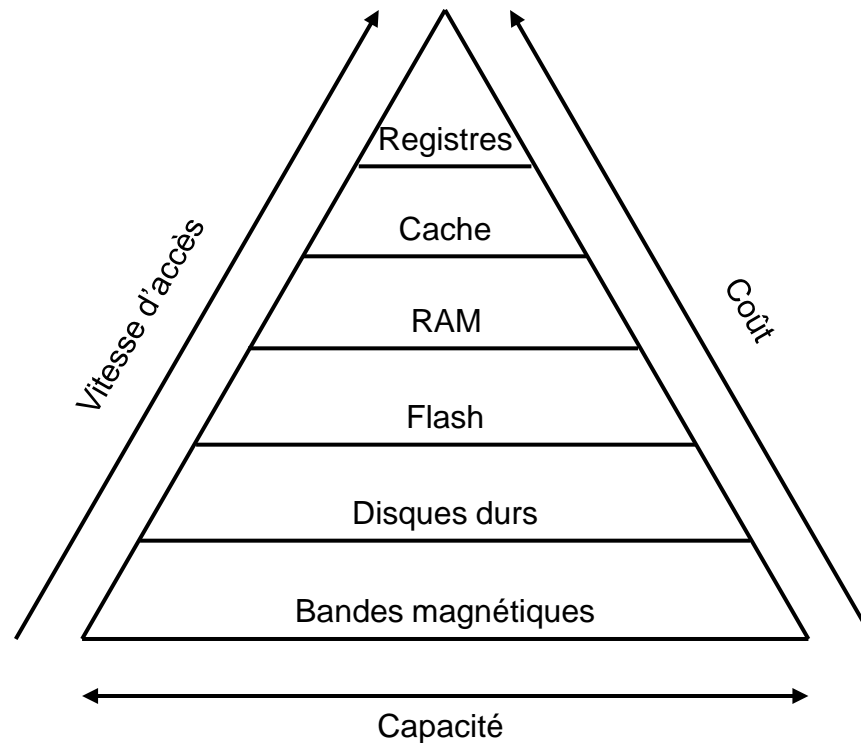
## 2.1. Hiérarchie mémoire

- Programmes (et données) sont stockés en mémoire
- Types de mémoire principale
  - RAM (Random Access Memory)
    - Stockage temporaire des programmes et données
    - Perd son contenu si plus d'alimentation
  - ROM (Read Only Memory)
    - Mémoire permanente, non inscriptible
    - Garde son contenu même sans alimentation
  - Cache (Mémoire rapide)
    - Permet d'accélérer le fonctionnement de l'ordinateur par copie locale des données/programmes souvent utilisés

## 2.1. Hiérarchie mémoire

- Autres mémoires
  - Disquettes
  - Disques durs
  - Disques flash (clé USB, Solid State Drives)
  - Bandes magnétiques...
- Permettent de stocker de grandes quantités de données
- Peu coûteuses
- Le processeur n'y a pas accès **directement**

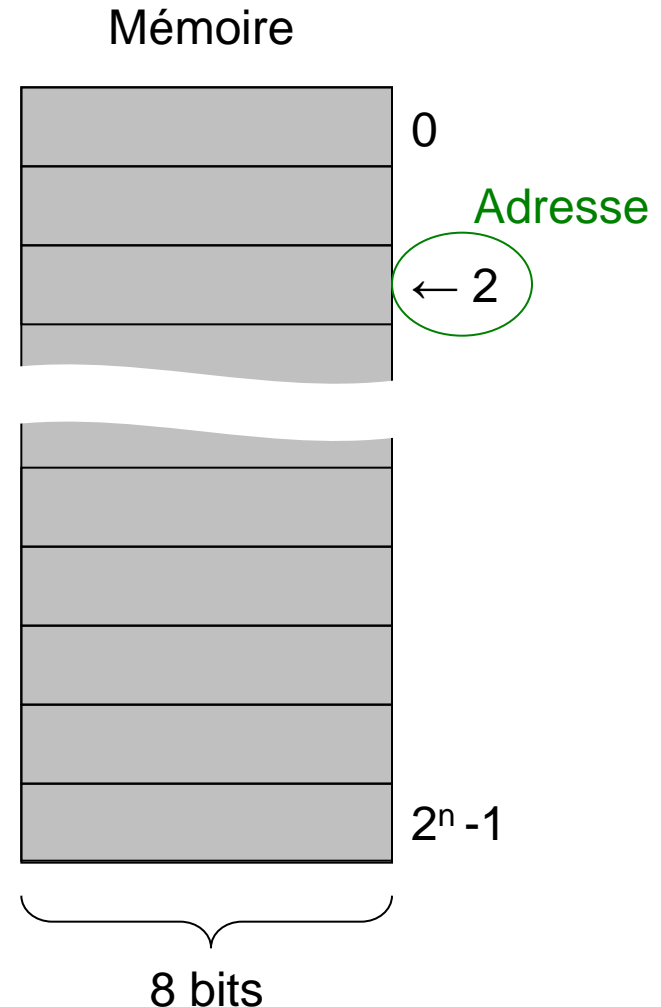
## 2.1. Hiérarchie mémoire



Plus la vitesse d'accès est grande, plus la capacité est petite et le coût élevé

## 2.2. Organisation de la mémoire

- Mémoire :
  - Séquence d'octets de 8 bits (en général)
  - Chaque octet est associé à une adresse **unique**
  - L'adresse d'un octet varie entre 0 et  $2^n - 1$   
=> Espace d'adressage
  - $n$  = nombre de bits d'adresse



## 2.2. Organisation de la mémoire

- La mémoire contient
  - Des octets (8 bits)
  - Des mots (16 bits)
  - Des doubles mots (32 bits)
  - Des quadruples mots (64 bits)
  - ... ou toute autre suite arbitraire d'octets contigus.
- Pas de frontières explicites : rien n'indique ce qui se trouve à une adresse donnée.

## 2.2. Organisation de la mémoire

00000000	56
00000001	52
00000002	41
00000003	49
00000004	4D
00000005	45
00000006	4E
00000007	54
00000008	44
00000009	45
0000000A	53
0000000B	47
0000000C	45
0000000D	45
0000000E	4B
0000000F	53

- Adresse de groupes de taille  $> 1$  = adresse du premier octet du groupe

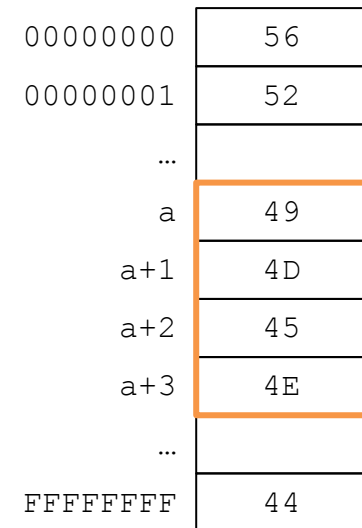
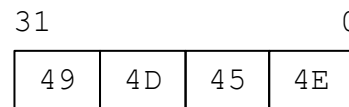
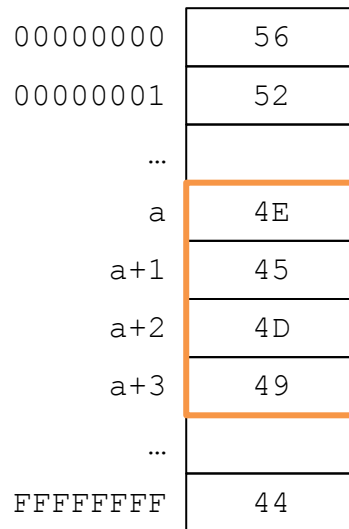
– Mot à l'adresse 7

– Double mot à l'adresse 7

– Double mot à l'adresse C

## 2.2. Organisation de la mémoire

- Stockage d'un nombre de plus de 8 bits :  
**exemple** mot long (32 bits) à l'adresse a



Ordre « Little Endian »

Ordre « Big Endian »



## 2.2. Organisation de la mémoire

- Organisation **logique** de la mémoire du point de vue d'un **programme** en cours d'exécution
  - Peut être différente de l'organisation logique vue du système de gestion de la mémoire du processeur
  - Peut être différente de l'organisation physique de la mémoire vue du matériel
- Seule vraiment utile pour la programmation d'applications.

On parle d'adresses **virtuelles**

# Sommaire

1. Systèmes à microprocesseur
2. La mémoire
- 3. Le processeur**
  - 3.1. Machine à accumulateur
  - 3.2. Machine à registres
  - 3.3. Machine à pile
  - 3.4. Classes de machines
4. Notre cas d'étude : la famille x86 d'Intel

# 3. Types de machines

Le rôle d'un processeur est de combiner des données en exécutant une instruction pour produire un résultat.

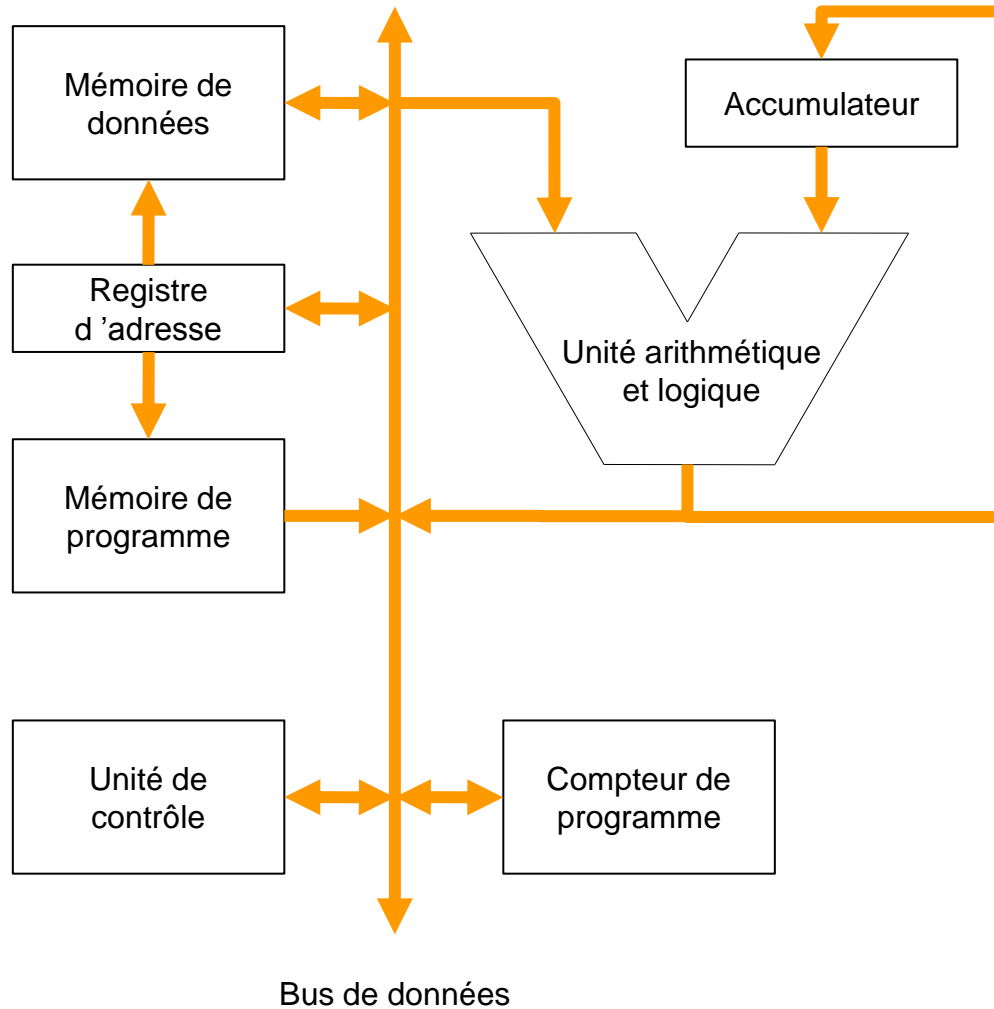
Trois structures dominantes :

- Machines à accumulateur
- Machines à registres
- Machines à pile

# 3.1. Machines à accumulateur

- « 1-operand machine »
  - L'accumulateur est un registre interne qui sert à la fois :
    - D'opérande source dans les instructions
    - De destination pour les résultats des instruction
- ⇒ l'accumulateur est impliqué dans la plupart des instructions
- ⇒ l'accumulateur permet l'enchaînement d'opérations successives sans rechargement depuis la mémoire
- Exemple : Microcontrôleurs PIC

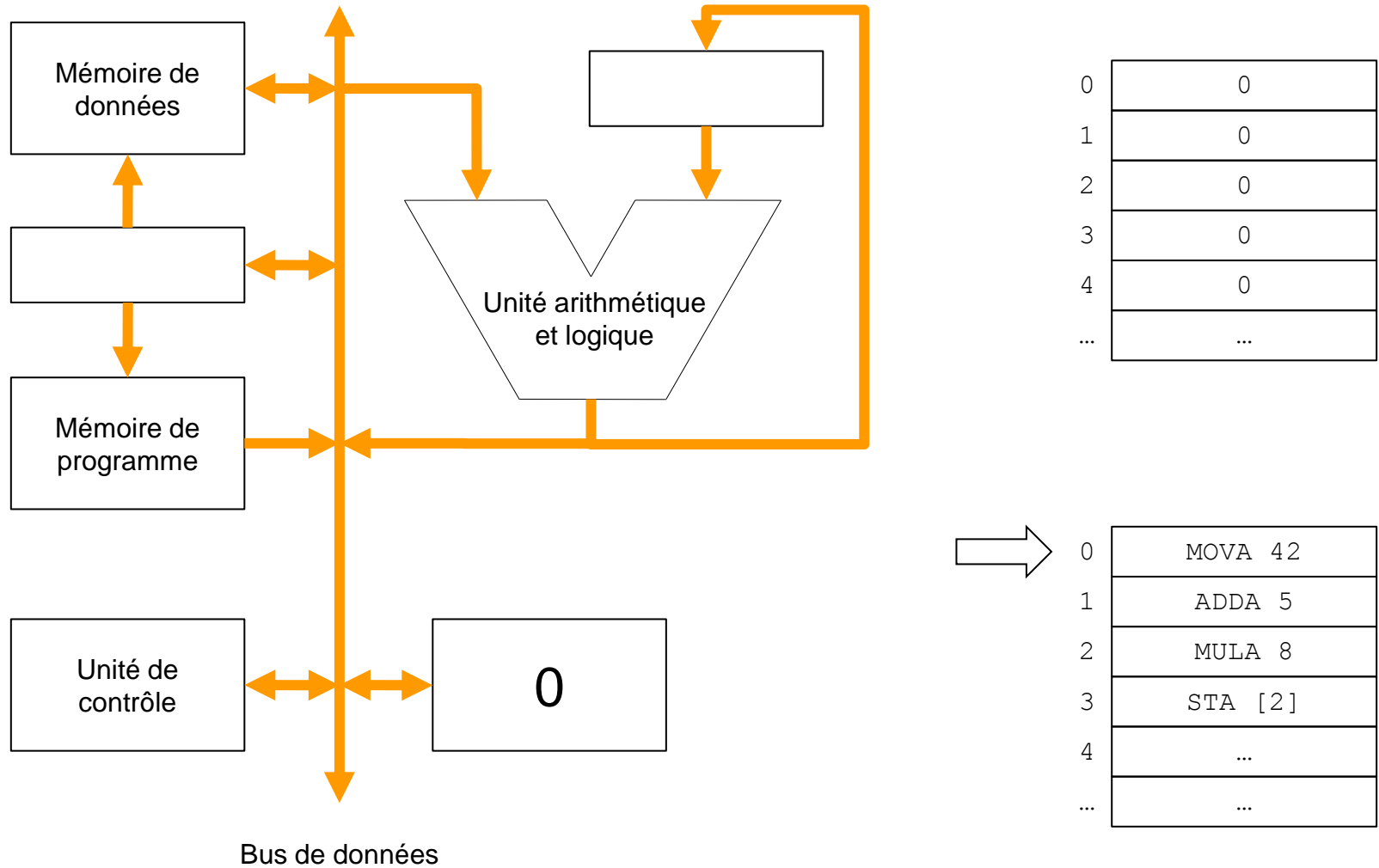
# 3.1. Machines à accumulateur



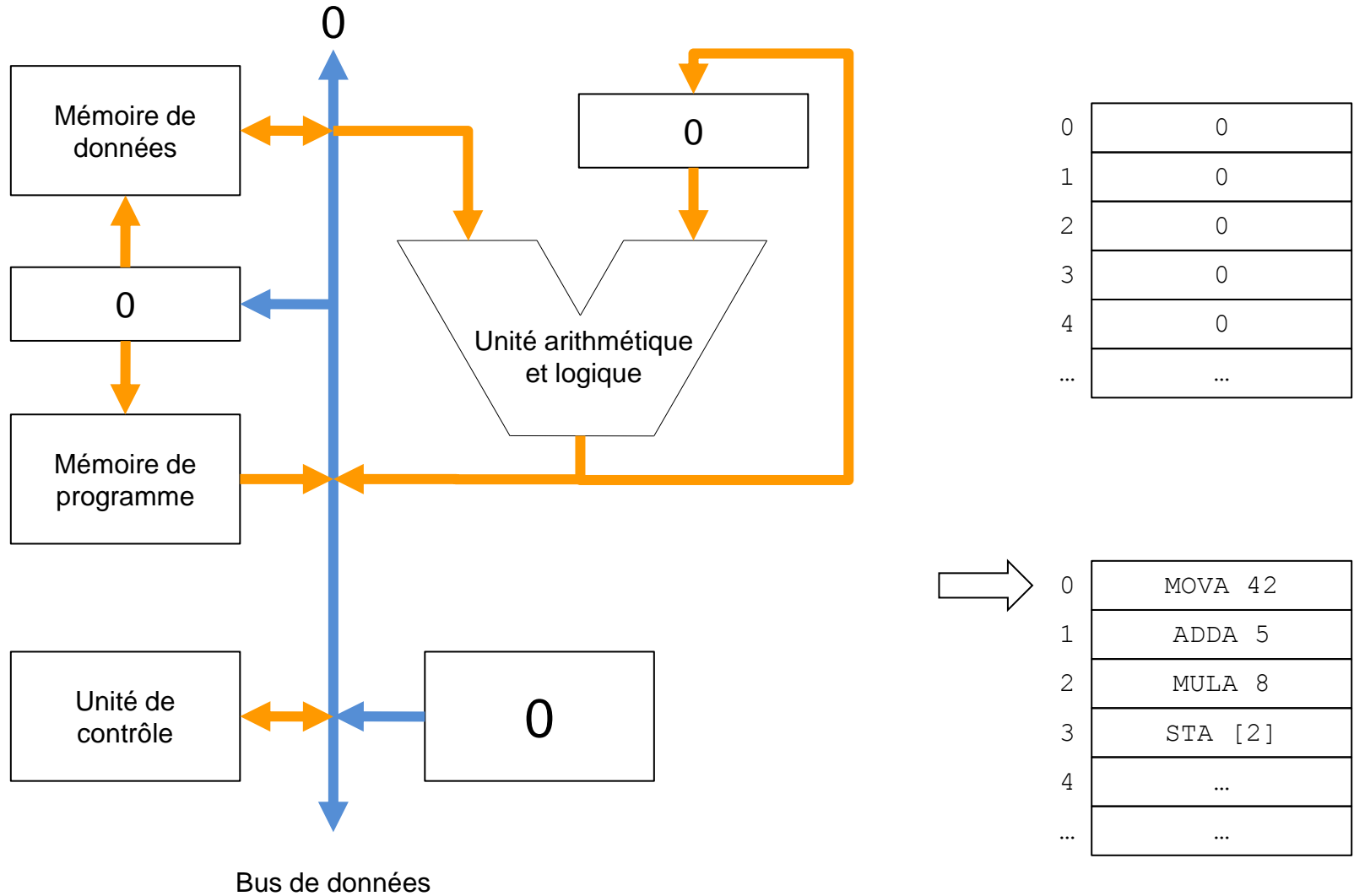
## Jeu d'instructions

- **LDA constante**
  - Charger constante dans A
- **LDA [adresse]**
  - Charger mémoire dans A
- **STA [adresse]**
  - Stocker A dans mémoire
- **ADDA constante**
  - Ajouter constante à A
- **ADDA [adresse]**
  - Ajouter mémoire à A
- **MULA constante**
  - Multiplier A par constante
- **MULA [adresse]**
  - Multiplier A par mémoire
- ...

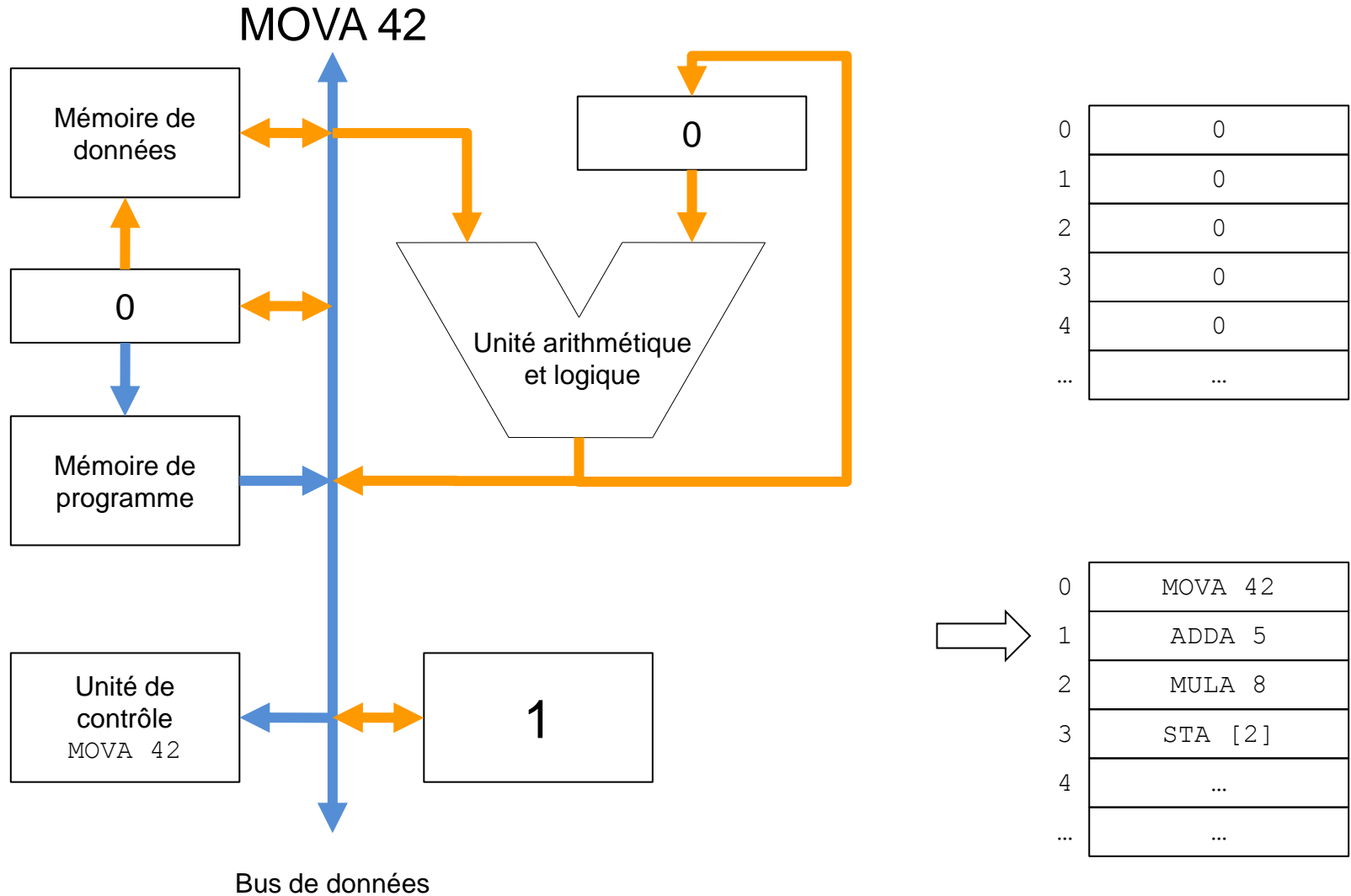
# 3.1. Machines à accumulateur



# 3.1. Machines à accumulateur

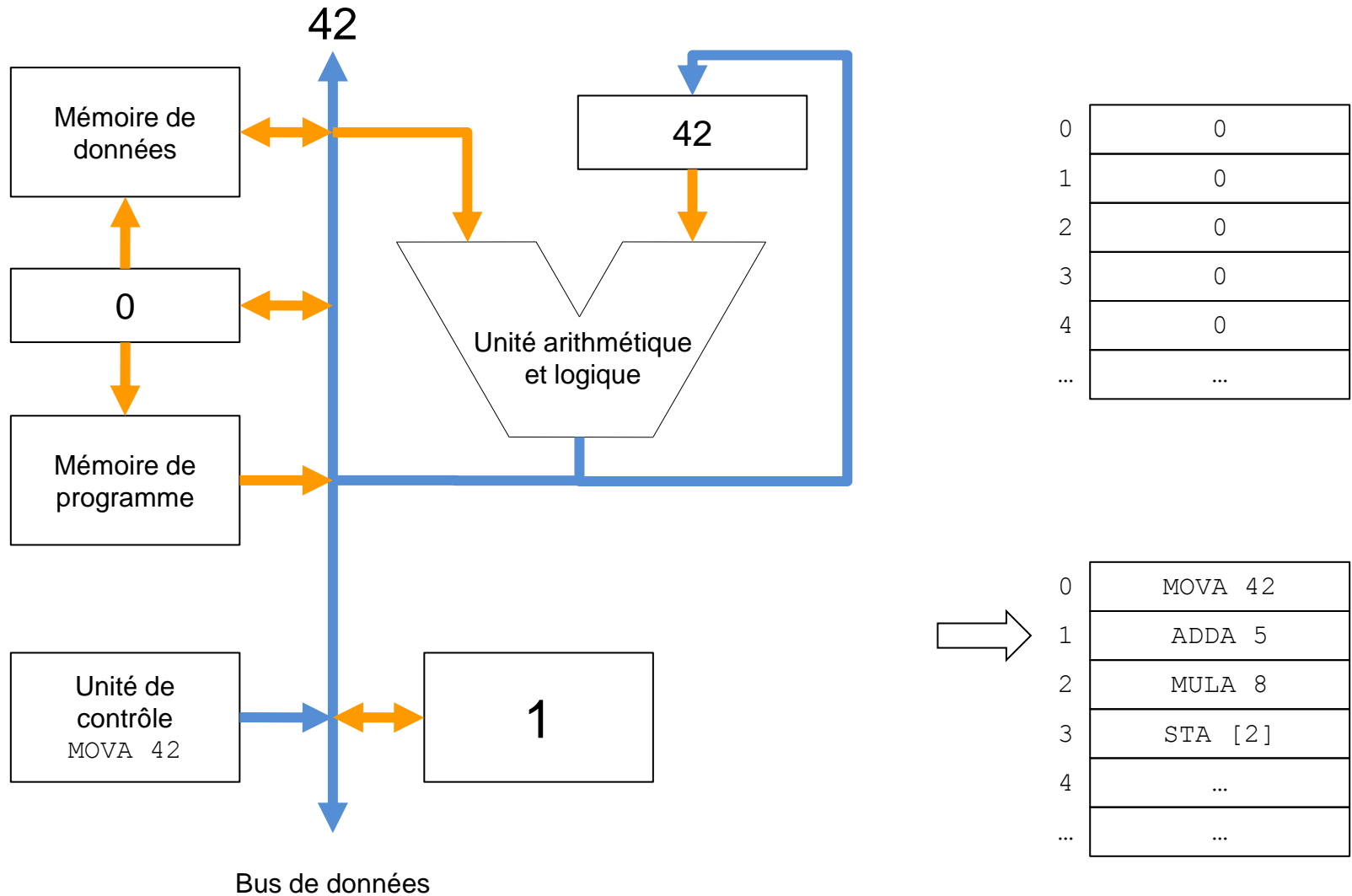


# 3.1. Machines à accumulateur

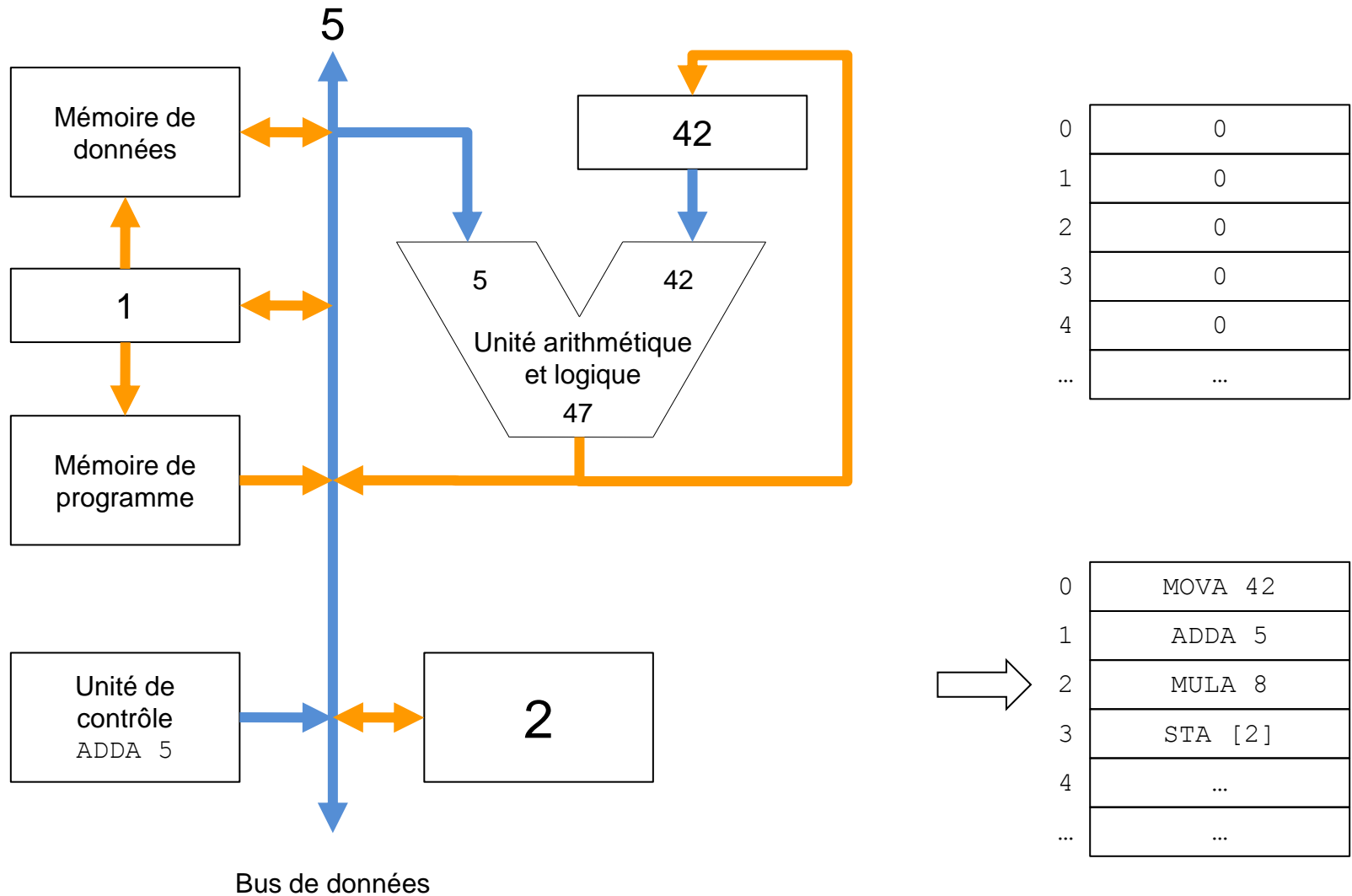




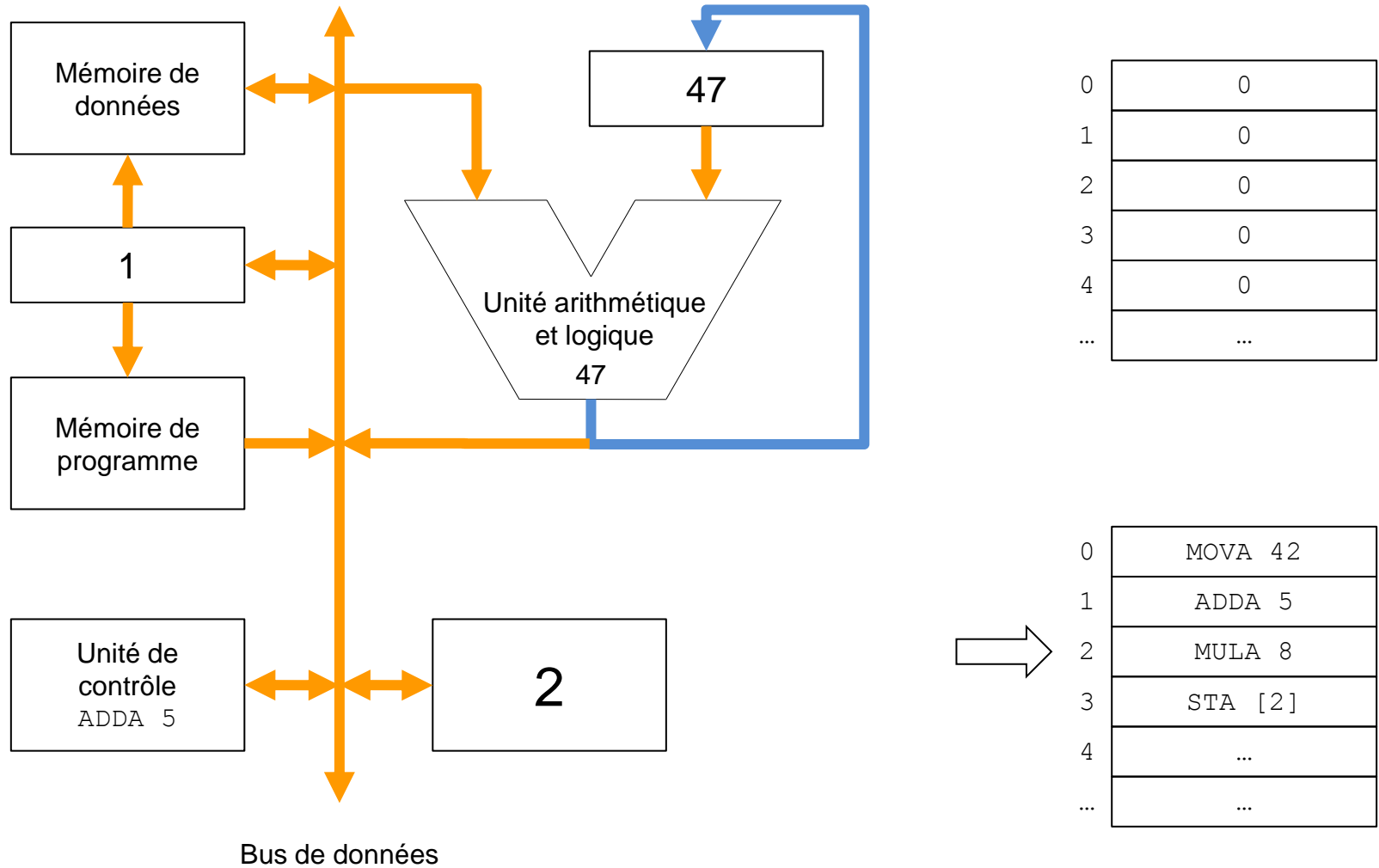
# 3.1. Machines à accumulateur



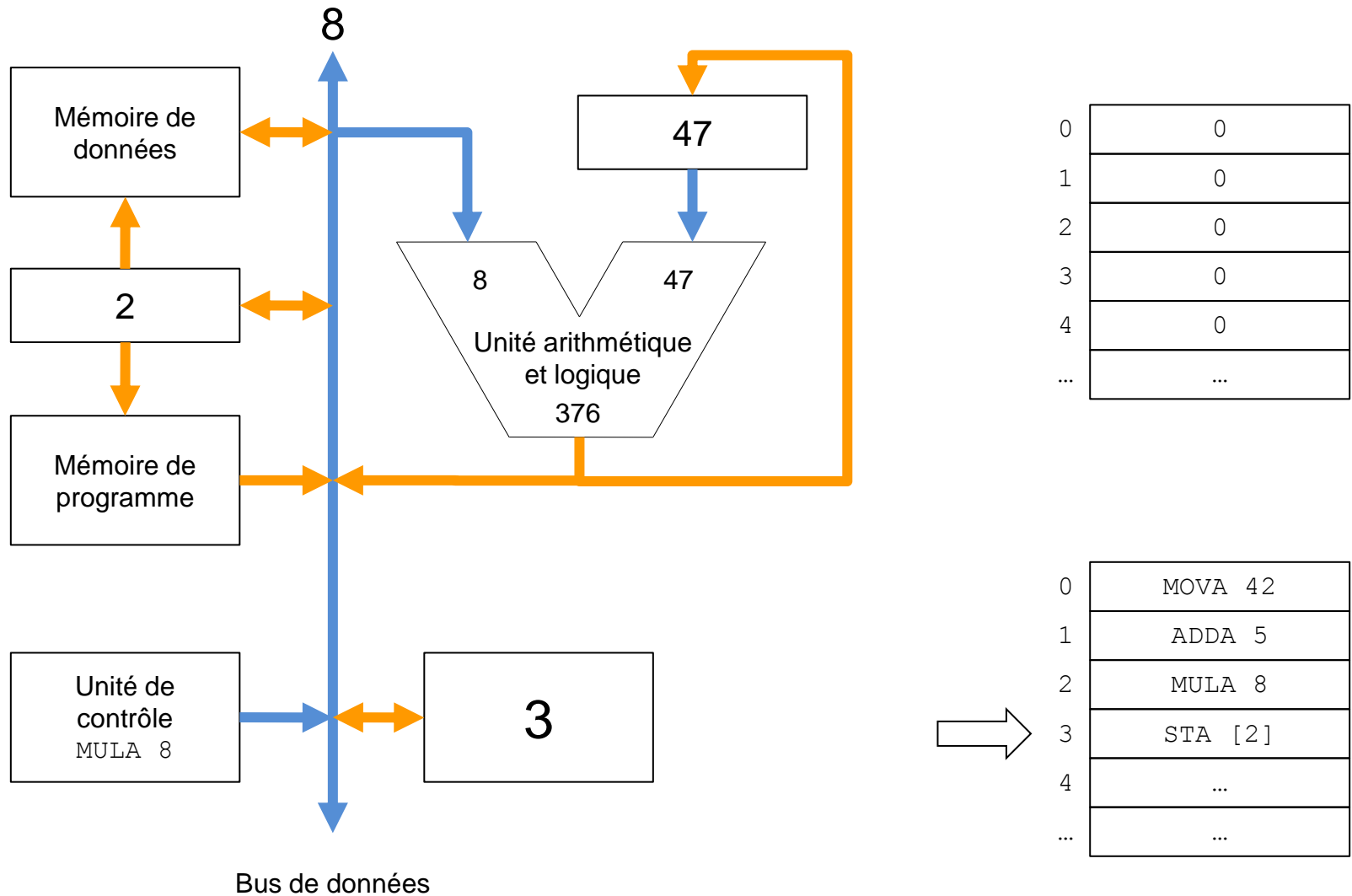
# 3.1. Machines à accumulateur



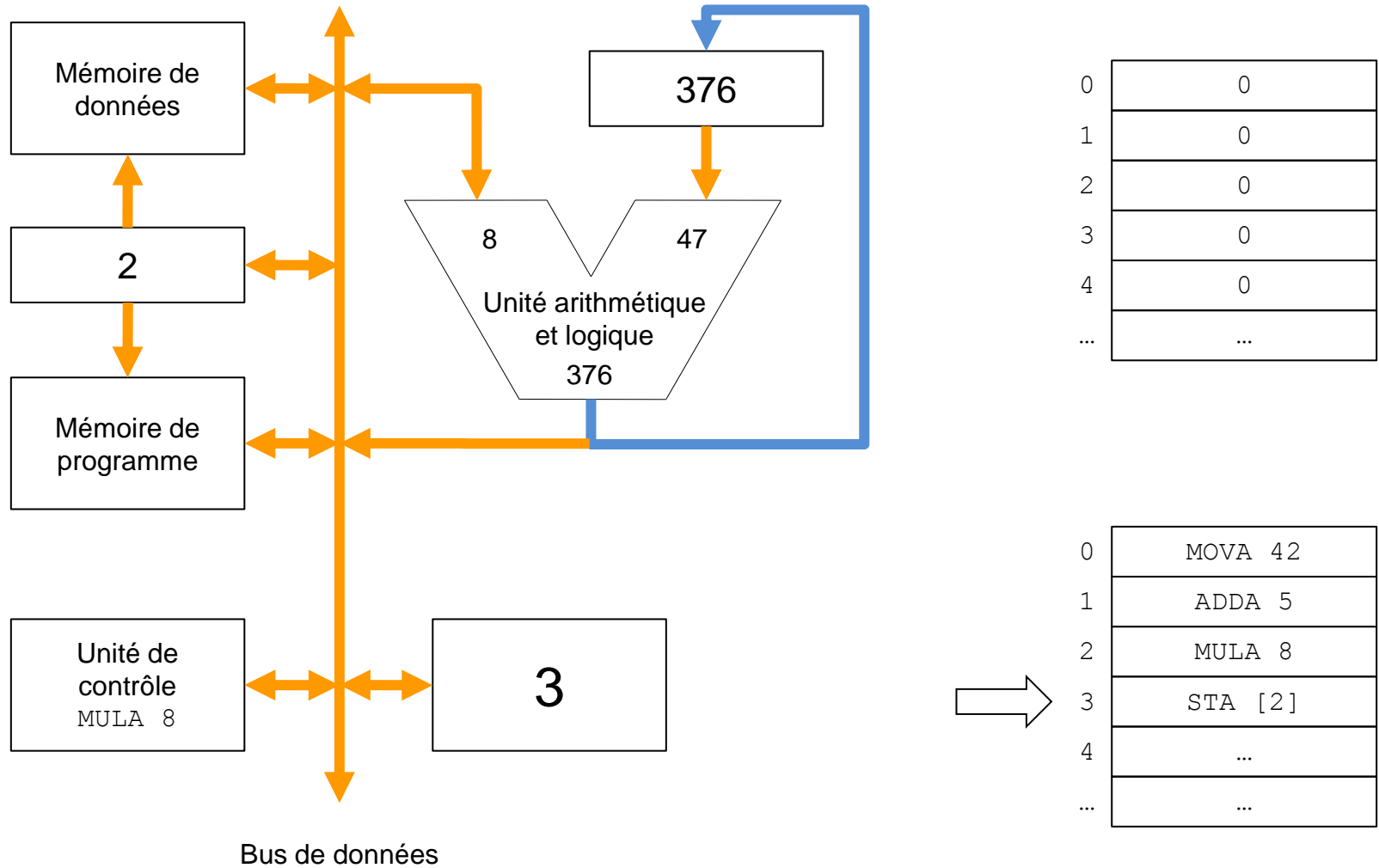
# 3.1. Machines à accumulateur



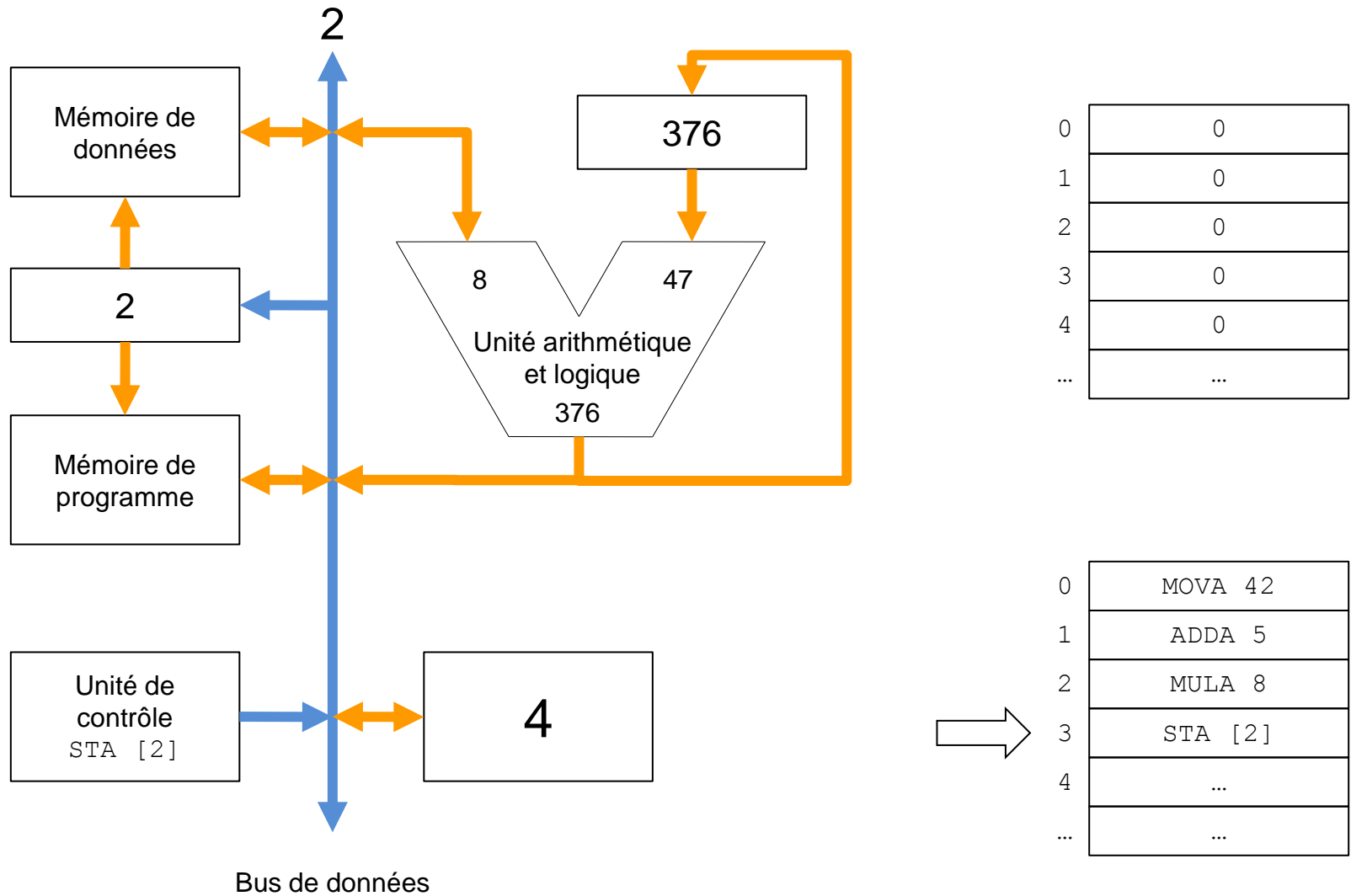
# 3.1. Machines à accumulateur



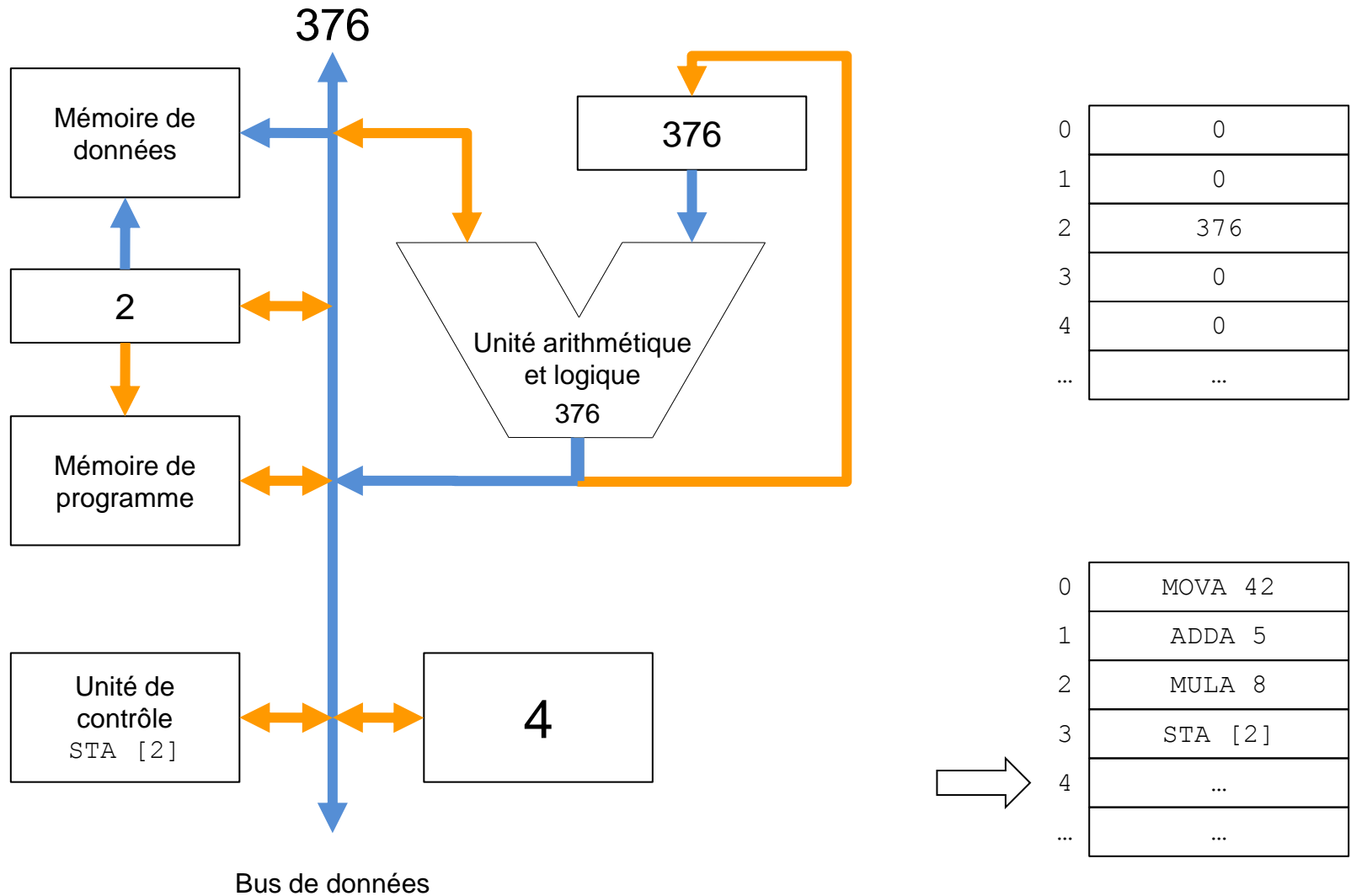
# 3.1. Machines à accumulateur



# 3.1. Machines à accumulateur



# 3.1. Machines à accumulateur

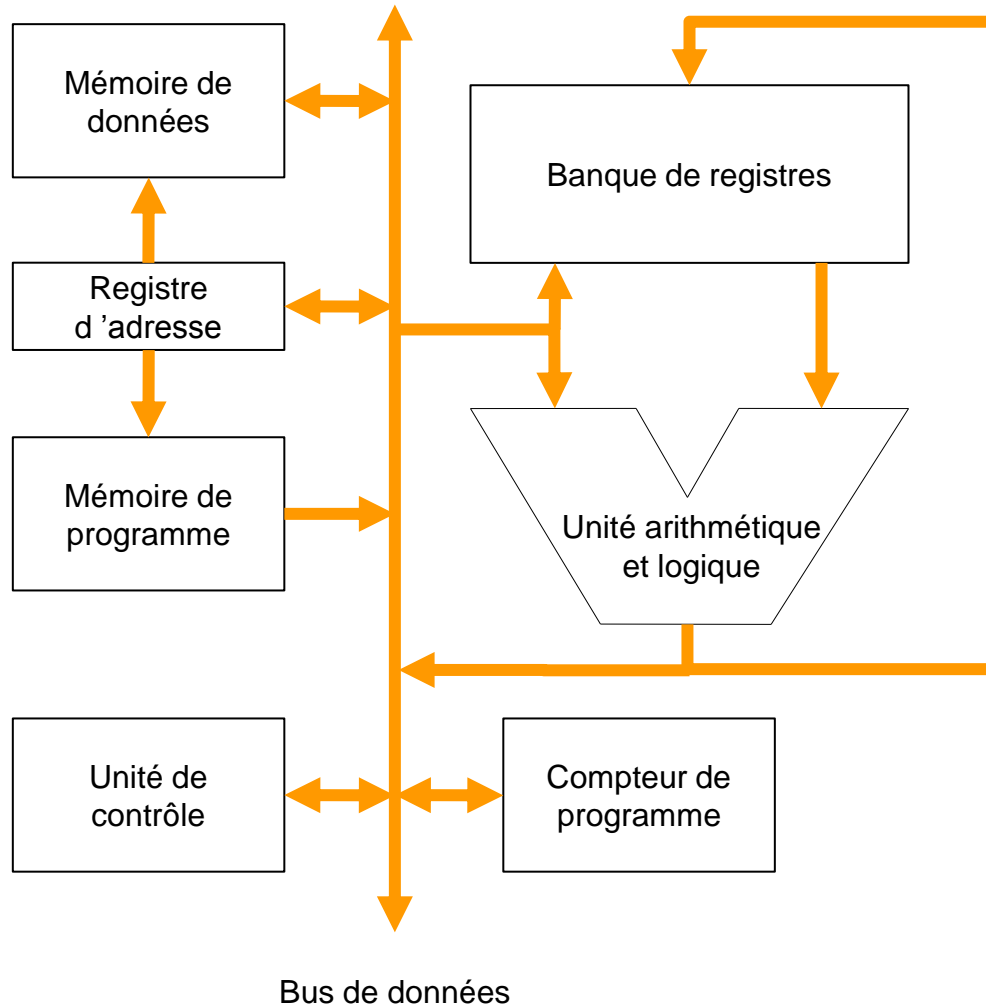


## 3.2. Machines à registres

- « 2-operand machine », « 3-operand machine »
- Machines à accumulateur limitées
  - ⇒ solution : plus d'accumulateurs!
  - ⇒ Registres multiples, qui peuvent être :
    - Opérandes sources
    - Opérandes destination
- Plus souple mais plus complexe
  - Exemples :
    - IA 32 (dans une certaine limite)
    - ARM
    - PowerPC...



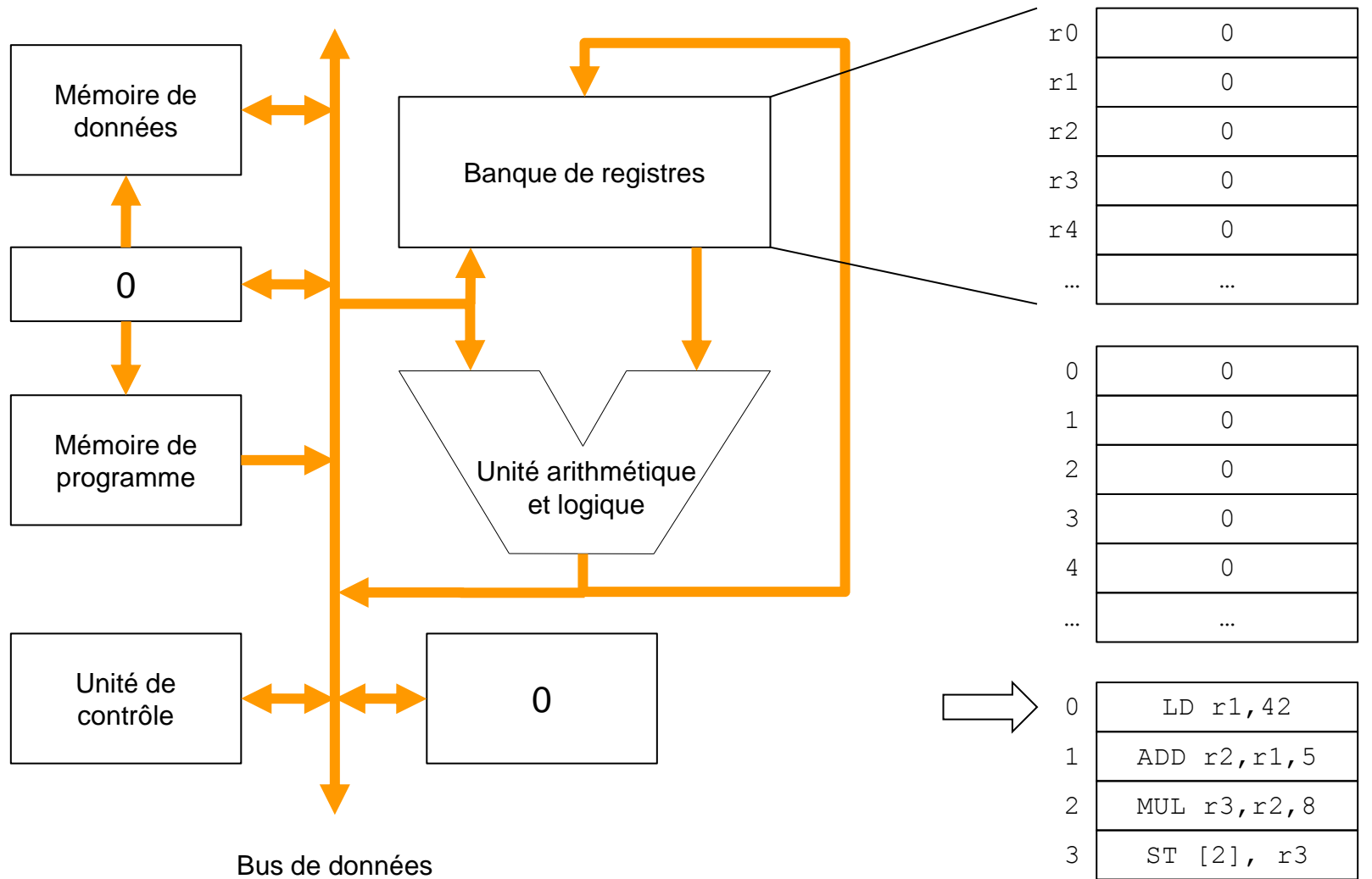
## 3.2. Machines à registres



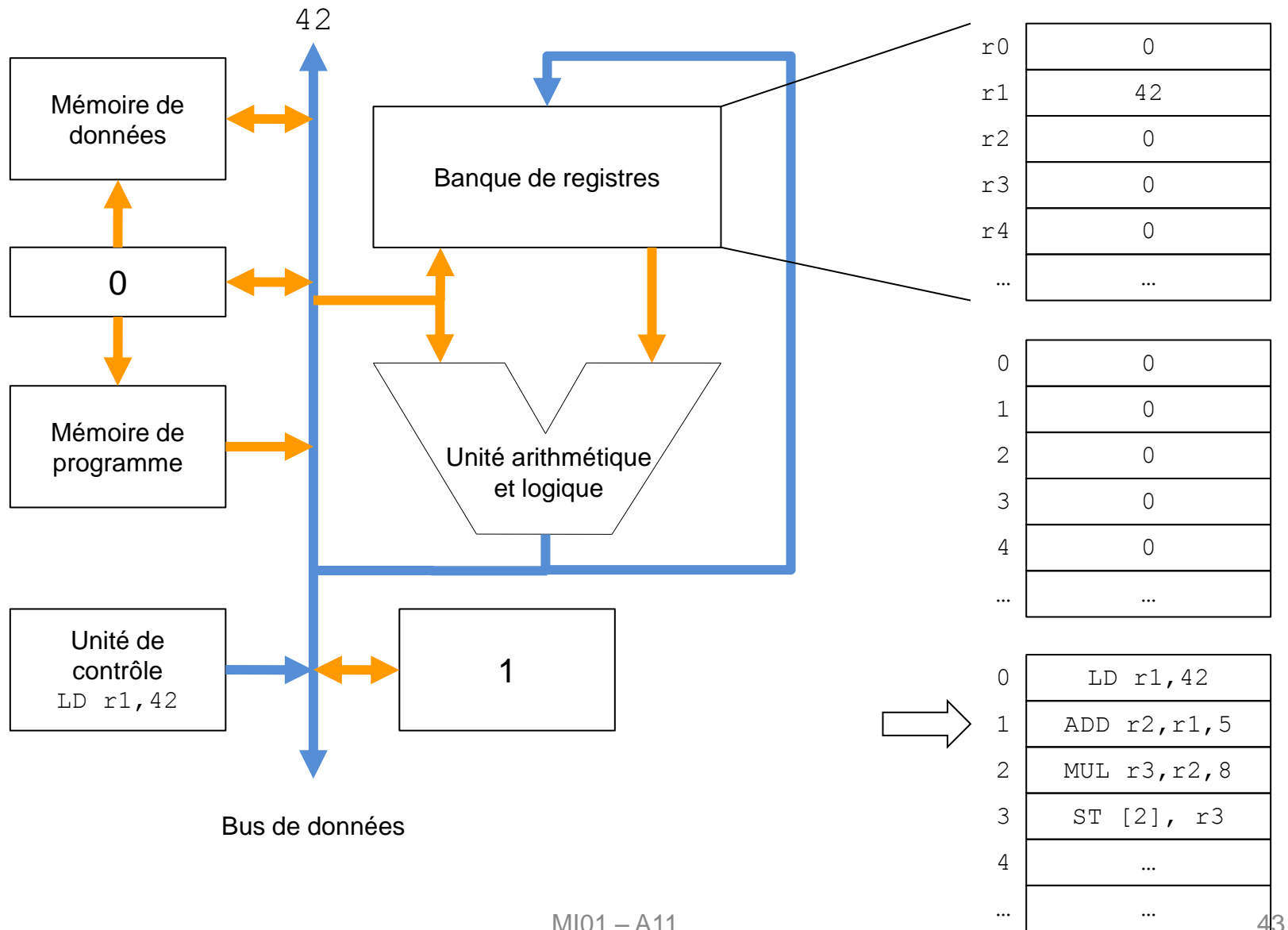
### Jeu d'instructions

- LD  $r$ , constante
  - $r \leftarrow \text{constante}$
- LD  $r$ , [adresse]
  - $r \leftarrow \text{mémoire}(\text{adresse})$
- ST [adresse],  $r$ 
  - $\text{mémoire}(\text{adresse}) \leftarrow r$
- ADD  $rd$ ,  $rs$ , constante
  - $rd \leftarrow rs + \text{constante}$
- ADD  $rd$ ,  $rs$ , [adresse]
  - $rd \leftarrow rs + \text{mémoire}(\text{adresse})$
- ADD  $rd$ ,  $rs1$ ,  $rs2$ 
  - $rd \leftarrow rs1 + rs2$
- MUL  $rd$ ,  $rs$ , constante
  - $rd \leftarrow rs \times \text{constante}$
- MUL  $rd$ ,  $rs$ , [adresse]
  - $rd \leftarrow rs \times \text{mémoire}(\text{adresse})$
- MUL  $rd$ ,  $rs1$ ,  $rs2$ 
  - $rd \leftarrow rs1 \times rs2$

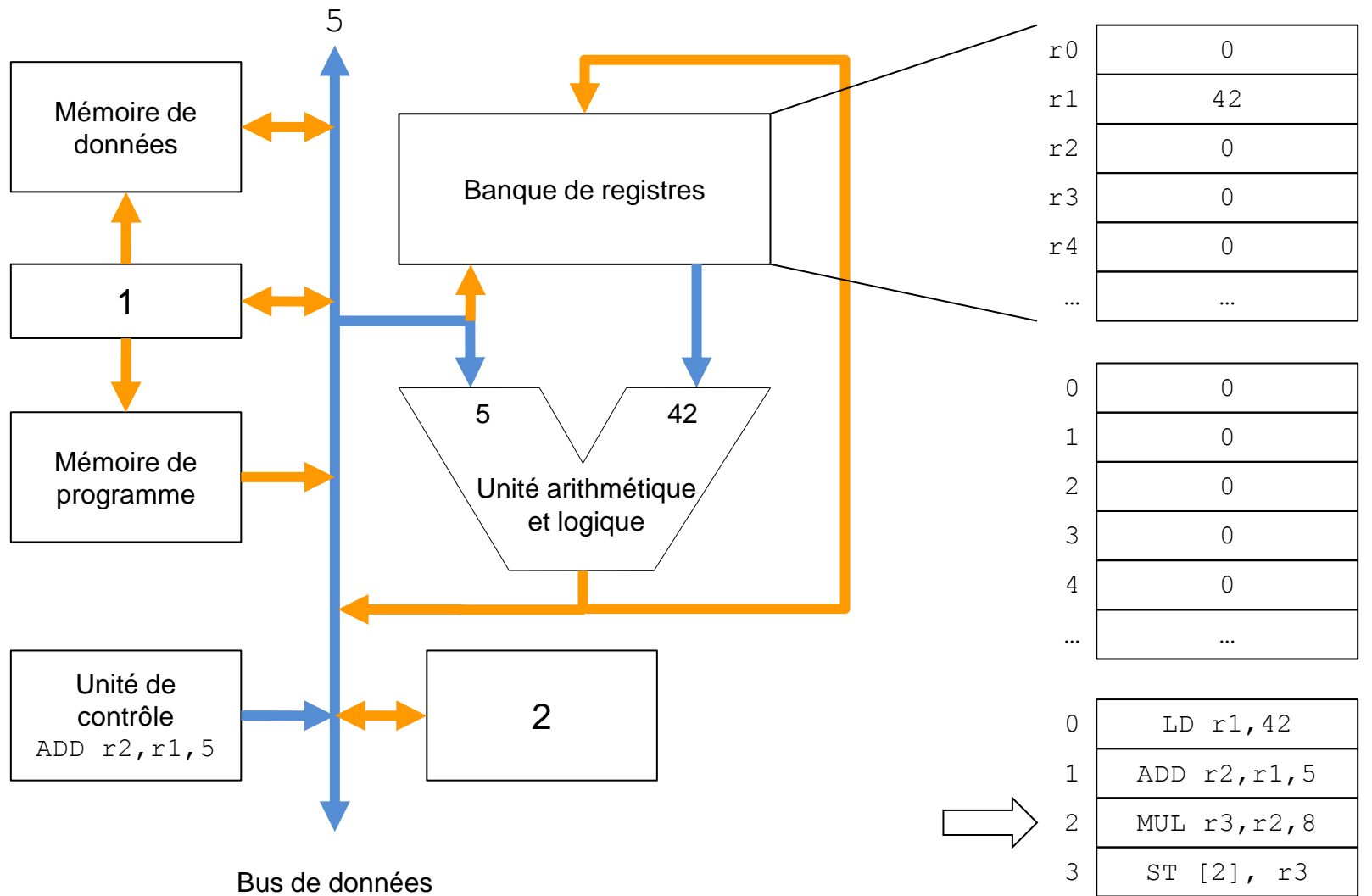
## 3.2. Machines à registres



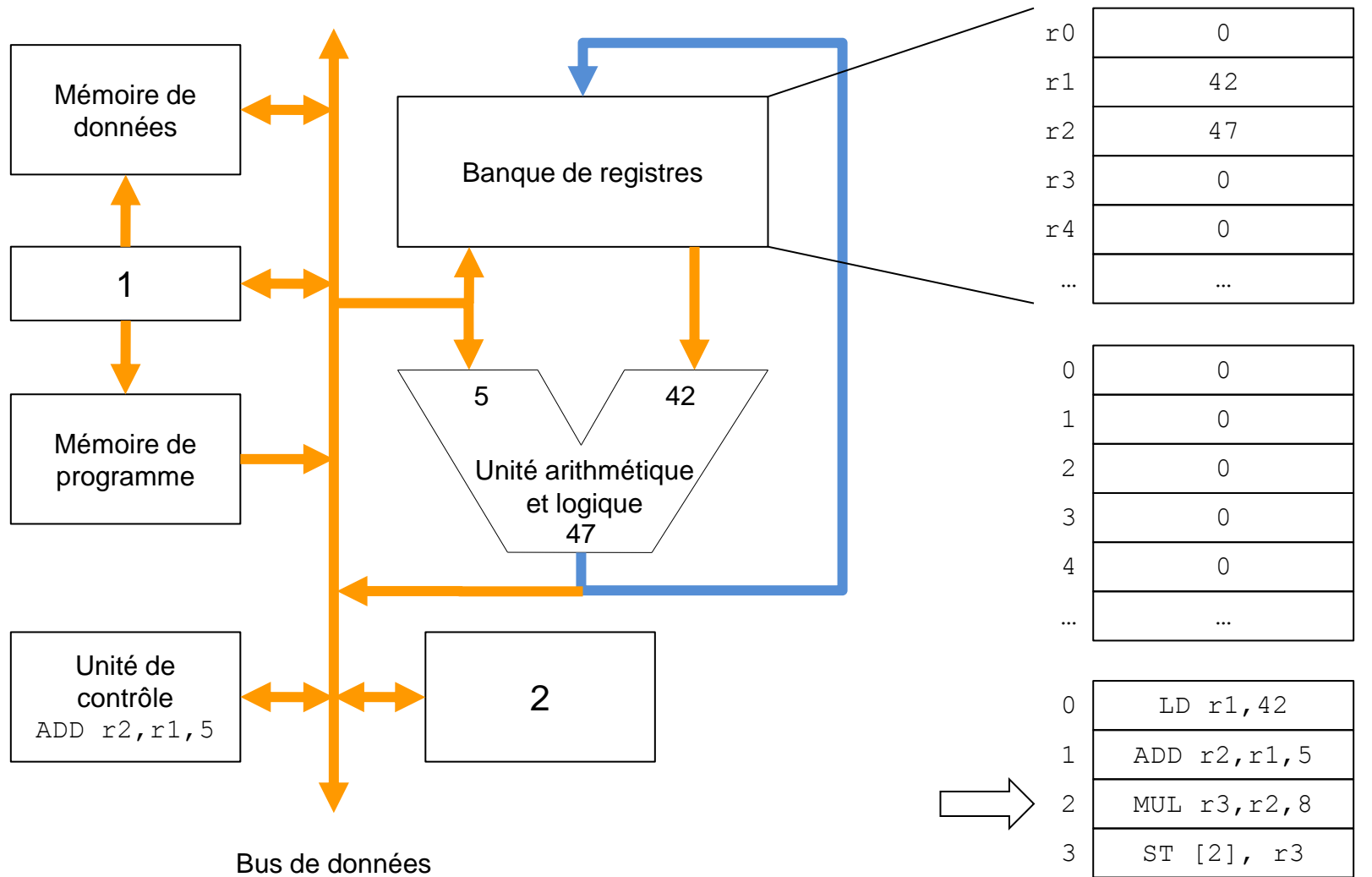
## 3.2. Machines à registres



## 3.2. Machines à registres



## 3.2. Machines à registres

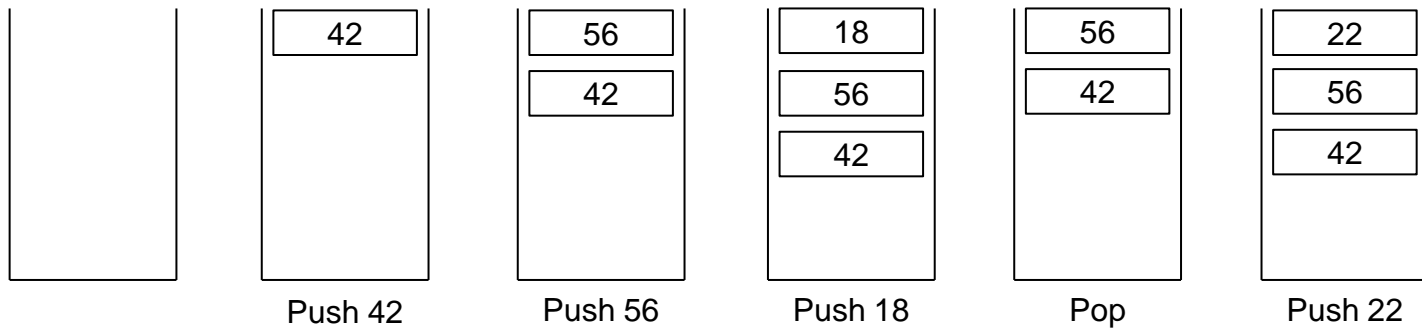


## 3.3. Machines à pile

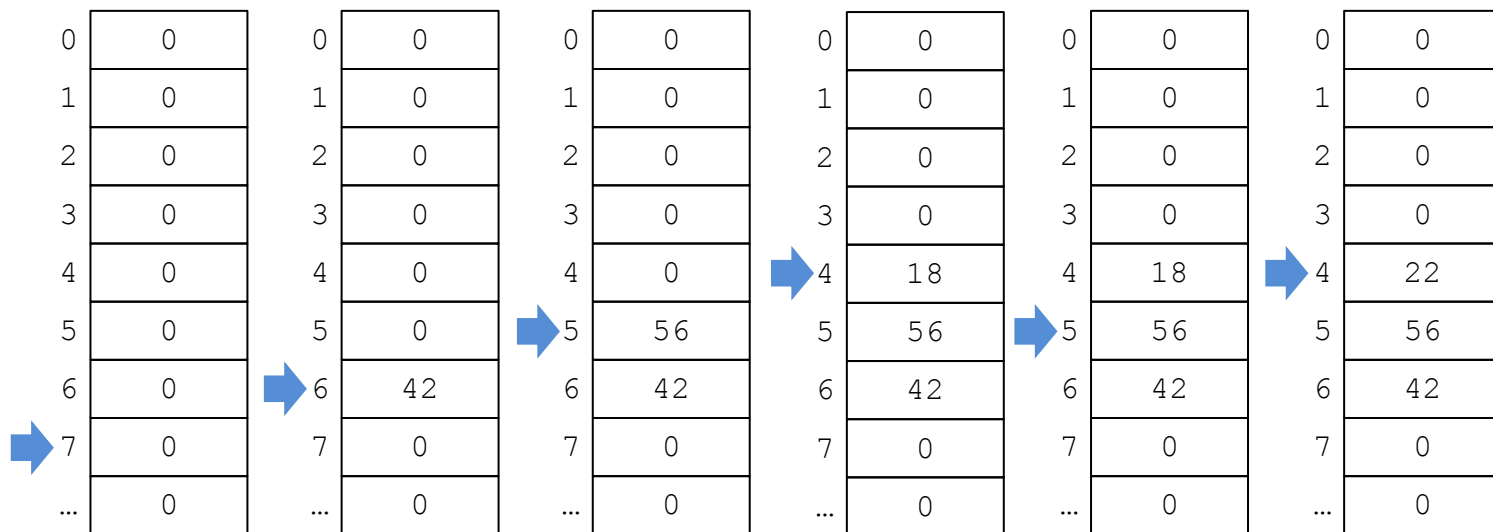
- « Zero-operand machine »
- Pas de registres, toutes les opérations sont faites sur une pile
- Une pile (LIFO, Last In first Out) est une structure de données qui supporte deux opérations :
  - Empiler une donnée en haut de la pile (« Push »),
  - Dépiler une donnée du haut de la pile (« Pop »)

# 3.3.1. Fonctionnement d'une pile

- Abstrait



- Concret (dans une mémoire) : pointeur de pile

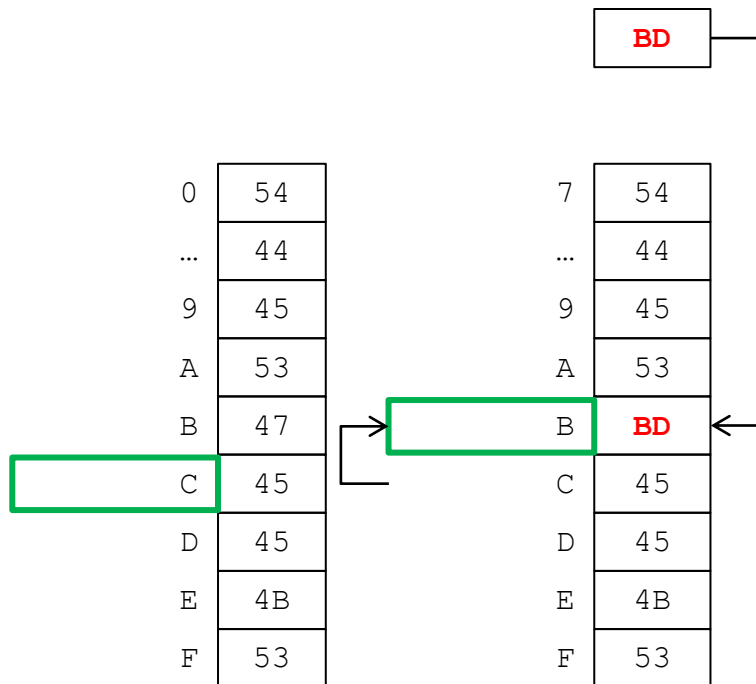


# 3.3.1. Fonctionnement d'une pile

## Ajout d'une donnée en haut de pile : empilage

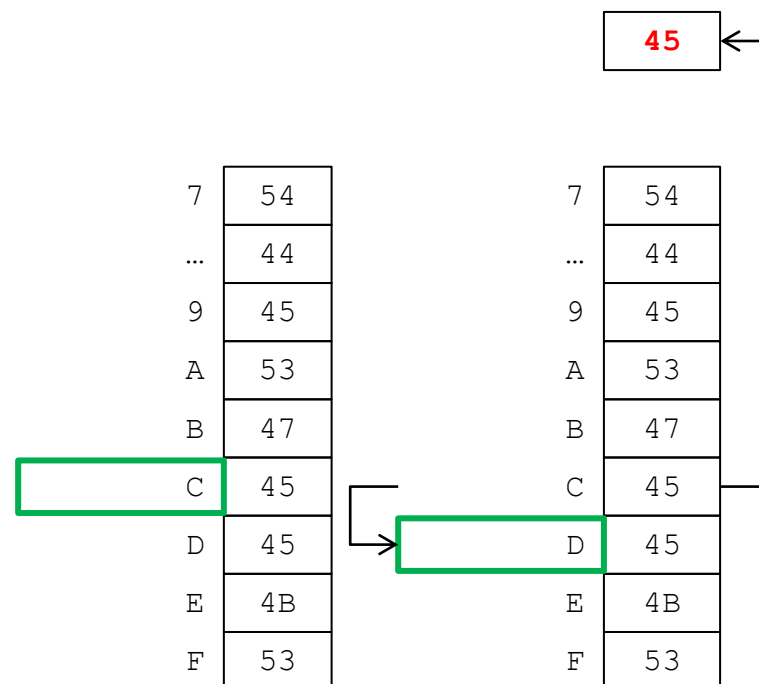
Le pointeur de pile référence toujours le dernier  
élément empilé => **haut de la pile** (Top of Stack)

1. **Création d'une nouvelle place en décrémentant le pointeur de la taille de la donnée**
2. Ecriture de la donnée à l'adresse correspondante



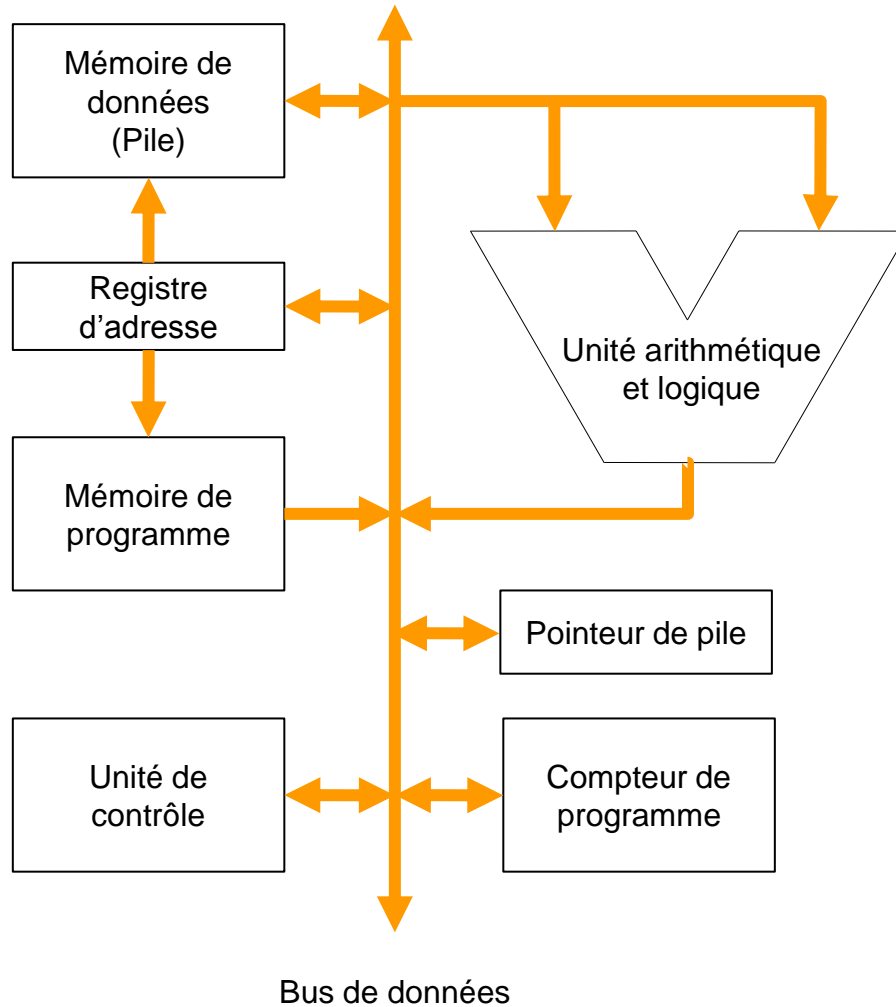
## Retrait d'une donnée en haut de pile : dépile

1. Lecture de la donnée à l'adresse correspondante au pointeur
2. **Suppression d'une place en incrémentant le pointeur de la taille de la donnée**





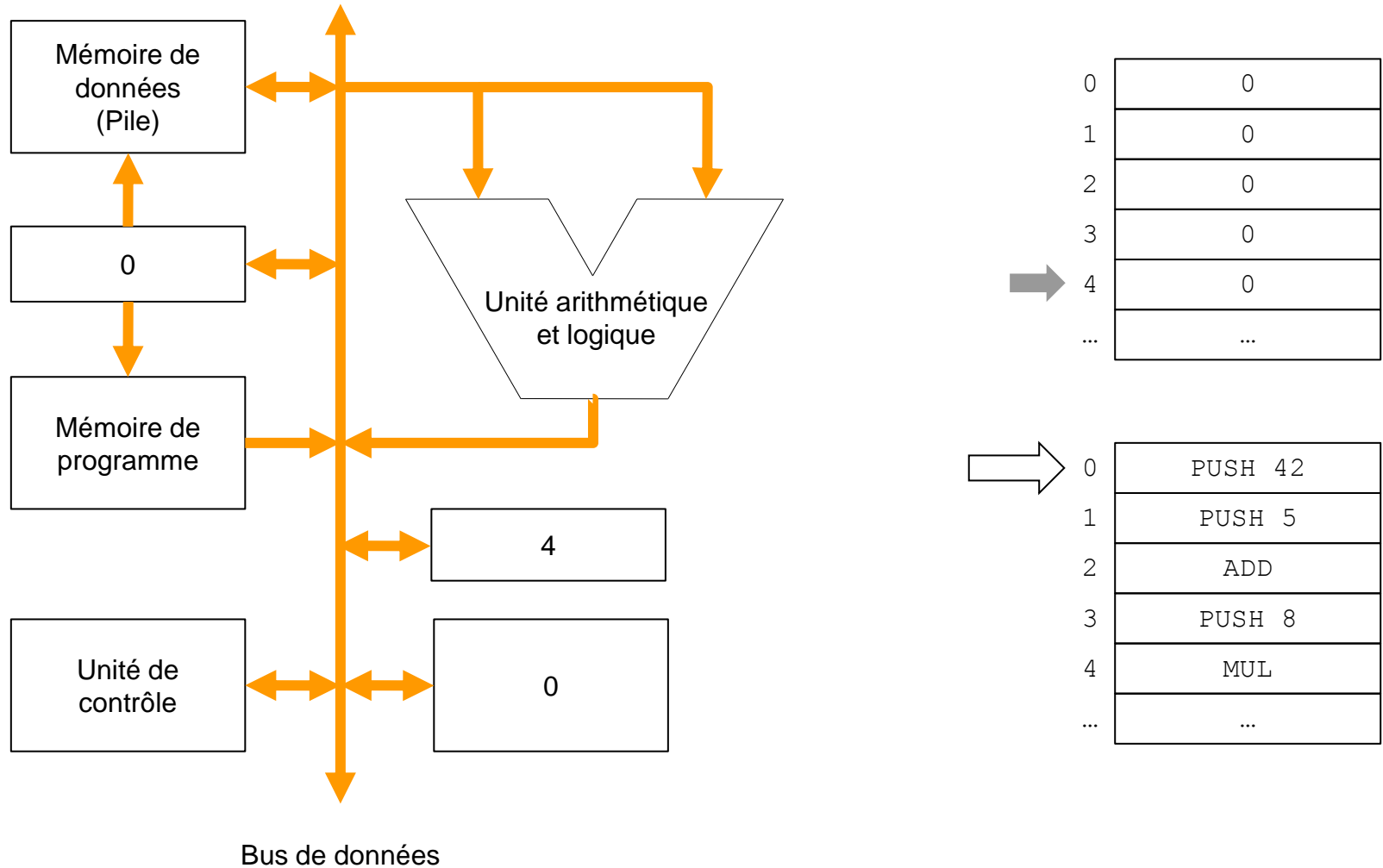
## 3.3.2. Machines à pile



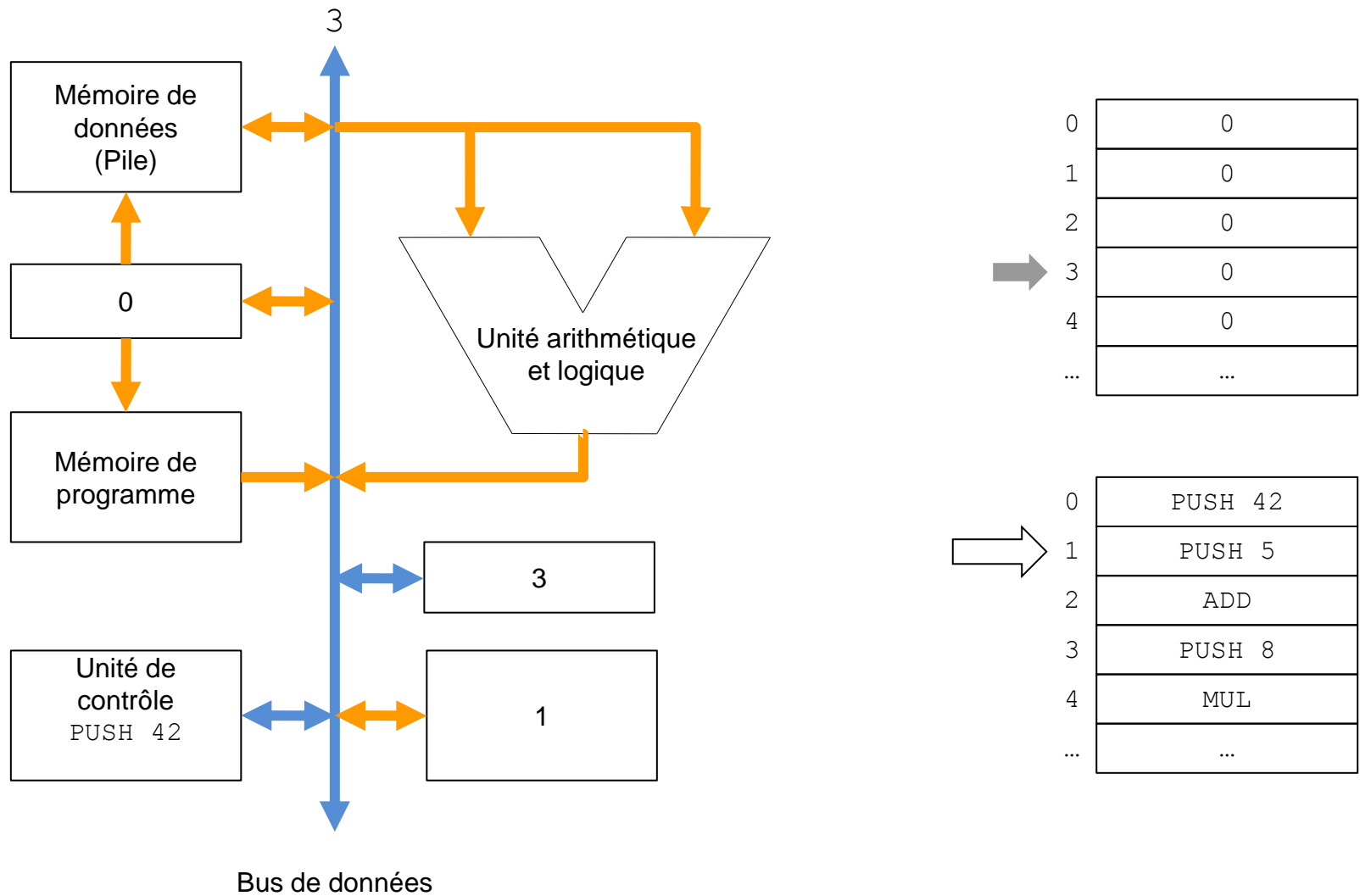
### Jeu d'instructions

- **PUSH constante**
  - Ajoute constante en haut de pile
- **ADD**
  - Dépile deux données, empile la somme
- **MUL**
  - Dépile deux données, empile le produit

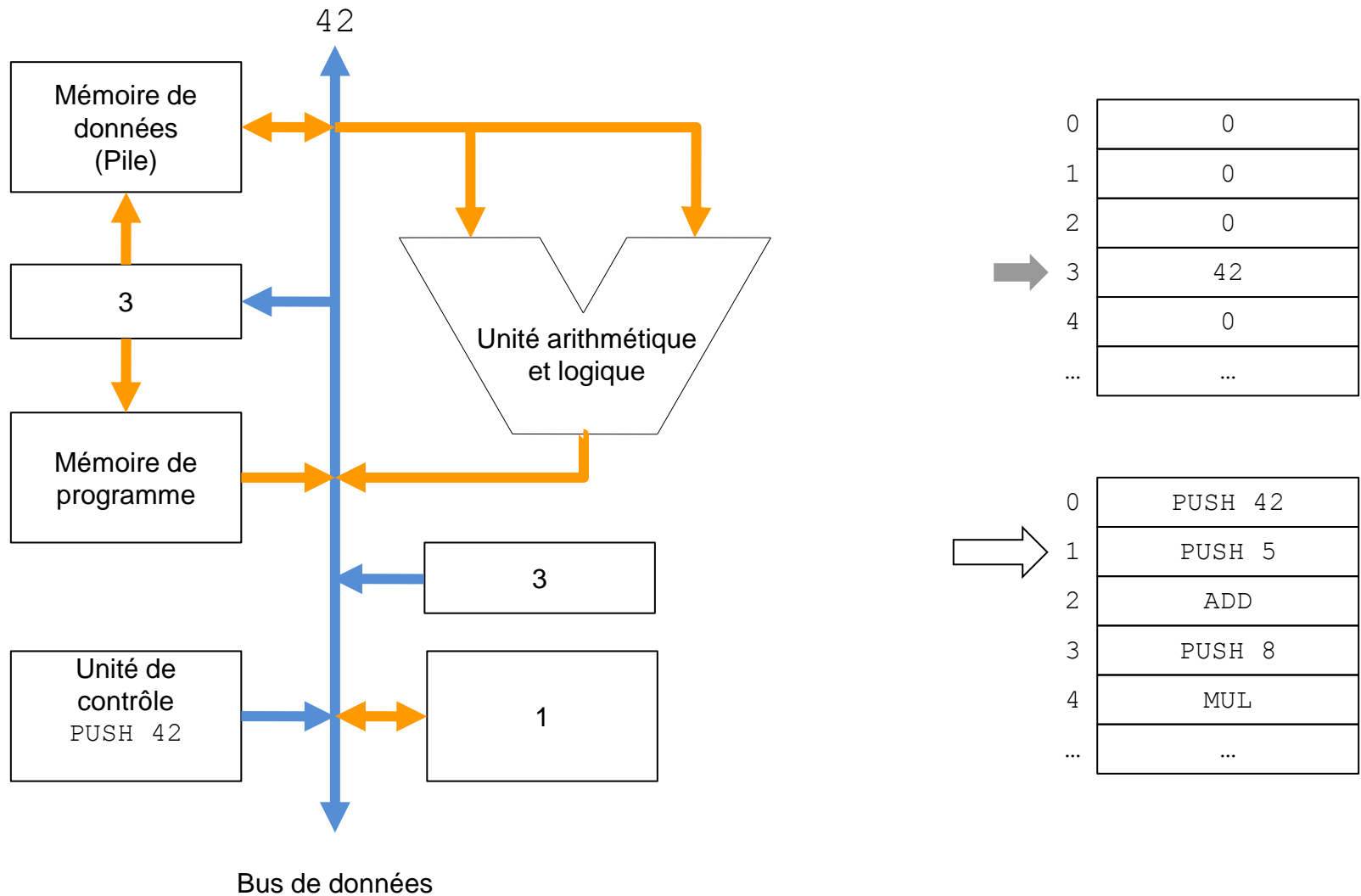
## 3.3.2. Machines à pile



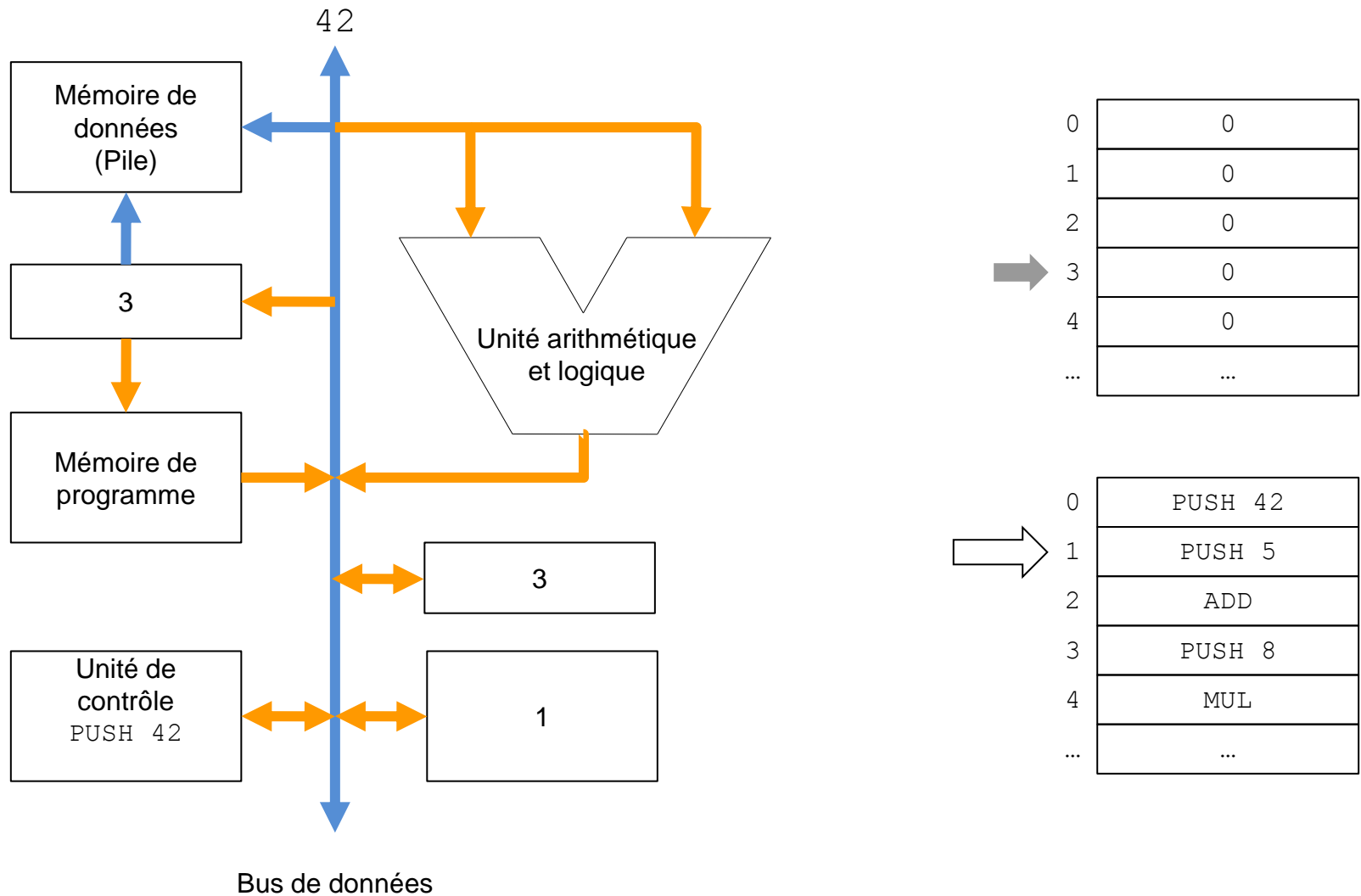
## 3.3.2. Machines à pile



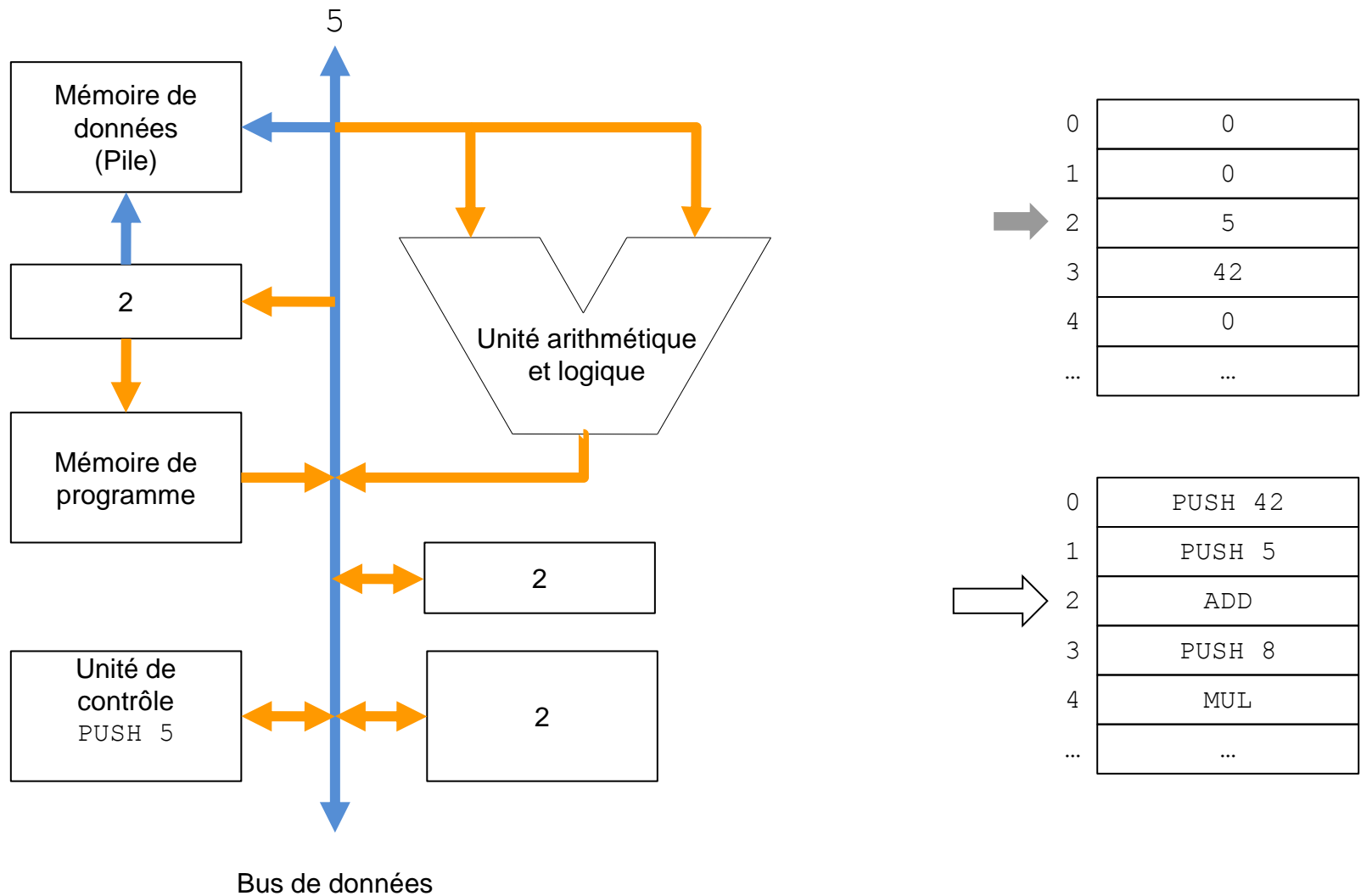
## 3.3.2. Machines à pile



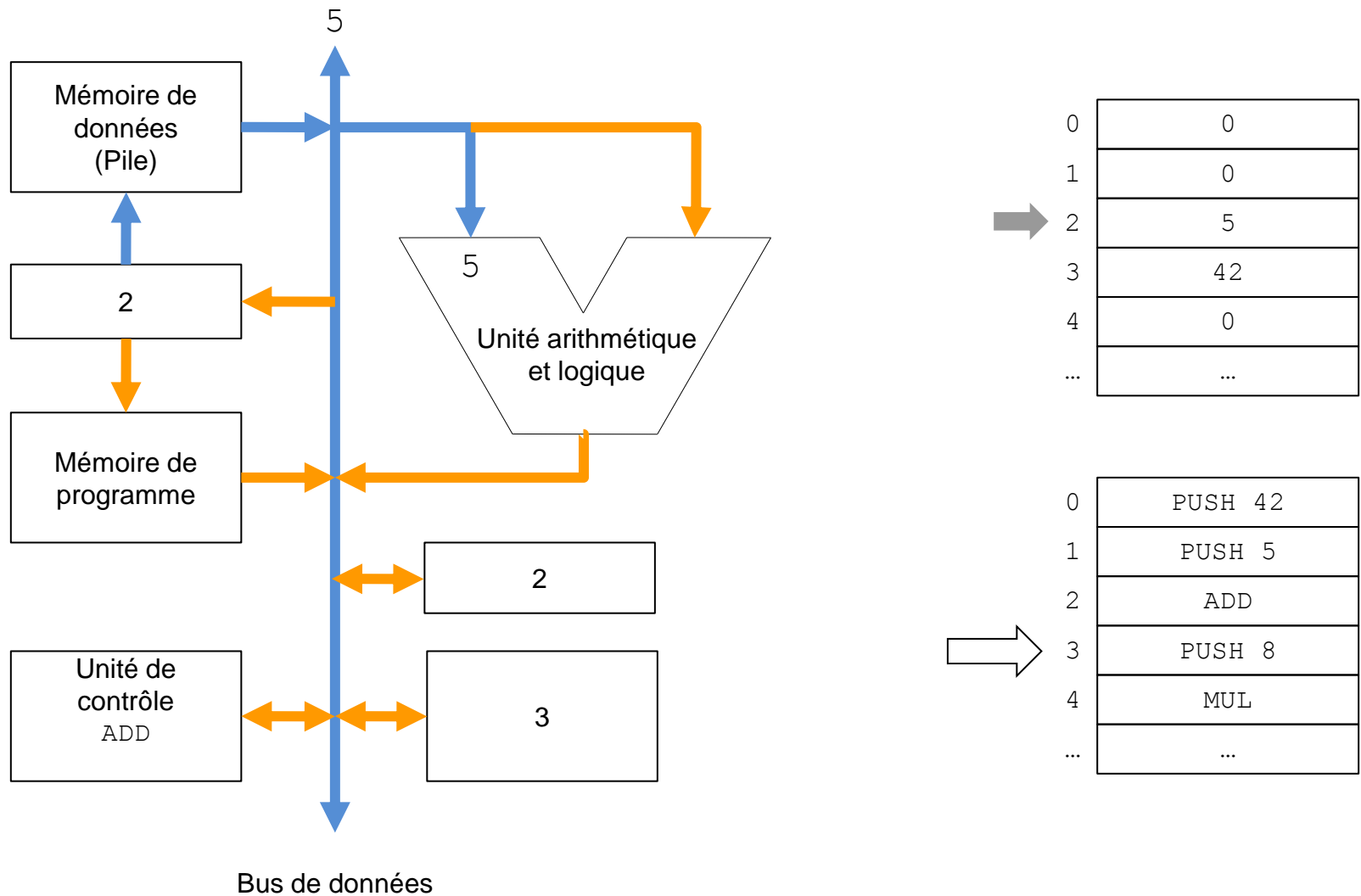
## 3.3.2. Machines à pile



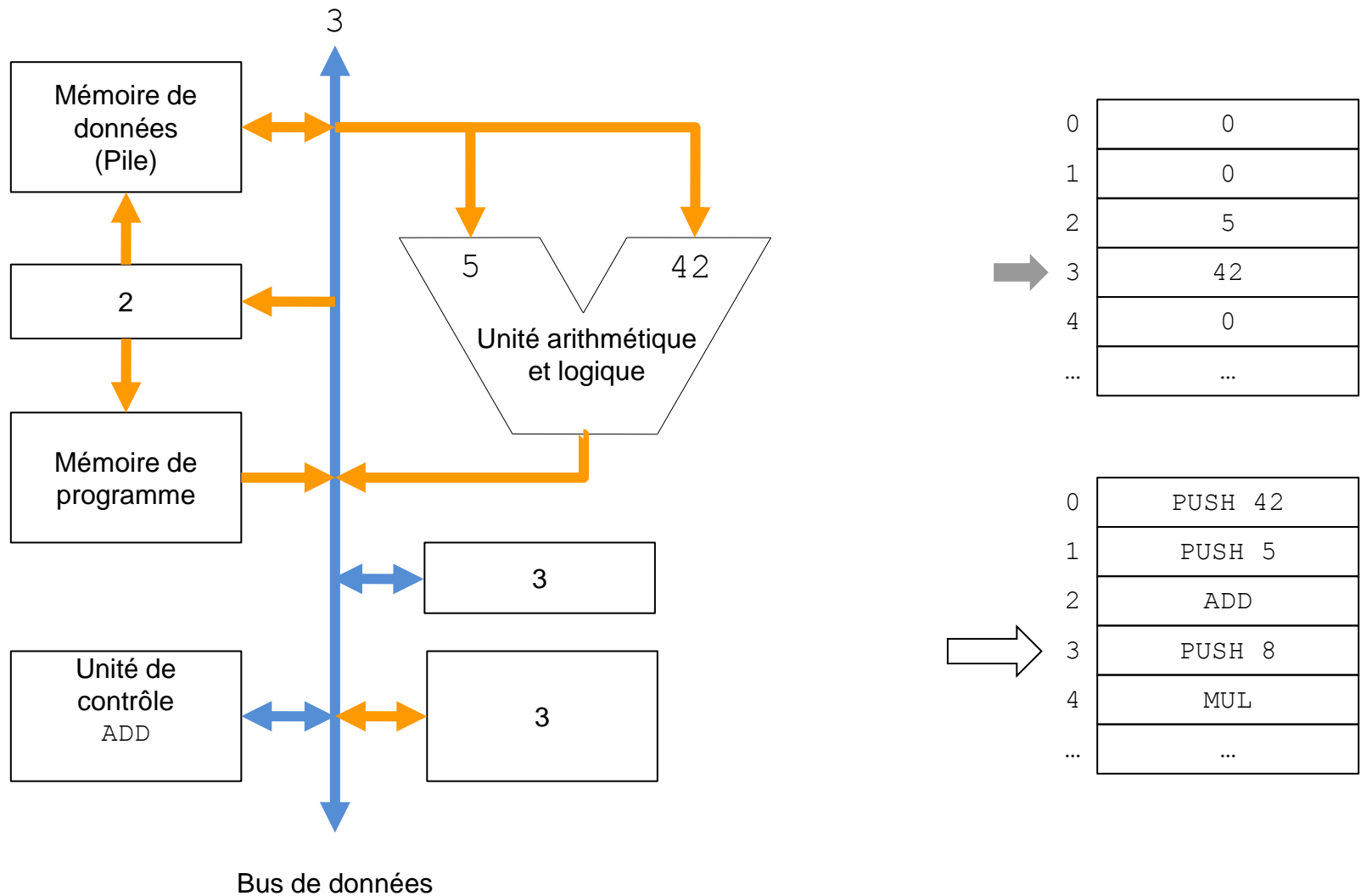
## 3.3.2. Machines à pile



## 3.3.2. Machines à pile

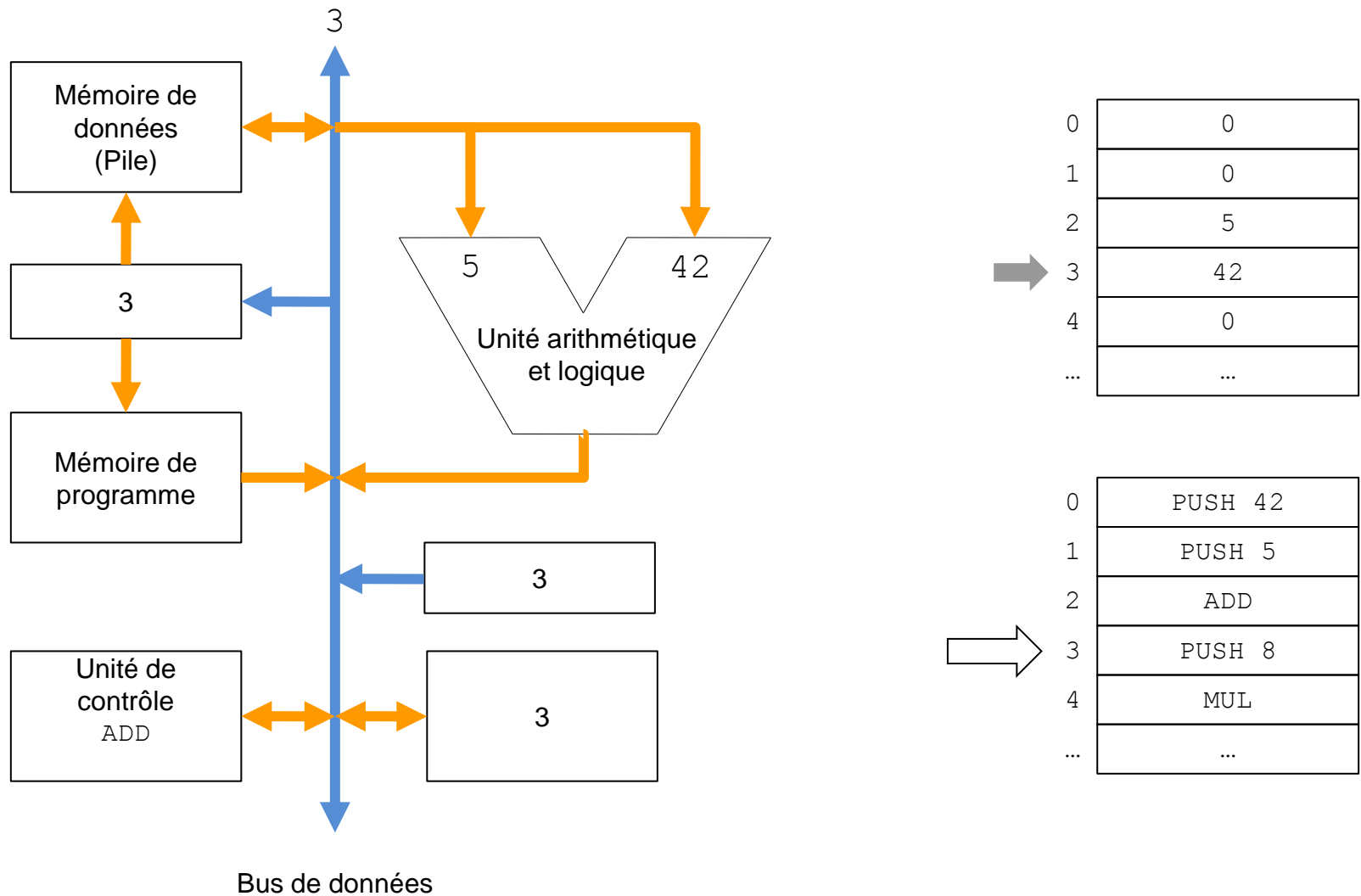


## 3.3.2. Machines à pile

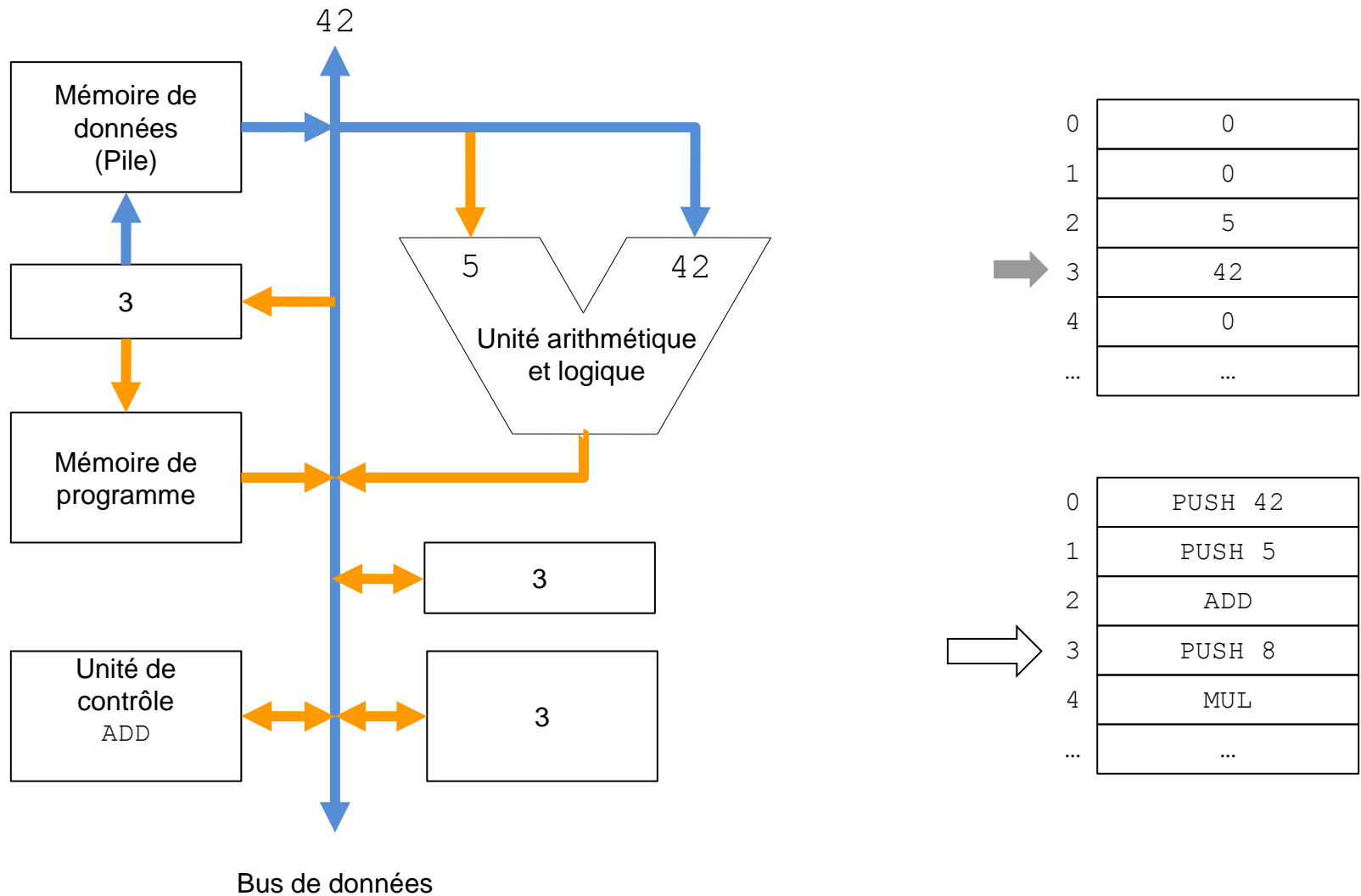




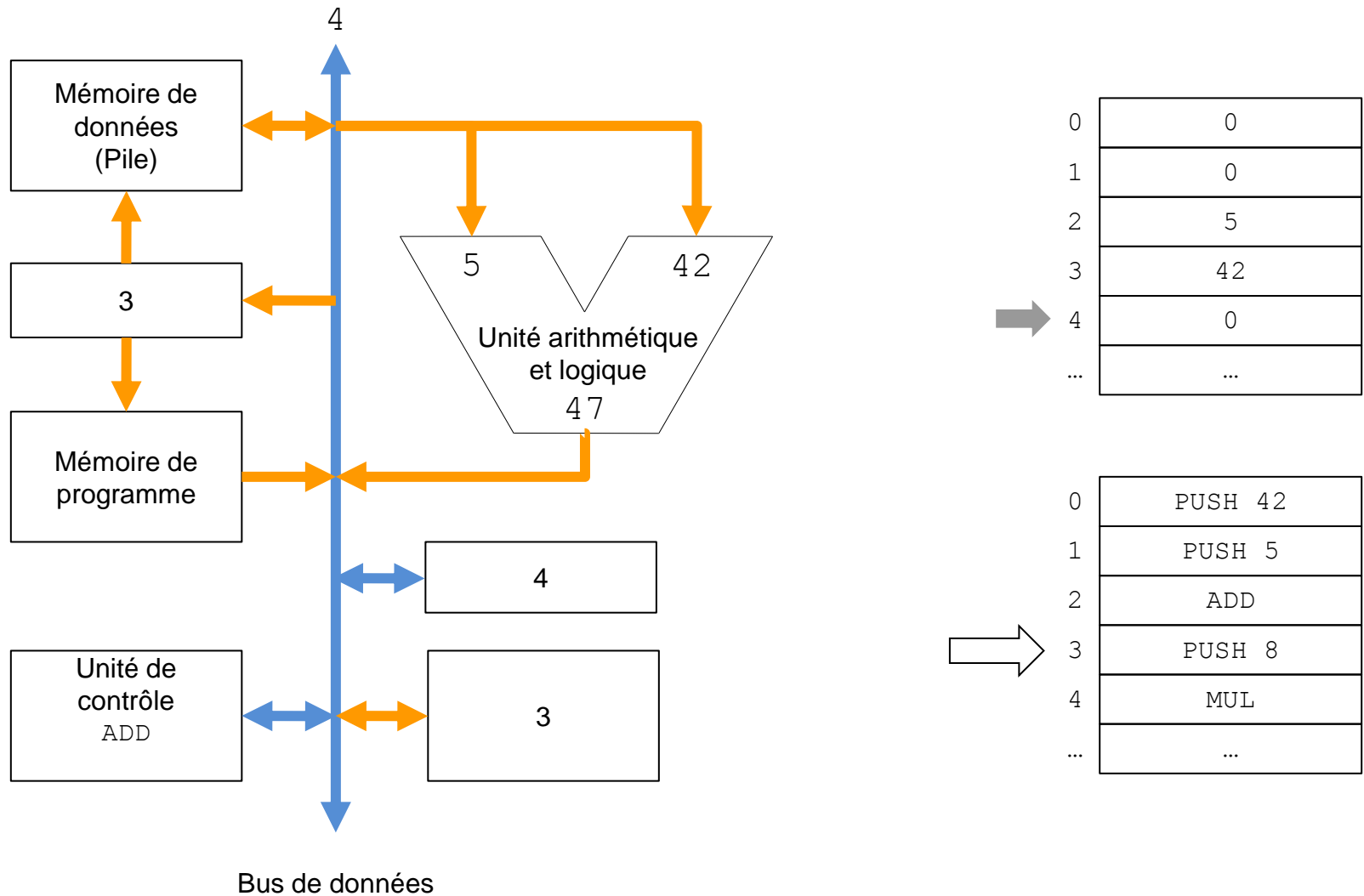
## 3.3.2. Machines à pile



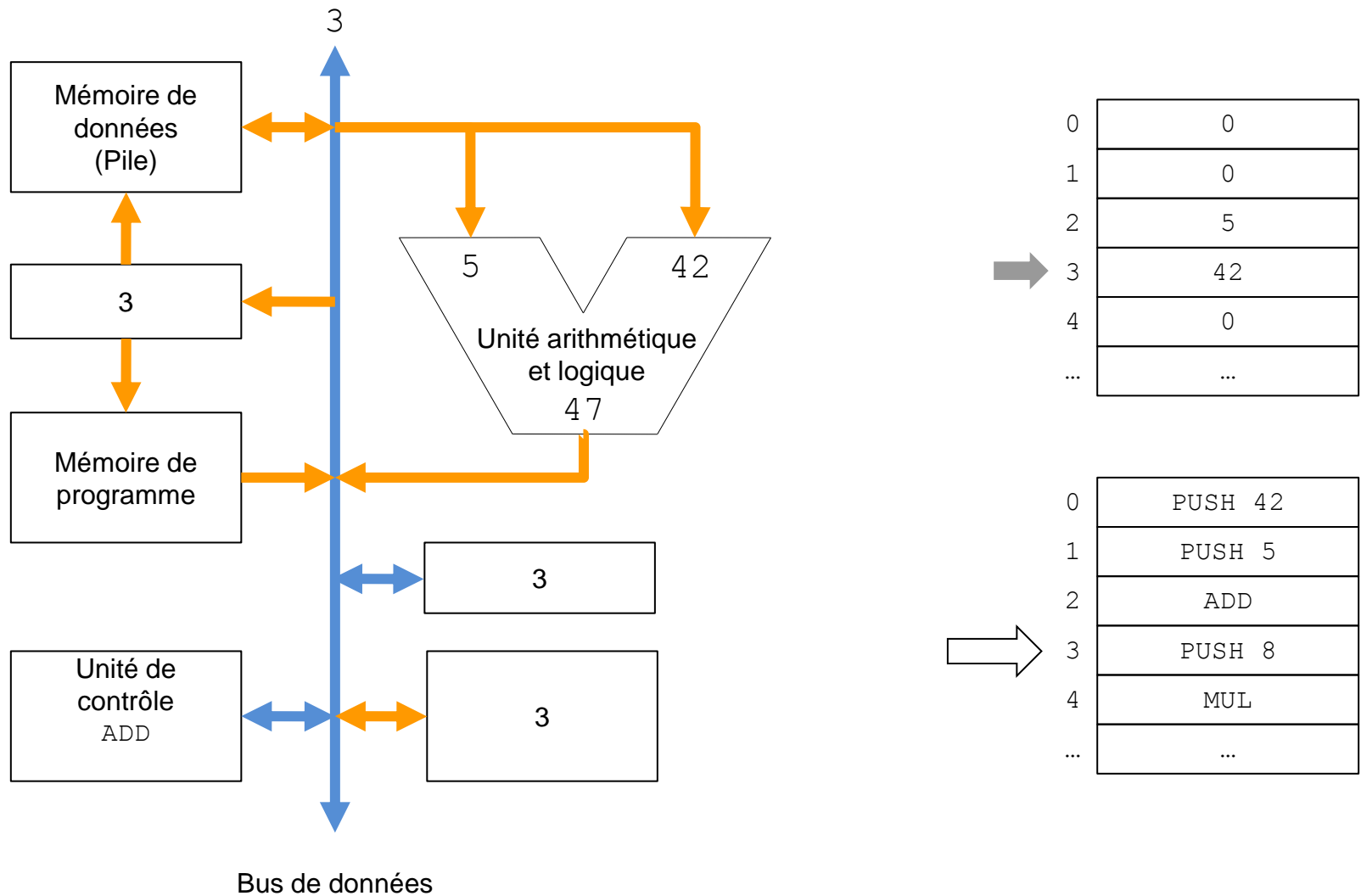
## 3.3.2. Machines à pile



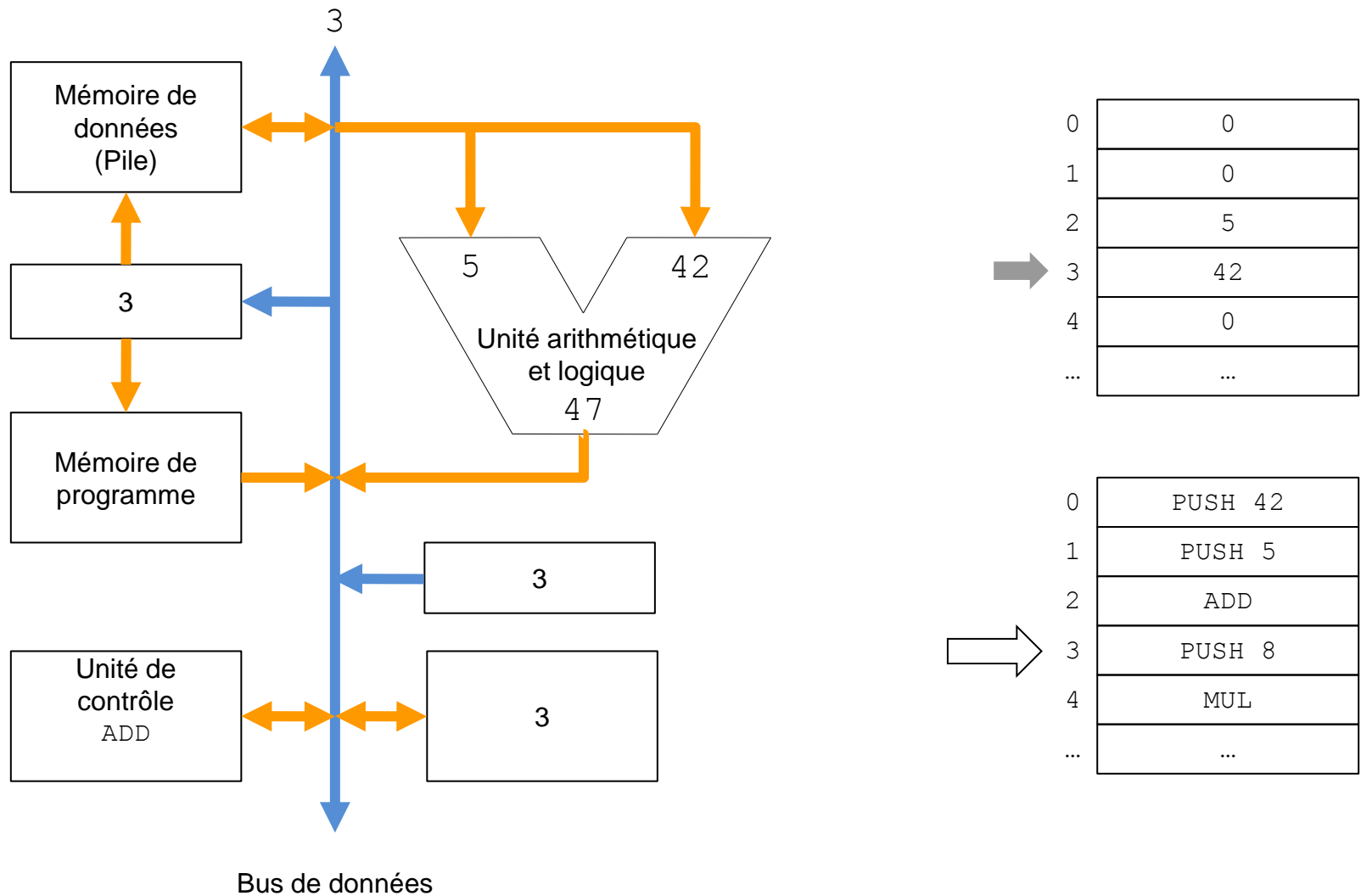
## 3.3.2. Machines à pile



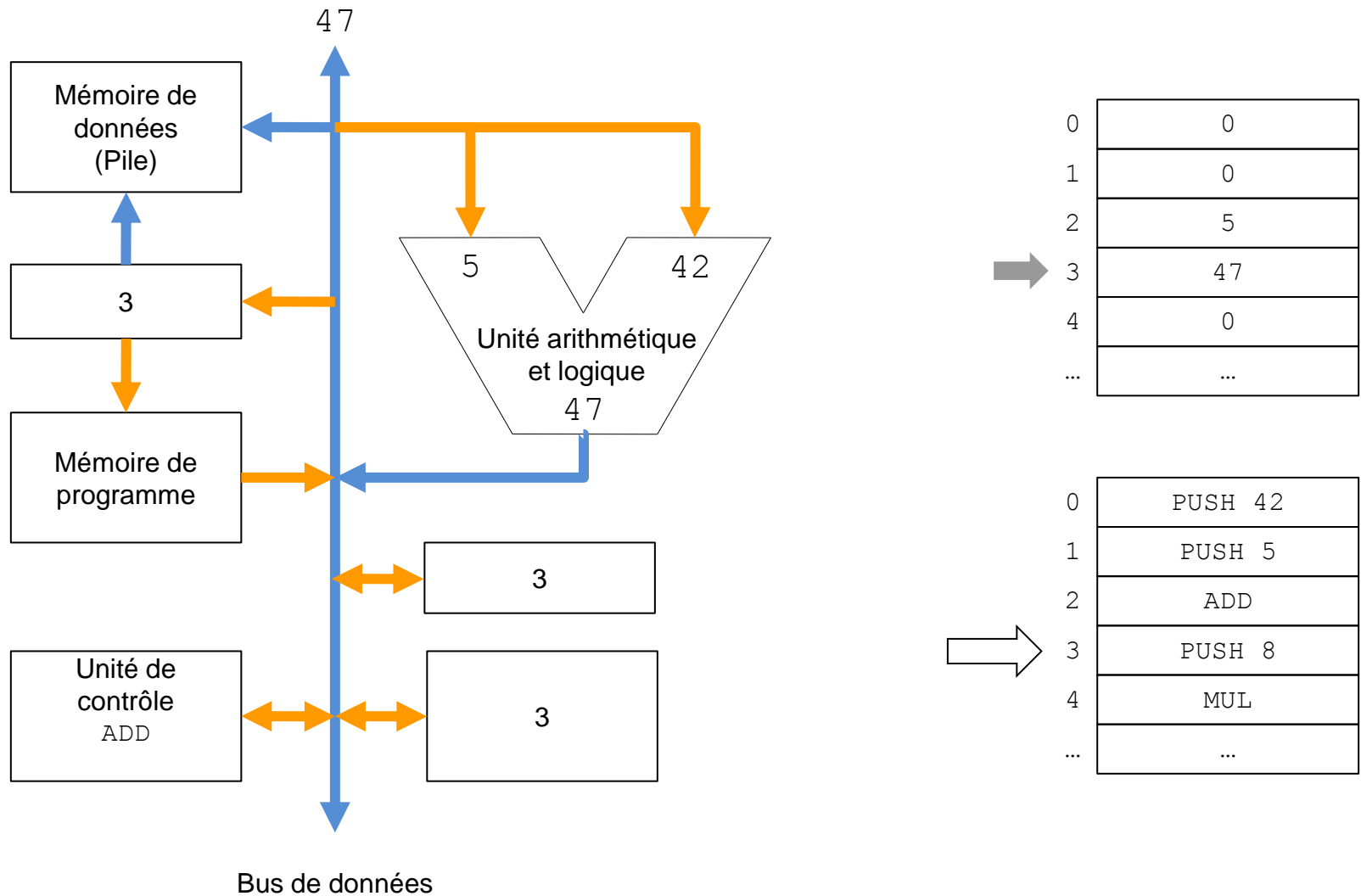
## 3.3.2. Machines à pile



## 3.3.2. Machines à pile



## 3.3.2. Machines à pile



## 3.4. Classes de machines

- Deux classes de machines en fonction de la complexité des instructions « élémentaires »
  - **CISC** (Complex Instruction Set Computer)
    - Années 60
    - Exemples : x86, 68000...
  - **RISC** (Reduced Instruction Set Computer)
    - Années 70-80
    - Exemples : ARM, PowerPC...

## 3.4.1. Machines CISC

- Instructions complexes
  - Plus d'une opération élémentaire par instruction, instructions multi-cycles
  - Opérandes mémoire pour toutes les instructions
  - Code dense
  - ⇒ Plus facile à programmer, pour l'homme et pour les compilateurs !
  - ⇒ Moins de mémoire pour stocker les programmes.
- Peu de registres
  - Plus de place disponibles sur le silicium pour des séquenceurs / stockage de microprogrammes



## 3.4.2. Machines RISC

- Instructions simples
  - Une seule opération élémentaire par instruction, un seul cycle d'horloge par instruction
  - Opérandes registres pour toutes les instructions
  - Chargements mémoire **explicites** (LOAD / STORE)
  - Code plus long
    - ⇒ Moins facile à programmer : il faut plus décomposer
    - ⇒ Plus facile d'optimiser le matériel (pipelining, exécution spéculative...)
- Plus de registres
  - Moins besoin de silicium pour les instructions

## 3.4.3. Un exemple

- Multiplier la case mémoire 2 avec la case mémoire 3, stockage du résultat en 2.

### CISC

#### Programme

MULT [2], [3]

#### Opérations

Chargement du premier registre interne de donnée avec [2]  
Chargement du second registre interne de donnée avec [3]  
Réalisation du produit dans le registre de sortie de l'ALU  
Stockage du résultat en [2]

=> 4 cycles

### RISC

#### Programme

LD r0, [2]  
LD r1, [3]  
MUL r0, r1  
ST [2], r0

#### Opérations

Chargement du premier registre interne de donnée avec [2]  
Chargement du second registre interne de donnée avec [3]  
Réalisation du produit dans le registre de sortie de l'ALU  
Stockage du résultat en [2]

=> 4 cycles

# Sommaire

1. Systèmes à microprocesseur
2. La mémoire
3. Le processeur
4. Notre cas d'étude : la famille x86 d'Intel
  - 4.1. La famille x86
  - 4.2. Modes de fonctionnement
  - 4.3. Structure et fonctionnement d'un processeur x86

## 4.1. La famille 80x86

- Les processeurs Intel 8086/8088 sont la base de tous les ordinateurs PC compatibles (8086 introduit en 1978, IBM-PC en 1981)
- Tous les processeurs évolués d'Intel et AMD sont compatibles avec le 8086.
- A la mise sous tension, les Pentium, Athlon, ... se comportent comme des 8086.

Compatibilité ascendante

## 4.1. La famille 80x86

- 8086
  - Processeur 16 bits
  - Bus de données 16 bits
  - Bus d'adresse 20 bits => Capable d'accéder à  $2^{20}$  octets = 1Mo
  - Modèle mémoire segmenté
  - Ordre de stockage des mots « little endian »

# 4.1. La famille 80x86

- Pentium
  - Processeur 32 bits
  - Bus de données de 64 bits
  - Bus d'adresse 32 bits => Accès à  $2^{32}$  octets (4 Go)
  - Compatible avec le 8086 (émulation à la mise sous tension)
  - Cette compatibilité implique des particularités et des complexités que nous n'étudierons pas...

## 4.2. Modes de fonctionnement

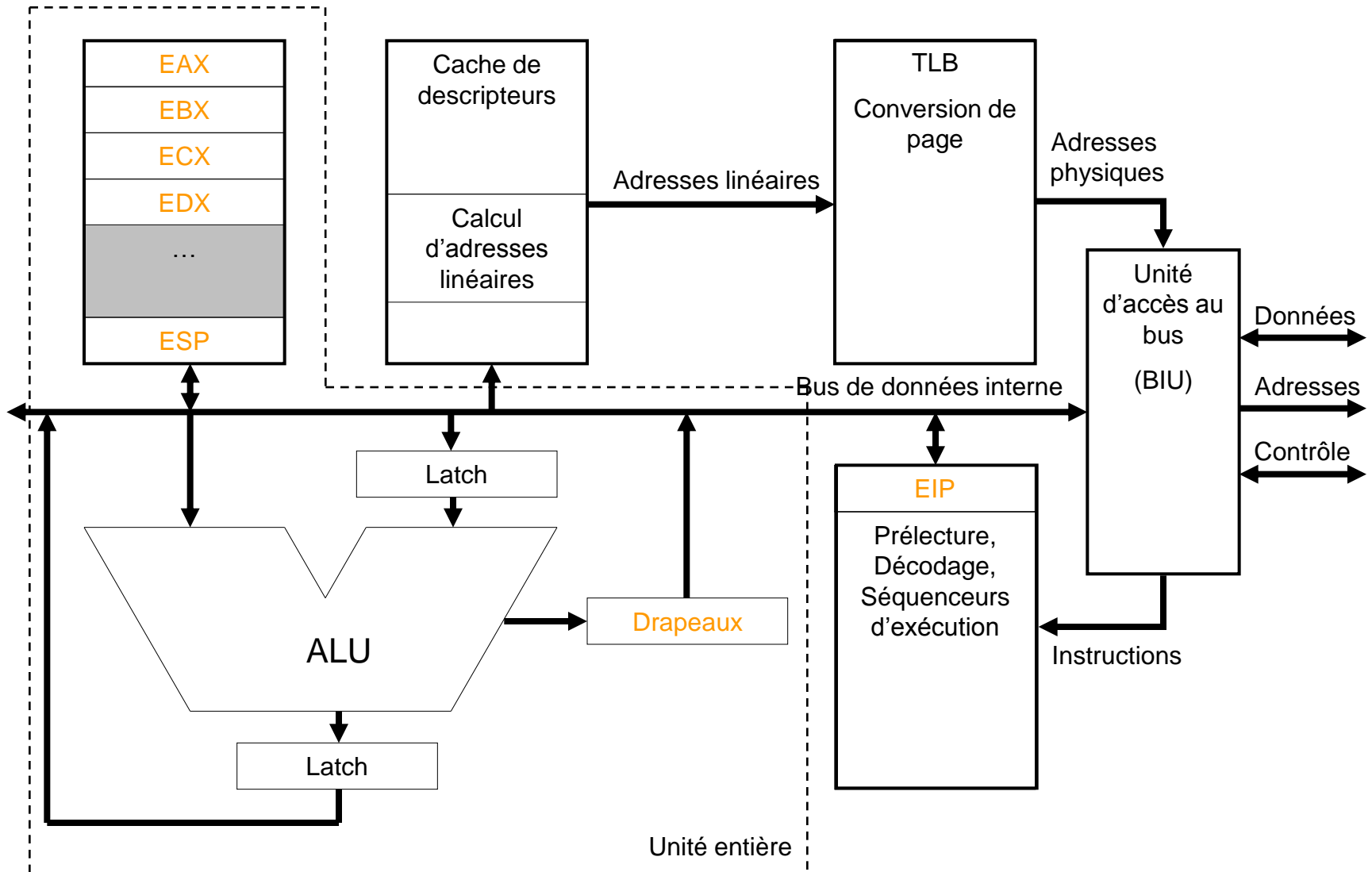
- Deux modes d'accès à la mémoire : réel et protégé
  - Mode réel : actif à la mise sous tension -  
Emulation d'un 8086  
C'est le seul mode de fonctionnement du 8086  
Ne permet l'accès qu'à 1Mo de mémoire
  - Mode protégé : introduit avec le 80286  
Dans ce mode, le processeur vérifie tous les accès à la mémoire.  
Permet l'accès à la totalité de l'espace d'adressage des processeurs (ex : 4Go sur un Pentium)

## 4.2.Modes de fonctionnement

- En fonction des processeurs, deux modes de taille d'opérande
  - 16 bits (seul mode disponible jusqu'au 80286)
  - 32 bits (à partir du 80386)
- Qui peut le plus peut le moins :
  - Tous les processeurs 32 bits (386 et supérieurs) peuvent fonctionner en mode 16 bits !



## 4.3. Structure fonctionnelle x86



## 4.3. Structure fonctionnelle x86

- Une (ou plusieurs) unités arithmétiques et logiques
- Un ensemble de registres internes destinés à recevoir les résultats des calculs : **registres d'usage général** (ici EAX, EBX,...)
- Des registres spécialisés : par exemple
  - Le **pointeur de pile** (ici ESP) contient l'adresse linéaire du haut de la pile
  - Le **pointeur d'instruction** (ici EIP) contient l'adresse linéaire de l'instruction à exécuter
  - Les **indicateurs** (ici EFLAGS) qui reflètent certaines propriétés du résultat d'un calcul
- Des unités de calcul et de conversion d'adresses
- Des unités de calcul supplémentaires (calculs flottants...)
- Une unité de contrôle qui pilote toutes les autres en fonction de l'instruction

## 4.3. Instruction

- Une instruction déclenche une **opération** élémentaire qui agit sur ses **opérandes**

Exemple :  $EAX \leftarrow EAX + [12]$




Registre      Registre    Opération      Mot mémoire

ou encore `ADD EAX, [12]`

## 4.3. Programme

- Un programme est constitué d'une suite d'instructions rangées en mémoire
- Le pointeur d'instructions désigne l'instruction à exécuter.

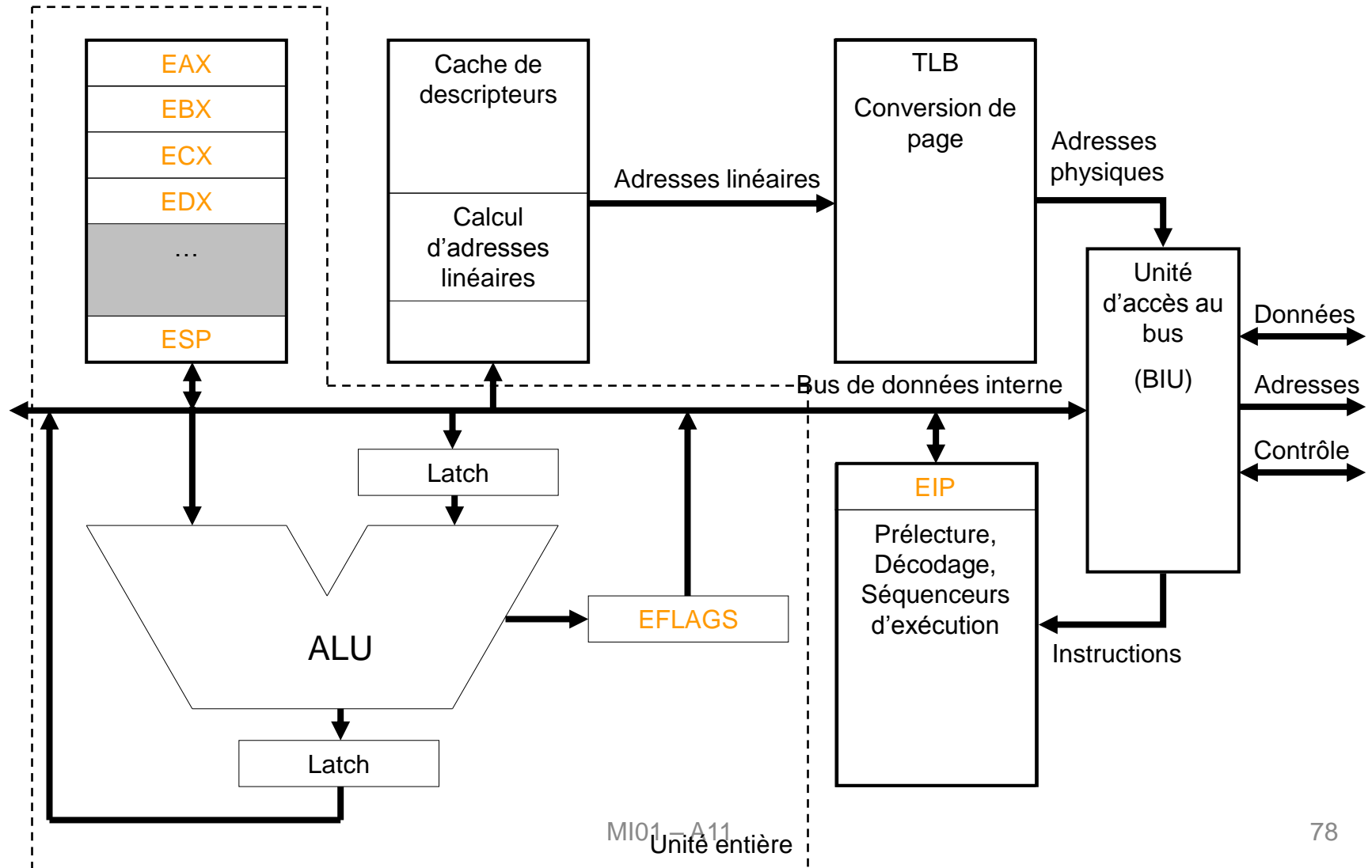
EIP  ...  
mov eax, 1  
add eax, [12]  
mov ebx, eax  
...

# Déroulement d'une instruction

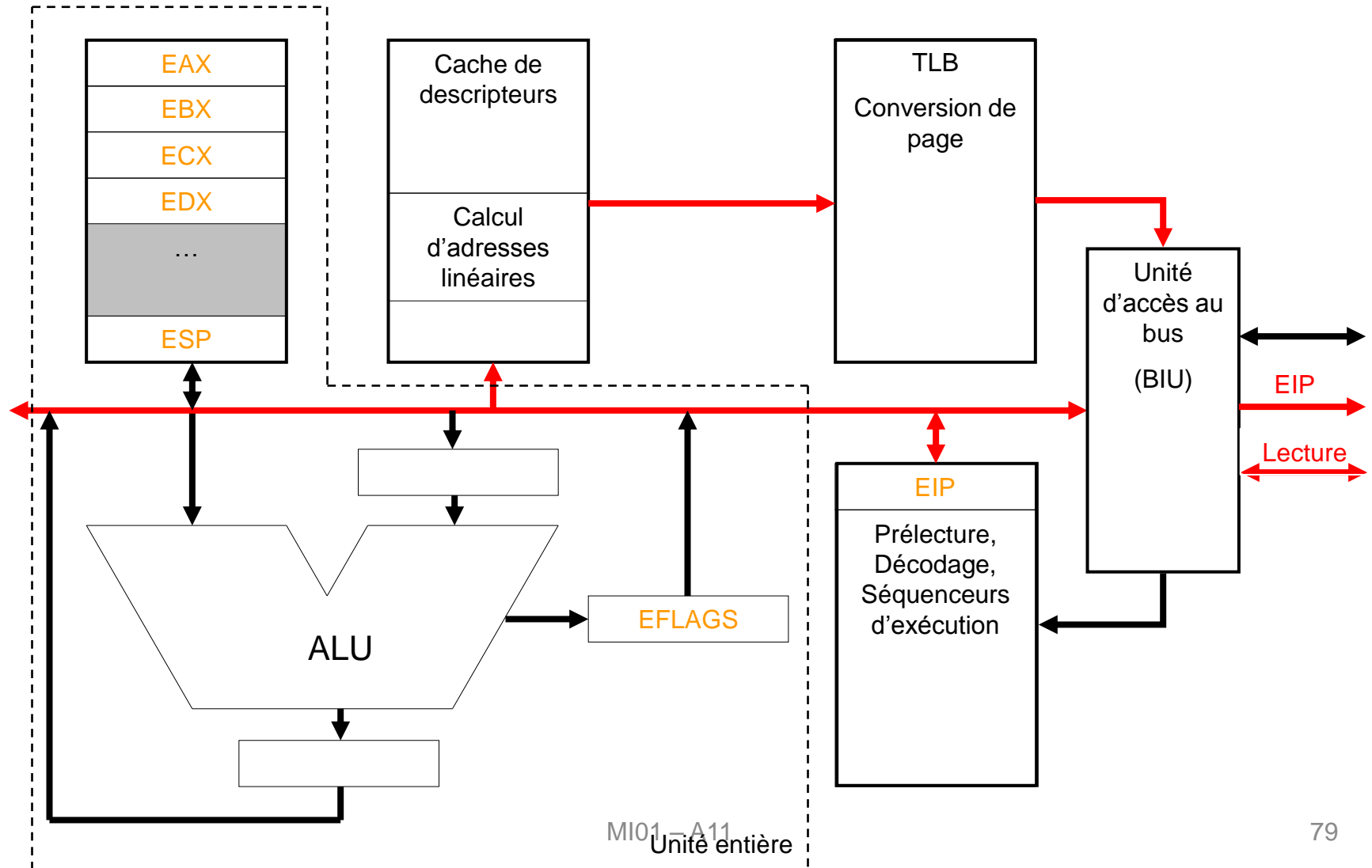


# Exécution d'une instruction

Cycle 0 (Etat initial)



### Cycle 1 (Lecture de l'instruction 1/2 : le processeur lance la lecture)



## Cycle 2 (Lecture de l'instruction 2/2 : le processeur lit l'instruction et la décode)

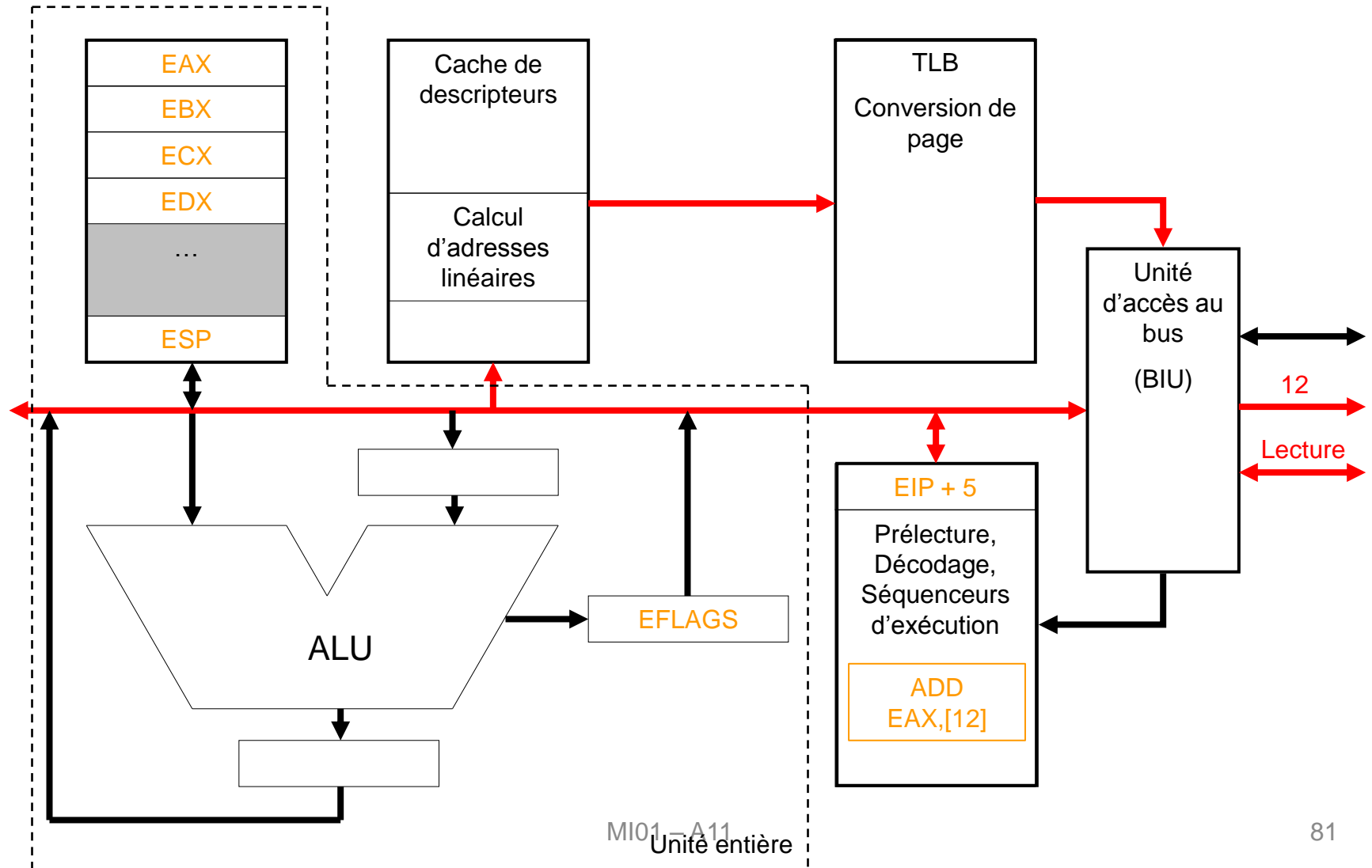
## Cycle 2 (Lecture de l'instruction 2/2 : le processeur lit l'instruction et la décode)





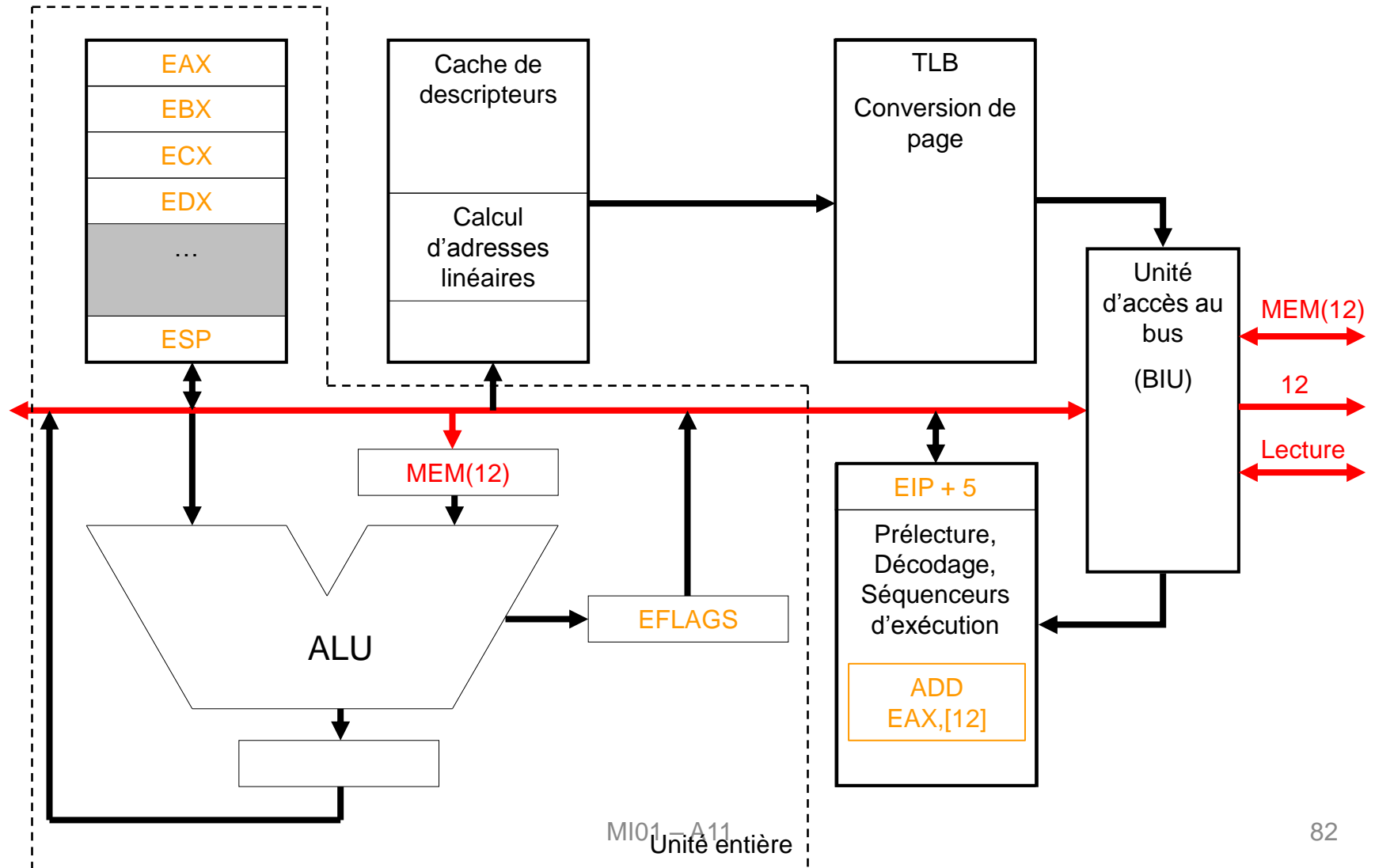
# 4.3. Exécution d'une instruction

Cycle 3 (Lecture des opérandes mémoire 1/2 : le processeur lance la lecture)



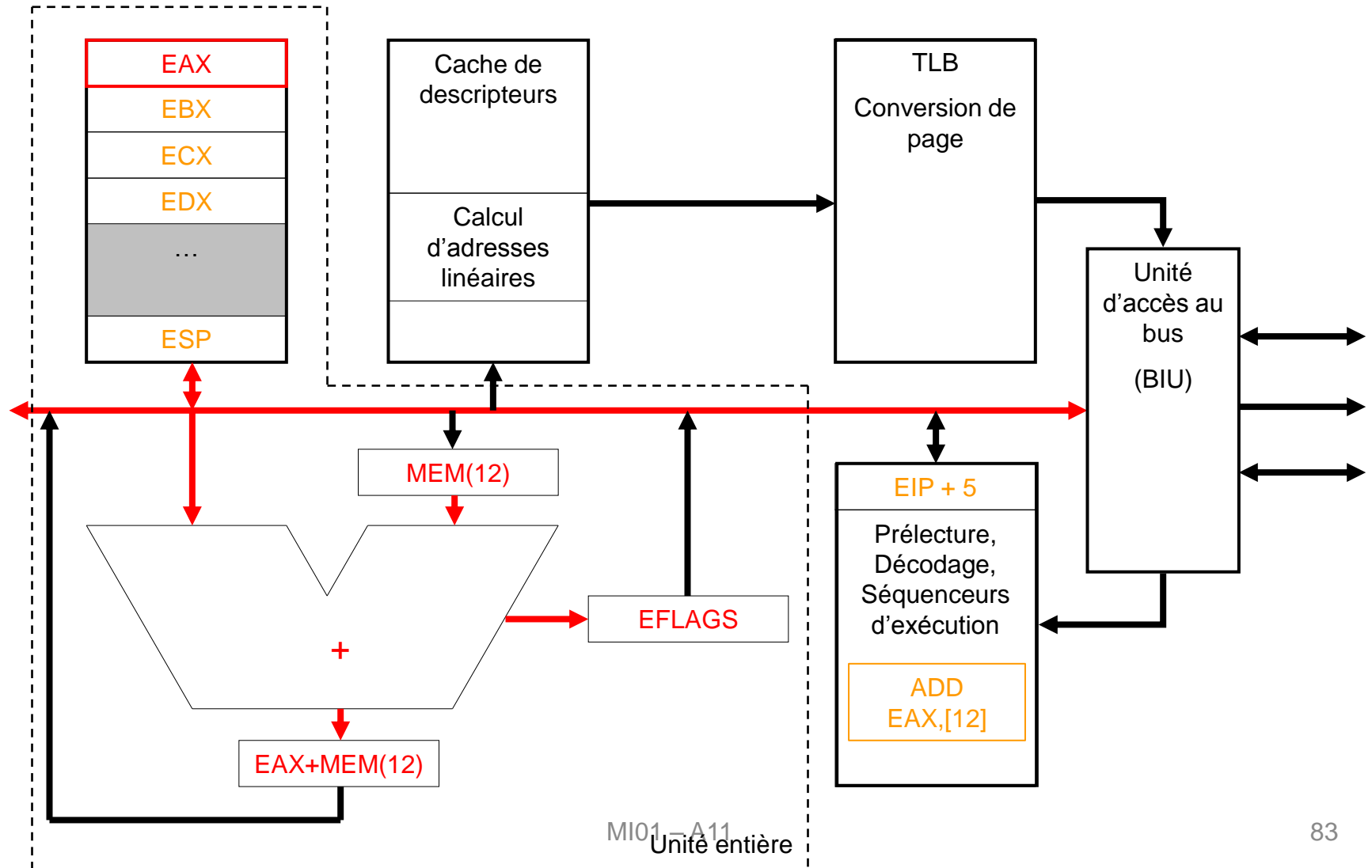
## 4.3. Exécution d'une instruction

Cycle 4 (Lecture des opérandes mémoire 2/2 : le processeur lance la lecture)



# 4.3. Exécution d'une instruction

Cycle 5 (Réalisation de l'opération et positionnement des indicateurs)



# 4.3. Exécution d'une instruction

Cycle 6 (Rangement du résultat)

