



Principes des Systèmes d'Exploitation

Licence en Ingénierie Informatique
Deuxième année

Résumé du cours

Connaitre:

- ▶ **La couche micro-architecture d'un ordinateur**
- ▶ **Les concepts fondamentaux des SE**
- ▶ **Les outils de base, les types et mode de fonctionnement des systèmes d'exploitation.**

Etre capable de:

- ▶ Comprendre le chemin des données et le langage machine
- ▶ Décrire le processus de gestion de ressources matérielles et logicielles
- ▶ Expliquer le cycle de vie d'un processus et l'interaction entre processus
- ▶ Décrire les algorithmes d'ordonnancement et de gestion de la mémoire
- ▶ Expliquer le principe et organisation des e/s
- ▶ Décrire le systèmes de fichiers et les opérations sur les fichiers

Systèmes d'Exploitation

Sommaire du Cours

① Architecture d'un micro-ordinateur

② Chemin des données

③ Gestion des processus

④ Les algorithmes d'ordonnancement

⑤ Parallélisme et interaction entre processus

⑥ La gestion de la mémoire

7-La gestion des Entrées/Sorties

Pré-requis

- ▶ **Connaître les fonctions logiques élémentaires et la programmation**

Evaluation

- ▶ **??????**

Documents

- ▶ **Slides**

Mots clés

- ▶ **Micro-architecture, processus, ordonnancement , taches, sémaphore, moniteur, segmentation, pagination, mémoire virtuelle**

Bibliographie-webographie

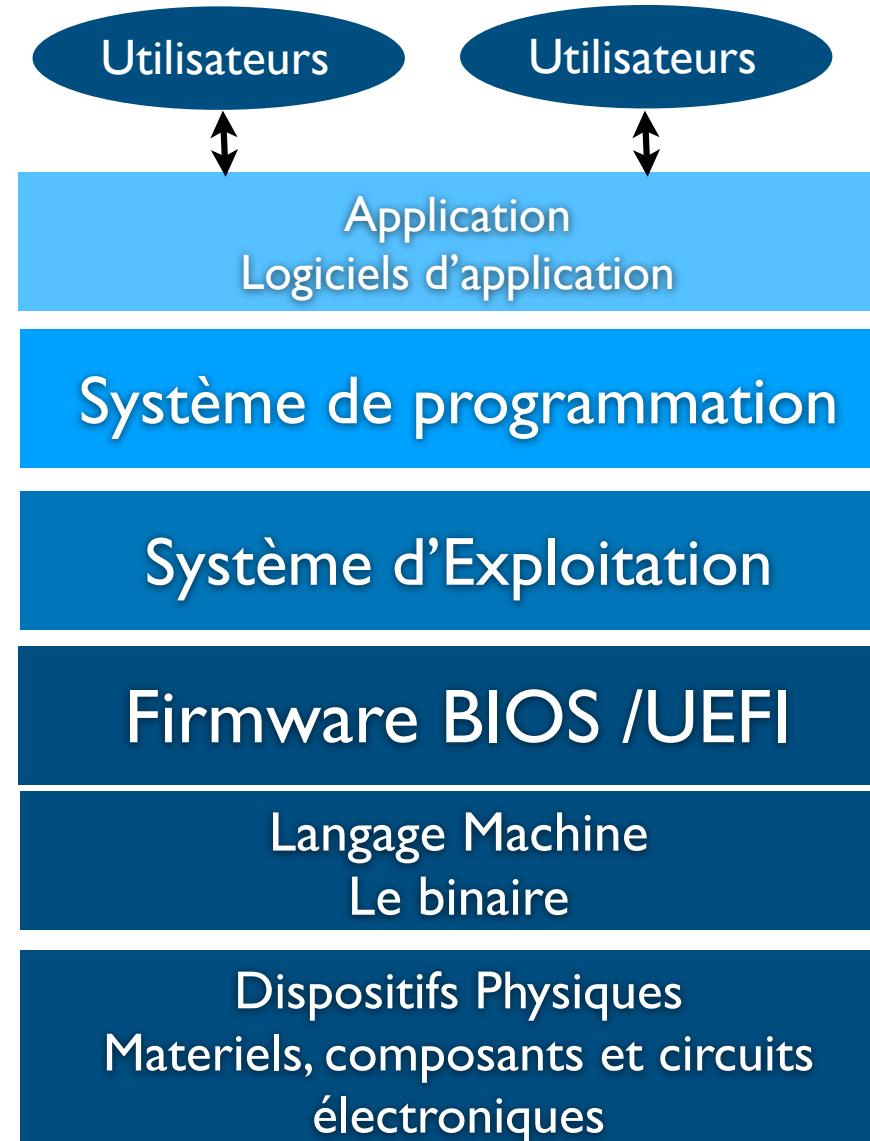
▶https://wiki.osdev.org/Expanded_Main_Page

▶<https://chamilo.grenoble-inp.fr/courses/ENSIMAG4MMPCSEF/document/projet.html>

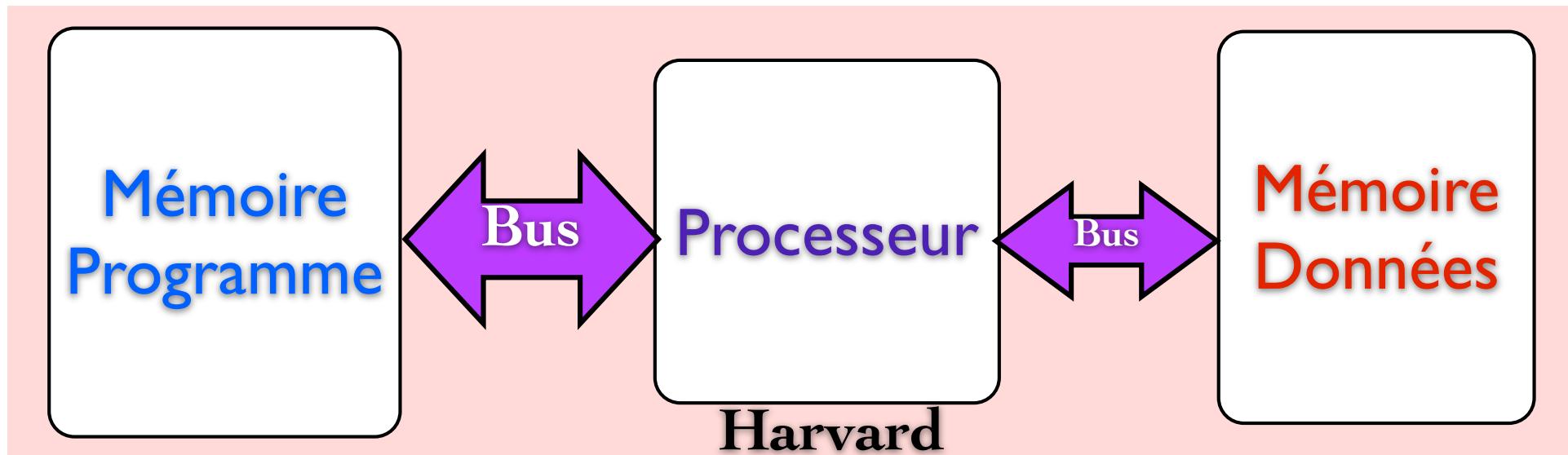
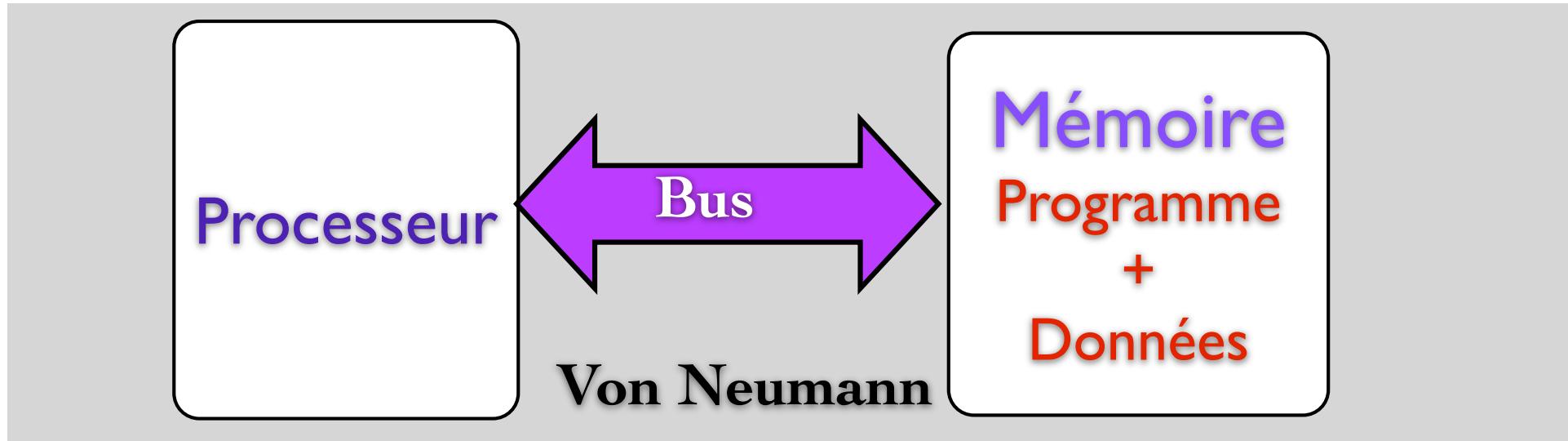
Chapitre I

Architecture d'un micro-ordinateur

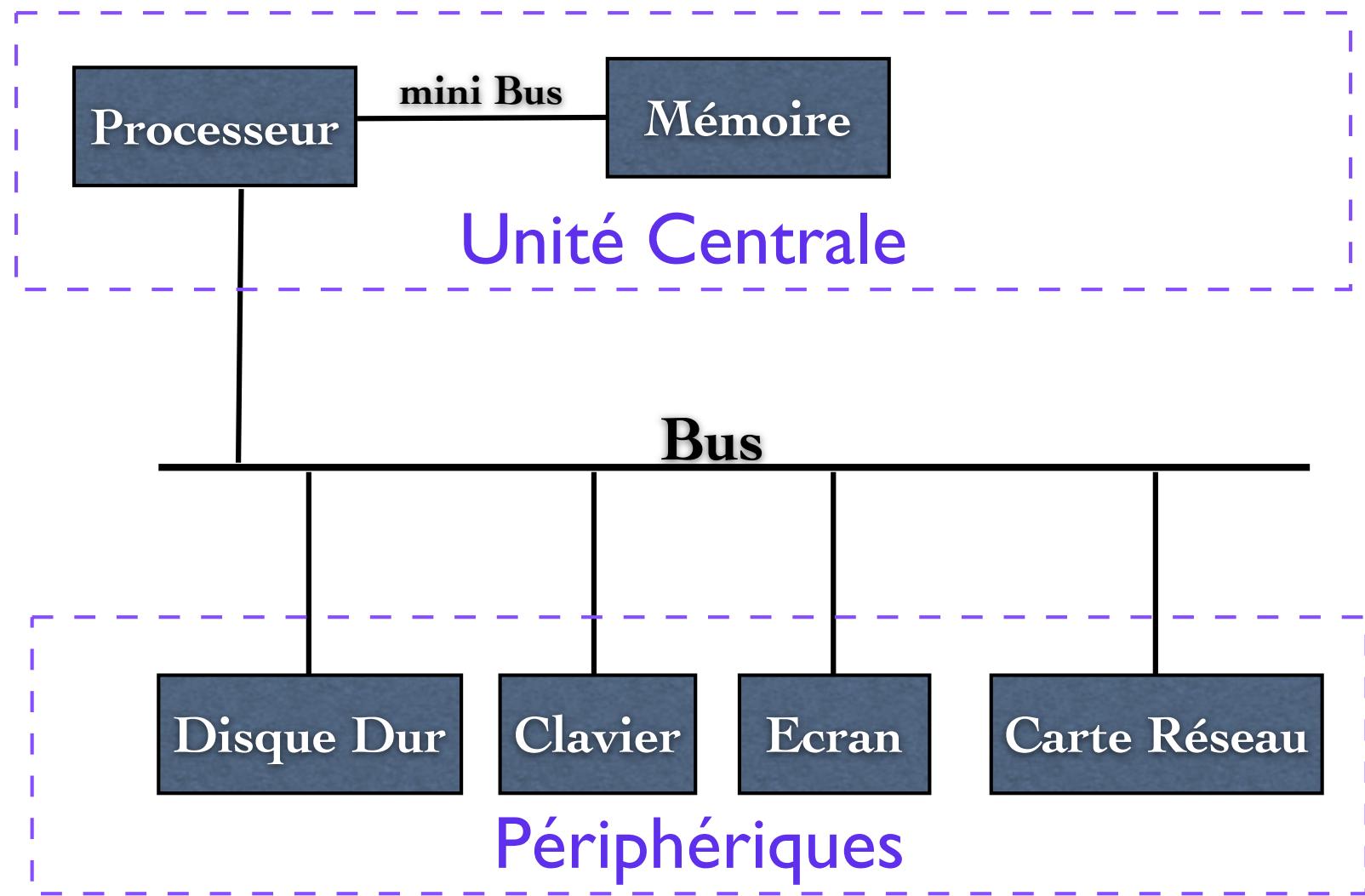
Architecture en couches



Rappel: Architecture de Von Neumann et de Harvard



Rappel: Architecture d'un micro-ordinateur



Le processeur CPU

- Unité arithmétique et logique
 - Dirige le processeur
 - Exécution des fonctions (arithmétiques, logiques)
- Unité de commande
 - Décodage des instructions
 - Coordonne l'exécution des instructions
- Registres: mémoires internes au CPU
 - Nombre dépend du CPU
 - Taille dépend du CPU
 - Exprimée en bits
 - Caractérise le processeur
 - Exemple processeur de 32 bits

Le processeur CPU

- La puissance du processeur
- Fréquence F du CPU est exprimée en MHz, ou GHz
- Cycle Par Instruction (CPI) nombre moyen de cycles d'horloge pour l'exécution d'une instruction pour un CPU donné
- Millions d'Instruction par seconde(MIPS) qui est la puissance de traitement du processeur
- $\text{MIPS} = \frac{F}{\text{CPI}}$, F exprimée en MHz



- Commercialisé par Intel en 2010
- 1 170 000 000 transistors
- 147 600 MIPS
- 3,47 GHz (cycles par seconde)
- Améliorer la performance d'un CPU
 - Augmenter la fréquence d'horloge
 - Diminuer le CPI

Le jeu d'instructions

- ▶ **Ensemble d'opérations élémentaires réalisables par un CPU**
- ▶ **Arithmétiques:** addition, soustraction, multiplication, division
- ▶ **Logique:** ET, OU, NON, NAND, Comparaison, Décalage etc..
- ▶ **Transfert de données:** charger ou sauver en mémoire
- ▶ **Contrôle de séquence:** saut/branchement etc...
- ▶ **Deux classes de processeurs**
- ▶ **CISC:** Complex Instruction Set Computer (Intel, AMD ..)....
- ▶ **RISC:** Reduced Instruction Set Computer (Apple, Sun....)

Architecture CISC

- ▶ **CISC:** Complex Instruction Set Computer (Intel, AMD ..)
- Instruction complexe= combinaisons de plusieurs instructions simples (élémentaires).
- Objectifs
 - ▶ Vitesse traitement du CPU >> traitement RAM:
 - ▶ Soumettre au CPU des instructions complexes
 - ▶ Réduire les accès multiples à la RAM
 - ▶ Faciliter la conception du compilateur pour les langages de haut niveau
- **Conséquences**
 - Plus d'instructions, de tailles différentes,
 - Une instruction dure plusieurs cycles d'horloge etc.....

Architecture RISC

- ▶ **RISC:** Reduced Instruction Set Computer (Apple, Sun....)
- CISC: des statistiques montrent que 80% des programmes évolués utilisent 20% des instructions du CPU
 - ▶ Réduire le jeu d'instruction à ces 20% utilisés
 - ▶ Soumettre au CPU des instructions simples
 - ▶ Instructions complexes réalisées à travers une combinaison d'instruction élémentaires
- **Conséquences**
 - Moins d'instructions (dites de base) de même taille,
 - Chaque instruction dure un seul cycle d'horloge etc..

Architecture CISC vs RISC

► RISC

- ① Instructions simples
 - ② Réduit le nombre de cycles par instruction au détriment du nombre d'instructions par programme
 - ③ Un cycle d'horloge par instruction
 - ④ Instruction moins proche des langages évolués
 - ⑤ Compilateur plus complexe
-

► CISC

- ① Instructions complexes
 - ② Minimise le nombre d'instructions par programme au détriment du nombre de cycles par instruction
 - ③ Plusieurs cycles d'horloge/instruction
 - ④ Instruction plus proche des langages évolués
 - ⑤ Compilateur plus simple
-;

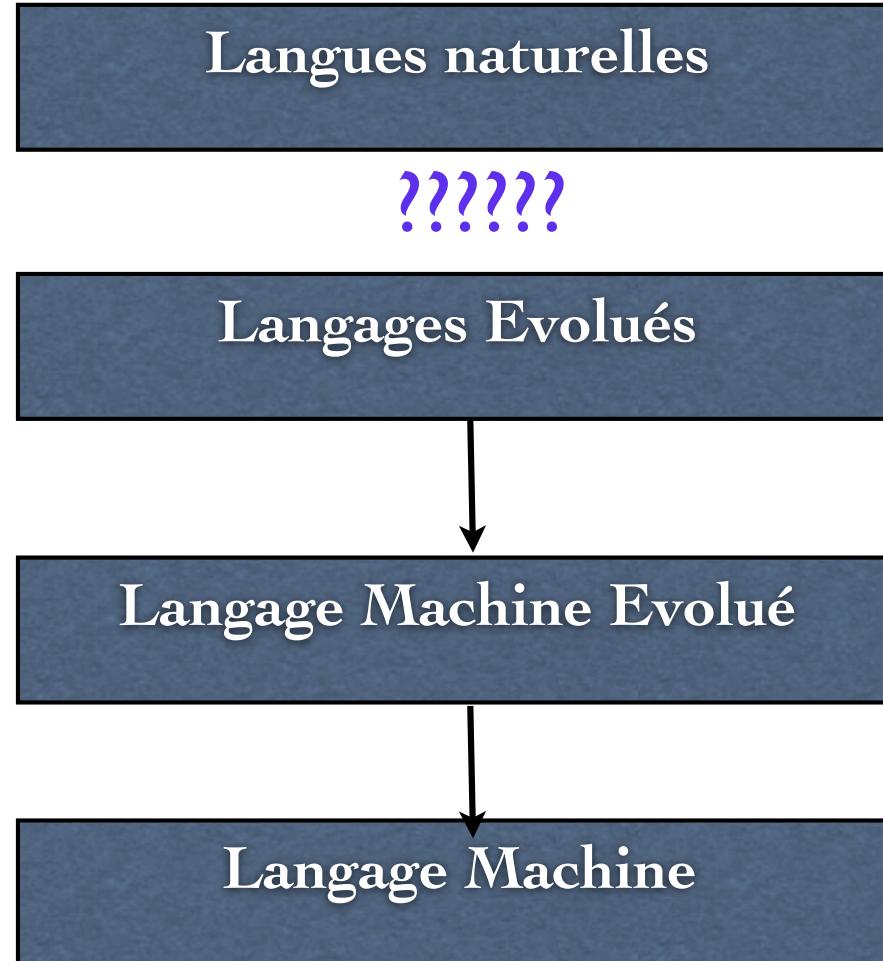
Langages

- ▶ Diola, Français,
Sérère, Wolof,
etc..

- ▶ C, C++
Pascal,
etc..

- ▶ Assembleur

- ▶ Jeu d'instructions
du CPU



- ▶ if, then, else
Begin,
etc..

- ▶ Move, Add,
Jump etc..

- ▶ 0110001000000000

Etapes des instructions

▶ Codage d'une instruction sur 2 parties en langage machine

Code Instruction	Code Opérande1	Code Opérande2
------------------	----------------	----------------

- **Code instruction:** qui indique au processeur quelle instruction réaliser
- **Code opérande:** qui contient la donnée, ou la référence à une donnée en mémoire (son adresse)

▶ 1. Lecture d'instruction

▶ 2. Décodage d'instruction

▶ 3. Lecture des opérandes

▶ 4. Exécution

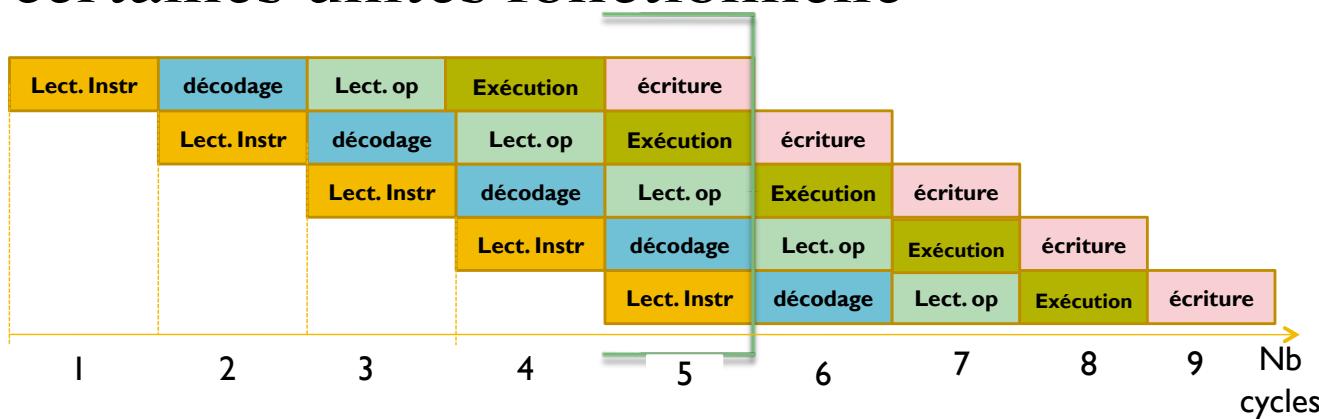
▶ 5. Ecriture éventuelle des résultats en mémoire

Notion de pipeline

- ▶ Modèle classique: séquence des instructions



- ▶ Modèle pipeline: tirer partie de l'indépendance de certaines unités fonctionnelle



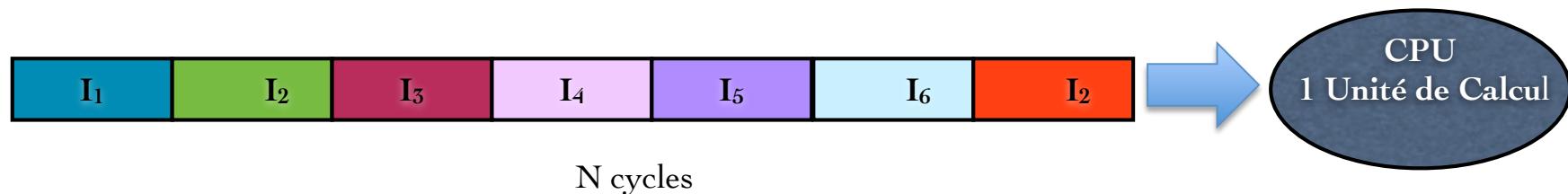
- ▶ On traite 5 fois plus d'instructions

Problèmes possibles....

- ▶ Généralement l'opération qui prend plus de temps est le **calcul** qui demande un développement dans l'UAL, la cadence du processeur sera soumis à la vitesse à laquelle l'UAL va exécuter le calcul
- ▶ Résultat de l'instruction (I_1) nécessaire pour faire I_2 :
 - Diffère I_2 tant que le résultat non disponible
- ▶ Même ressource utilisée par I_1 et I_2
 - Diffère I_2 tant que ressources non libre
- ▶ Aléas de branchements
 - Si test...Sinon....
 - Pari sur le résultat du test
 - ▶ Vrai: exécution optimale
 - ▶ Faux: annule les opérations commencées depuis le pari

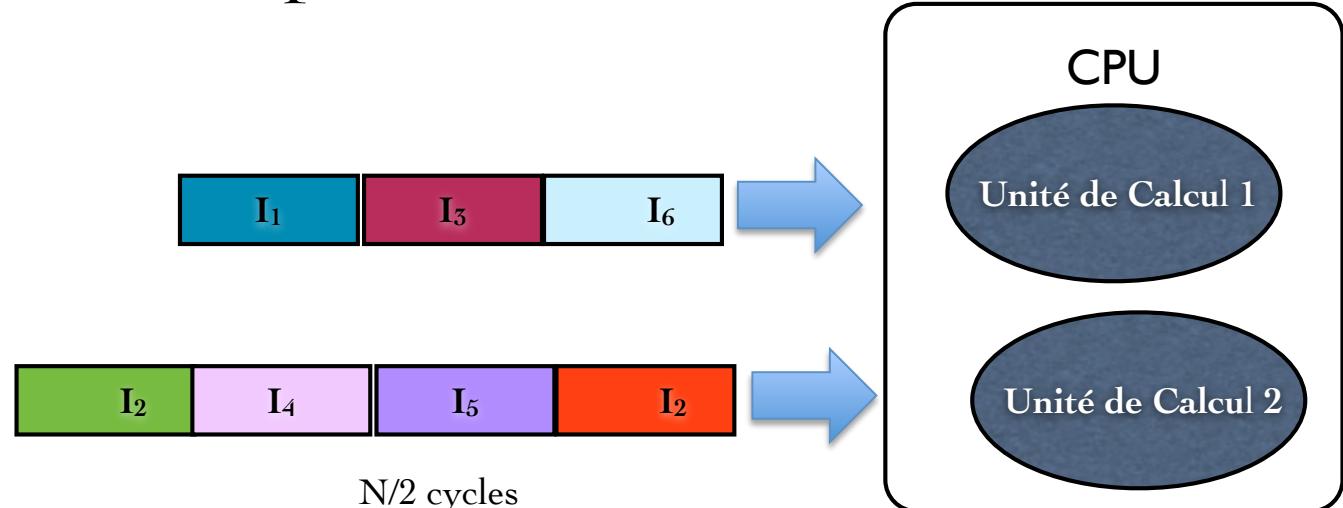
Architecture super-scalaire

- ▶ **Architecture scalaire:** une seule unité de calcul



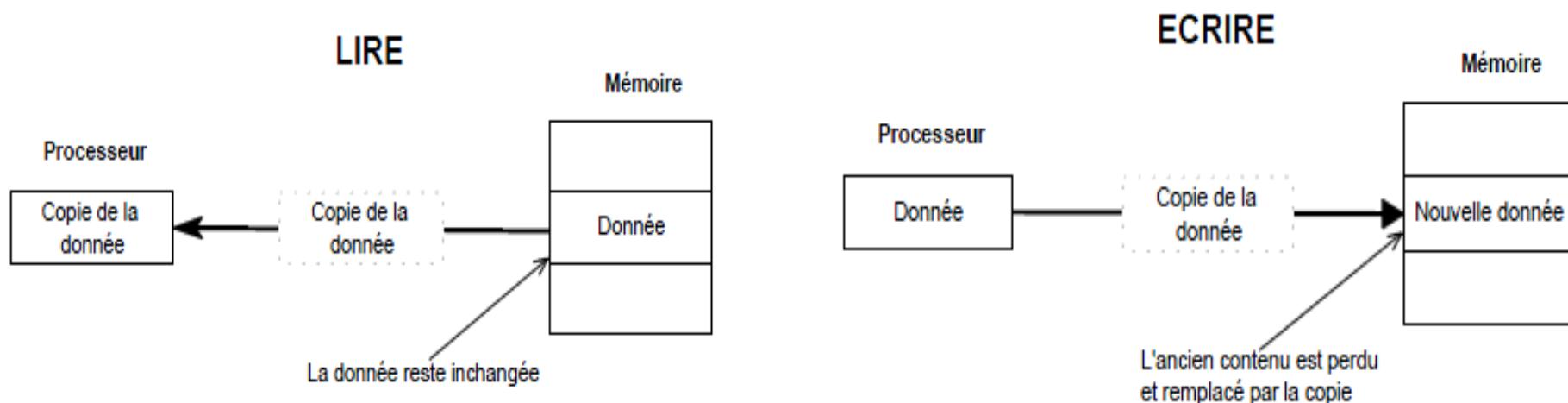
- ▶ **Architecture super-scalaire:** doter le CPU de plusieurs unités de calcul travaillant en parallèle

Ne résout pas le problème du calcul qui doit apporter un résultat pour la deuxième opération, mais celui de perte de temps dû à l'occupation de l'unité de calcul par une opération complexe



Les Mémoires Centrales: RAM, ROM (1)

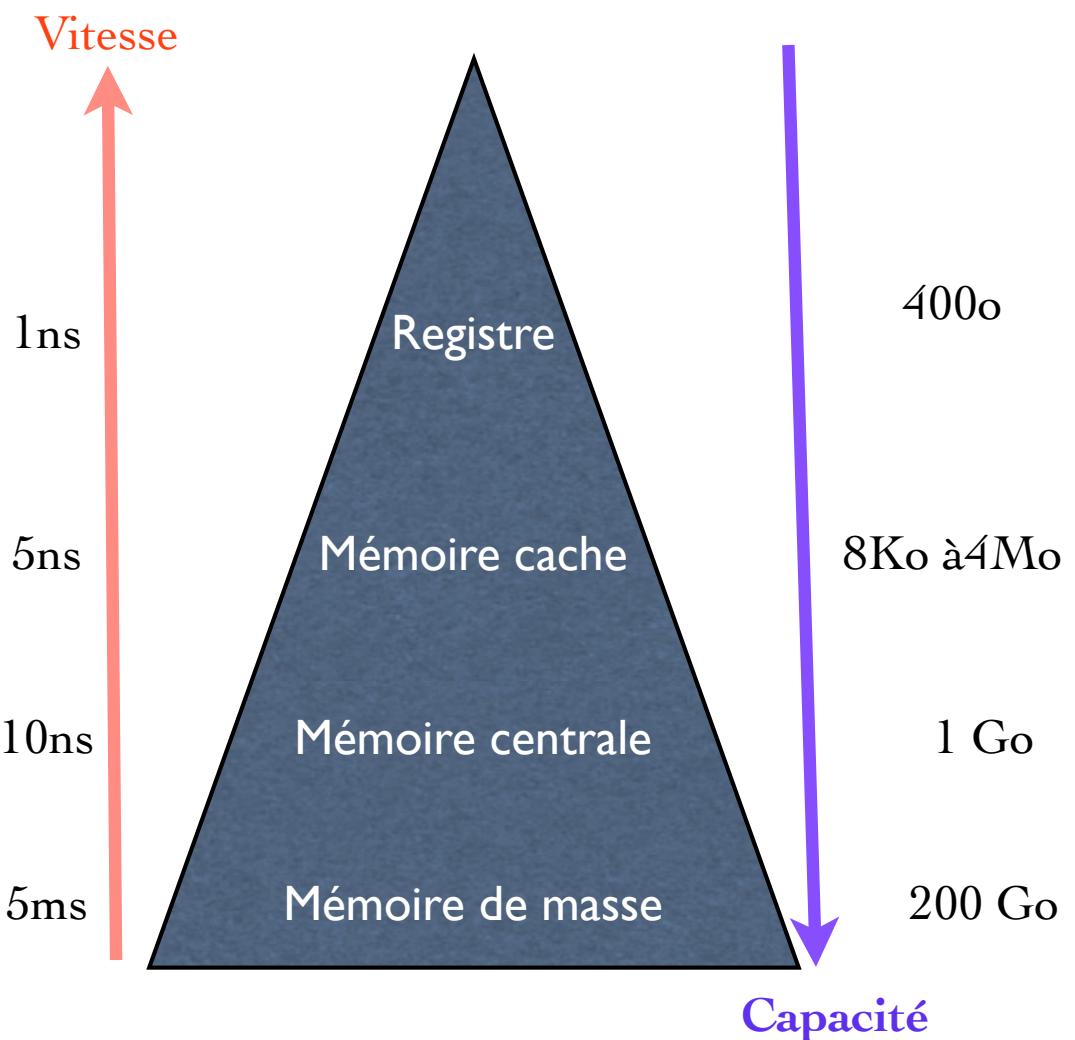
- ▶ **ROM (Read Only Memory):** L'ensemble de programmes et données qui restent à jamais dans la ROM Constituent le BIOS
- ▶ **RAM (Random Access Memory)**
 - Mémoire volatile
 - Processeur y accède en lecture/écriture



Les Mémoires Centrales: RAM, ROM (2)

► Hiérarchie Mémoire

- **Registre:** situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- **Cache:** destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- **Centrale:** contient les programmes (instructions et données)
- **Masse:** mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations.

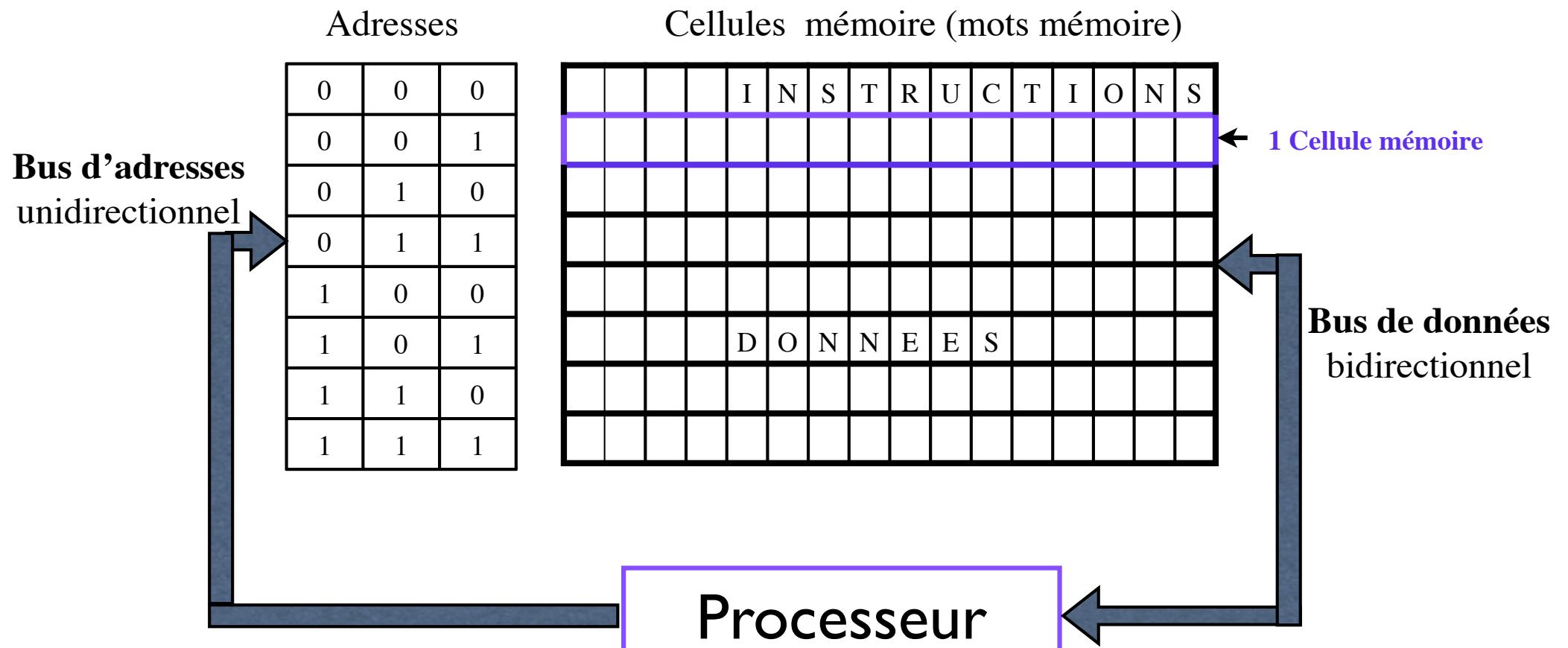


Chapitre 2

Chemin des données

Architecture Générale d'une Unité Centrale (1)

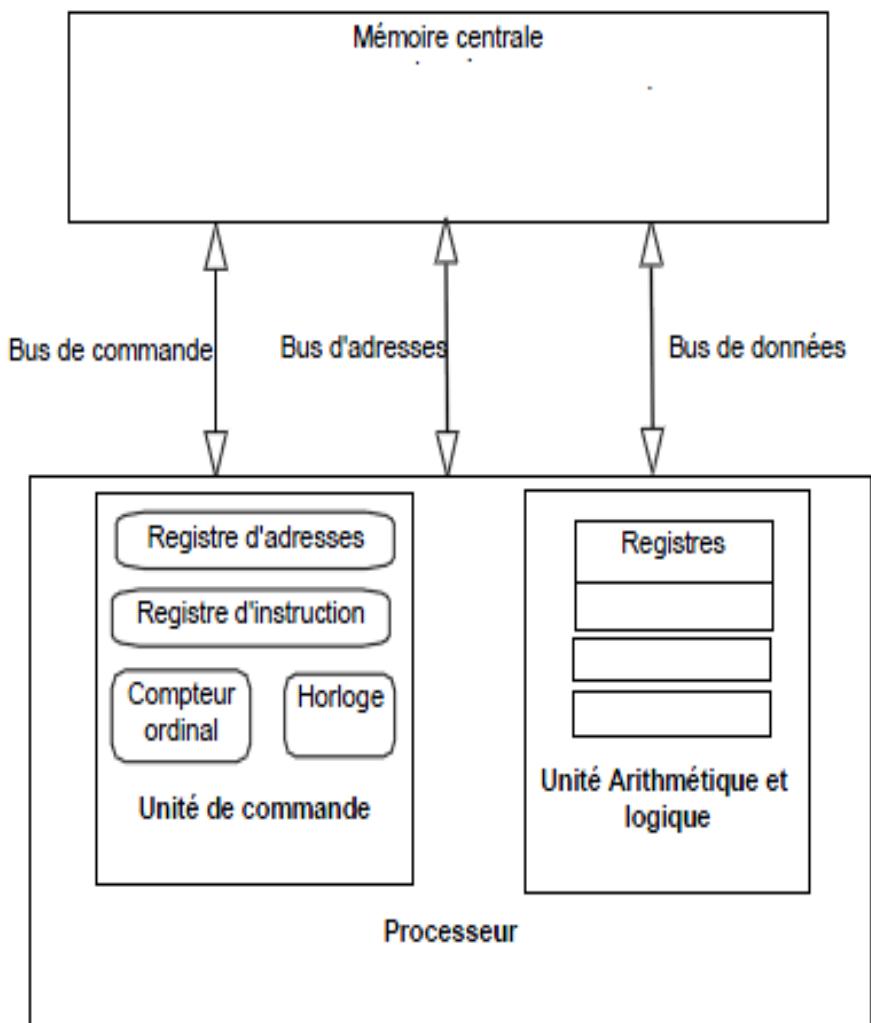
▶ Organisation physique de la RAM



Architecture Générale d'une Unité Centrale (2)

▶ Composants du processeur

- **Horloge:** ses battements cadencent le travail du processeur, l'accès à la mémoire etc., sa fréquence est exprimé en Méga ou giga Hertz
- **Compteur Ordinal:** registre qui contient toujours l'adresse de l'instruction à exécuter
- **Registre d'instruction:** contient l'instruction à exécuter qui y est rangée puis décodée par le décodeur d'instruction.
- **Registre d'adresse:** contient l'adresse d'une cellule à laquelle le processeur souhaite accéder
- **Registre de données de l'UAL:** contient les opérandes et les résultats issus des opérations



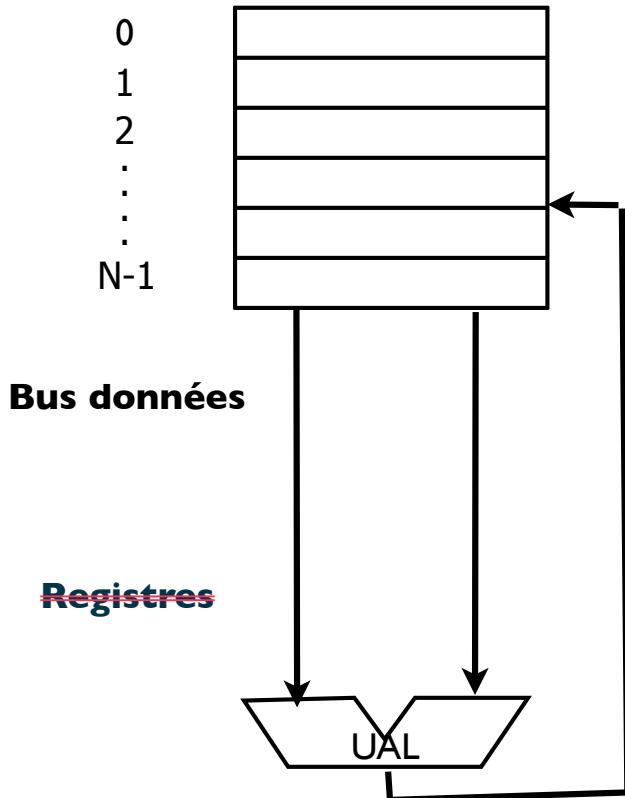
Modèles d'exécution ou architectures

- ▶ **Le modèle d'exécution définit le jeu d'instructions**
- Il correspond à l'organisation générale des échanges d'information entre processeur, registres et mémoire.
- **Principaux modèles d'exécution ou types d'architectures**

Architecture	Nb opérandes spécifié dans l'instruction	Nb opérandes en mémoire	Exemples
Mémoire-mémoire (3,3)	3	3	VAX
Mémoire-accumulateur (1,1) ou à accumulation	1	1	Machines anciennes
Mémoire-pile (0,0) ou à pile	0	0	Machines anciennes
Mémoire-registre (1,2)	2	1	Intel 80x86, Motorola 68000, TIC54x, IBM360
Registre-registre (0,3) ou à chargement / rangement	0	3	Alpha, ARM, MIPS, PowerPC, Sparc.

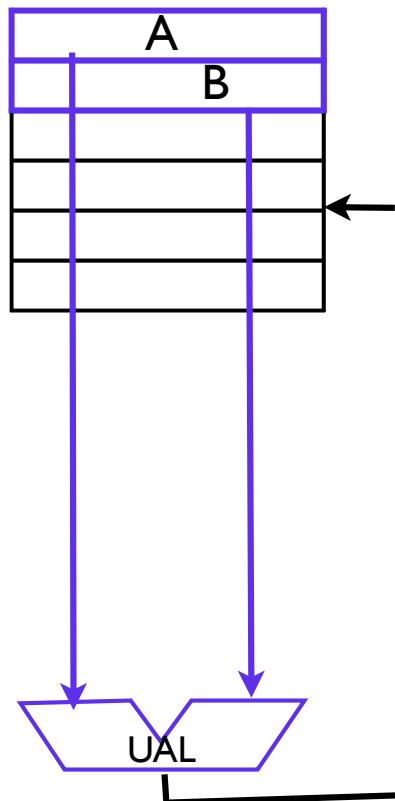
Modèle mémoire/mémoire

Adresses Mémoire (prog+données)

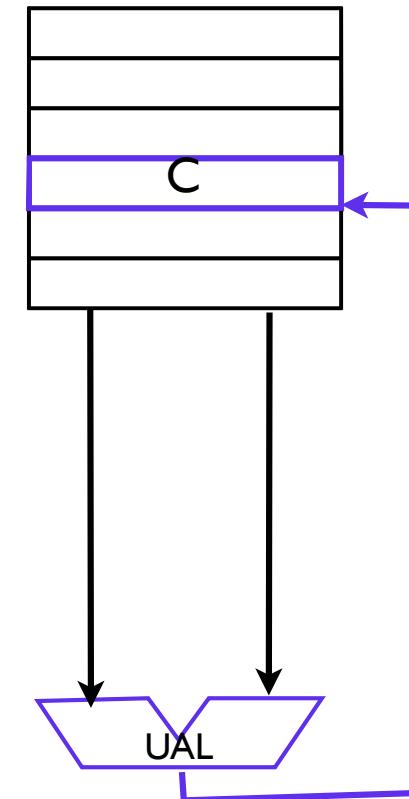


$$C = A + B$$

ADD @C @A @B



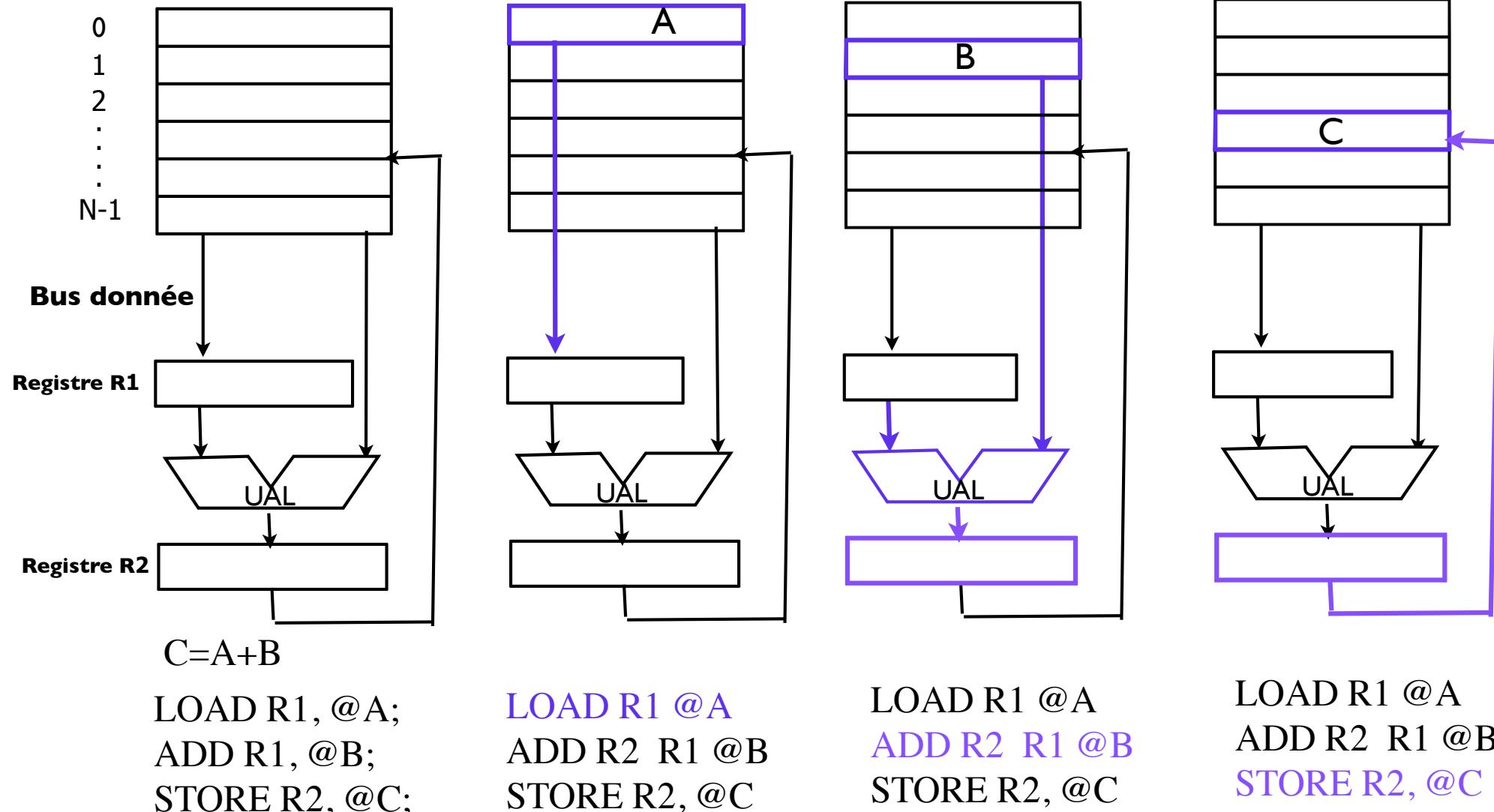
ADD @C @A @B



ADD @C @A @B

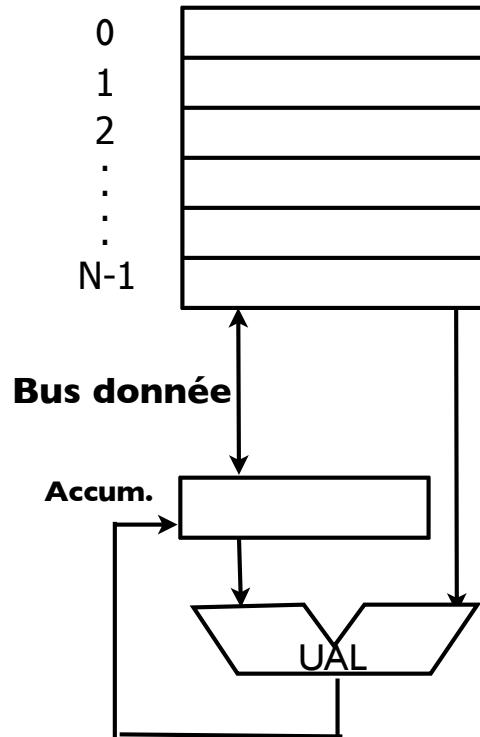
Modèle mémoire/registres

Adresses Mémoire (prog+données)



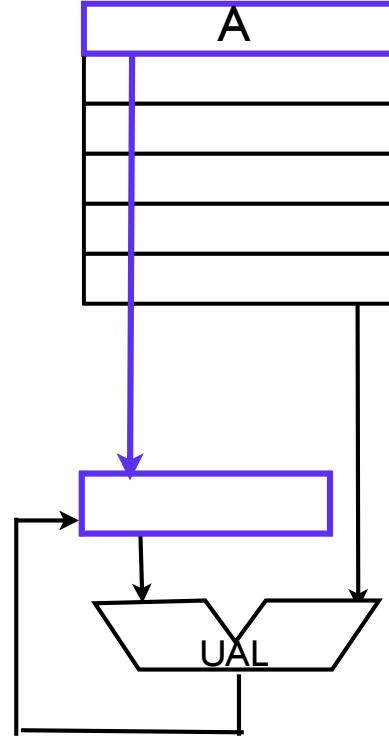
Modèle mémoire/accumulateur

Adresses Mémoire (prog+données)

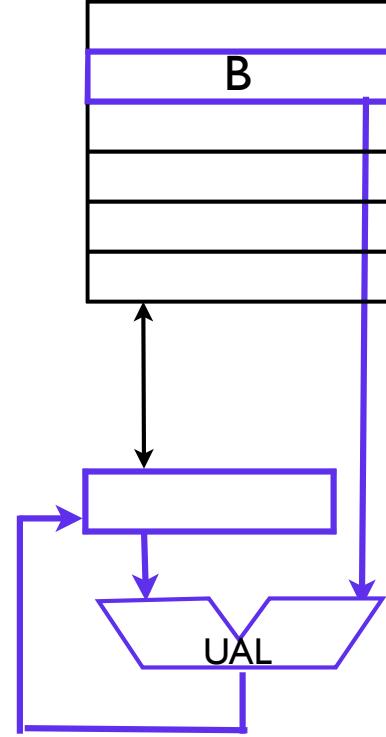


$$C = A + B$$

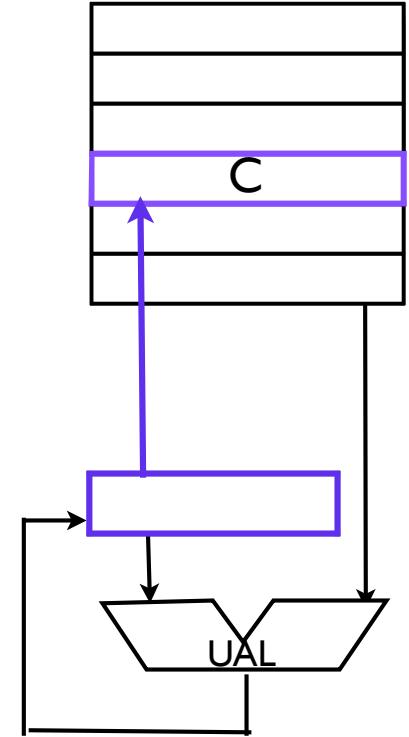
LOAD @A
ADD @B
STORE @C



LOAD @A
ADD @B
STORE @C



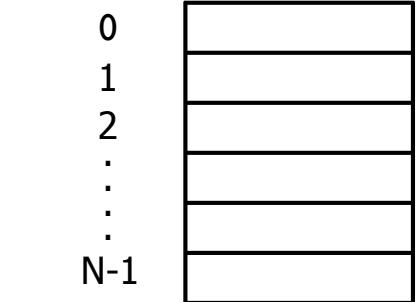
LOAD @A
ADD @B
STORE @C



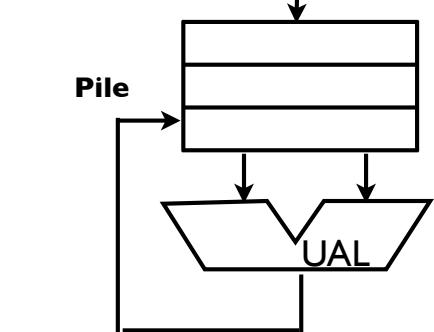
LOAD @A
ADD @B
STORE @C

Modèle pile

Adresses Mémoire (prog+données)



Bus donnée



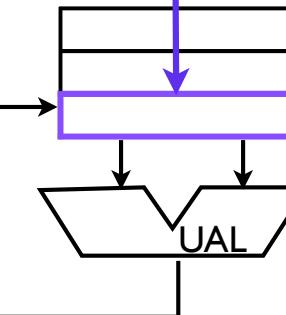
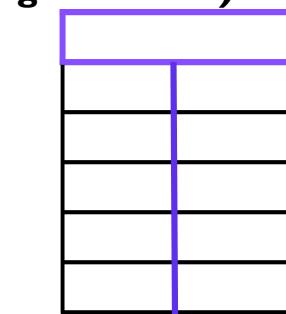
$C = A + B$

PUSH @A

PUSH @B

ADD

POP @C

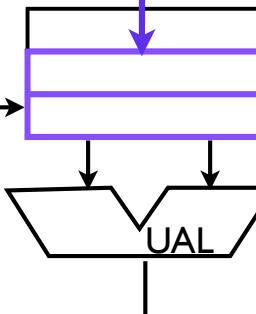
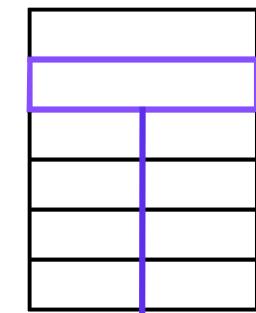


PUSH @A

PUSH @B

ADD

POP @C

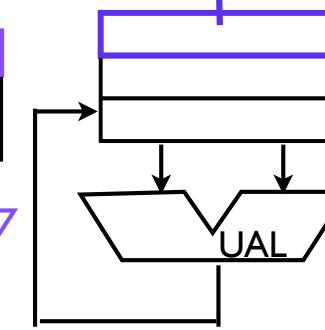
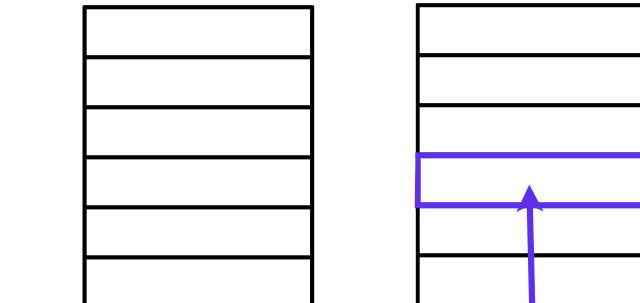
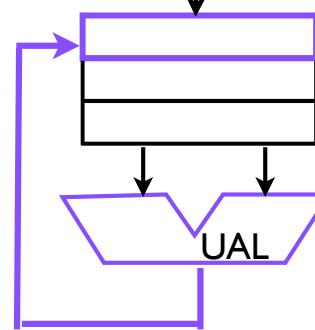
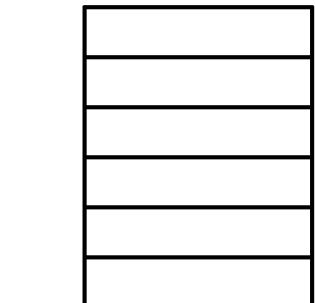
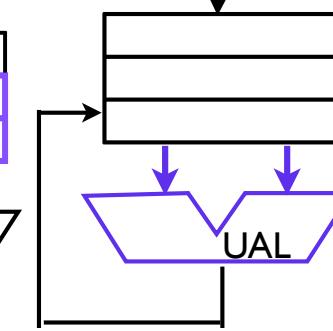
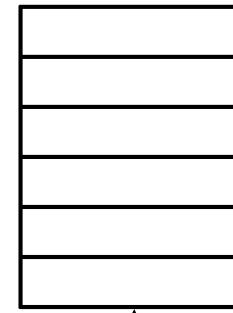


PUSH @A

PUSH @B

ADD

POP @C



PUSH @A

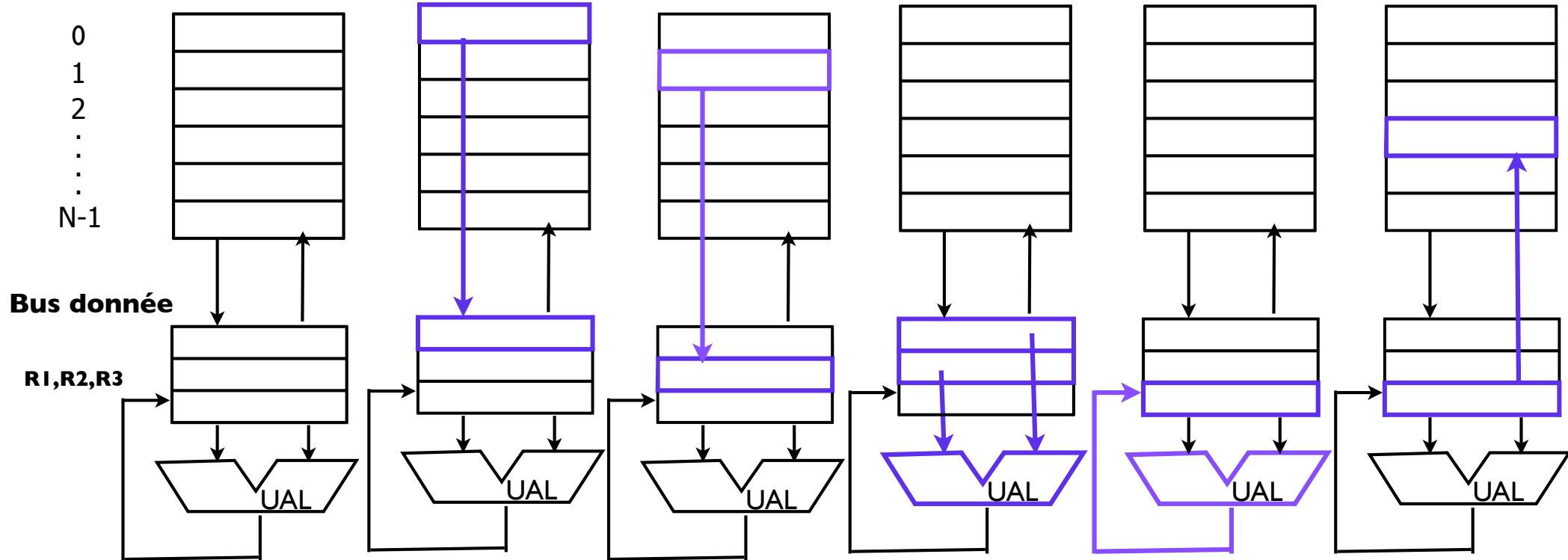
PUSH @B

ADD

POP @C

Modèle registre/registre

Adresses Mémoire (prog+données)



$$C = A + B$$

LOAD R1 @A
LOAD R2 @B
ADD R3 R1 R2
STORE R3 @C

LOAD R1 @A
LOAD R2 @B
ADD R3 R1 R2
STORE R3 @C

LOAD R1 @A
LOAD R2 @B
ADD R3 R1 R2
STORE R3 @C

LOAD R1 @A
LOAD R2 @B
ADD R3 R1 R2
STORE R3 @C

LOAD R1 @A
LOAD R2 @B
ADD R3 R1 R2
STORE R3 @C

LOAD R1 @A
LOAD R2 @B
ADD R3 R1 R2
STORE R3 @C

Cycle d'Exécution d'une Instruction: calcul de (a+b)

► Programme en langage évolué

```
/*variables globales*/  
static int a;  
static int b;  
static int c;  
/*fonction principale*/  
int main()  
{  
    a=40;  
    b=18;  
    c=a+b; /*l'étape que nous tenterons de montrer après chargement des opérandes en mémoire*/  
}
```

- Nous utiliserons l'architecture registre-registre
 - On écrit en mémoire du registre vers la mémoire ou en indiquant l'adresse à laquelle on veut écrire et les données à écrire
 - On lit en mémoire vers les registre
 - On peut également charger un registre à partir d'une valeur indiquée (dans une instruction par exemple...)
 - On effectue les opérations sur des données déjà disponibles en registres

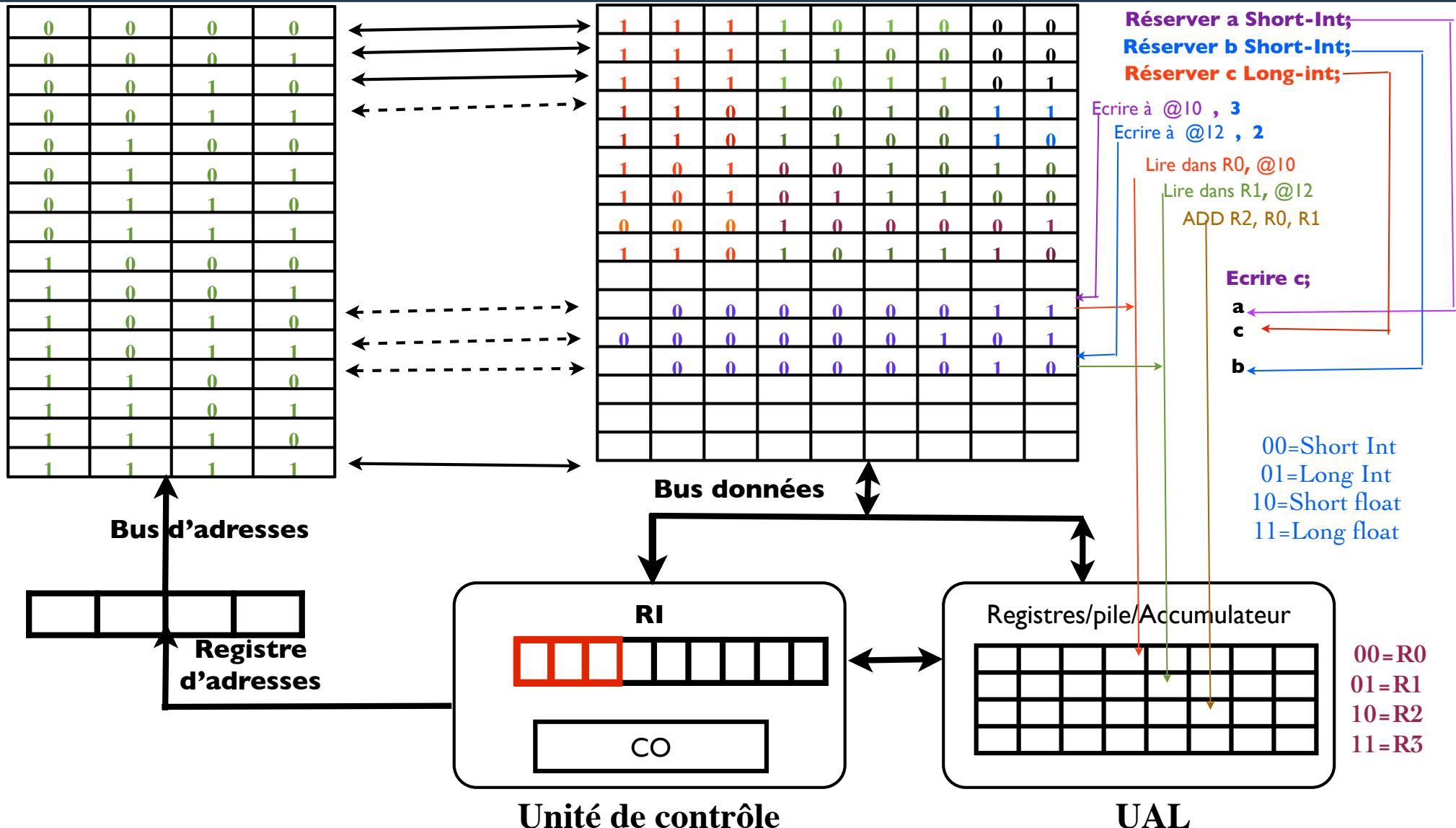
Cycle d'Exécution d'une Instruction: calcul de (a+b)

► (a+b): Code en langage machine

► Jeu d'instructions du processeur

- **Addition (000)**: additionne le contenu de 2 opérandes **logés dans 2 registres** (registre-registre) ou en mémoire (mémoire-mémoire) ou registre-mémoire ou pile-pile ou accumulateur et range **le résultat dans un registre** ou la mémoire ou la pile ou l'accumulateur.
- **Soustraction (001), Multiplication (010), Division (011)** ont la même approche que l'addition.
- **Chargement (100)**: permet de charger une valeur indiquée dans un registre
- **Lecture (101)**: elle copie le contenu d'une cellule mémoire et l'amène à travers le bus de données dans **un registre** ou la pile ou l'accumulateur de l'UAL.
- **Ecriture (110)**: copie dans une cellule mémoire, une valeur indiquée, ou le contenu d'**un registre** ou de la pile ou de l'accumulateur de l'UAL.
- **Réservation ou Déclaration (111)**: elle réserve une taille mémoire égale à la taille du type (type entier, type réel, type caractère etc..) déclaré.

Cycle d'Exécution d'une Instruction: calcul de (a+b)



Cycle d'Exécution d'une Instruction: calcul de (a+b)

▶ Programme en langage évolué

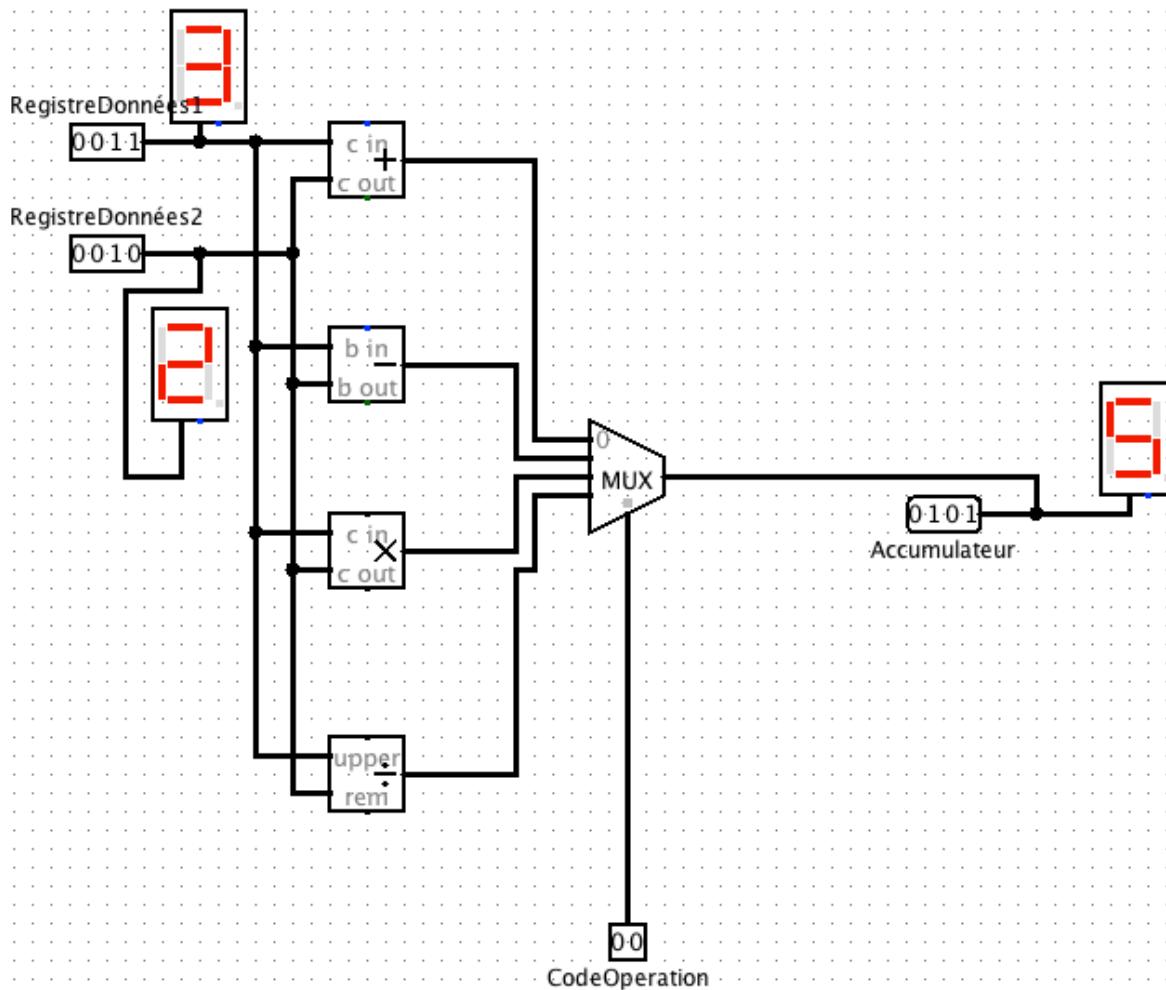
```
/*déclaration de variables
globales*/
static int a;
static int b;
static int c;
/*déclaration de la fonction
principale*/
int main()
{
a=40;
b=18;
c=a+b; // l'étape que nous
tenterons de montrer après
chargement des
```

▶ Simulation avec Logisim

- **UAL** qui implémnte une addition, soustraction, multiplication et division
- **Un banc de registres** de 4 registres
- **Une UCT** composée de l'UAL est du banc de registres
- **Une mémoire** qui contient déjà le programme à exécuter
- **Un format et jeux d'instructions** composé des 4 opérations de l'UAL et une opération de chargement.

Cycle d'Exécution d'une Instruction: calcul de (a+b)

► UAL

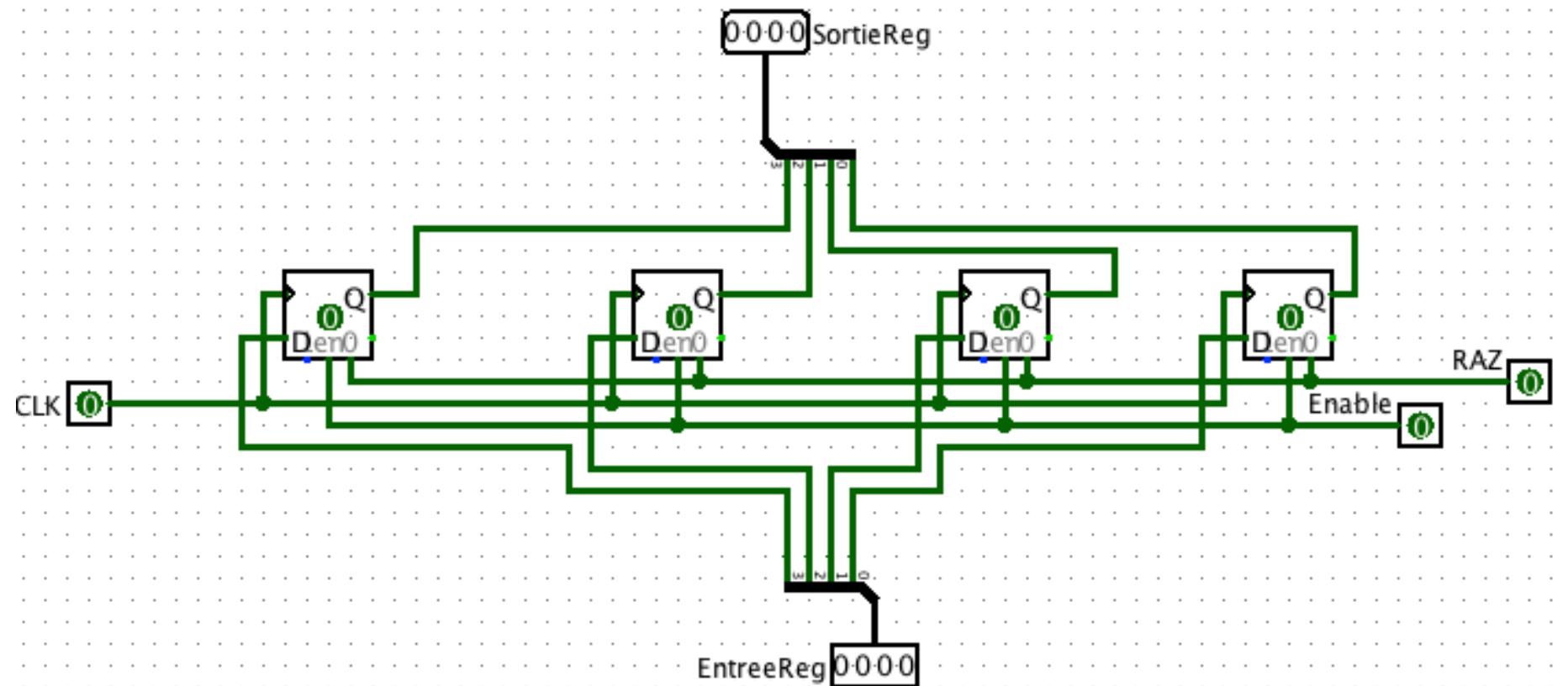


un MUX de 4 bits de données et 2 bits de sélection permet de sélectionner l'entrée de données à rediriger vers la sortie du MUX

00=Addition,
01=Soustraction,
10=Multiplication et
11=Division

Cycle d'Exécution d'une Instruction: calcul de (a+b)

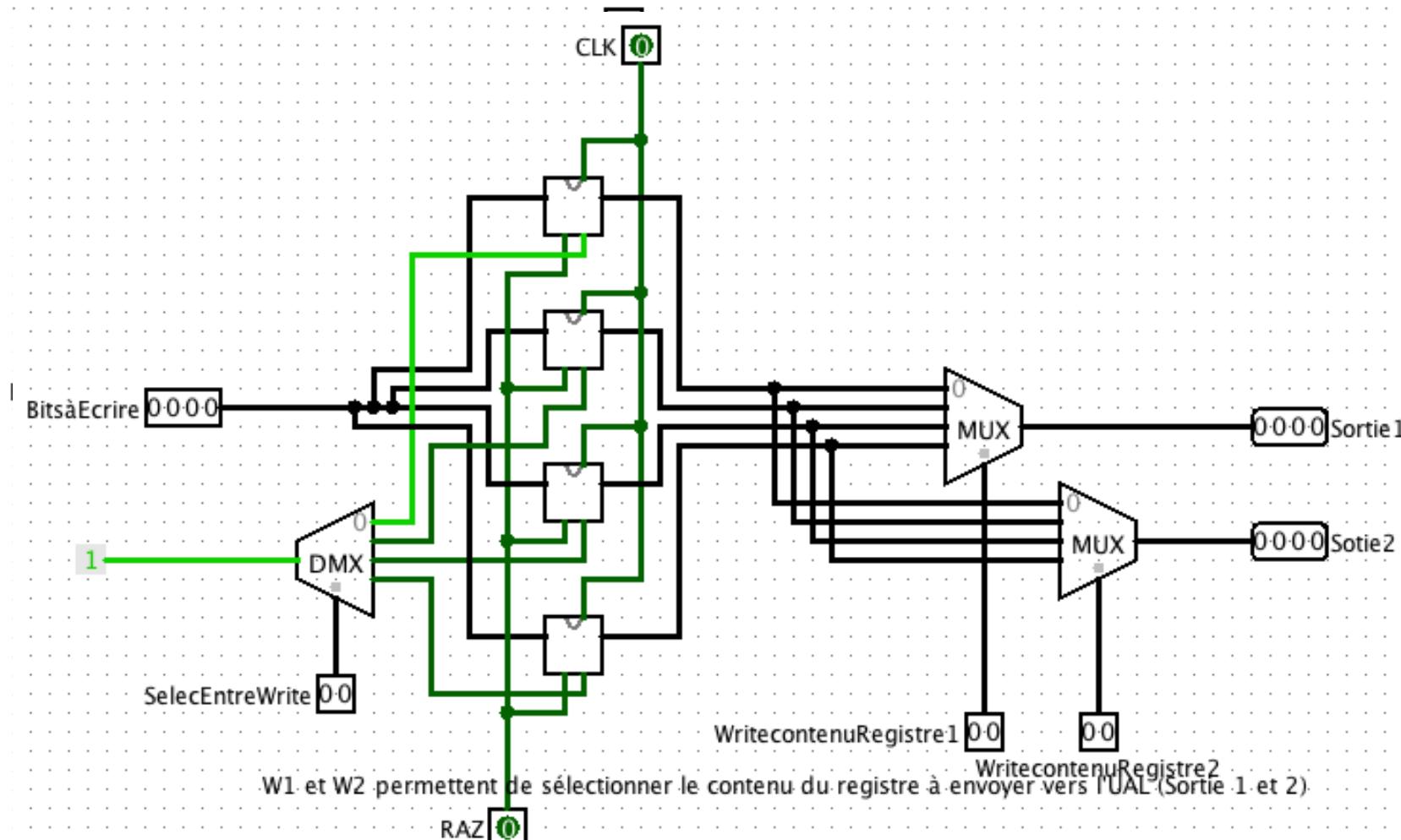
► BANC DE REGISTRES



Exemple du registre qui sera utilisé dans le banc de registre

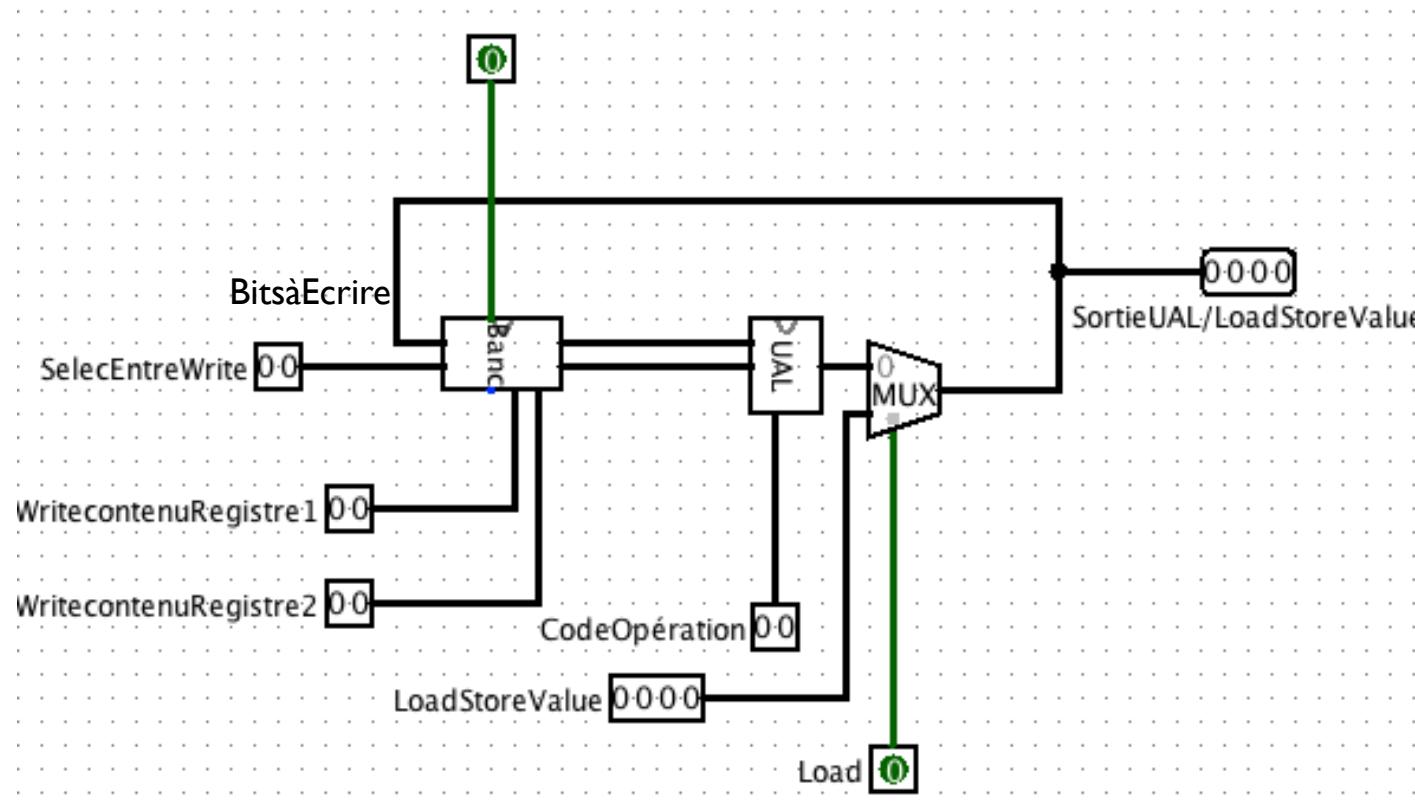
Cycle d'Exécution d'une Instruction: calcul de (a+b)

► BANC DE REGISTRES



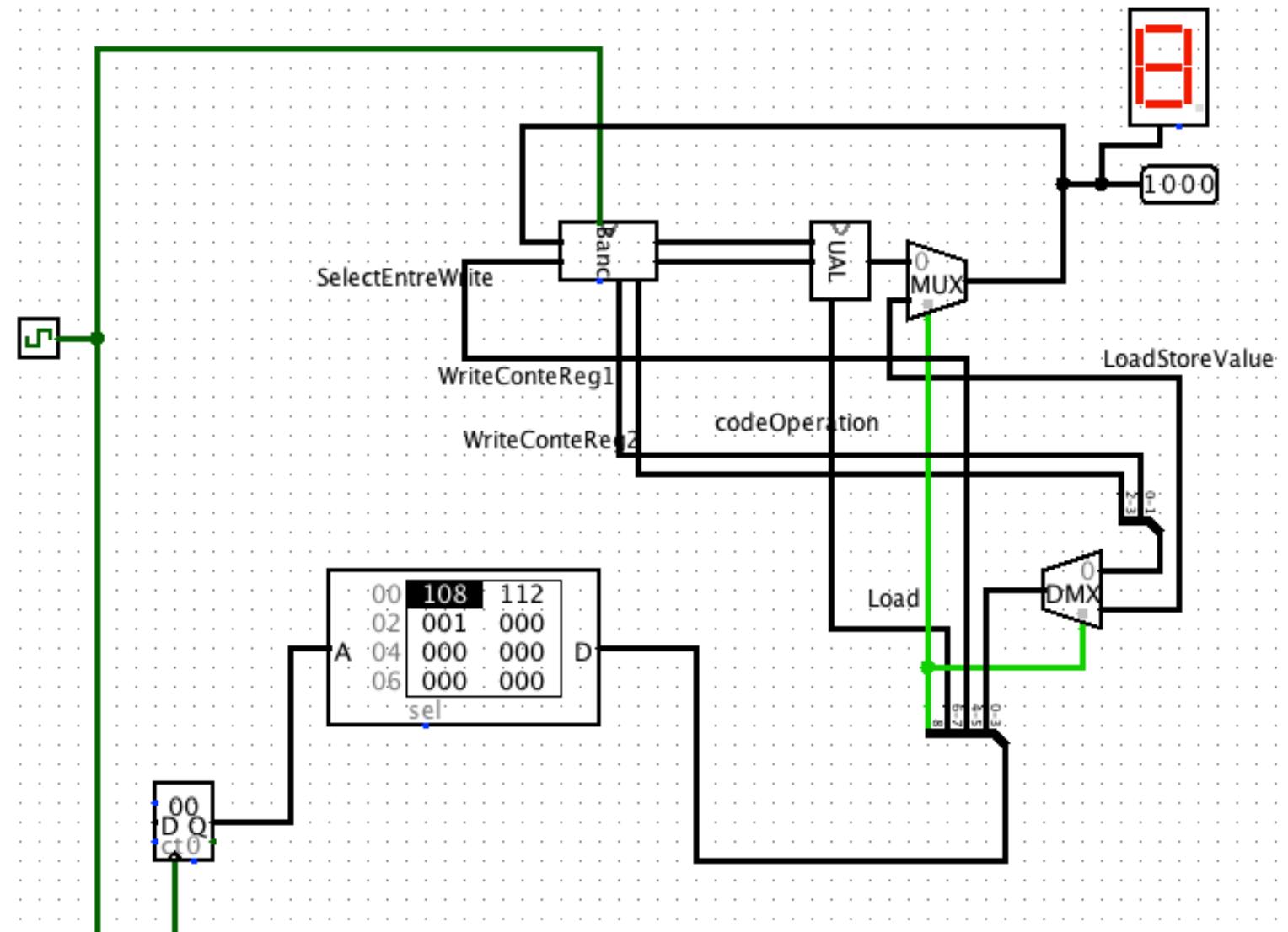
Cycle d'Exécution d'une Instruction: calcul de (a+b)

► Interconnexion Banc de registre et UAL



Cycle d'Exécution d'une Instruction: calcul de (a+b)

► UCT



Cycle d'Exécution d'une Instruction: calcul de (a+b)

- ▶ **Format et jeux d'instructions**
- ▶ **Partie Code Instruction**
- ▶ **Composition: 4 instructions arithmétiques (+, -, x, ÷) et une instruction de chargement LDR (Load).**
 - **La partie code instruction est composée de 3 bits**
 - Load sur 1 bit et le code des opérations arithmétiques sur 2 bits. ainsi on a comme code instruction:
 - **Addition 00 et Load doit être à 0 = 000**
 - **Soustraction 01 et Load doit être à 0 = 001**
 - **Multiplication 10 et Load doit être à 0 = 010**
 - **Division 11 et Load doit être à 0 = 011**
 - **On a une instruction de chargement LDR où Load doit être à 1=100**

Cycle d'Exécution d'une Instruction: calcul de (a+b)

► Format et jeux d'instructions

► Partie Code opérande

- **Composition:** 1 Registre Résultat et 2 Registres opérandes ou une valeur ou constante sur 4 bits.
 - **Registre Résultat:** codé sur 2 bits car le banc compte 4 registres
 - **Chaque registre opérande:** codé sur 2 bits car le banc compte 4 registres
 - **Une valeur:** sur 4 bits puisque les registres opérandes sont sur 4 bits, ce qui nécessite load à 1 pour être traité). Cela permettra également d'avoir des instructions de même longueur

Cycle d'Exécution d'une Instruction: calcul de (a+b)

► Format et jeux d'instructions en assembleur

- Exemple de l'instruction **ADD**

- En “Langage Assembeur”: **ADD R1 R1 R2**// **ADD** (additionner 000) **R1**(ranger le résultat dans R1, registre codé par 00), **R1**(contenu de R1, opérande 1 codé par 00) **R2** (contenu de R2, opérande 2, codé par 01)

- En langage Binaire: 000 00 00 01

- Exemple de l'instruction **LDR**

- **LDR R1 4** // **LDR** (charger 100) **R1** (ranger la valeur à charger dans R1), **4** (valeur à charger dans R1 comme seule opérande code sur 4 bits).

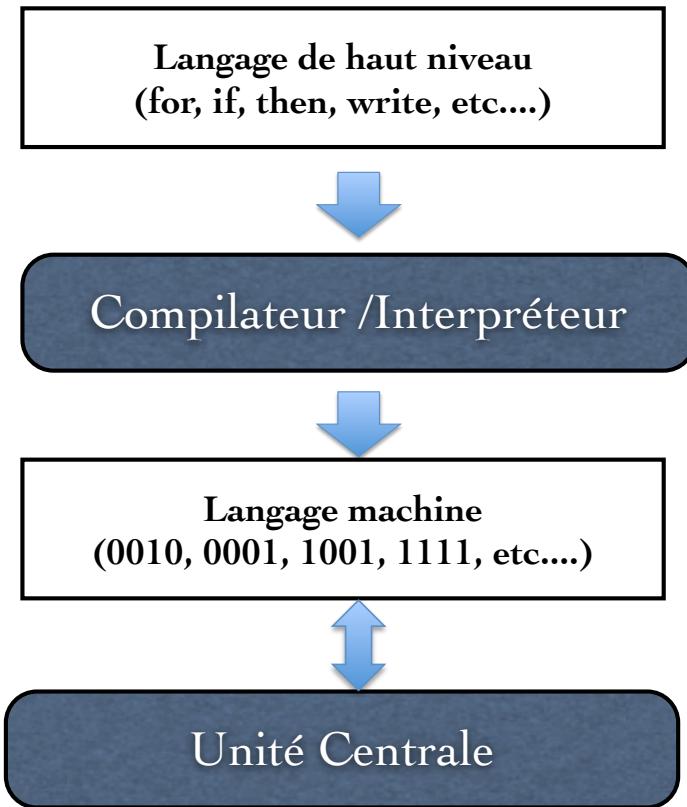
- En langage Binaire: 100 00 0100

Cycle d'Exécution d'une Instruction: calcul de (a+b)

- ▶ Exemple: a=8 et b=2
 - ▶ On suppose que le programme est déjà en mémoire et que les instructions d'affectation a=8 et b=2 déjà faites
 - Opérations Assembleur
 - ▶ 1-Charger l'opérande1(8) dans R1 1-LDR R1 8
 - ▶ 2-Charger l'opérande2 (2) dans R2 2- LDR R2 2
 - ▶ 3- Additionner R1 (opérande 1) avec R2 (opérande 2) 3- ADD R3 R1 R2 et ranger le résultat dans R3 (opérande 3)
 - Binaire
 - ▶ 1-100 00 1000
 - ▶ 2-100 01 0010
 - ▶ 3-000 10 00 01
 - Hexadécimal (à charger dans Logisim)
 - ▶ 1-108
 - ▶ 2-112 NB: les instructions sont codées sur 9 bits
 - ▶ 3-021 Nous avons choisi un adressage sur 9 bits

La chaîne de production d'un Programme (1)

- ▶ **Interpréteur:** traduit le programme instruction par instruction en langage machine
- ▶ **Compilateur:** traduit tout le programme globale en langage machine



Le travail du **compilateur** se divise en plusieurs phases :

- **l'analyse lexicale** (reconnaissance des mots du langage, c'est-à-dire appréhension du vocabulaire) ;
- **l'analyse syntaxique** (vérification de la syntaxe, c'est-à-dire appréhension de la grammaire) ;
- **l'analyse sémantique** (vérification de la sémantique, c'est-à-dire appréhension du sens) ;
- **l'optimisation et la génération** du code objet.

La chaîne de production d'un Programme (2)

► Hiérarchie de traduction

```
{
    in a =8, b=4, c;
    c = a+b;
}
```

prog.c

gcc -o prog prog.c

Compilateur

prog.o

Bibliothèque

LD A, (F800h)
ADD A, (F810h)
st (F820h), A

RAM

08	F800
04	F810
---	F820
3AF800	FB00
C6F810	FB01
32F820	FB02

Exécutable sur Disque.D

08
04

3AF800
C6F810
32F820

Chargeur

prog.exe

Assembleur