



Architecture des systèmes à microprocesseur

Informatique embarquée



Séance 1

- Séance 1 : Processeurs ?
- Séances 2 : Entrées-sorties génériques – GPIO
 - GPIO
- Séance 3 : Interruptions
- Séance 4 : Périphériques de conversion
 - CAN
 - CNA
- Séances 5 : BUS de communication synchrone et asynchrone
 - UART
 - I2C
 - SPI



☐ Séances de TP (18h- 6x3h)

- TP 1 : Introduction à l'environnement de développement Keil μ vision
- TP 2 : Interruptions / Scrutation
- TP 3 : GPIO
- TP 4 : ADC
- TP 5 : Timer
- TP 6 : Liaison Série - DMA

- Mini-projet (12h-4x3h)



- Règles du jeu
 - Notes de DS
 - Notes de TP
 - Note de TP Contrôle



❑ Les documents

- Les documents papiers seront distribués en temps utile pendant les séances
 - Diapositives feront office de poly et seront distribués en CM
 - Textes de TP
- Documents fortement conseillés pour les séances de TP et projet

❑ Les sources

- Site de STMicroelectronics (www.st.com)
- Site de ARM (www.arm.com)
- Documentations annexes :
 - Getting started with STM32 Nucleo board software development tool
 - Mastering STM32 de Carmine Noviello



- Capable de mettre en œuvre une grande partie des périphériques internes/externes nécessaires à un objet connecté,
- Comprendre l'architecture et fonctionnement d'un microprocesseur, son organisation mémoire, les mécanismes indispensables d'interruption, ...
- Savoir utiliser l'environnement de développement Keil uVision (compilation, debug,...) pour concevoir/tester/valider votre code



1. Historique : Evolution des machines de calcul
2. Taxonomie d'un processeur : éléments d'architecture
3. Famille de processeur
4. STM32
5. Outils

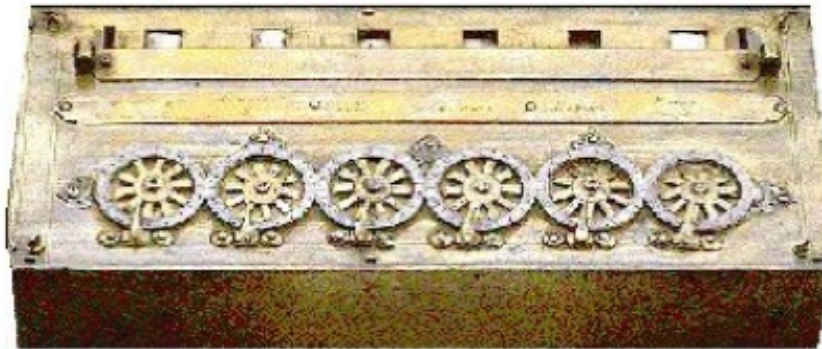


□ Les premiers calculateurs (1642-1945)

- Mécaniques

- **Blaise Pascal (1623-1662)** : Première calculatrice mécanique (1642 à 19ans)

- Addition, soustraction



La pascaline

- **Leibnitz (1646-1716)**

IUT GEII - Université de Cergy-Pontoise ➤ Ajout de l'opération de multiplication et de division (1670) O.Romain & J.Lorandel affari & S.Zuckerman - 2019

□ Les premiers calculateurs (1642-1945)

-Charles Babbage (1792-1871)

- Machine analytique (1840):
 - . Lecture de cartes perforées
 - . Différents algorithmes exécutables !!



Charles Babbage

-Ada Lovelace (1815-1852)

- Développeuse du premier programme informatique destiné à être exécuté par une machine (1843)
- Langage d'assemblage



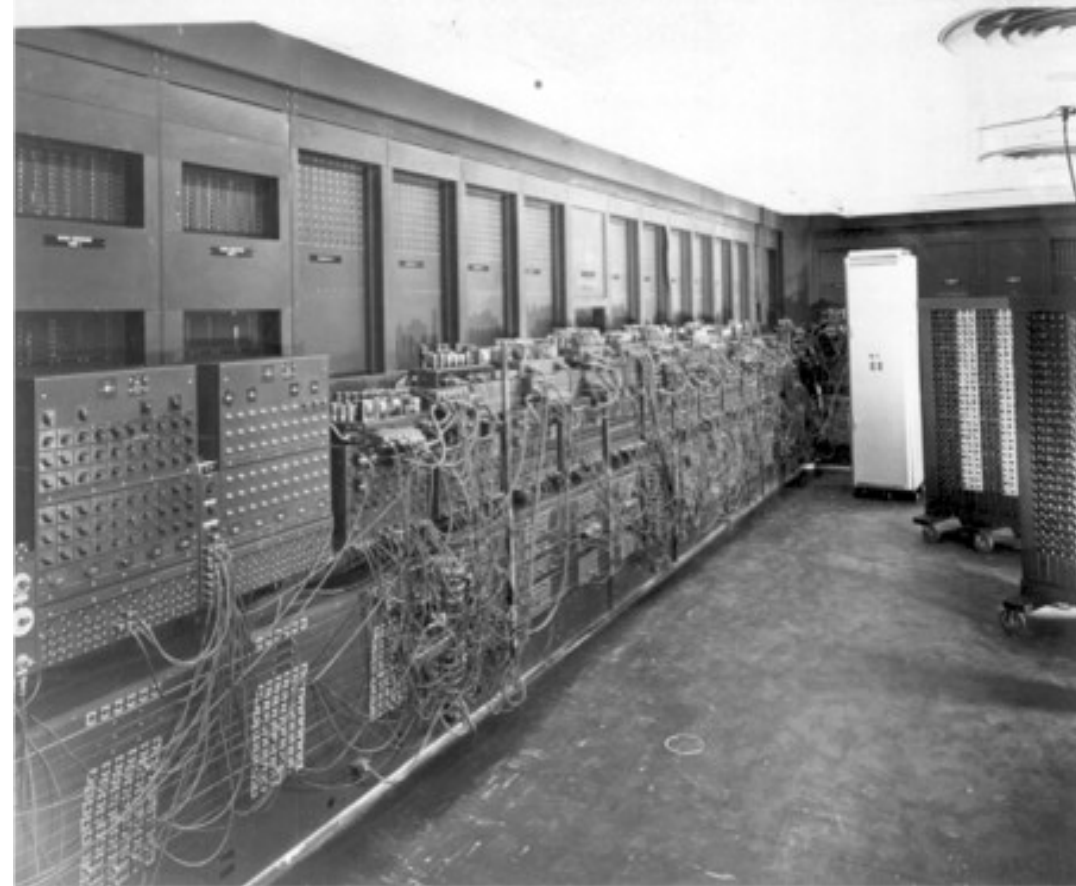
Première Machine analytique (1840)

1. Historique

- ❑ Le premier ordinateur : **Eniac** (1946)
 - Développé par Mauchly et Eckert
 - Projet de l'US Army (**Participant Von Neumann**)
 - Premier ordinateur électronique pouvant être reprogrammé par branchement
 - Vendue 500k\$ à 1 unité
 - 1 cycle = 200 ms soit 5 cycles/s

En Quelques Chiffres

- 30m de long, 2.5m de haut, 18k tubes à vide,
- 1500 relais, 30 tonnes, 140kW,
- 20 registres de 10 chiffres décimaux,
- 6000 commutateurs pour la programmation



O.Romain & J.Lorandel
Màj: F.Ghaffari & S.Zuckerman - 2019

□ Von Neumann (1903-1957)

- Mathématicien célèbre
- il s'est greffé au projet ENIAC après 1946



➤ Son idée :

- Représenter un programme sous sa forme numérique et le ranger en mémoire comme les données (création d'un jeu d'instructions machines)
- Introduction à l'arithmétique binaire
- Ancêtre à la base des architectures actuelles

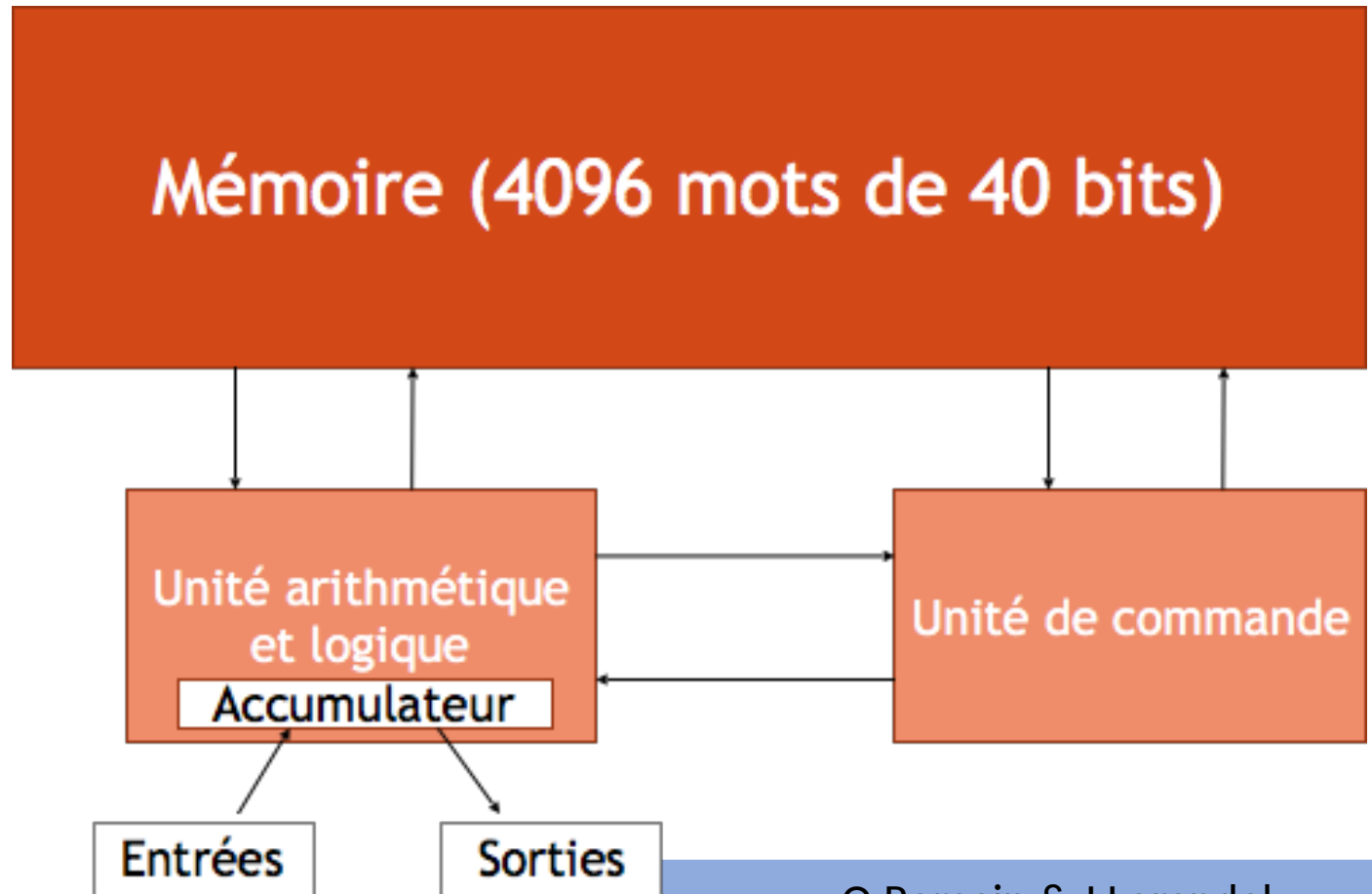
➤ Résultat :

- l'IAS (1952) : 1^{er} ordinateur à programme enregistré

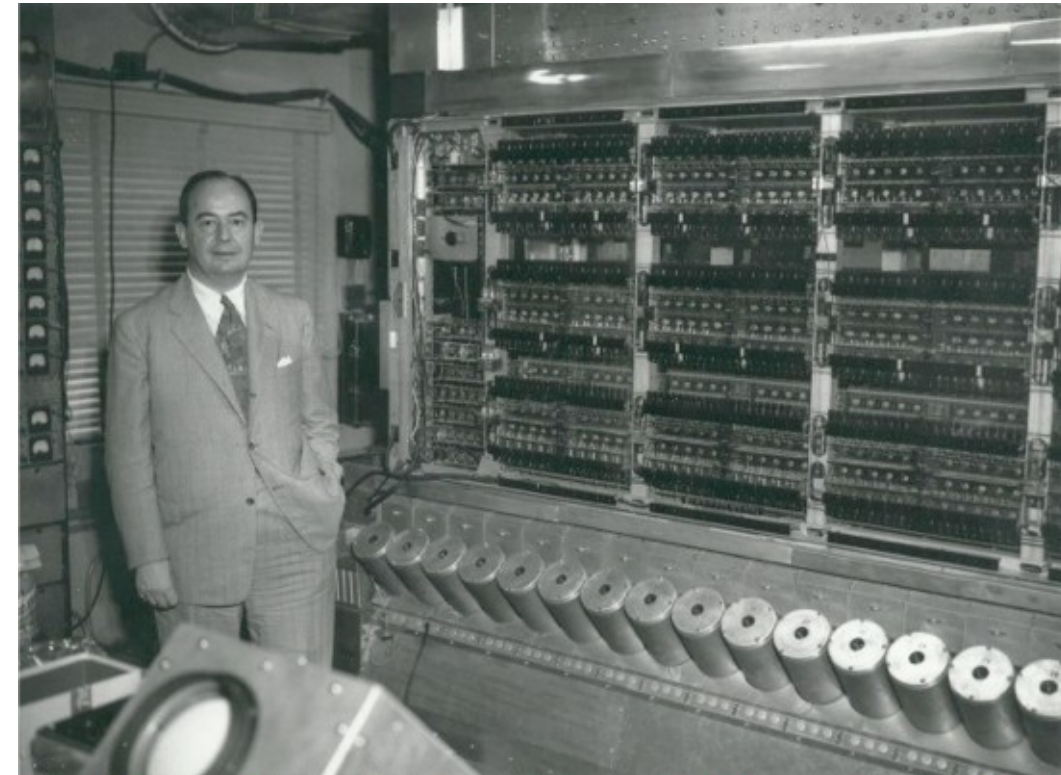
□ Schéma de l'architecture

- 5 parties :

.L'UAL,
.L'unité de
commande,
.La mémoire
.Les entrées
.Les sorties



- La machine de Von Neumann (1903-1957)
- *Von Neumann's Bottleneck*
 - Congestion des communications entre le processeur et sa mémoire
 - John Backus en 1977 dans son *ACM Turing award lecture*, évoque les prémices de **nouvelles architectures plus distribuées** (inventeur du langage de programmation FORTRAN)



O.Romain & J.Lorandel
Màj: F.Ghaffari & S.Zuckerman - 2019

□ Les transistors (1955-1965)

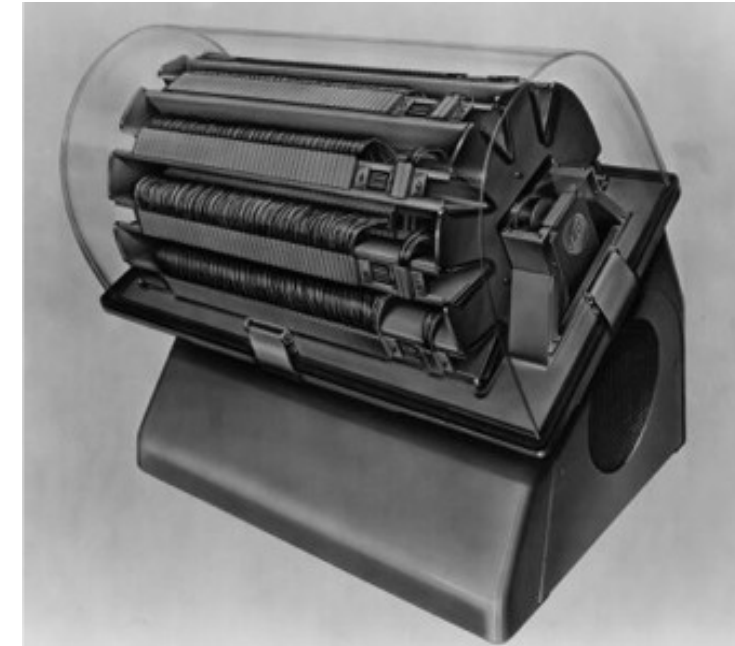
- Inventés par Schockley, Bardeen, Brattain (1947)
 - Interrupteurs commandés électroniquement
 - Beaucoup moins encombrant que les tubes à vide
- Débuts des grandes firmes, domination forte d'IBM
- Apparition des systèmes d'exploitation (Assembleur/Fortran)
- Exemple d'architecture
 - IBM 650

2ème ordinateur d'IBM

Tambour de 2000 mots pour stocker les instructions
60 mots de mémoire à ferrite

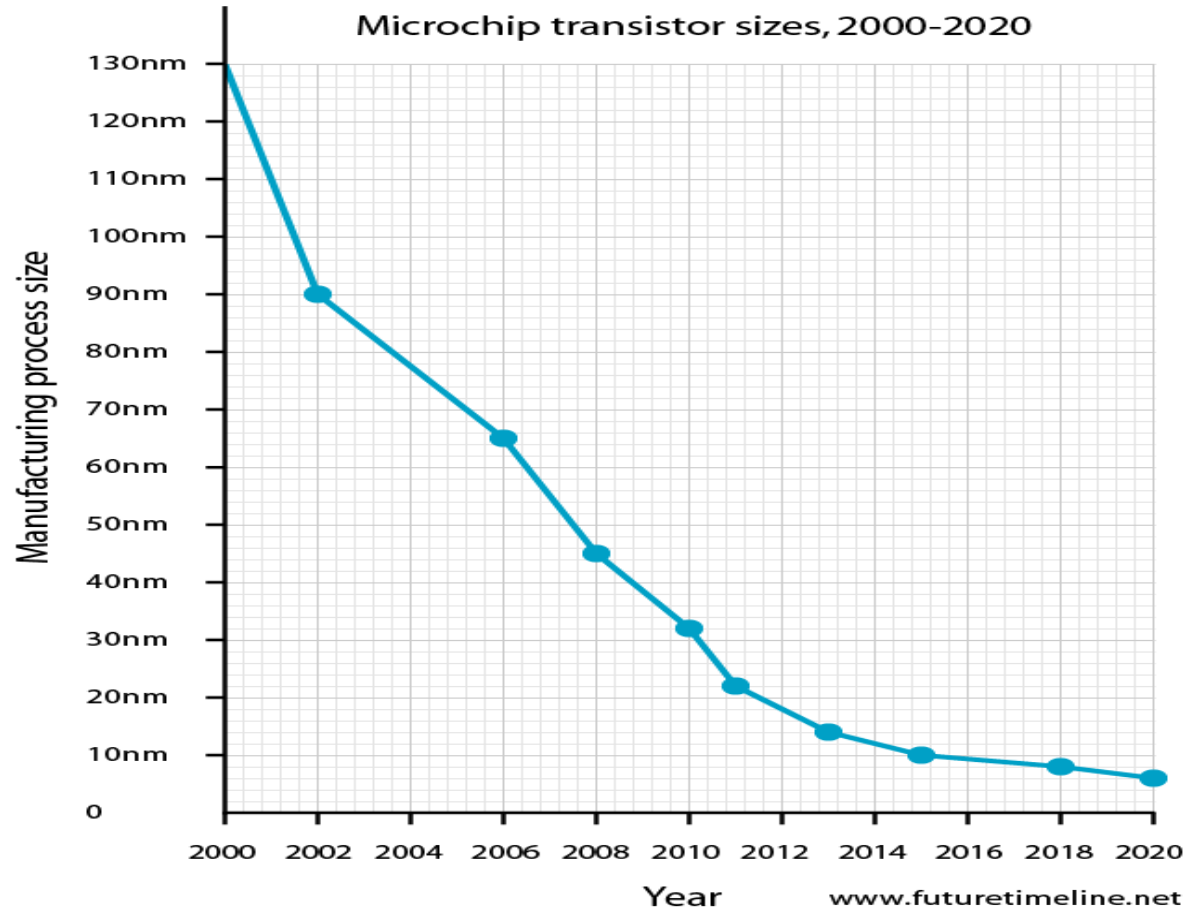
Addition : 1,63ms

Multiplication : 12,96ms

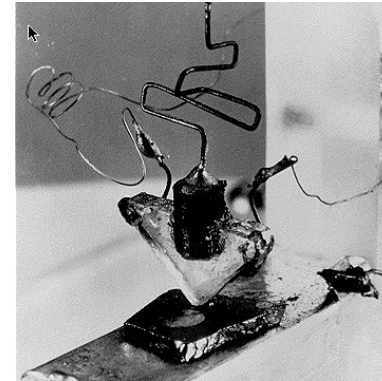


O.Romain & J.Lorandel
Màj: F.Ghaffari & S.Zuckerman - 2019

Evolution des transistors

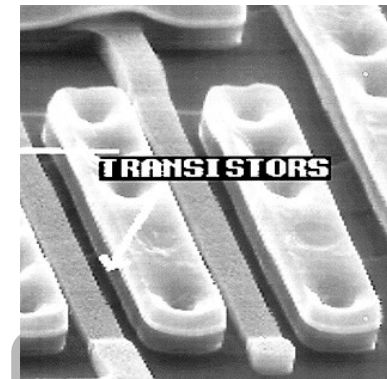


1947



Quelques cm

2014



Quelques nm

1. Historique

Le Circuit Intégré (1965-1980)

- 1961 : Brevets d'invention du circuit intégré par Kilby et Noyce
 - Consommation et encombrement plus faibles - gains en performances
- 1968 : Naissance d'Intel Corporation par Noyce, Moore et Rock
- 1971 : Premier microprocesseur (Intel 4004)
 - Toutes les fonctions essentielles d'un ordinateur sur un même circuit intégré



En Quelques Chiffres

- Microprocesseur 4 bits,
- 2300 transistors
- 108 kHz,
- 60 000 instructions / s

- Le nombre de transistors par unité de surface double tous les 18 mois

□ Dans les années qui suivirent

- Intel 8008 : processeur 8 bits (1973)
- Intel 8086 : processeur 16 bits (1978)
- ...

Augmentation de la
densité d'intégration

□ Les VLSI (1980-maintenant)

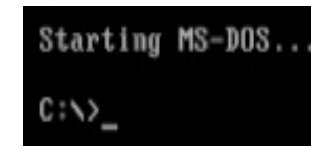
- Dizaines de milliers de transistors -> millions de transistors sur une même puce
- Passage du centre de calcul à l'ordinateur personnel :



1977 Apple 2
Millions d'exemplaires
(Steve Jobs)



1981 IBM PC
-Intel 8088, MS-DOS



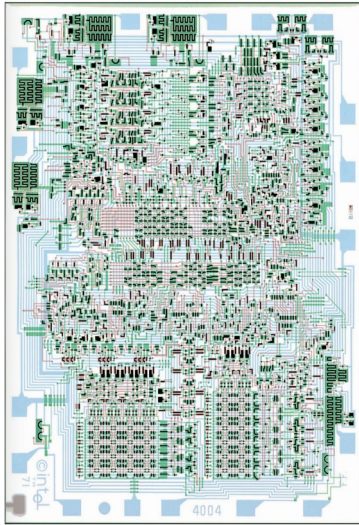
O.Romain & J.Lorandel
Màj: F.Ghaffari & S.Zuckerman - 2019

1. Historique



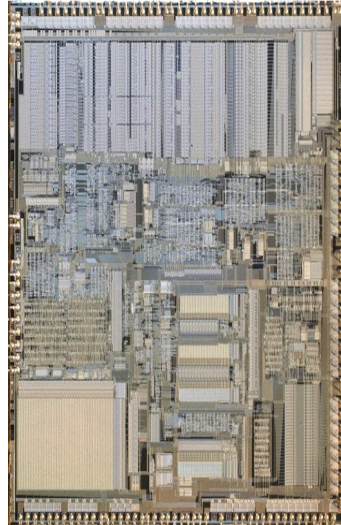
Augmentation de la densité d'intégration

1970's
8 bits uP



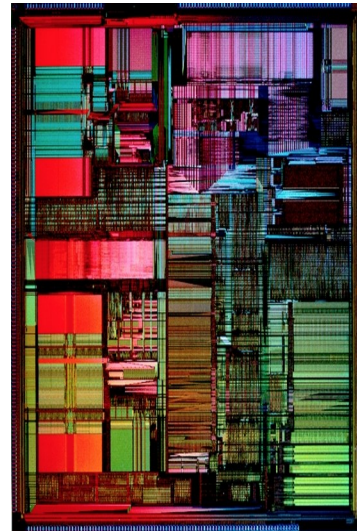
Intel 4004
2300 transistors

1980's
32 bits uP



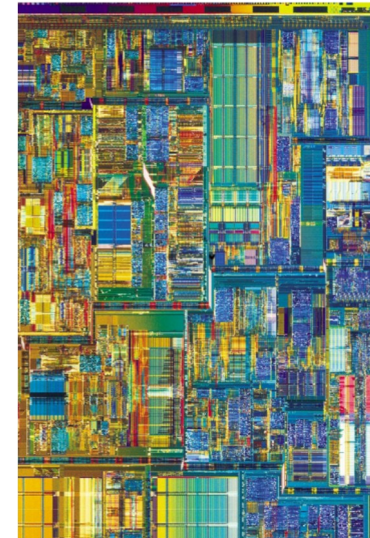
Intel 80386
275k transistors

1990's
32 bits uP



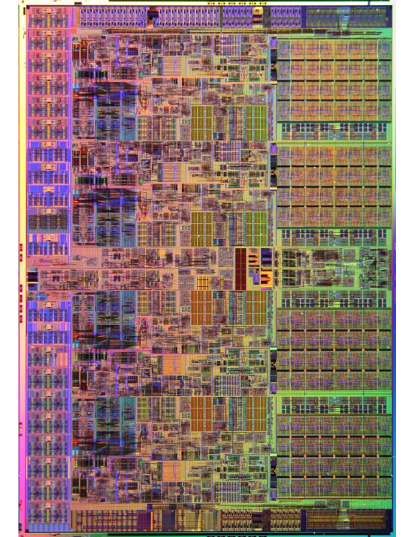
Pentium (1993)
3,1M transistors

2000's
64 bits uP



Pentium IV
[42-184]M transistors

2010's
64 bits uP



Core i7
[731M et + (>Milliard)
transistors

1. Historique

Evolution des technologies

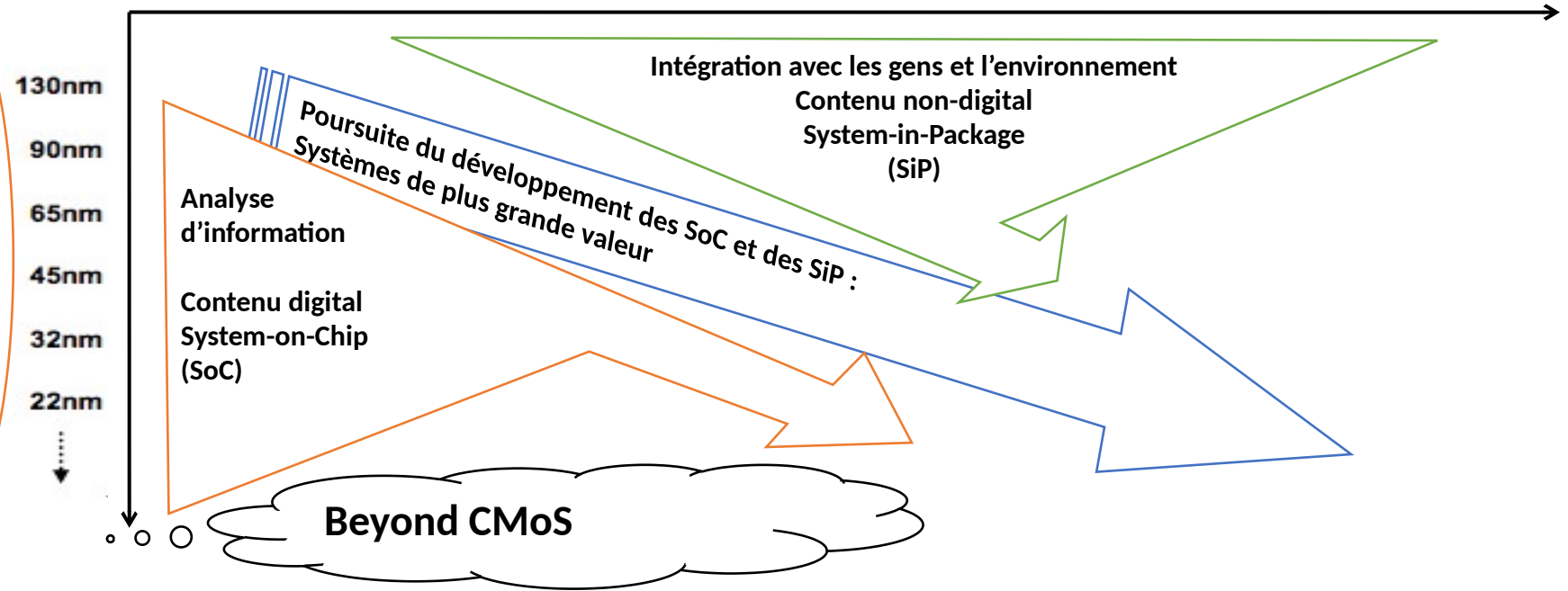
More than Moore



- Analogue/RF
- Composants passifs
- Haute tension alimentation
- Capteurs actuateurs
- BioChips

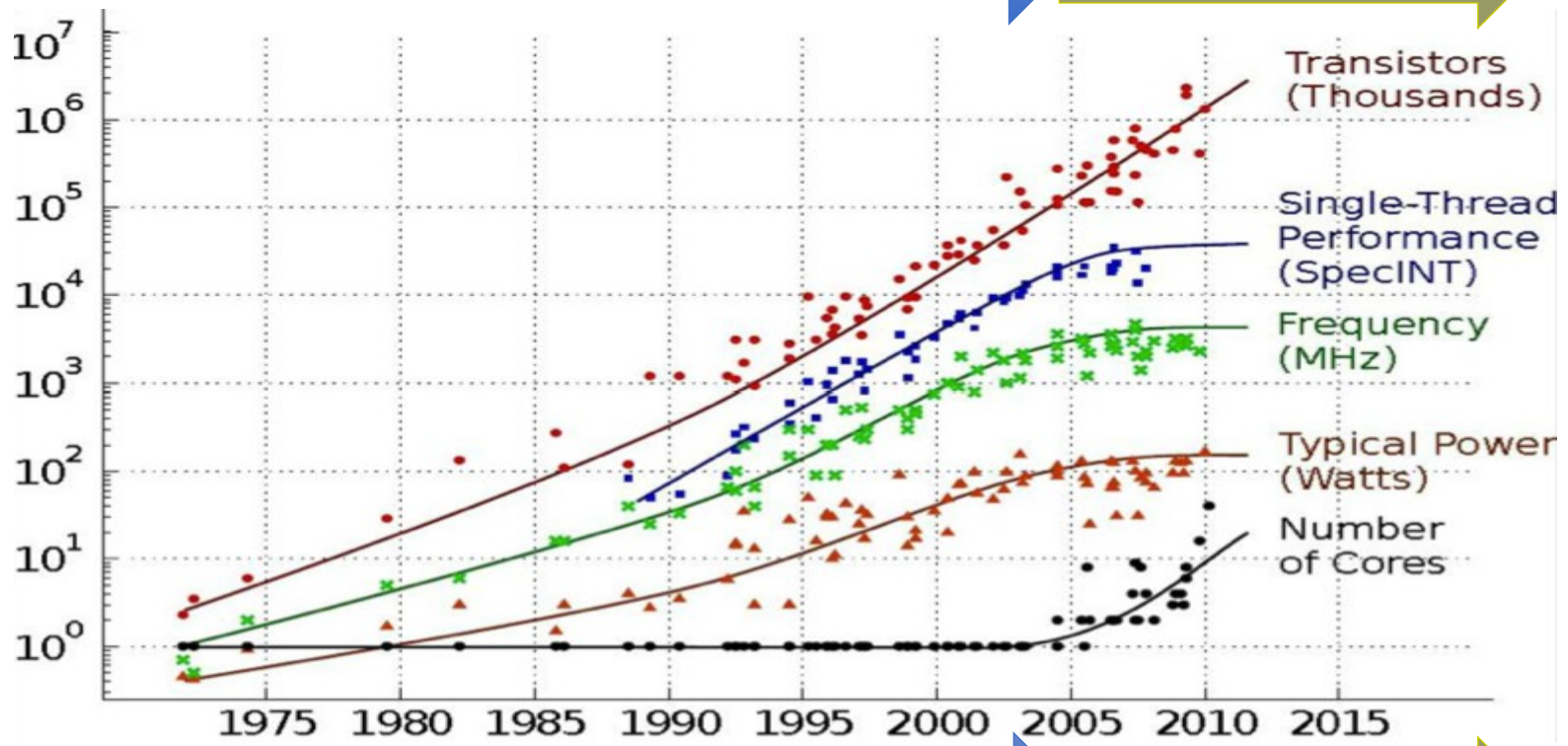


Coeur CMOS: CPU, mémoire, logique



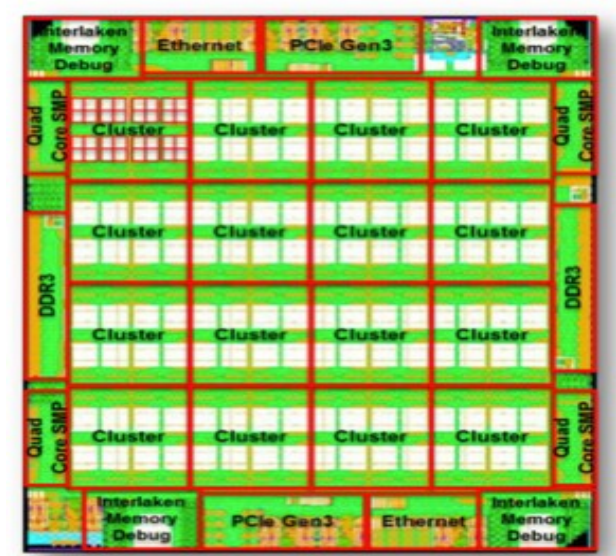
More Moore

1. Historique



Emergence des architectures MPSoC

Augmentation de la puissance de calcul

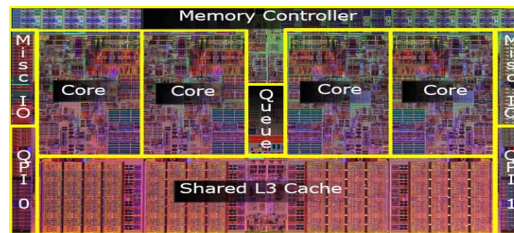
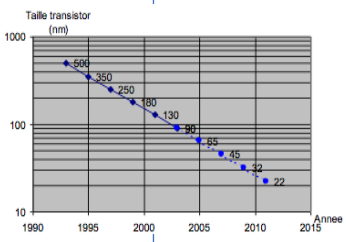


Evolution des technologies

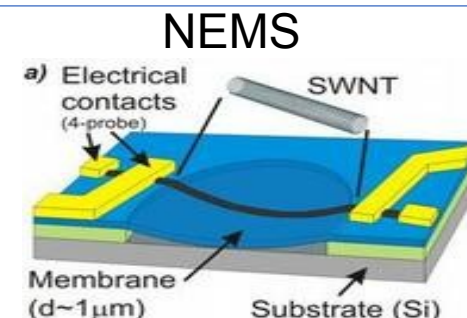
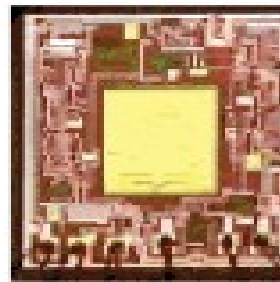
Hier

Aujourd'hui

Demain



MEMS



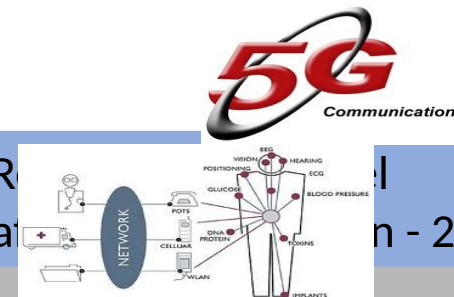
NEMS



Cergy-Pontoise

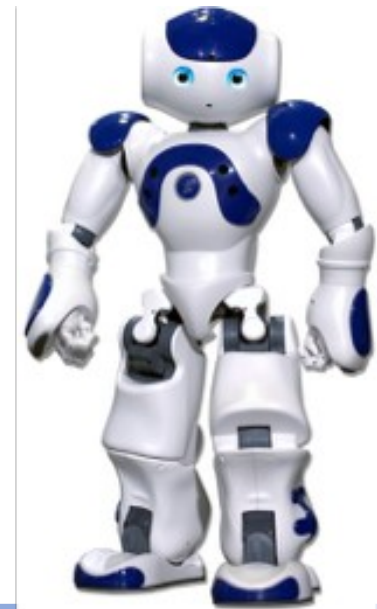
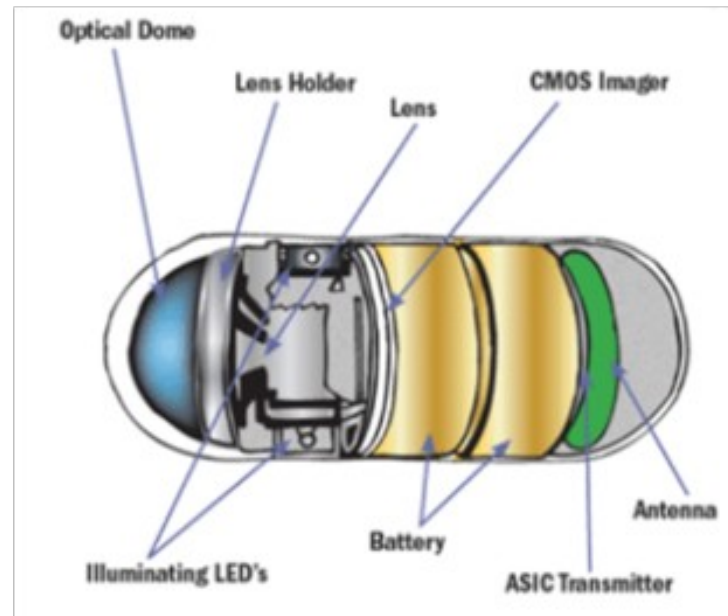


O.R
Maj: F.Gha



2019

- Processeurs fabriqués en masse et à faible coût
- Ils sont omniprésents dans nos vies et ils sont le cœur des objets connectés pour « **l'internet des objets** »



□ Fonctionnalités d'un microprocesseur

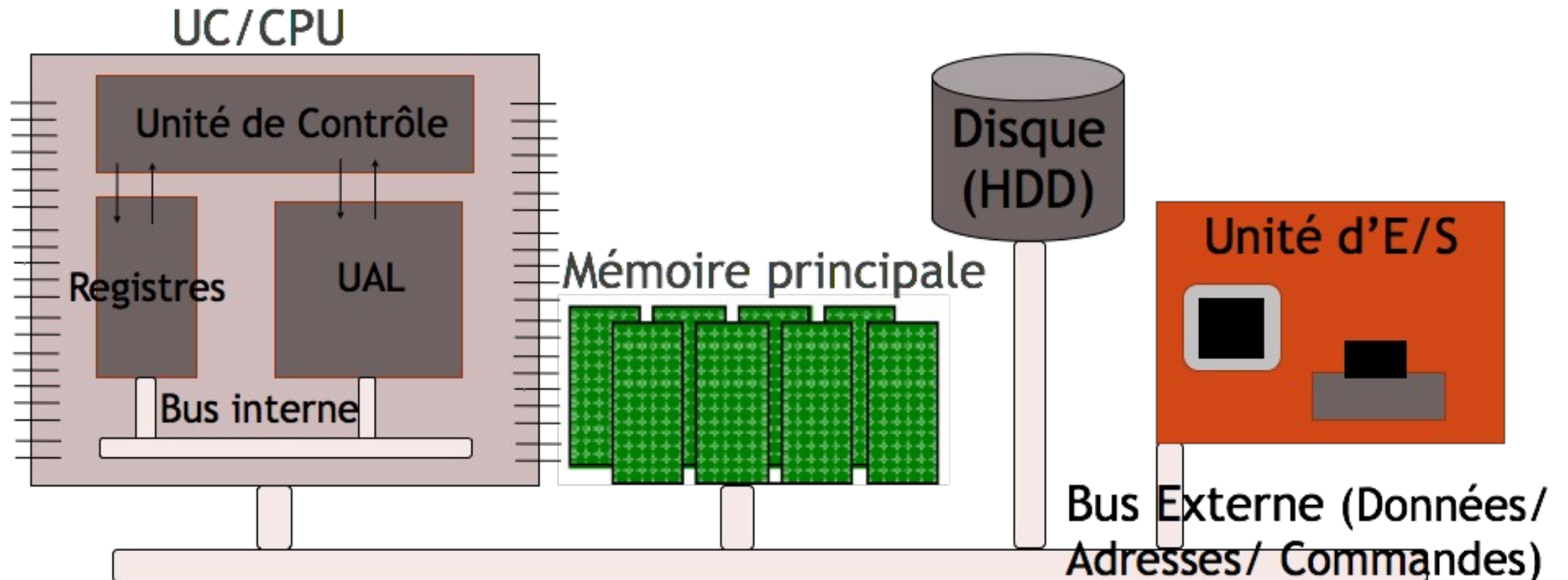
- Acquisition de signaux issus du monde 'extérieur'
 - Convertisseur analogique-numérique
- Traitement des données
 - processeur
- Envoie de commandes extérieures
 - GPIO, convertisseur numérique-analogique
- Communication
 - Liaison I2C, SPI, USART, USB, Ethernet, ...
- Gestion de périphériques internes
 - Mémoires, Timer, ...

2) Taxonomie d'un processeur

- Architecture actuelle de l'ordinateur (aspects externe/interne)
- Cycle de vie d'une instruction
- Définition des différents composants élémentaires (ALU, registre, mémoire, bus)

3. Architecture

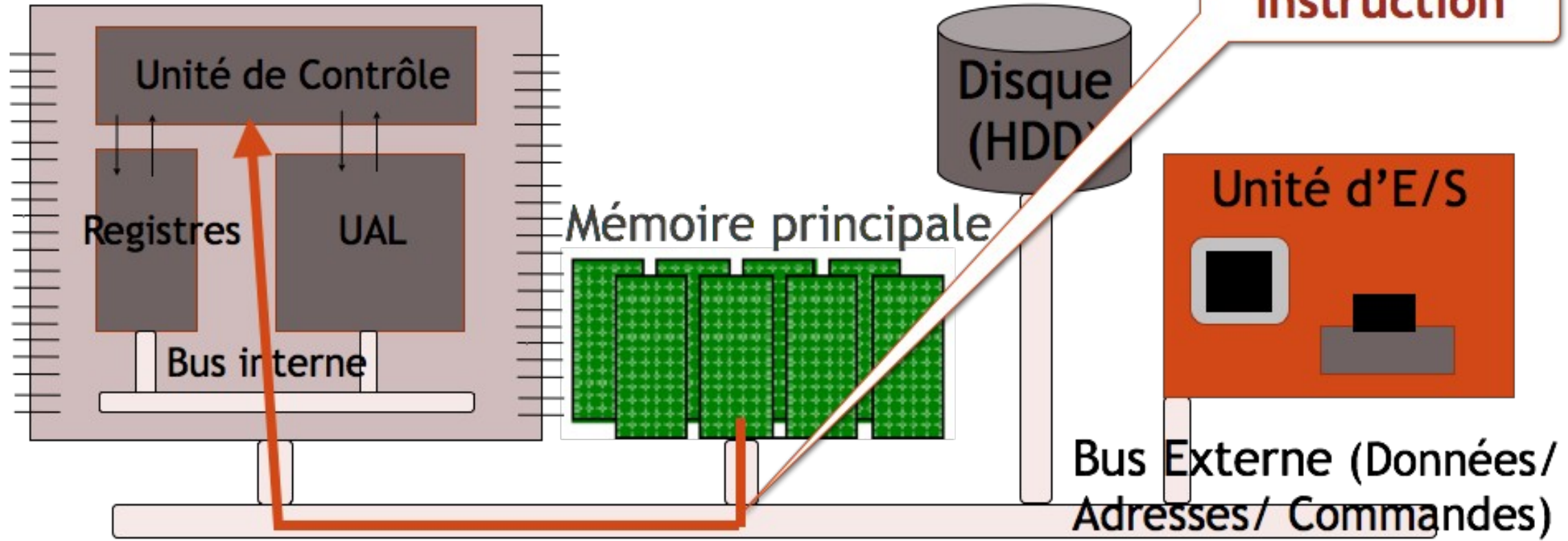
- Architecture actuelle de l'ordinateur : aspect externe



- ❑ Architecture actuelle de l'ordinateur
 - ❑ Le processeur exécute un programme
 - ❑ Programme écrit en mémoire
 - ❑ Transfert **d'instructions**
 - ❑ Le processeur manipule des variables
 - ❑ Transfert de **données**
 - ❑ Toutes ces informations sont rangées à un certain emplacement mémoire
 - ❑ Transfert **d'adresses**

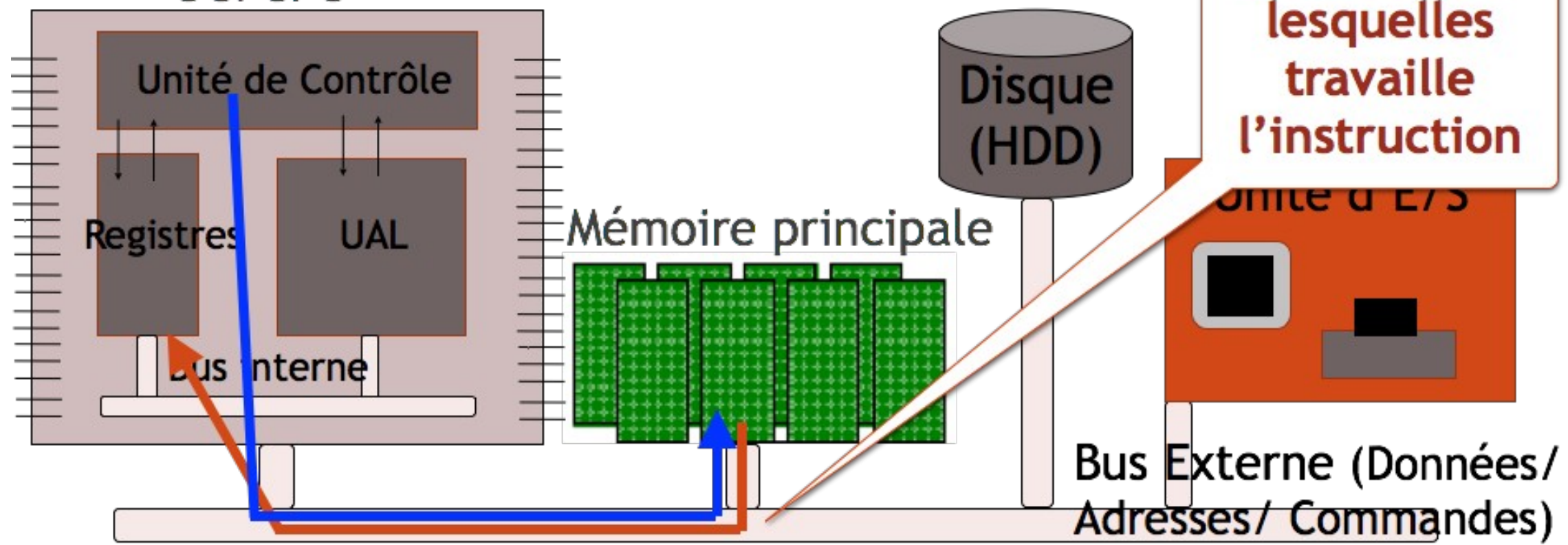
2. Architecture

□ Principe général d'exécution
UC/CPU

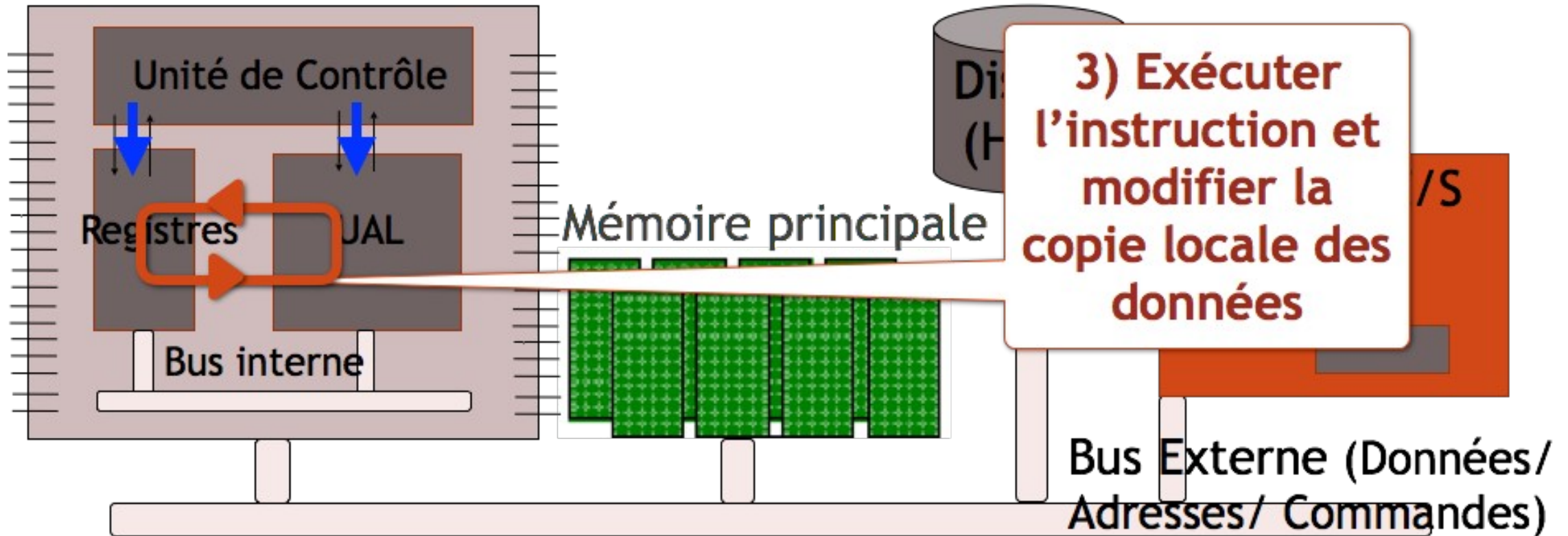


2. Architecture

Principe général d'exécution UC/CPU

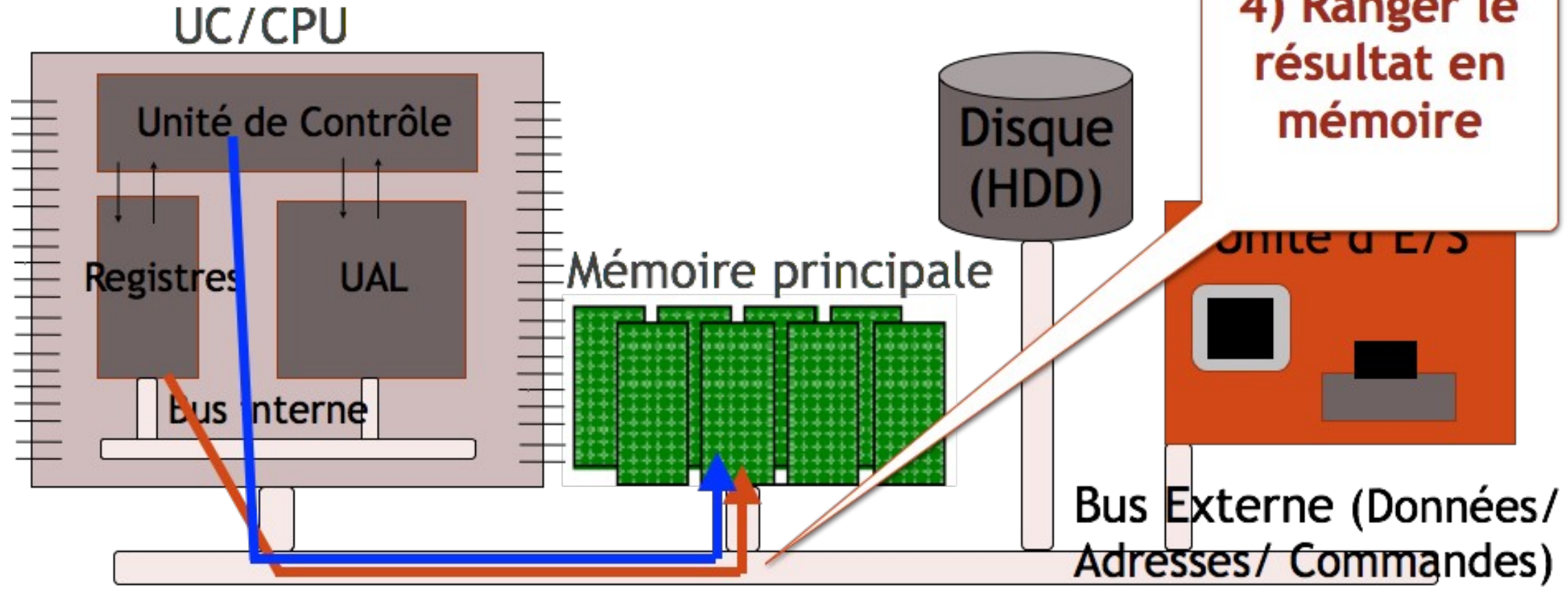


□ Principe général d'exécution UC/CPU

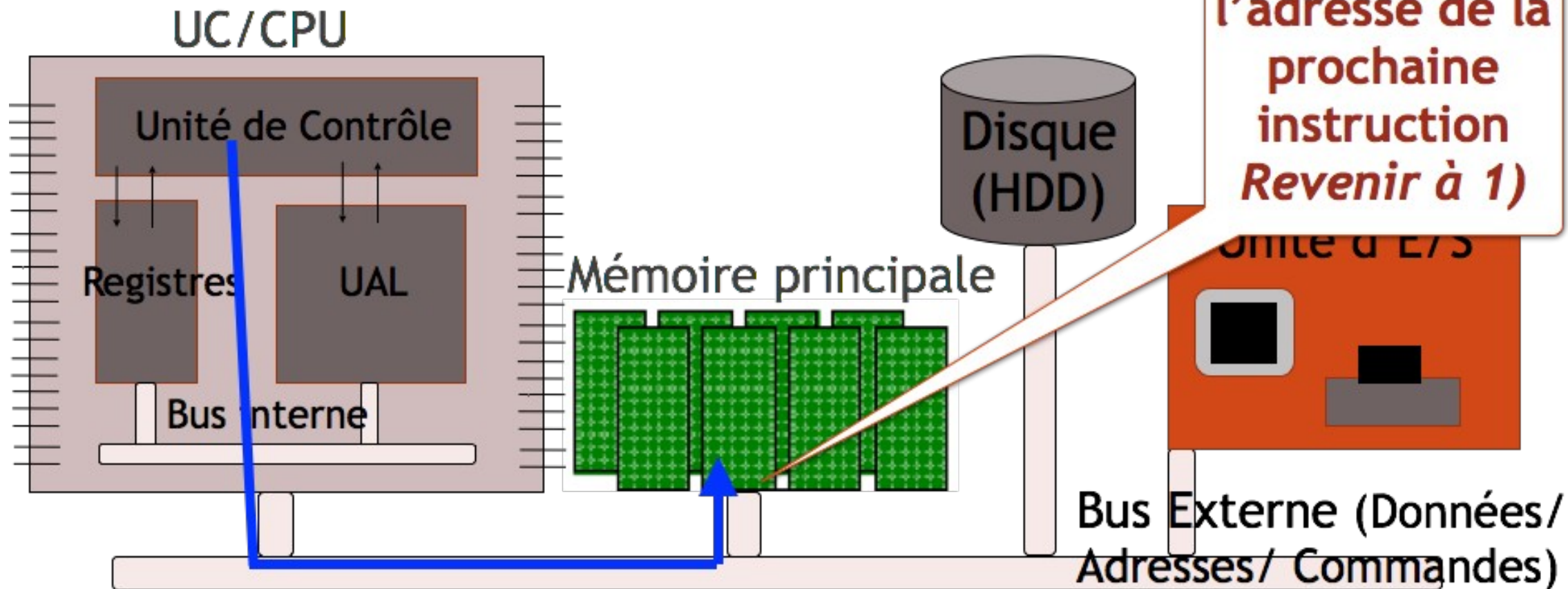


2. Architecture

Principe général d'exécution



□ Principe général d'exécution



□ Première vision du cycle d'exécution machine

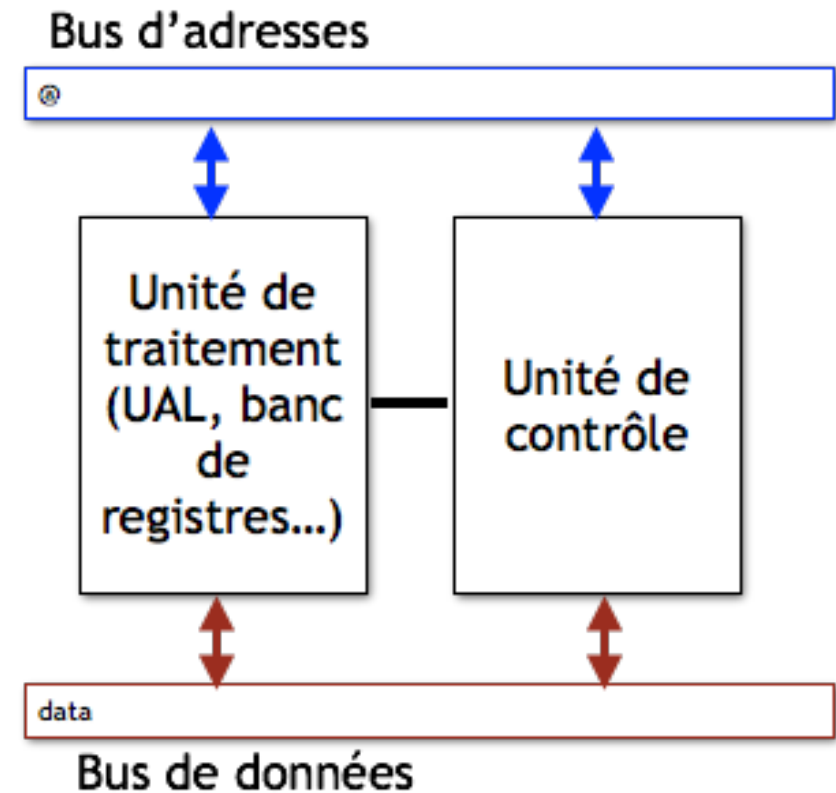
Un cycle d'exécution machine consiste à :

1. Charger l'instruction
2. Charger ses données
3. Faire un traitement sur ces données
4. Ranger le résultat du traitement en mémoire
5. Désigner la prochaine instruction

Séance 2

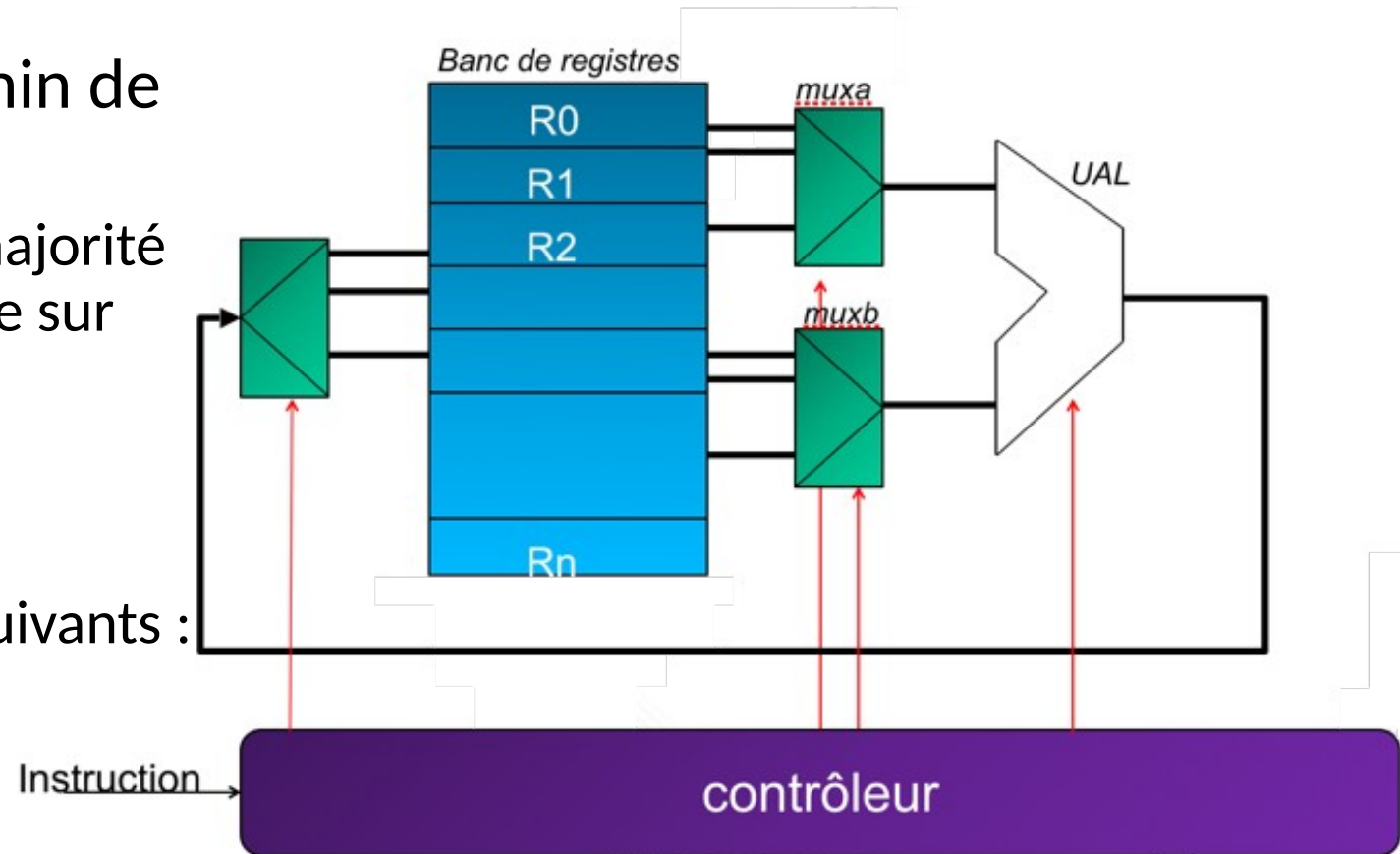
□ Aspect interne du microprocesseur : le chemin de données

- On trouve au cœur du microprocesseur
 - **Unité de traitement** : regroupe les unités permettant les traitements nécessaires à l'exécution des instructions
 - **Unité de contrôle** : séquence le déroulement des instructions



□ Aspect interne du microprocesseur : le chemin de données

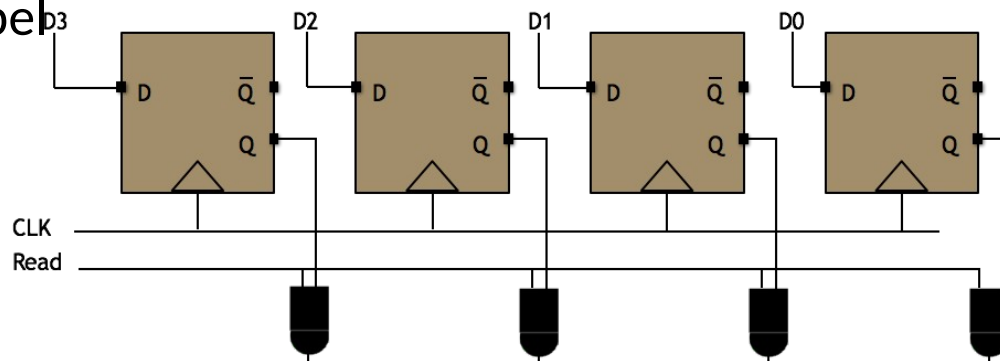
- Depuis les années 80, la majorité des architectures est basée sur une architecture à chargement/rangement
- Utilisation des éléments suivants :
 - banc de registres
 - UAL
 - multiplexeurs



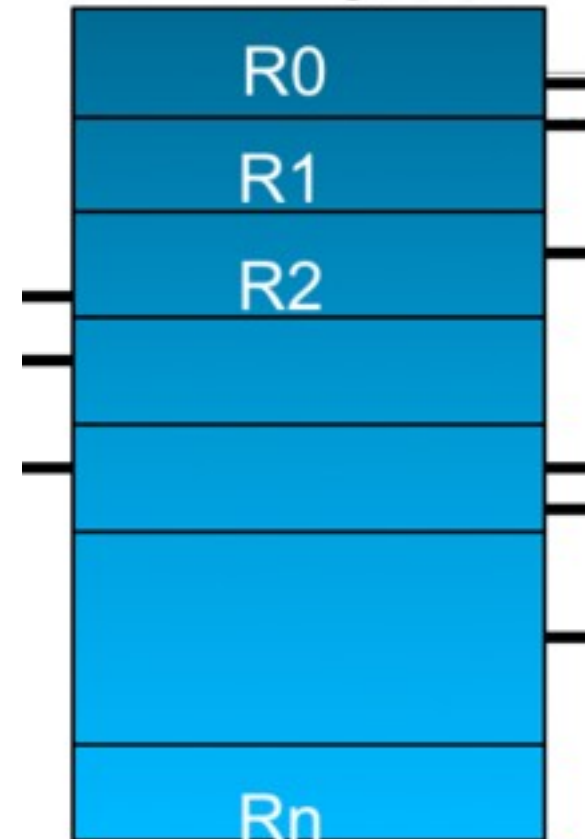
□ Le banc de registres

- Ces éléments sont des registres de travail qui permettent le stockage d'opérandes au début d'une opération et le résultat d'une opération
- Les registres sont accessibles très rapidement en lecture/écriture en comparaison à une mémoire

• Rappel



Banc de registres

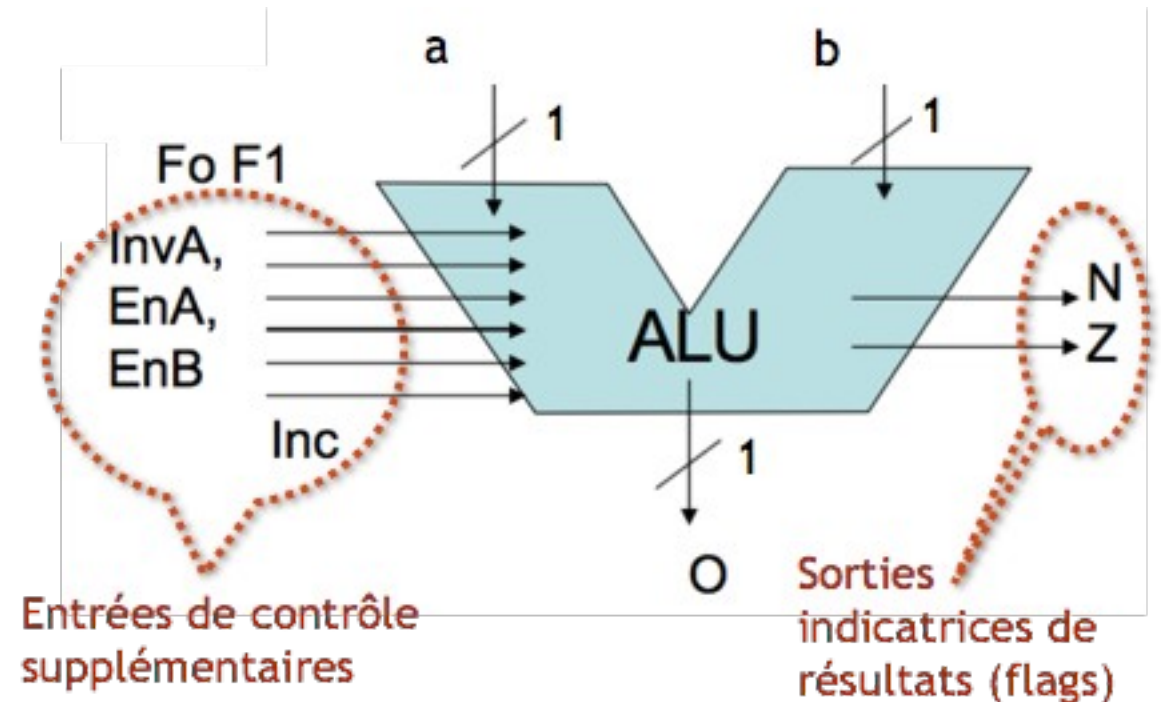
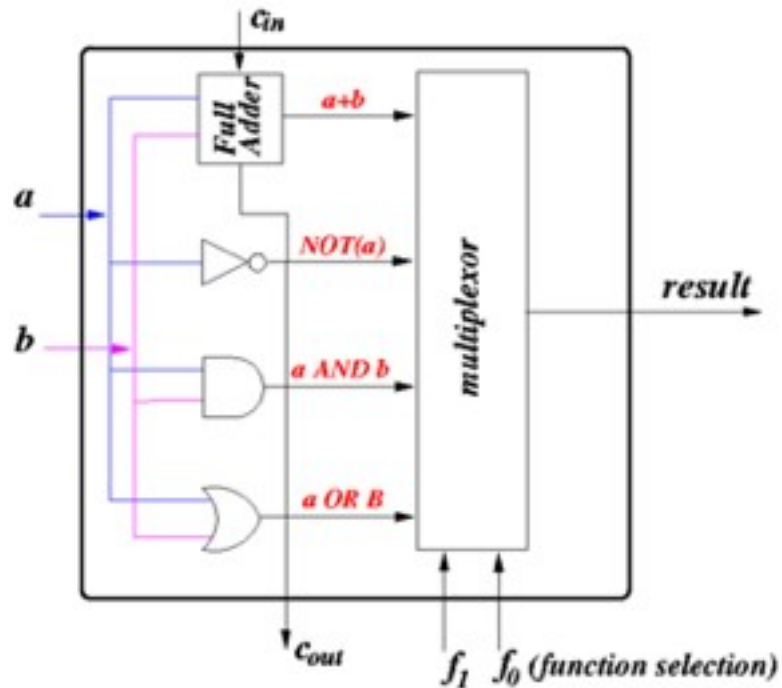
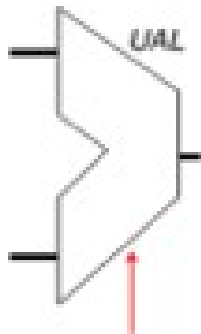


O.Romain & J.Lorandel
Màj: F.Ghaffari & S.Zuckerman - 2019

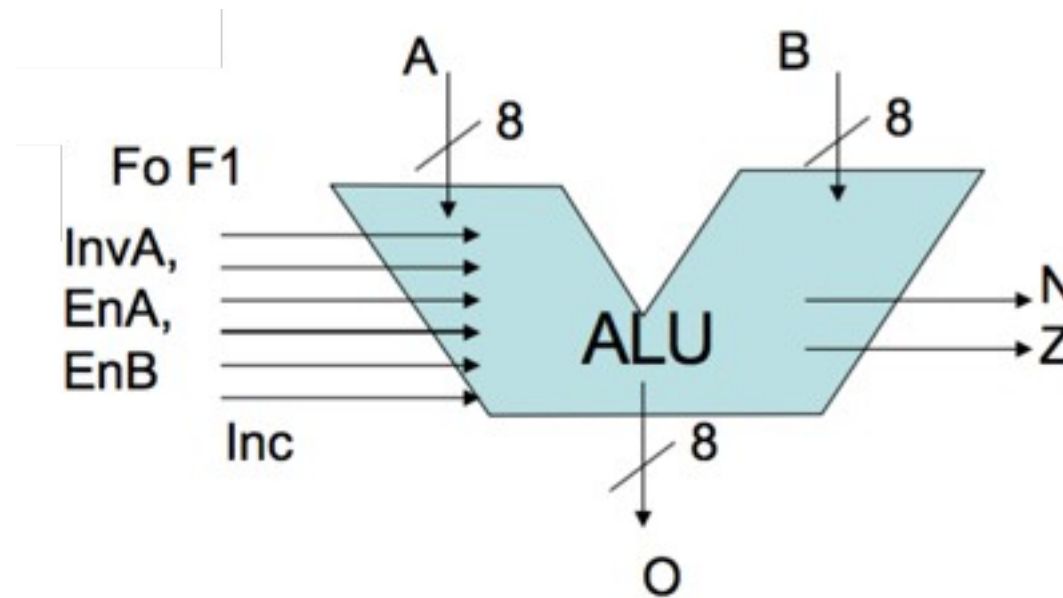
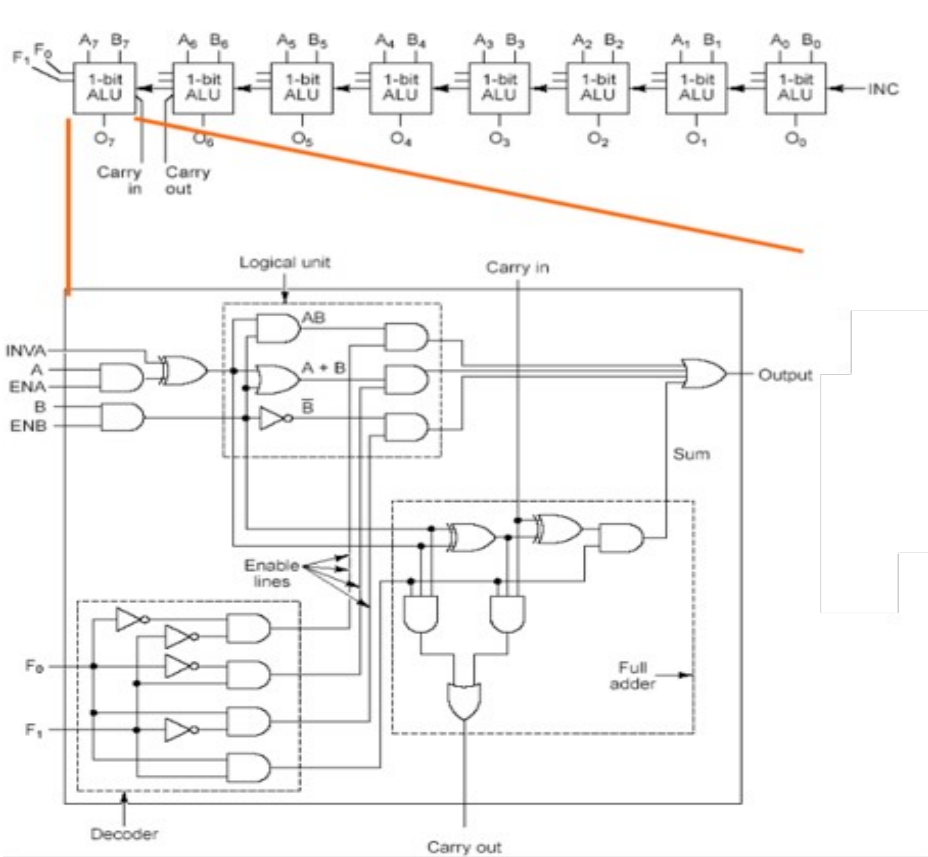
□ L'Unité Arithmétique et Logique (UAL)

- Permet de réaliser des calculs arithmétiques (additions, soustractions) et logiques (AND, OR,...)

ALU 1 bit
simplifiée



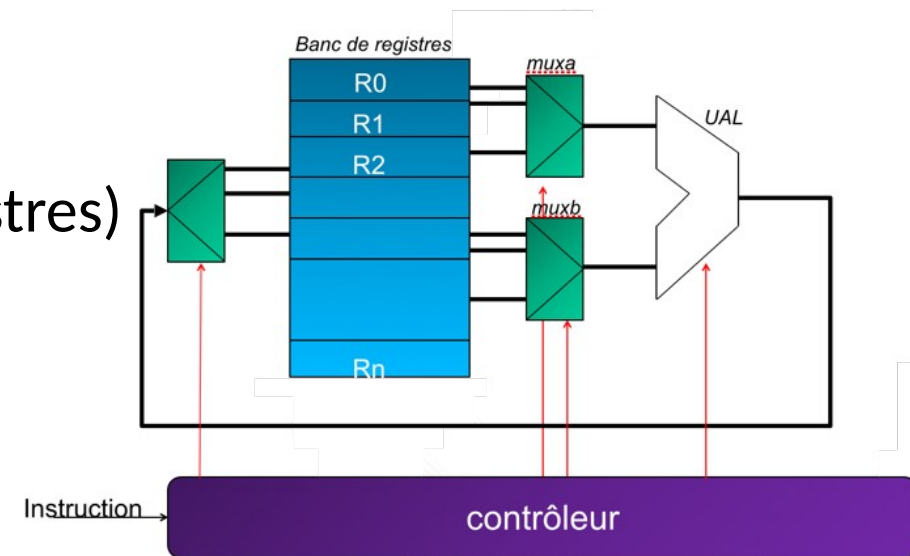
□ L'Unité Arithmétique et Logique (UAL) – 8 bits



Pour une UAL 8 bits = 8 UALs 1 bit

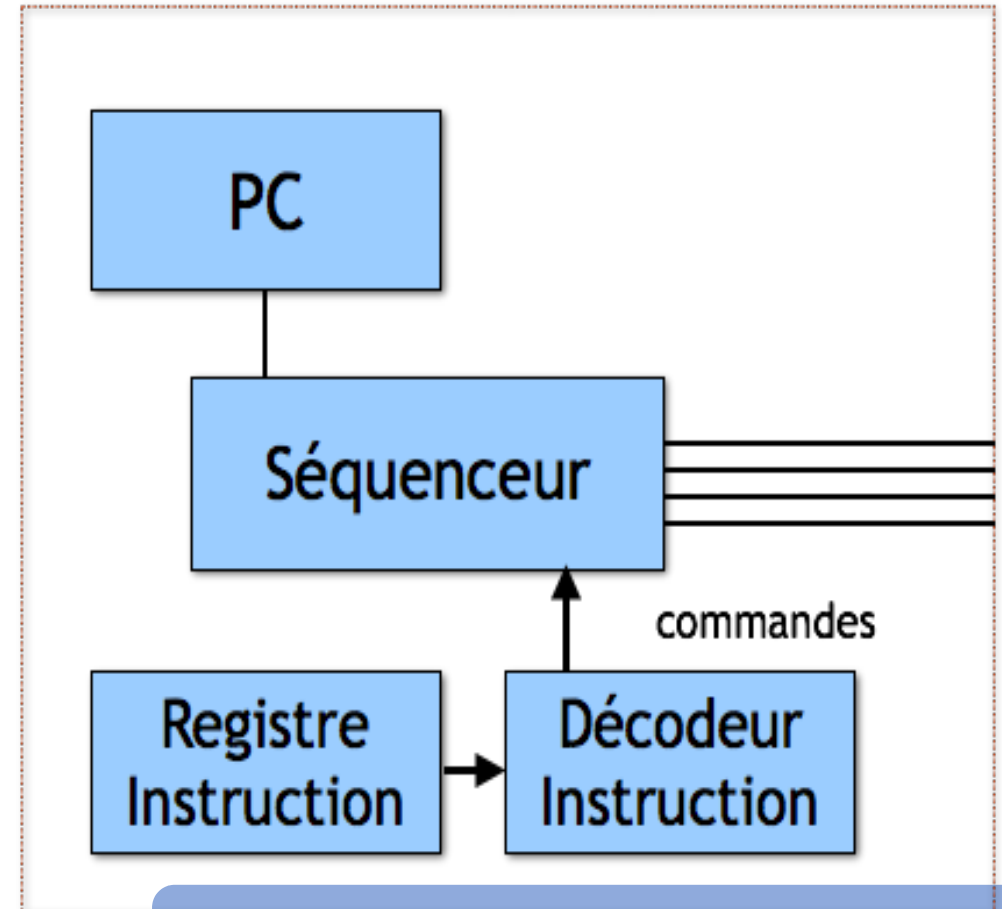
□ Le contrôleur

- Le contrôleur est une machine à états dont le rôle est de générer des signaux de contrôle à des instants précis
- Il doit :
 - Recevoir l'instruction à exécuter
 - Commander les opérations de l'ALU
 - Sélectionner les multiplexeurs (pour les registres)
 - Placer le résultat dans le registre adéquat
 - Charger la prochaine instruction



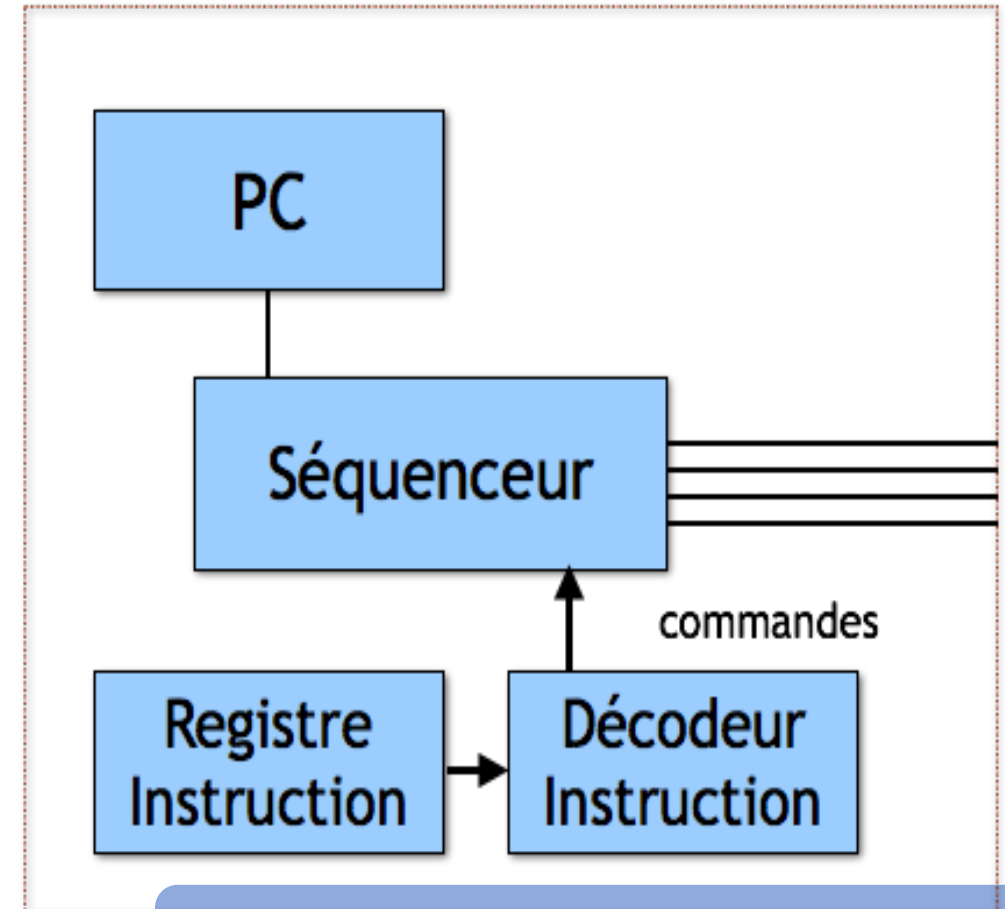
□ Le contrôleur : Architecture

1. **Le registre d'instruction (RI)** : les instructions venant de la mémoire y sont stockées
2. **Le décodeur d'instruction** : a pour rôle de décoder l'instruction et d'envoyer des signaux de commande au séquenceur
3. **L'unité de contrôle/commande (ou séquenceur)** : permet d'organiser le déroulement de l'instruction. Il est synchronisé par rapport à une horloge. C'est un automate.



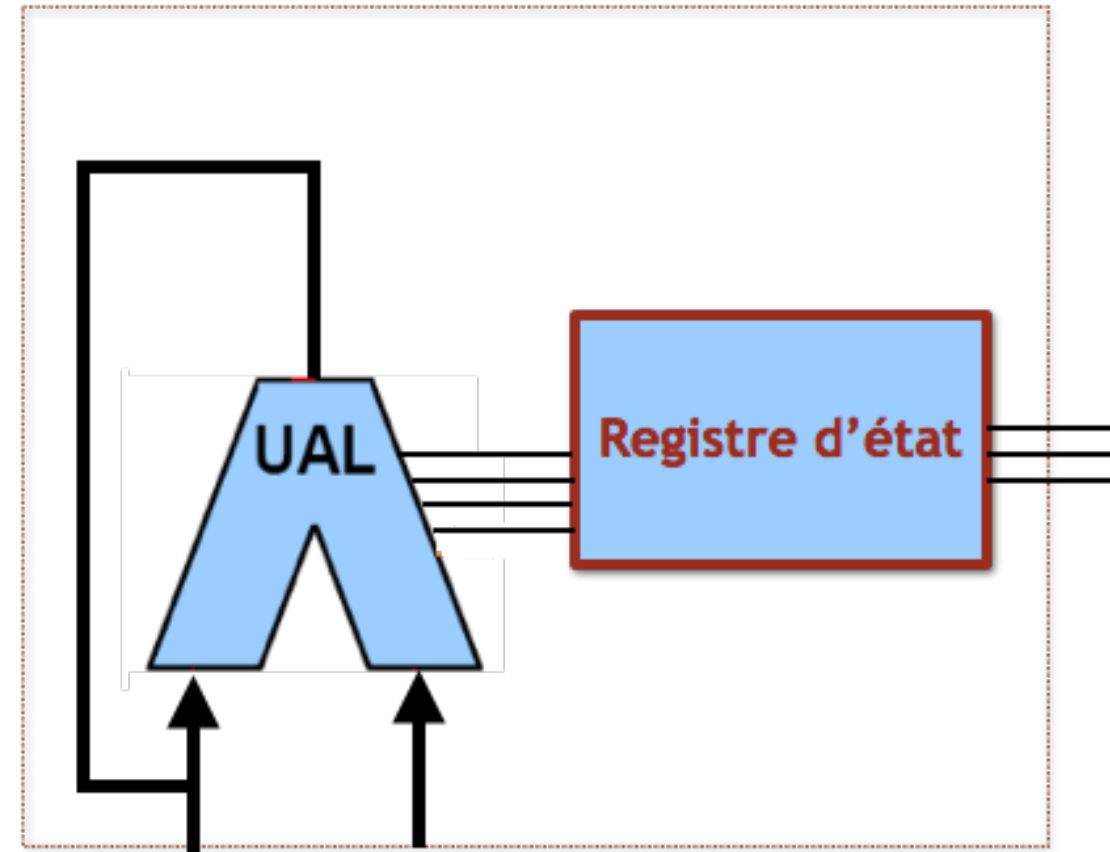
O.Romain & J.Lorandei
Màj: F.Ghaffari & S.Zuckerman - 2019

- Des registres spécifiques (1/2)
 1. **Le compteur programme (PC)** ou encore compteur ordinal est un registre. Il contient l'adresse de la prochaine instruction à exécuter. Il est initialisé au reset avec l'adresse de la 1^{ière} instruction du programme.

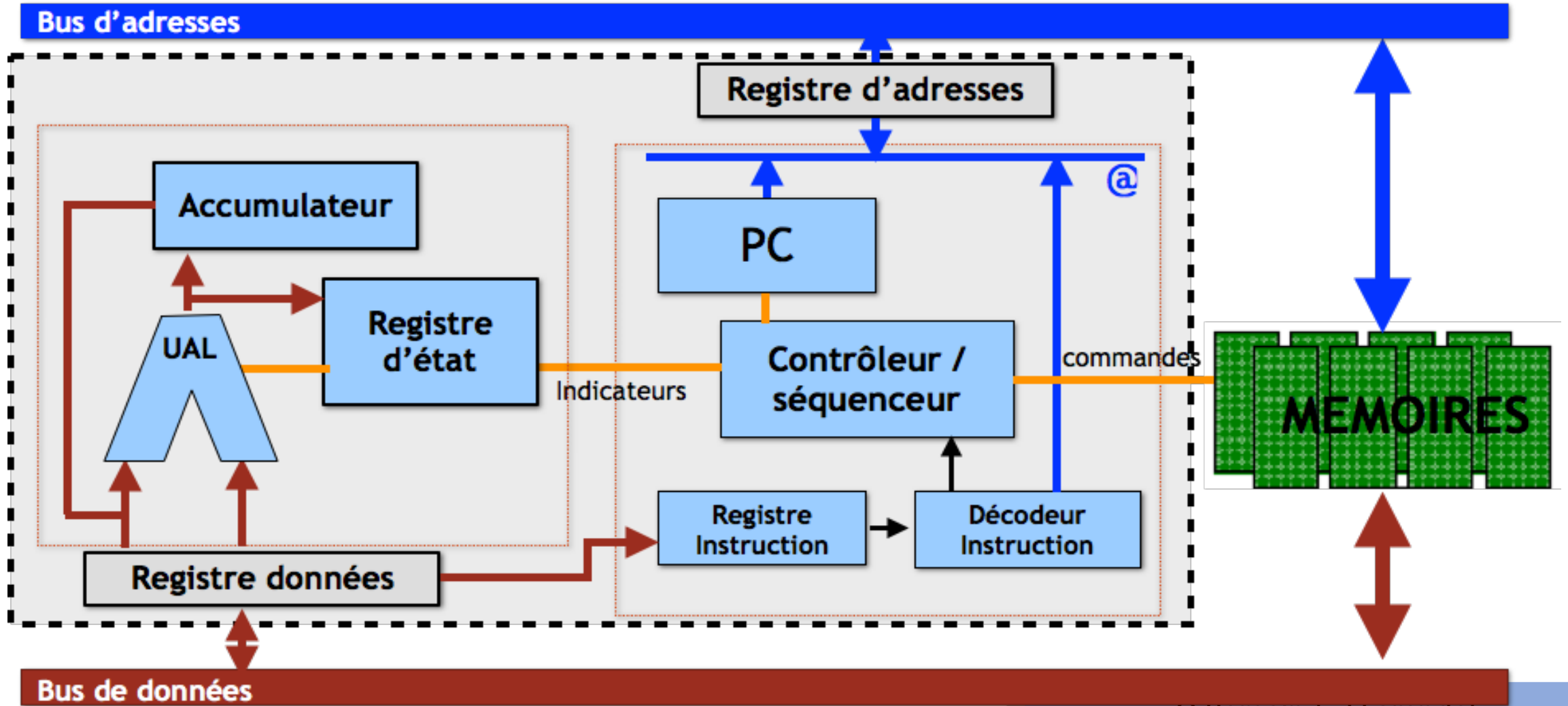


O.Romain & J.Lorandiel
Màj: F.Ghaffari & S.Zuckerman - 2019

- Des registres spécifiques (2/2)
 1. **Le registre d'état** (de *status*) est un registre contenant des bits, dont les états changent en fonction du résultat précédent de l'UAL. Ces flags/indicateurs conditionnent généralement le déroulement d'un programme.
 - Zéro (bit Z)
 - Négatif (bit N)
 - Carry (C)
 - Débordement (O ou OV)
 -



2. Architecture

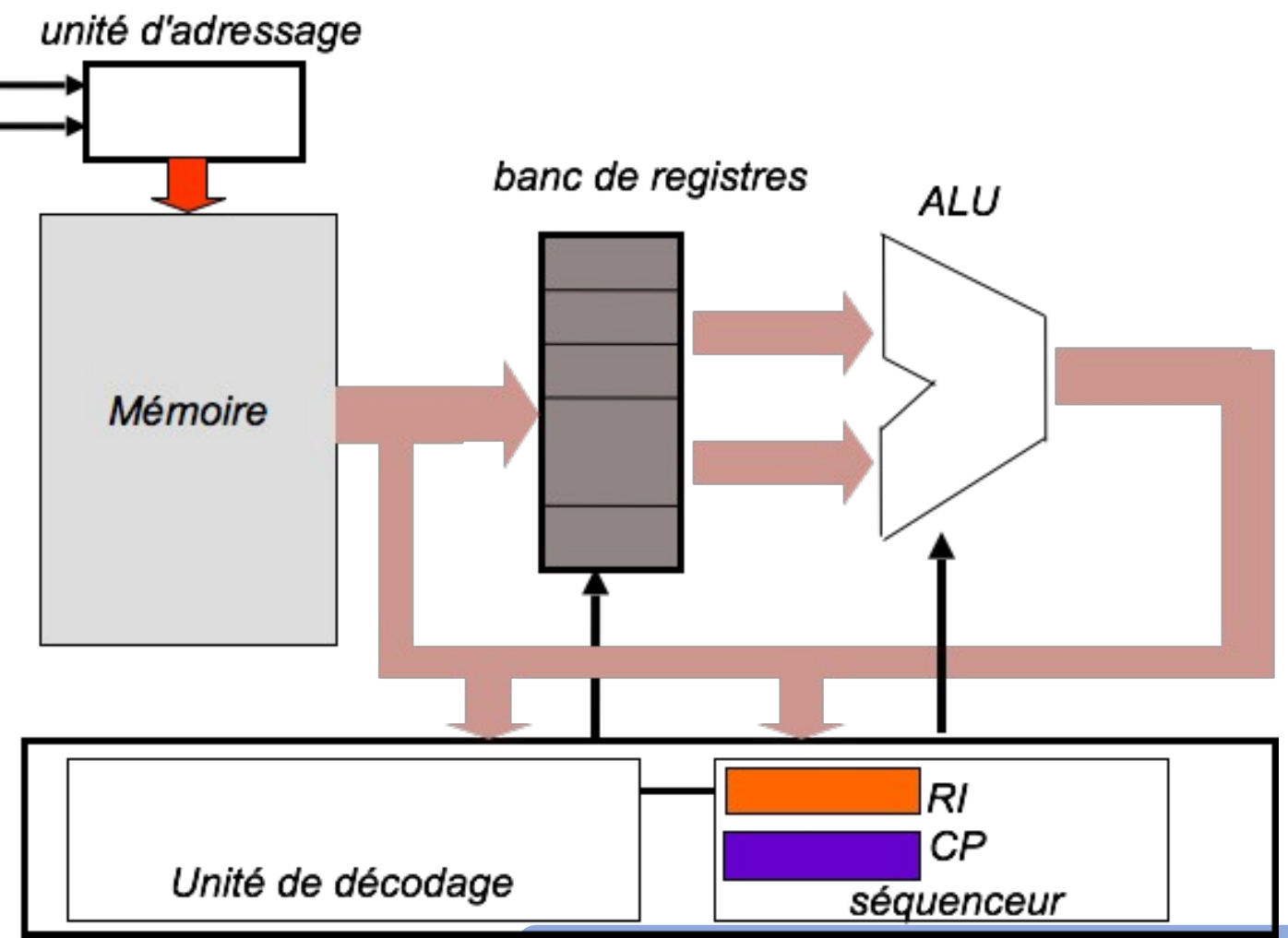


Bus de données

2. Cycle de vie d'une instruction

Exemple du cycle d'exécution machine

1. Recherche d'instruction
2. Incrémentation de PC
3. Décoder l'instruction
4. Rechercher les données
5. Exécuter l'opération
6. Ranger le résultat
7. Retour



2. Cycle de vie d'une instruction

☐ Exemple du cycle d'exécution machine

1. Recherche d'instruction
2. Incrémentation de PC
3. Décoder l'instruction
4. Rechercher les données
5. Exécuter l'opération
6. Ranger le résultat
7. Retour

Exemple de programme

Langage C

Assembleur

A=3;

Mov #3, R0

B= 4;

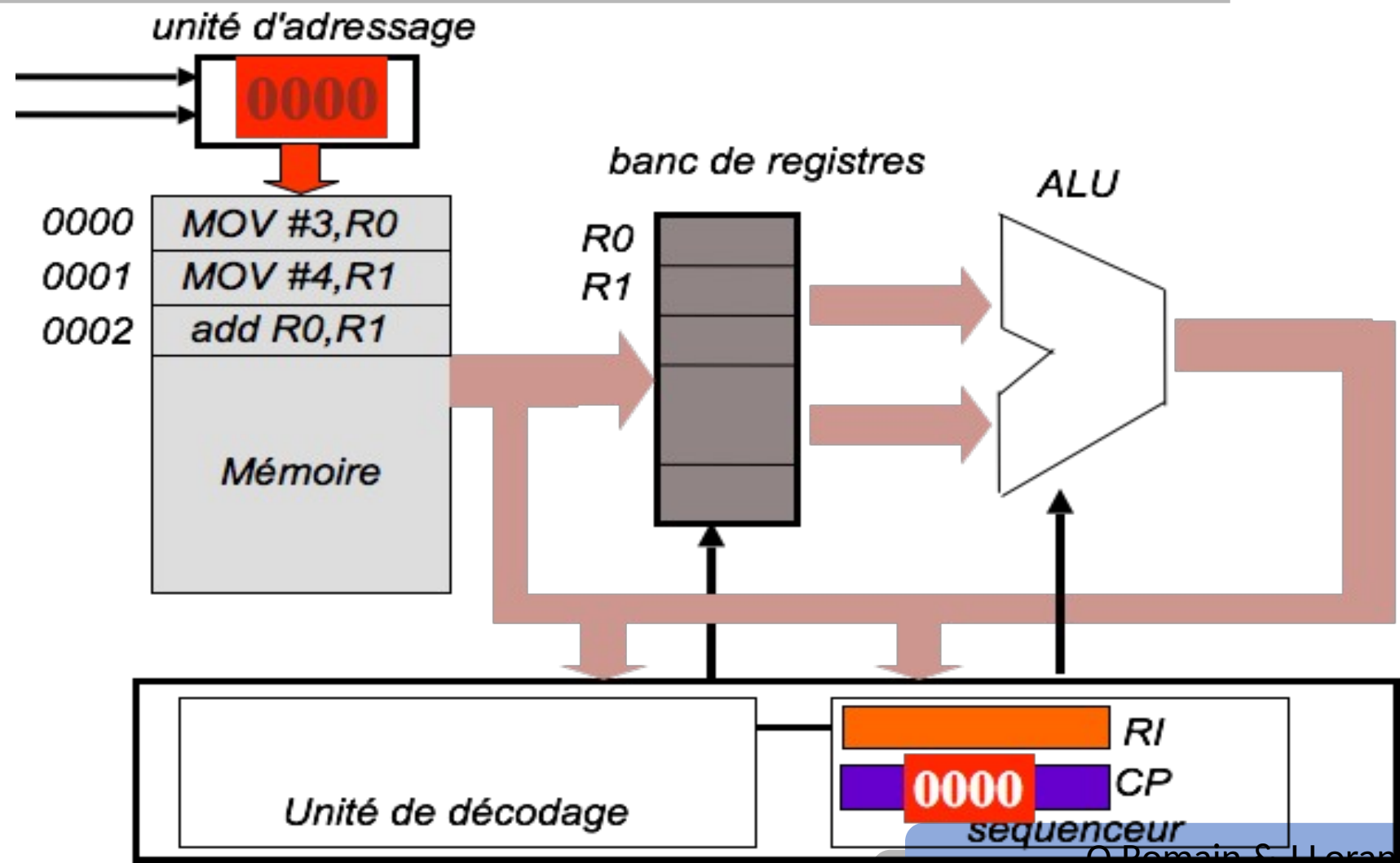
Mov #4, R1

B = A+B

ADD R0,R1

2. Cycle de vie d'une instruction

□ Etat initial

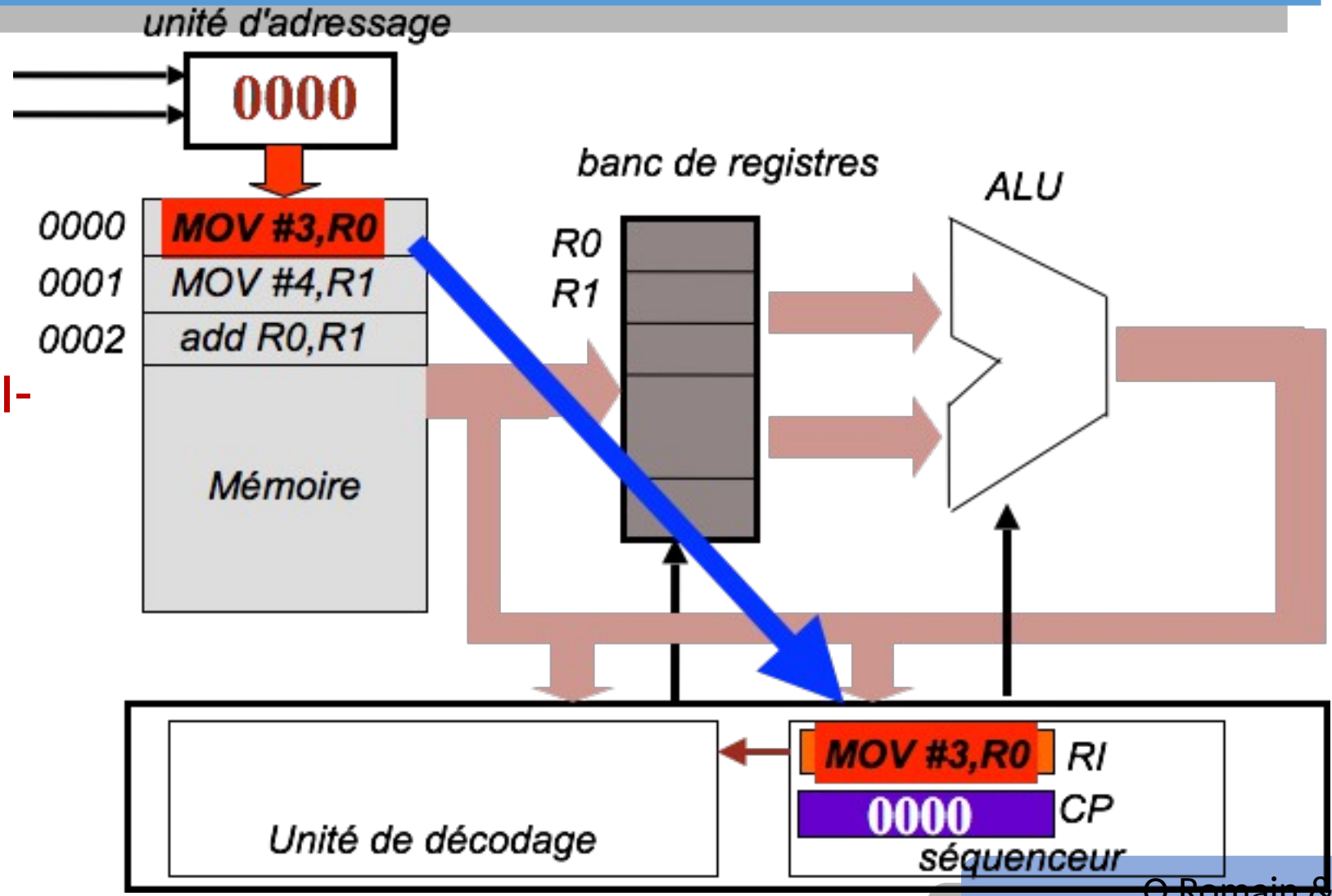


O.Romain & J.Lorandel

Maj: F.Ghaffari & S.Zuckerman - 2019

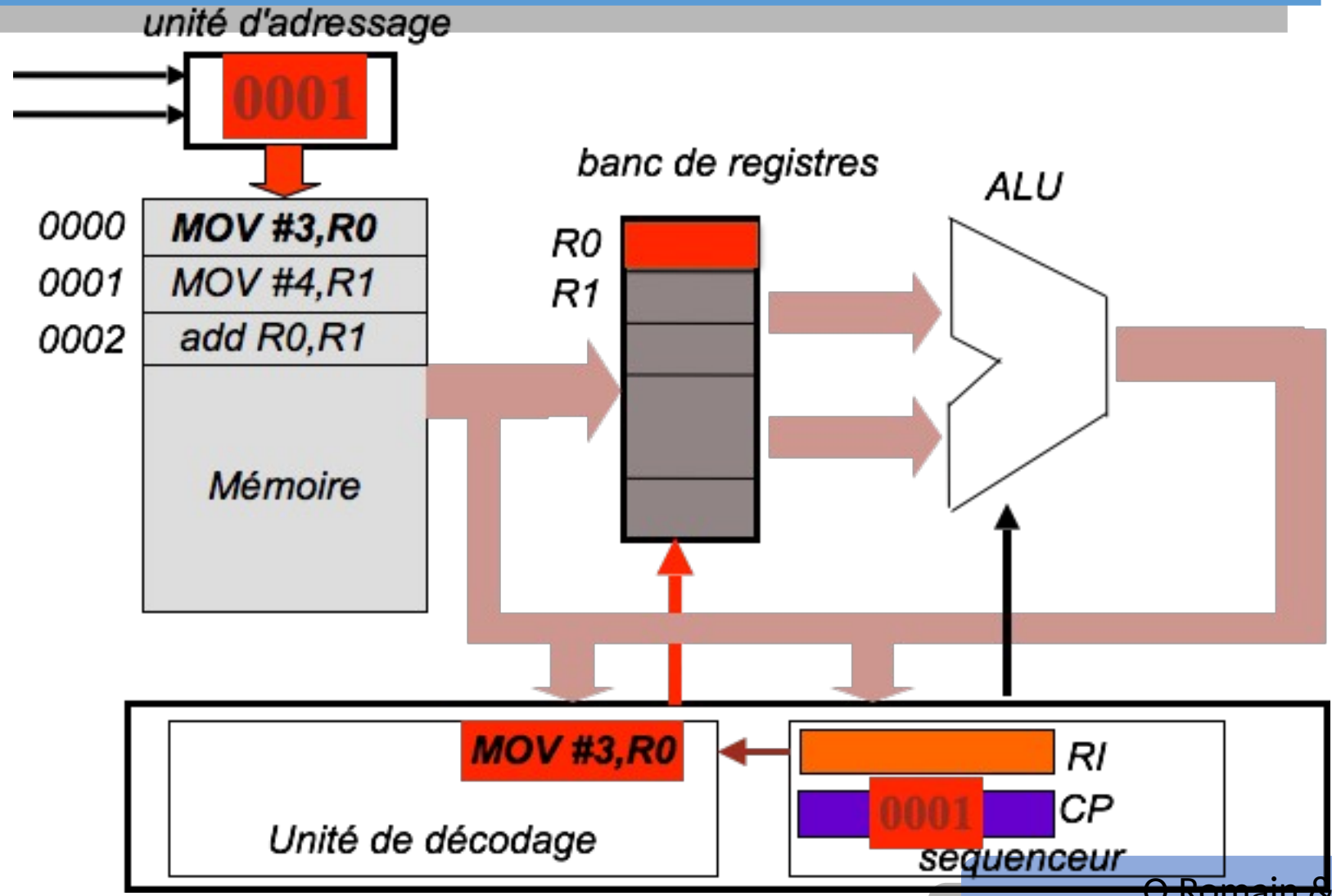
2. Cycle de vie d'une instruction

Recherche d'instruction (I-
Fetch)



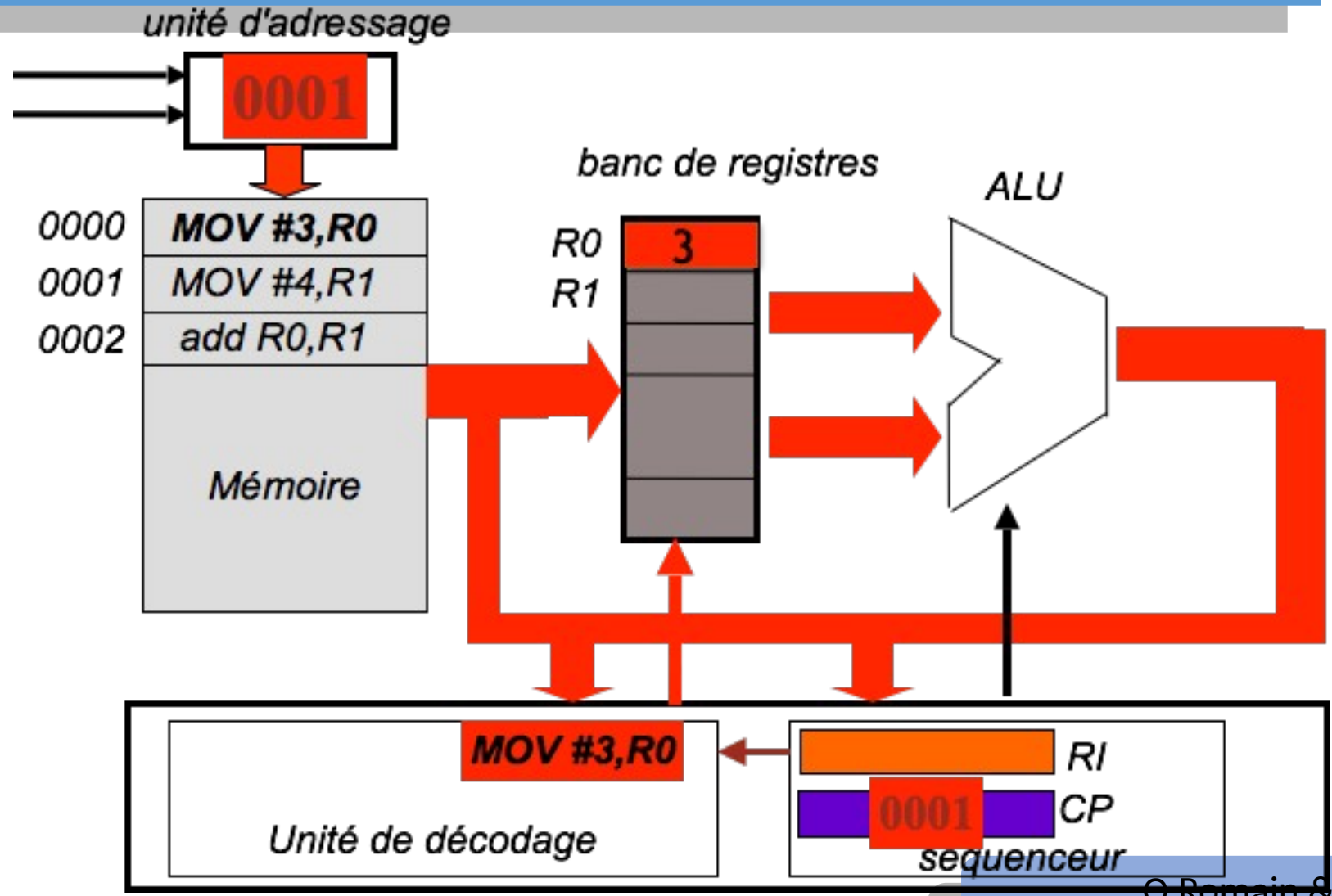
2. Cycle de vie d'une instruction

□ Décodage
(Decode)



2. Cycle de vie d'une instruction

Exécution
(Execute)



2. Cycle de vie d'une instruction

Exemple de programme

Langage C

A=3;

B= 4;

B = A+B

OK

Idem pour cette instruction

Assembleur

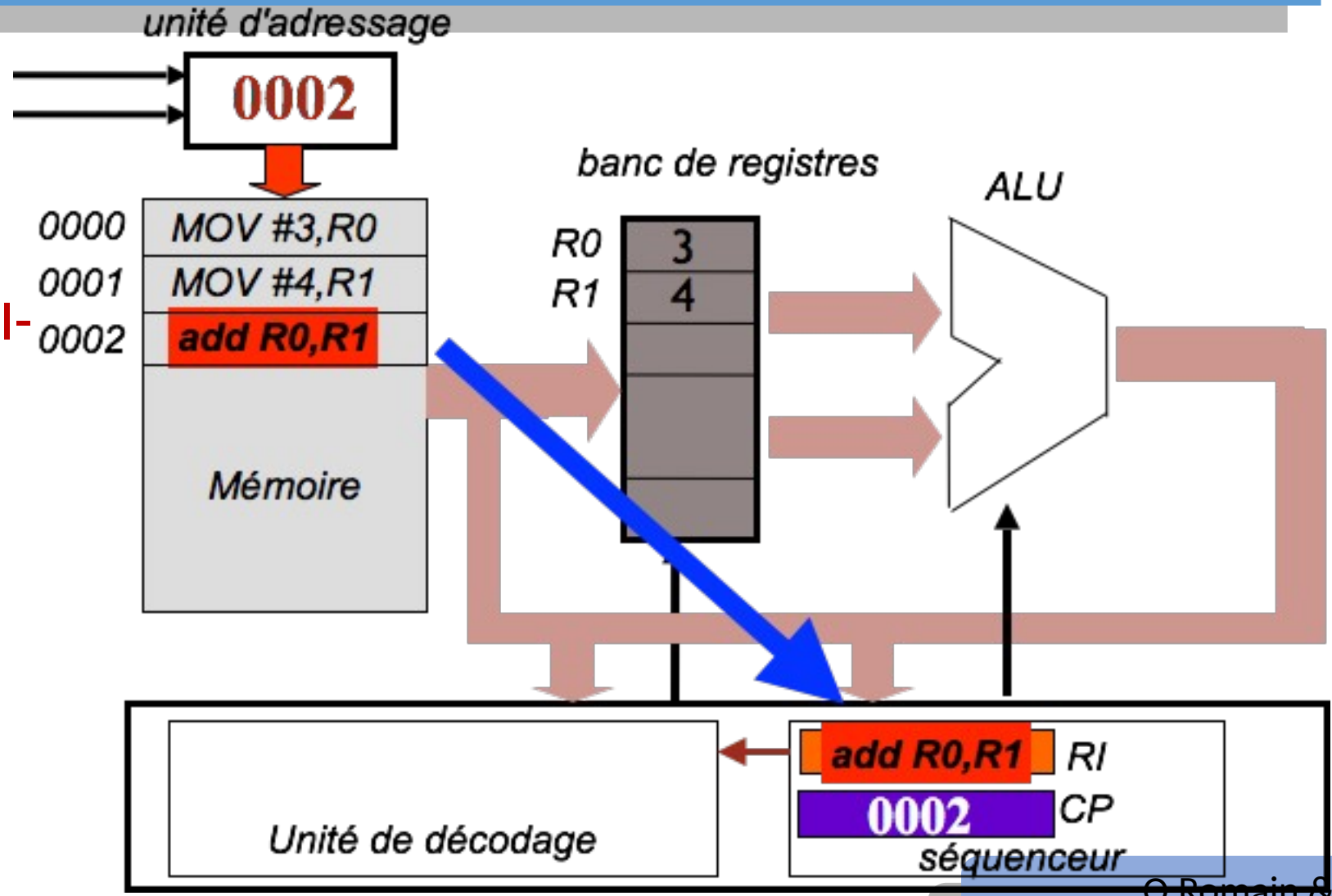
Mov #3, R0

Mov #4, R1

ADD R0,R1

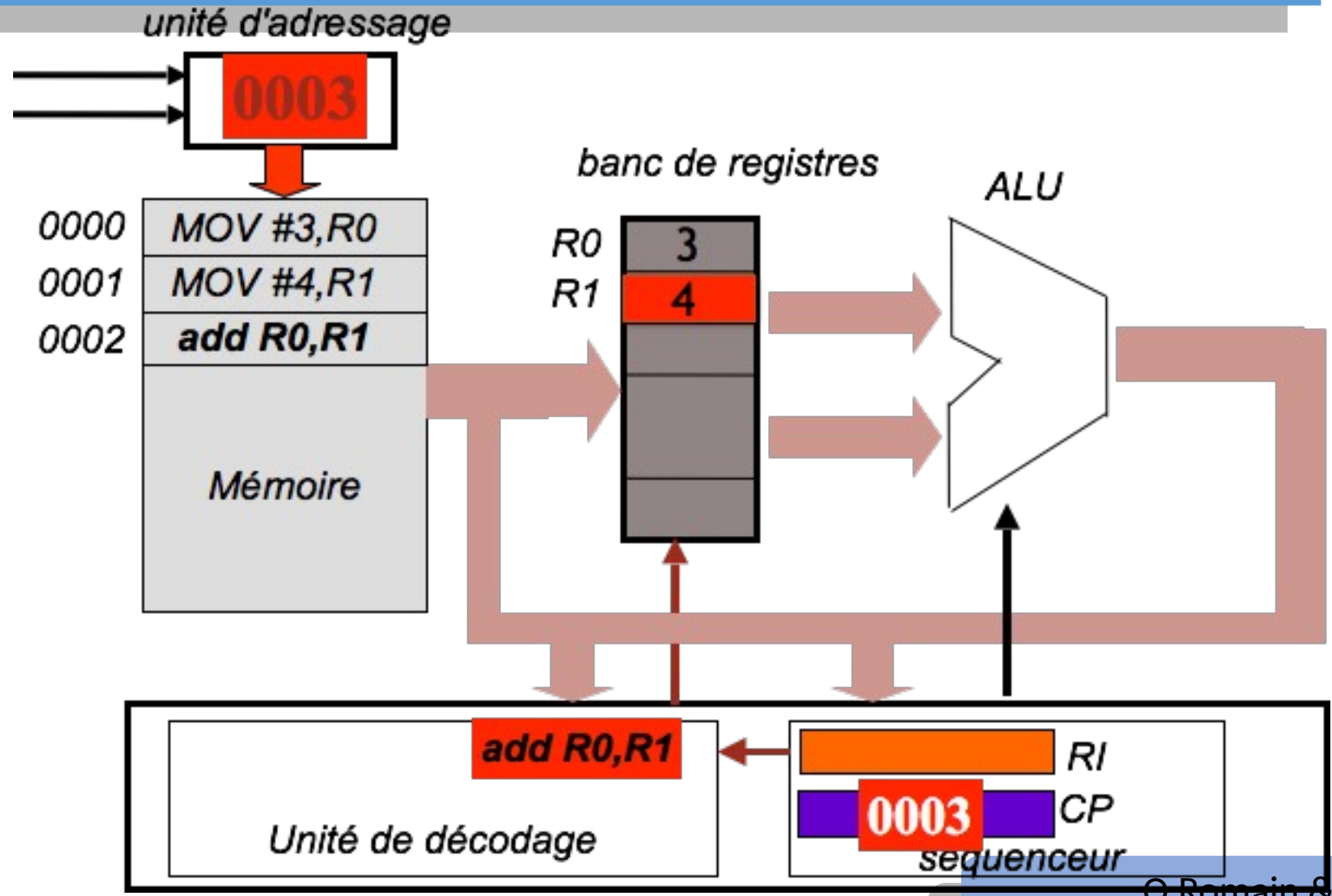
2. Cycle de vie d'une instruction

Recherche d'instruction (I-
Fetch)



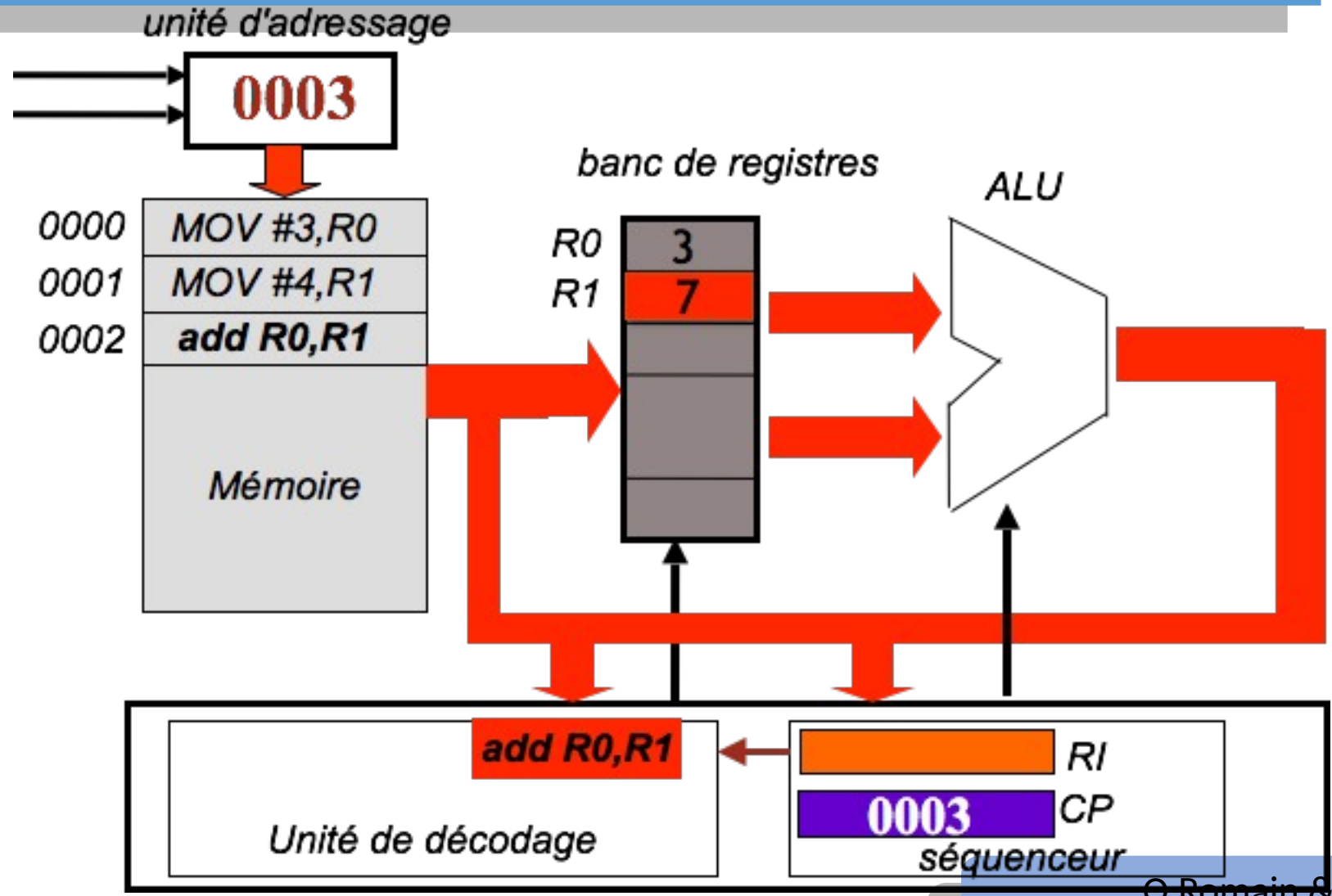
2. Cycle de vie d'une instruction

□ Décodage
(Decode)



2. Cycle de vie d'une instruction

Exécution
(Execute)



2. Cycle de vie d'une instruction

□ Cas d'accès mémoires

Exemple de programme

Algorithme

Tab[100] : integer

A <= tab[0]

B <= 2

tab[0] <= A+B

Assembleur

Mov #Tab, R2

Mov @R2, R0

Mov #2, R1

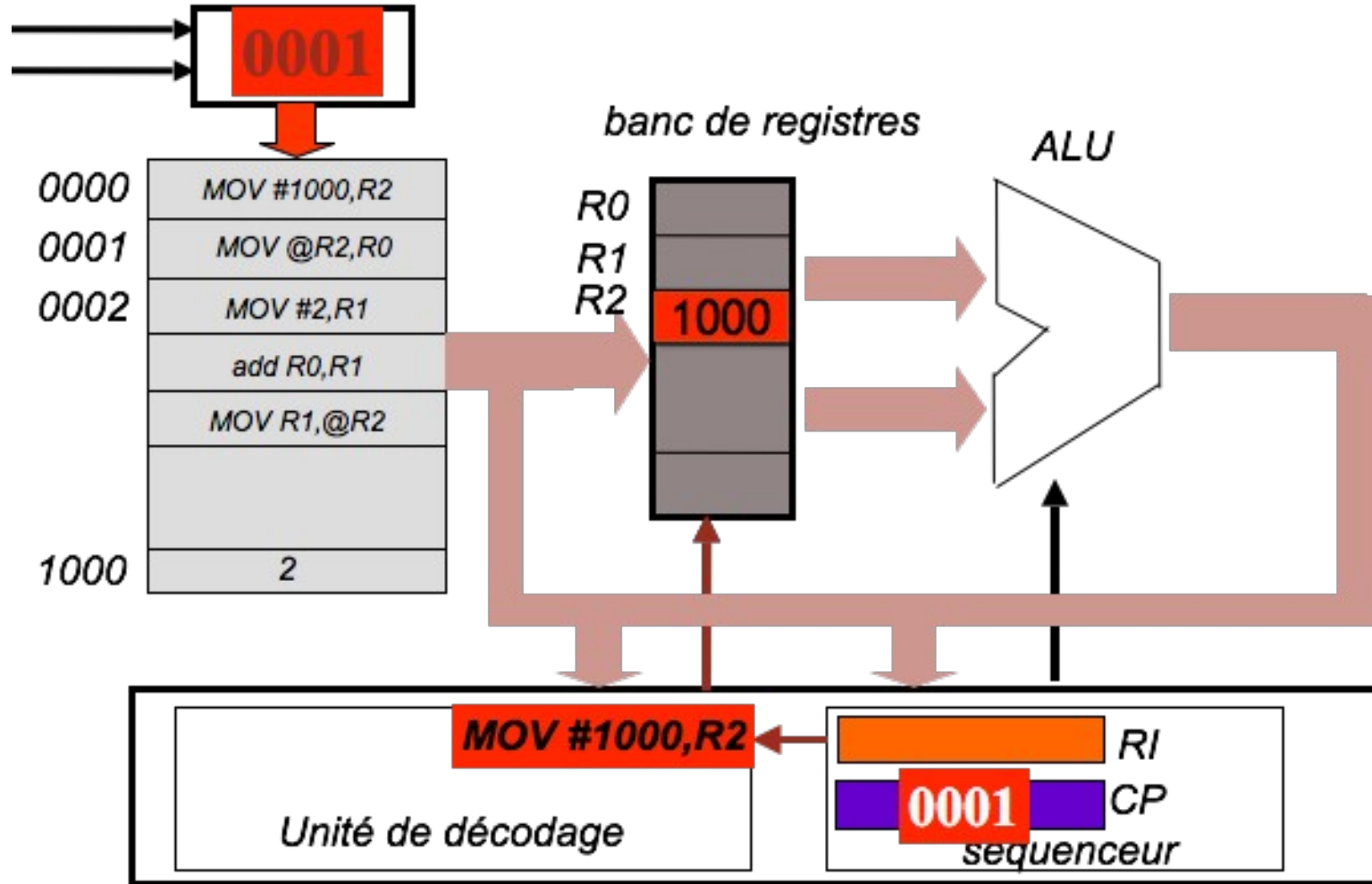
Add R0,R1

Mov R1,@R2

Le compilateur doit effectuer une allocation d'espace mémoire aux structures de données (statiques) utilisées dans le programme.

2. Cycle de vie d'une instruction

unité d'adressage



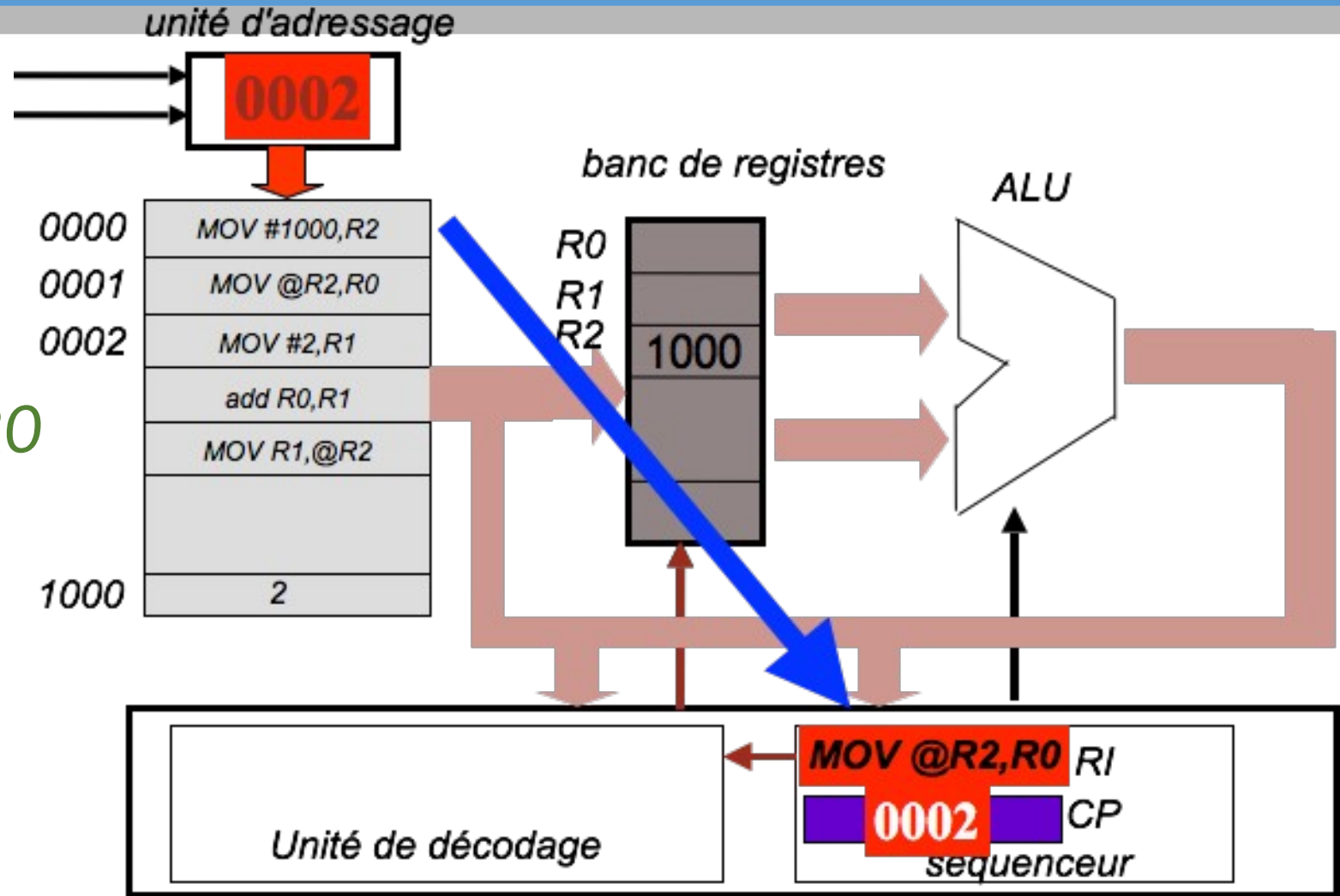
□ Résultat

Mov #Tab, R2

2. Cycle de vie d'une instruction

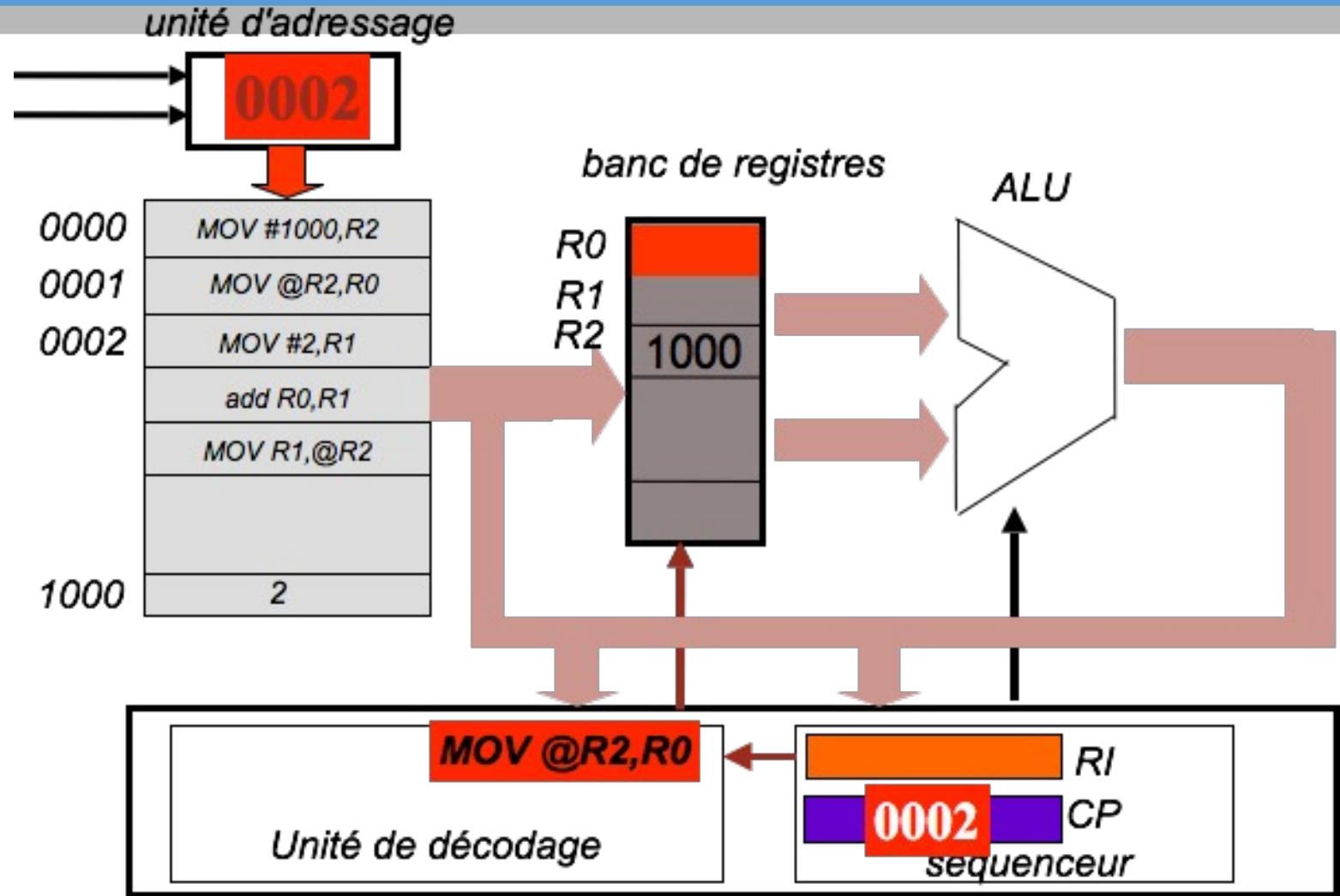
I-Fetch

Mov @R2, R0



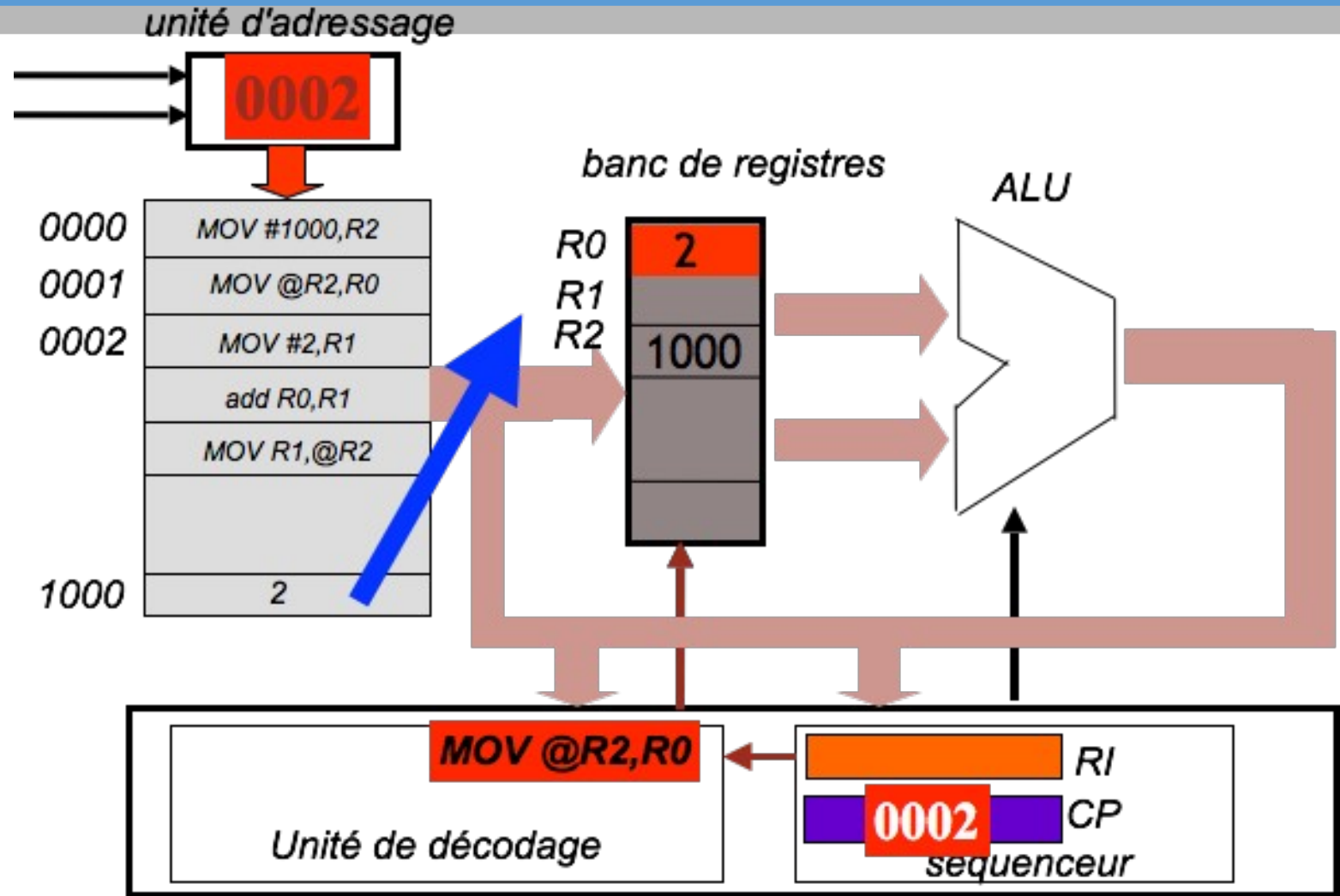
2. Cycle de vie d'une instruction

Decode



2. Cycle de vie d'une instruction

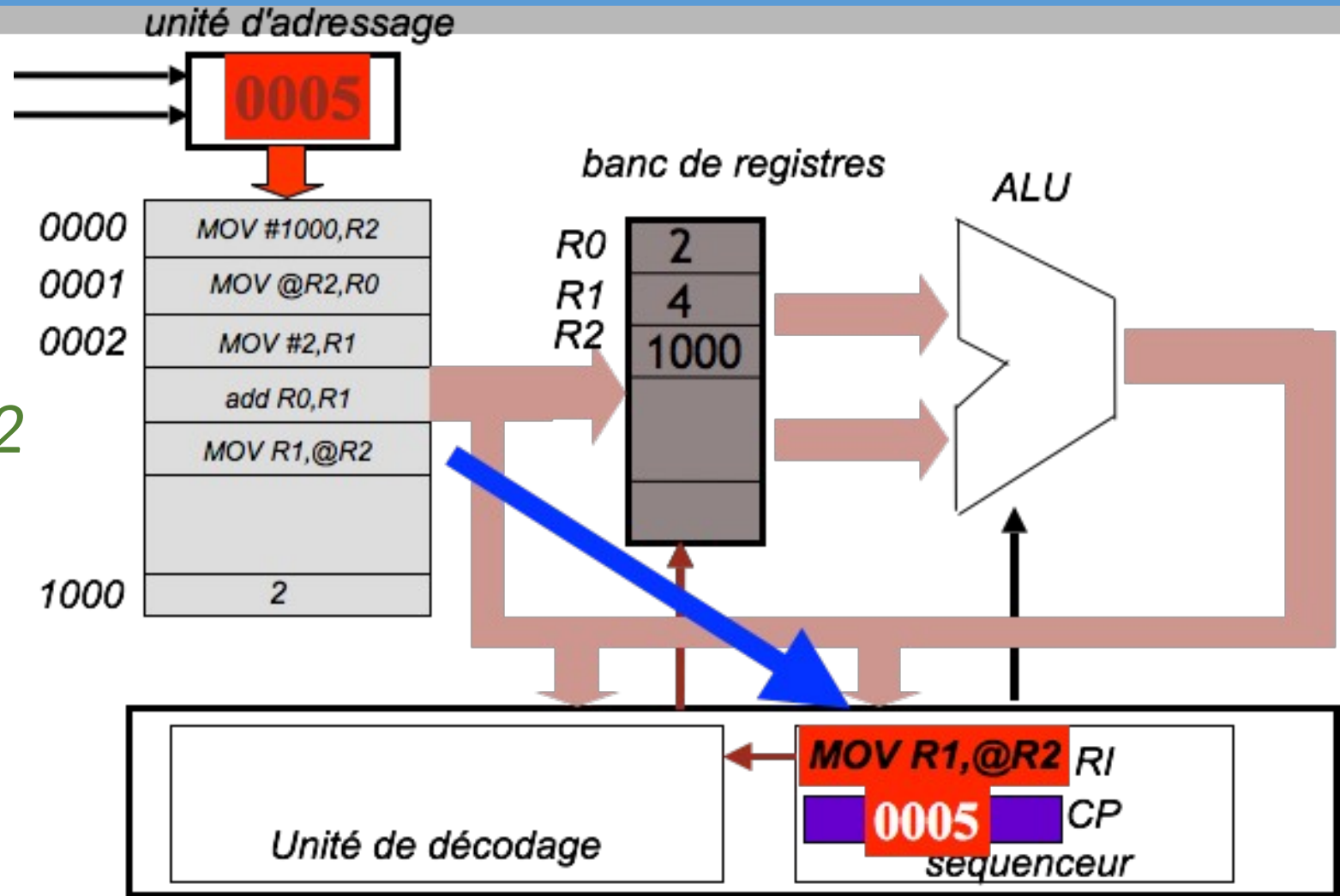
Execute



2. Cycle de vie d'une instruction

I-Fetch

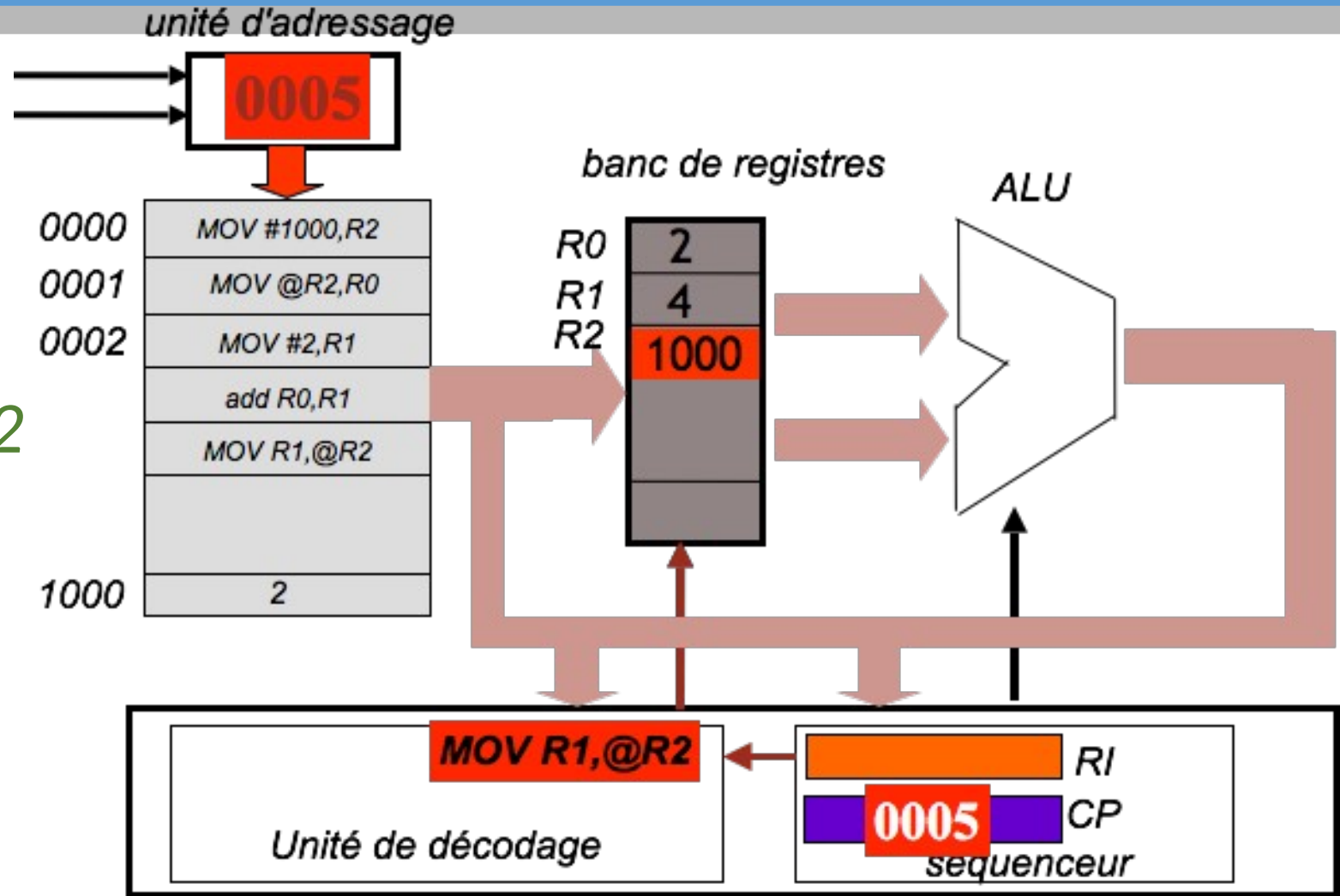
Mov R1, @R2



2. Cycle de vie d'une instruction

Decode

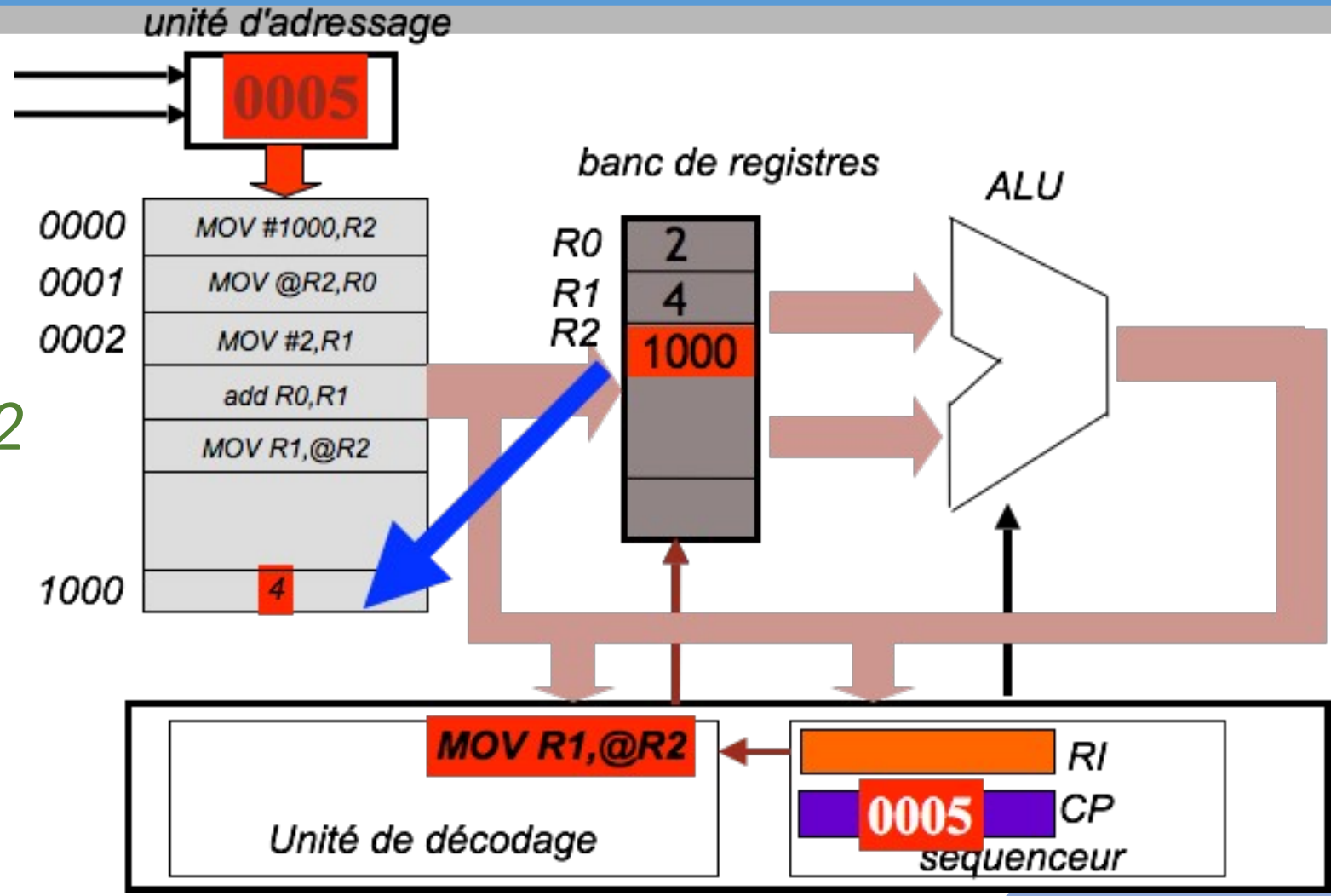
Mov R1, @R2



2. Cycle de vie d'une instruction

Execute

Mov R1, @R2



2. Cycle de vie d'une instruction

□ Cas des branchements conditionnels

Exemple de programme

Algorithme

A,B,C: integer

A<=4, B<=4

If (A !=B)

C=12

Assembleur

CMP R0,R1

JEQ suite

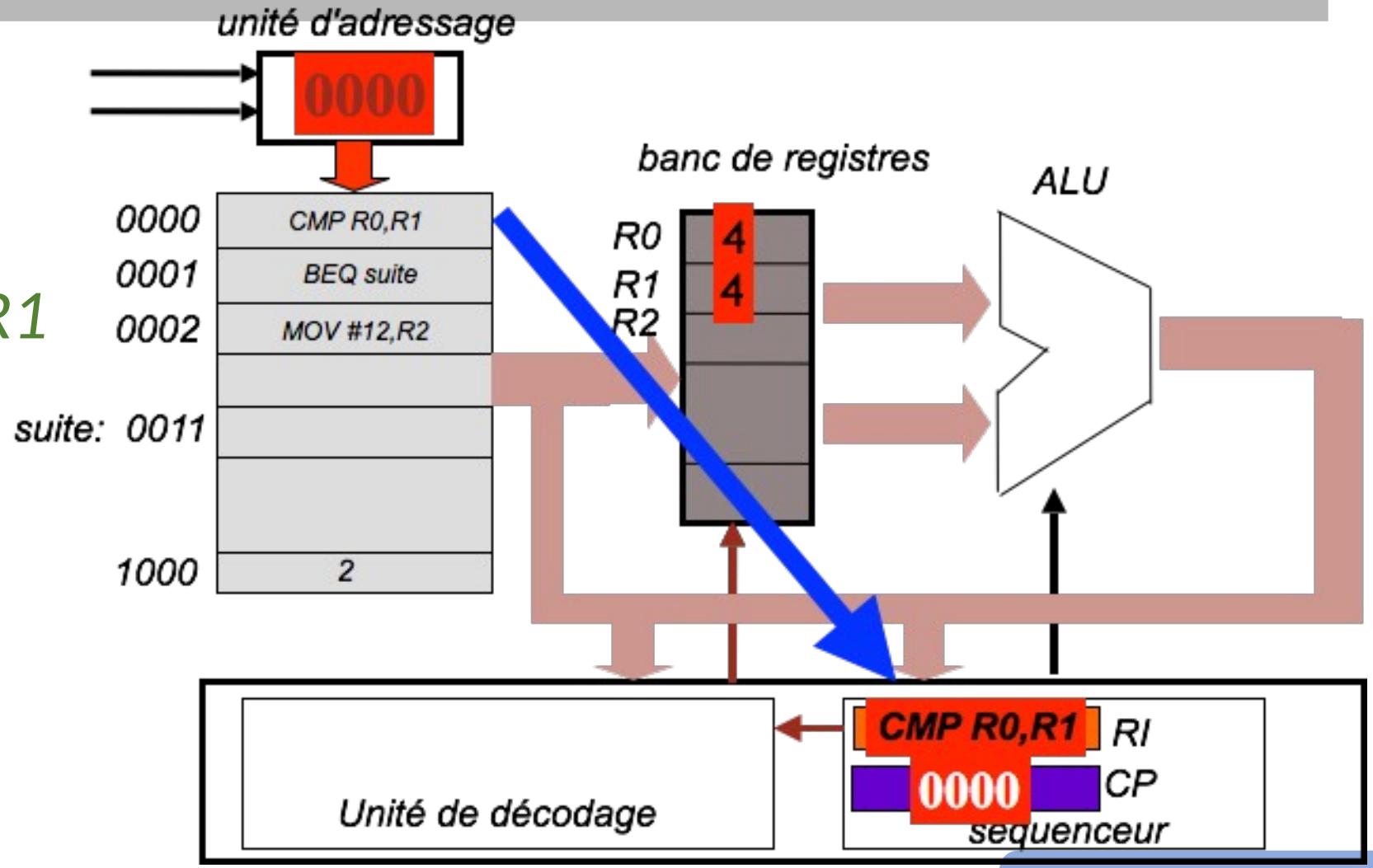
Mov #12, R2

suite : ...

Ici, le compilateur traduit des étiquettes dans le programme permettant de localiser les suites d'instructions exécutées de manière conditionnelle. De plus, il traduit le code en remplaçant la condition (A!=B) par un branchement (BEQ, JEQ,...).

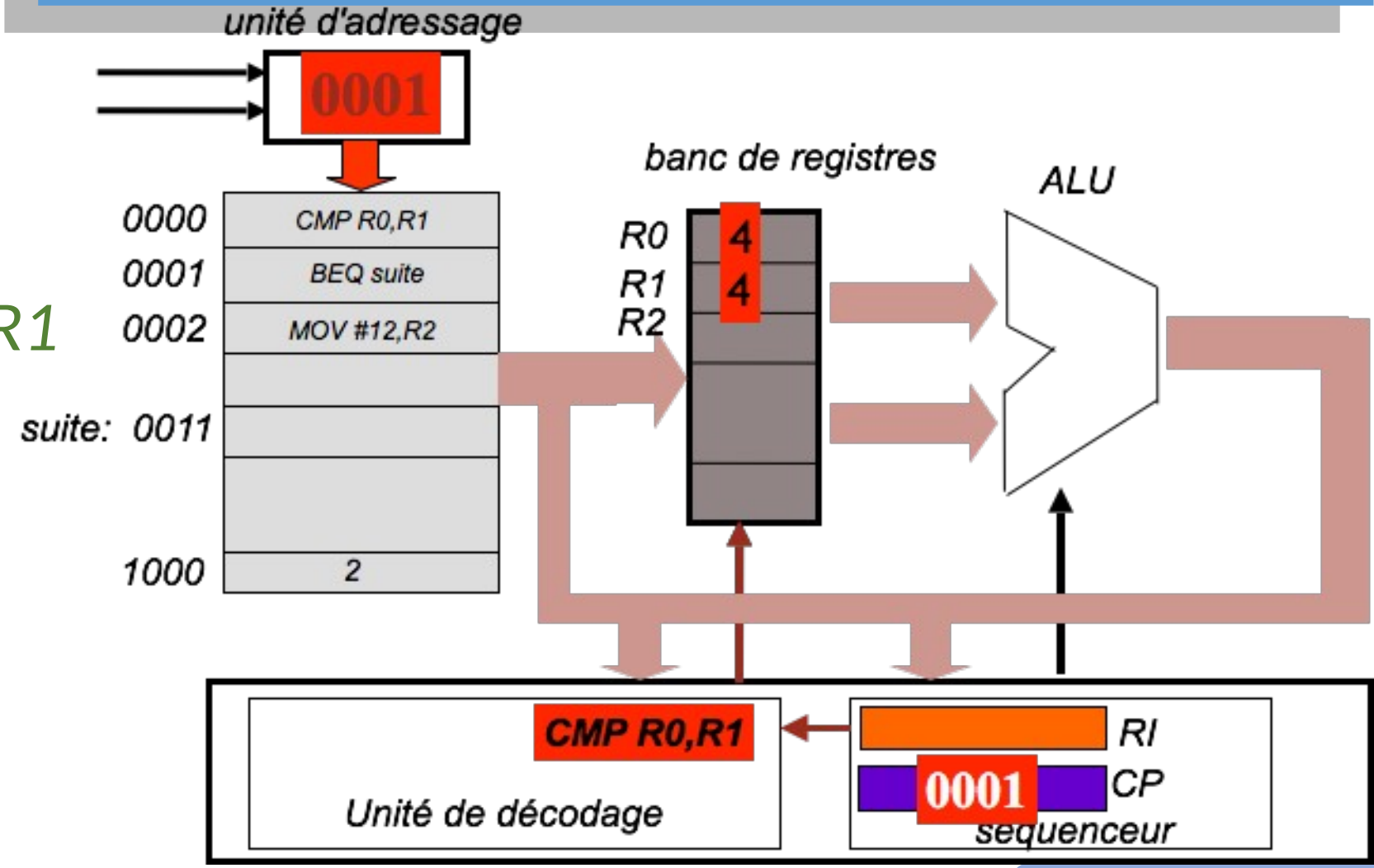
2. Cycle de vie d'une instruction

I-Fetch
CMP R0,R1



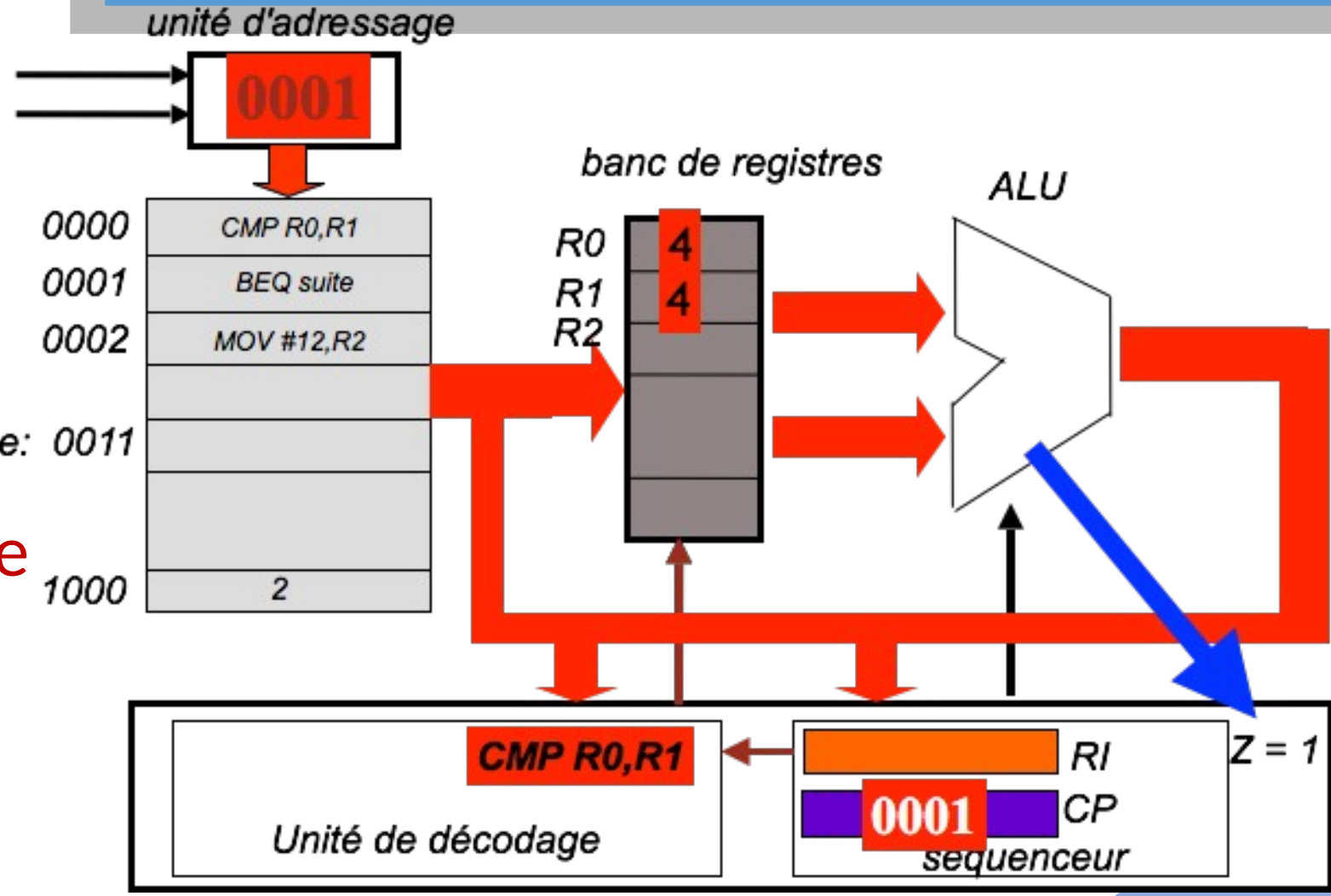
2. Cycle de vie d'une instruction

Decode
CMP R0,R1



2. Cycle de vie d'une instruction

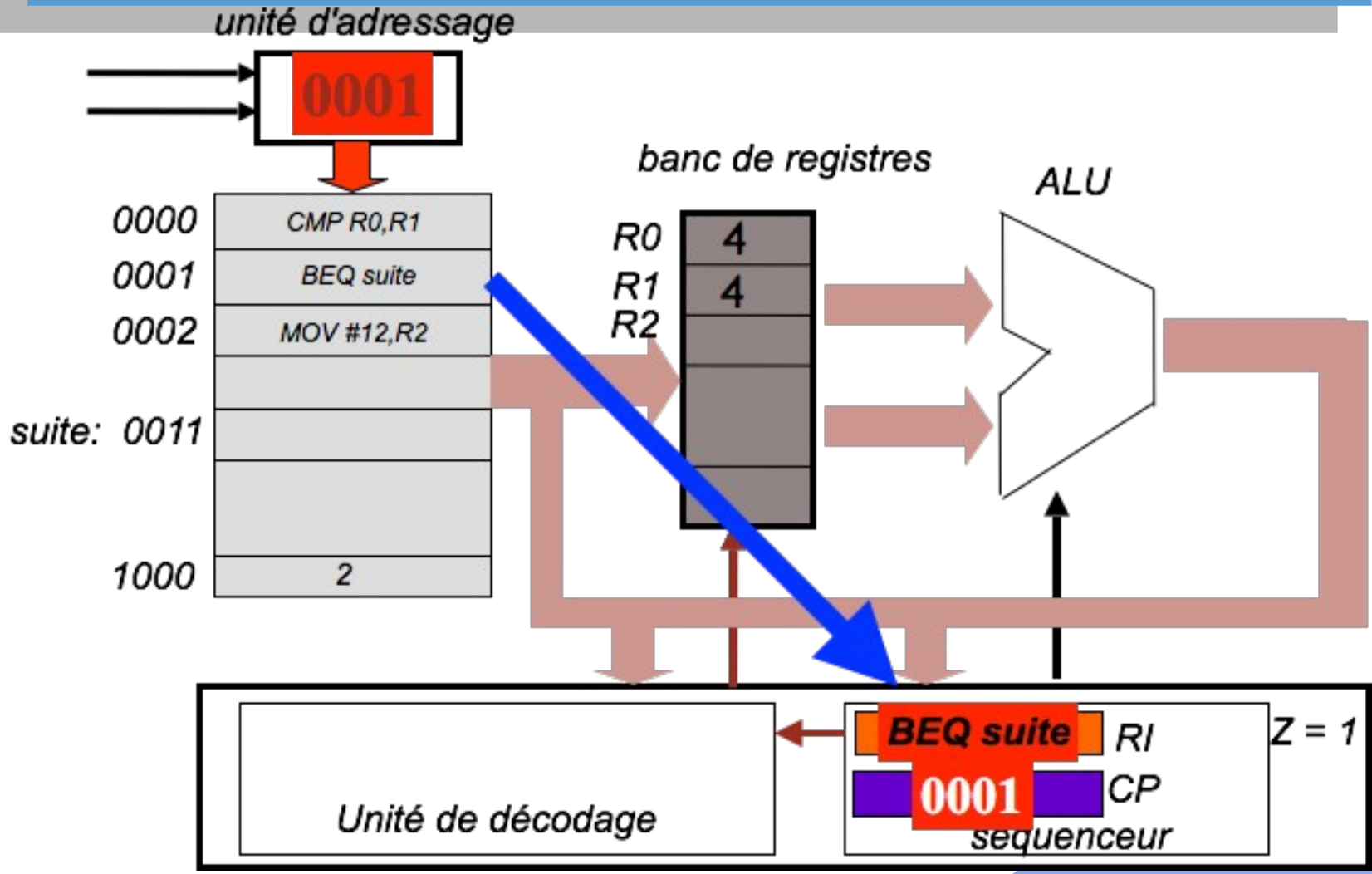
Execute



Bit Z (zéro) du
registre d'état est
mis à 1
car `CMP A,B` est
traduit par `A-B`

2. Cycle de vie d'une instruction

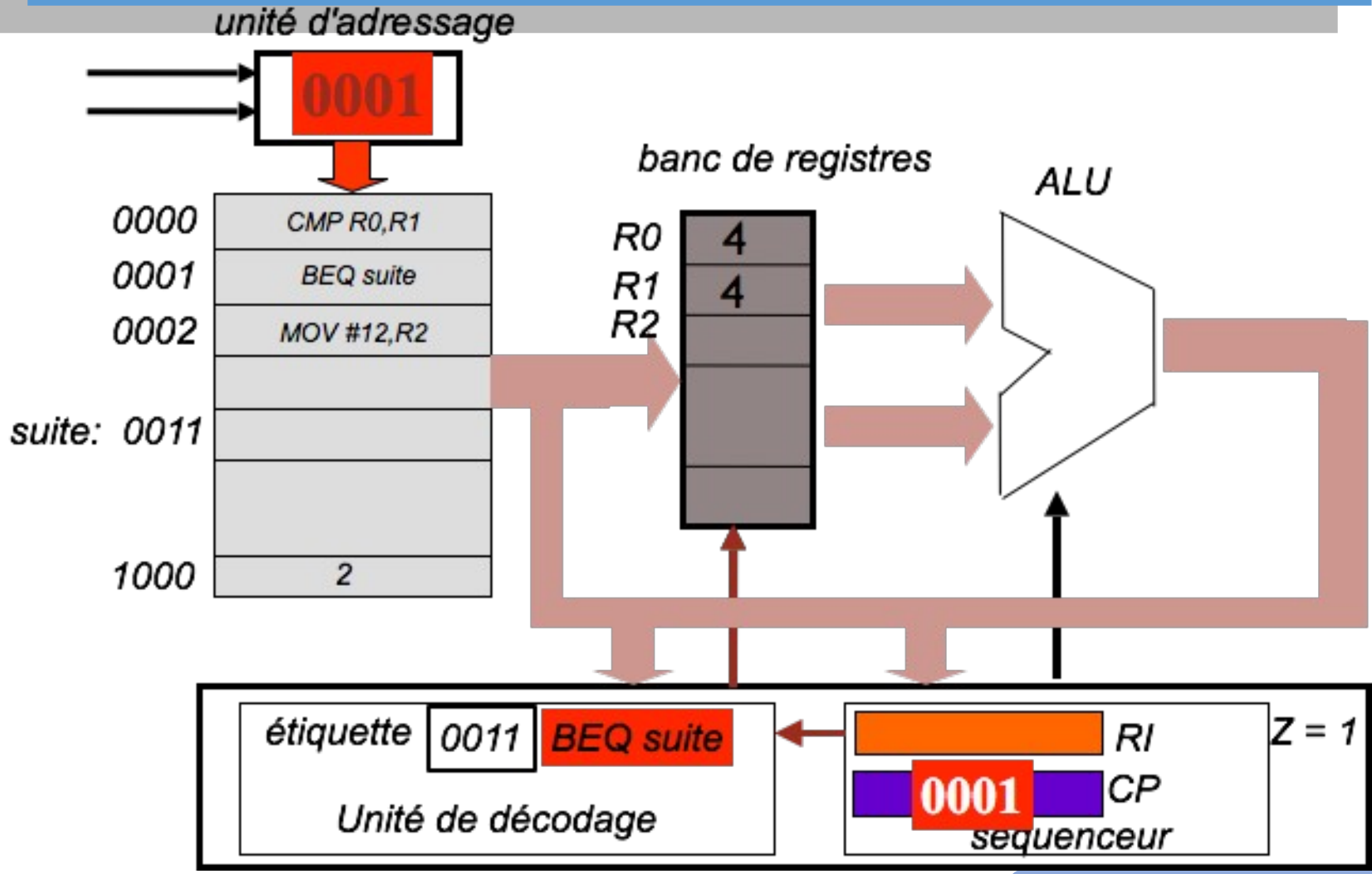
I-Fetch
JEQ suite



2. Cycle de vie d'une instruction

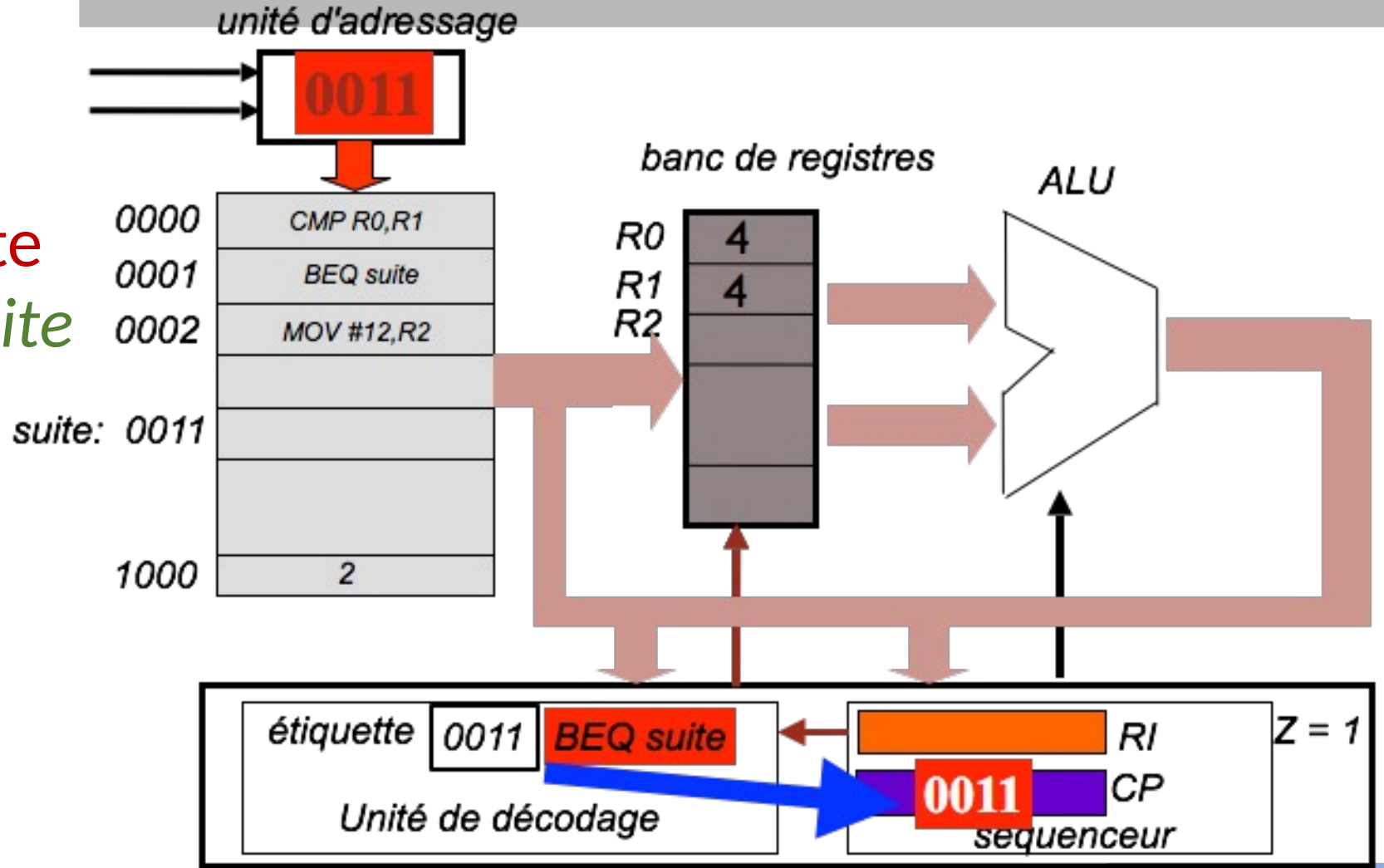
Decode

JEQ suite



2. Cycle de vie d'une instruction

Execute
JEQ suite



Modification de l'adresse contenue dans le PC

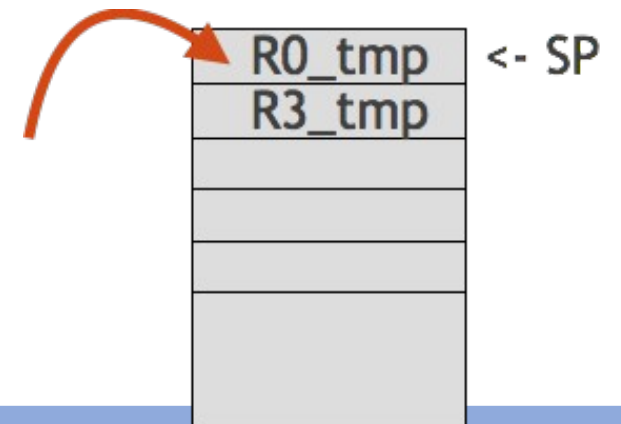
2. Cycle de vie d'une instruction

- ❑ Cas des appels sous programme
- ❑ Lors d'un appel sous-programme, le programme appelant doit sauvegarder son adresse de retour (prochaine instruction à exécuter après le retour sous-programme). Cette adresse peut être stockée dans un registre
- ❑ Dans certaines architectures, des registres dédiés sont prévus pour le passage de paramètre lors des appels et retours de sous programme

□ La pile

- La pile est un emplacement mémoire dédié pour la sauvegarde de l'état des registres
- Cet emplacement mémoire doit être connu du processeur
 - On utilise un pointeur SP (Stack Pointeur) qui stocke l'adresse du dernier mot stocké

- La pile fonctionne comme une mémoire de type LIFO
- Elle est généralement créée en RAM
- Le 'sommet' de la pile est repéré par le pointeur de pile et évolue au fil des accès



Séance 3

3. Famille de processeur

3. Famille de processeur

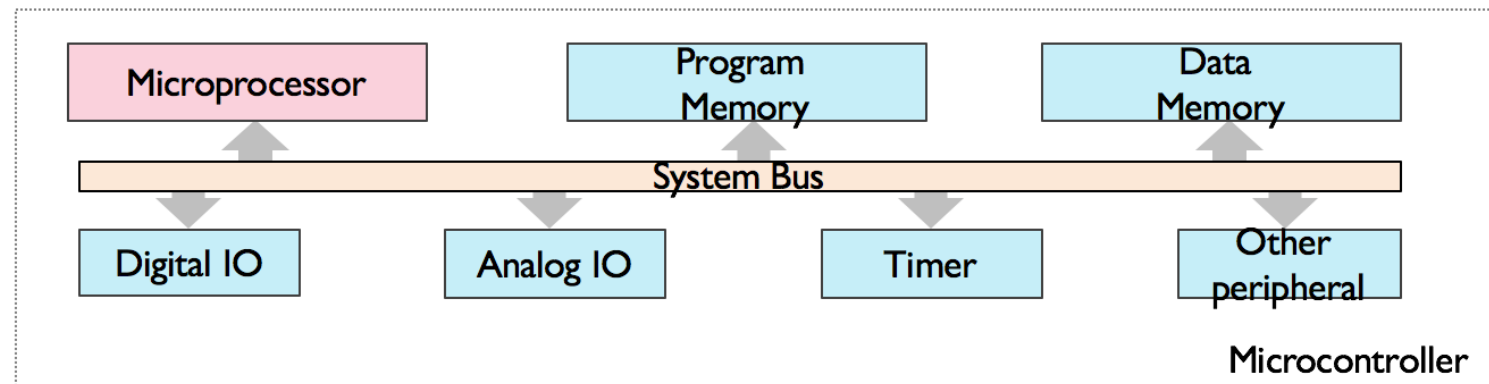
- Microcontrôleur vs Microprocesseur
- RISC vs CISC
- Parallélisme d'instructions

3. Famille de processeur

□ Microcontrôleur vs Microprocesseur

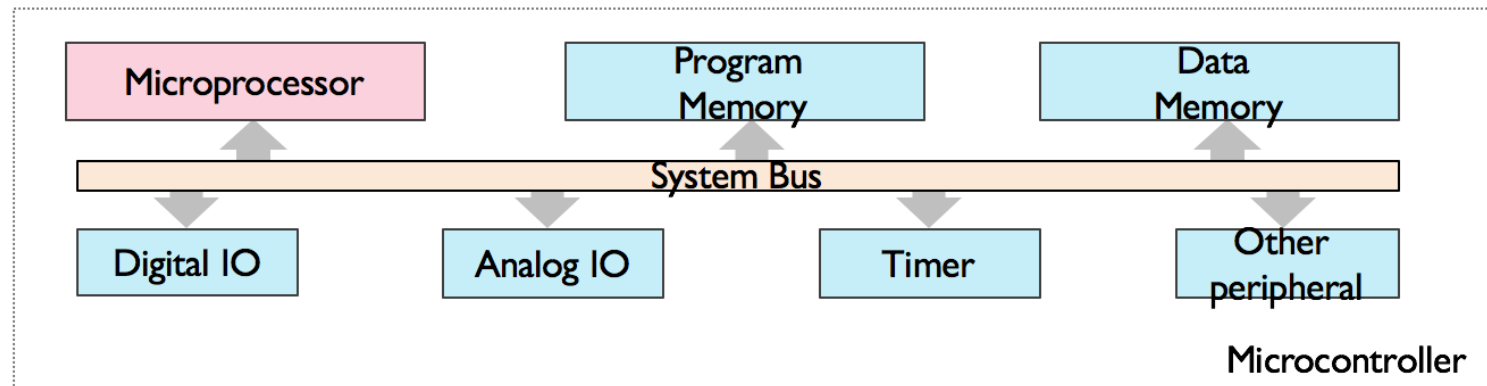
- Microcontrôleur :

- C'est un circuit intégré qui contient un microprocesseur, des mémoires, des entrées/sorties ainsi que des périphériques additionnels. Le microcontrôleur contient tous les éléments d'un mini-ordinateur sur une même puce.



3. Famille de processeur

- Microprocesseur :
 - élément clé permettant d'exécuter les instructions d'un programme. A noter qu'un microprocesseur seul ne peut fonctionner. Il a besoin de mémoires (ROM/RAM) ainsi que de périphériques extérieurs.



□ Architecture RISC vs CISC

- **CISC : *Complex Instruction Set Computers***

Jeu étendu d'instructions complexes où chaque instruction peut effectuer plusieurs opérations élémentaires (chargement mémoire, opération arithmétique,...). Le temps d'exécution varie en fonction de la taille et le format de l'instruction.

- **RISC : *Reduced Instruction Set Computers***

Jeu d'instruction réduit avec uniquement des instructions simples pouvant effectuer une opération élémentaire. Chaque instruction prend en général 1 cycle d'horloge et elles sont codées avec le même nombre de bits.

□ Architecture RISC vs CISC

- Comparatif :

CISC (ex: x86)	RISC (ex: PowerPC)
Multiples instructions de tailles et formats différents	Instructions simples, en nombre réduit et de taille fixe
Utilisation de moins de registres	Couteux en registres
Beaucoup de modes d'adressage	Nombre limité de modes d'adressage
Compilation simple	Compilation complexe
Temps d'exécution d'instruction variant	Temps d'exécution fixe (1 cycle)

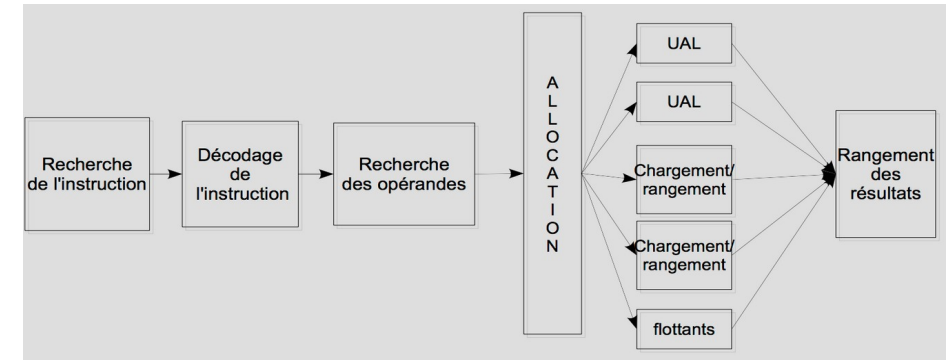
Pour RISC, le compilateur doit fournir un effort supplémentaire pour optimiser le code (allocation des registres, optimisations des boucles, du pipeline, du choix d'instruction...)

3. Famille de processeur

□ Parallélisme d'instructions

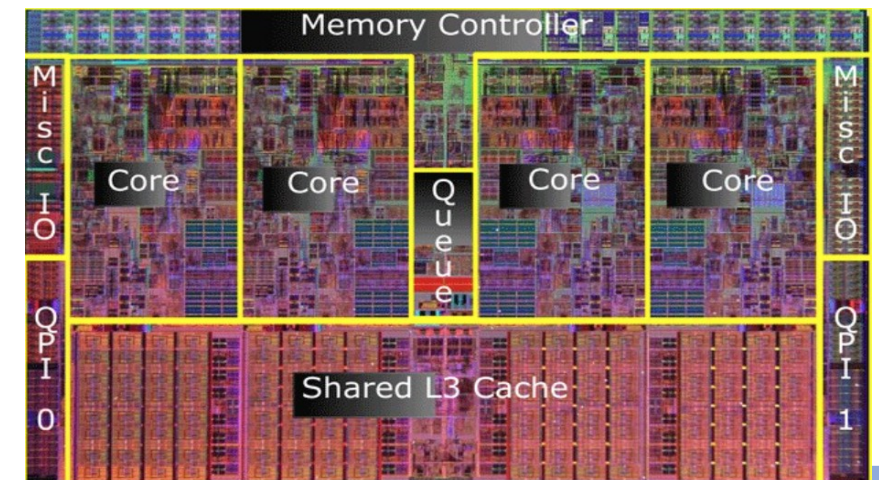
- **Processeurs superscalaires**

Ce type de processeurs cherche à exploiter le code de manière efficace en déterminant quelles instructions peuvent être exécutées en parallèle. L'objectif étant d'augmenter le nombre d'instructions exécutées par cycle



- **Le multi-cœurs**

Idée : n cœurs = n architectures superscalaires (le plus souvent identiques)



O.Romain & J.Lorandel
Màj: F.Ghaffari & S.Zuckerman - 2019

3. Famille de processeur

- **Processeurs VLIW (Very Long Instruction Word)**

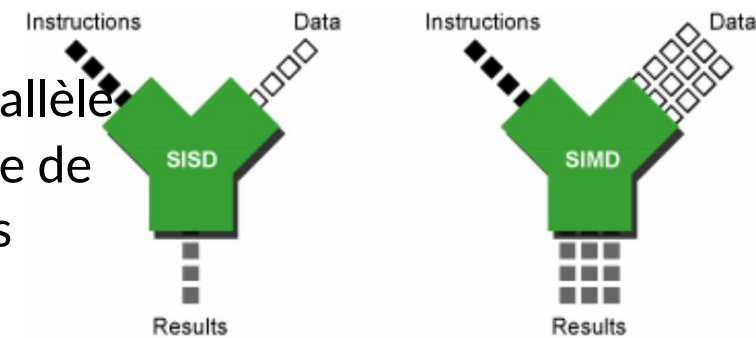
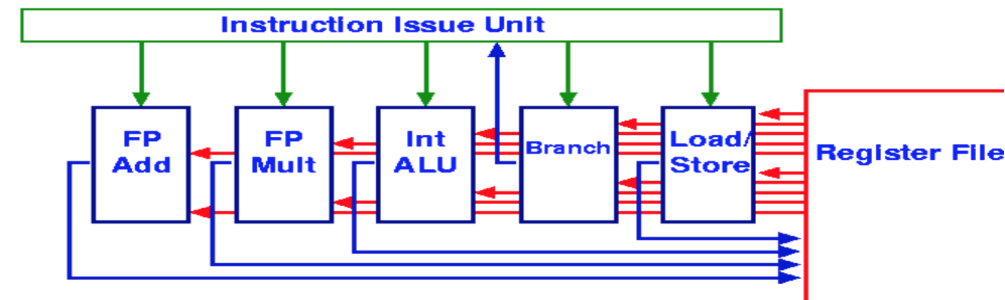
Les processeurs VLIW possèdent une architecture permettant de manipuler des instructions codées sur 128 bits ou plus. Ces instructions sont en réalité des suites d'instructions élémentaires qui seront exécutées en parallèle.

□ **Parallélisme de données**

- **Architecture SIMD :**

Ce type d'architecture permet d'exécuter une instruction en parallèle sur plusieurs données. Les données sont alors regroupées sous forme de blocs de taille fixe appelés vecteurs. Des langages ou bibliothèques dédiés peuvent être utilisés pour faciliter le travail du compilateur (CUDA, OpenCL...)

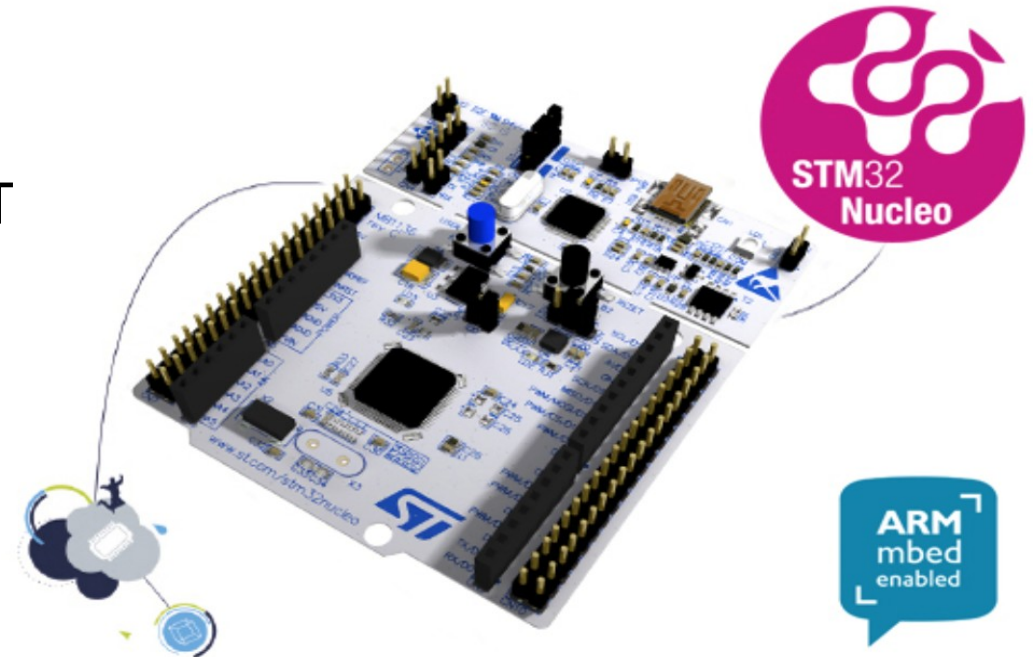
Instruction Format



3. Famille de processeur

□ Pour les travaux pratiques, la plateforme que nous utiliserons est la **nucleo board 64** qui contient un microcontrôleur **STM32F4xx** de STMicroelectronics (ST)

- Architecture du cœur : Cortex-M4 d'ARM
- Périphériques et plateforme conçus par ST
 - Connecteurs Arduino, ST Morpho
 - 3 Leds, 2 boutons poussoirs, ...
- Programmation/Debug via ST-LINK/V2-1
- Compilateur en ligne Mbed possible
- Prix ≈ 12\$



3. Famille de processeur

□ Environnements possibles de développement

- **µVision de Keil**

- Utilisé en TP

- Version gratuite mais limitée en taille de code (32ko)

- Possède un très bon débbuger !!

- Coocox

- IAR

- Compilateur en ligne avec Mbed

□ OS disponibles

- PowerPac (IAR)

- FreeRTOS

- uClinus sur Cortex-M4

- ARTX-ARM (Keil)

- C/OS-333 (Micrium)

- embOS (Segger)

3. Famille de processeur

ARM

- Conçoit et vend des architectures de processeurs RISC depuis les années 80 (pas de circuit) :

« An ARM architecture is a set of specifications regarding the instruction set, the execution model, the memory organization and layout, the instruction cycles and more, which describes precisely a machine that will implement said architecture » [Mastering STM32, Carmine Noviello]

ARM Cortex Family



Cortex-A

Highest performance
Optimized for rich operating systems

Application



Cortex-R

Fast response
Optimized for high-performance, hard real-time applications

Real-Time



Cortex-M

Smallest/lowest power
Optimized for discrete processing and microcontroller

eMbedded



SecurCore

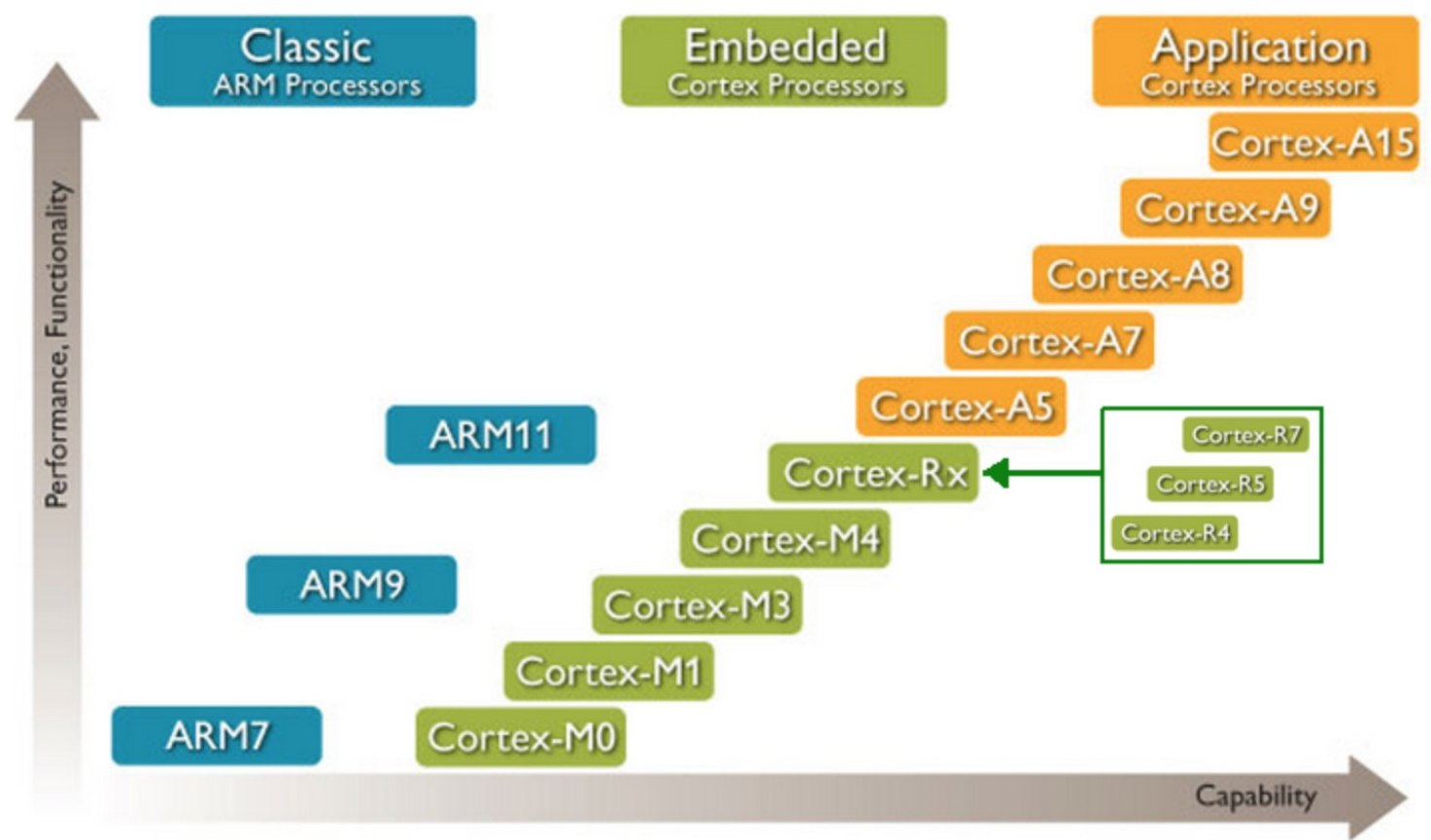
Tamper resistant
Optimized for security applications

Security

[source :
arm.com]

3. Famille de processeur

La famille Cortex d'ARM

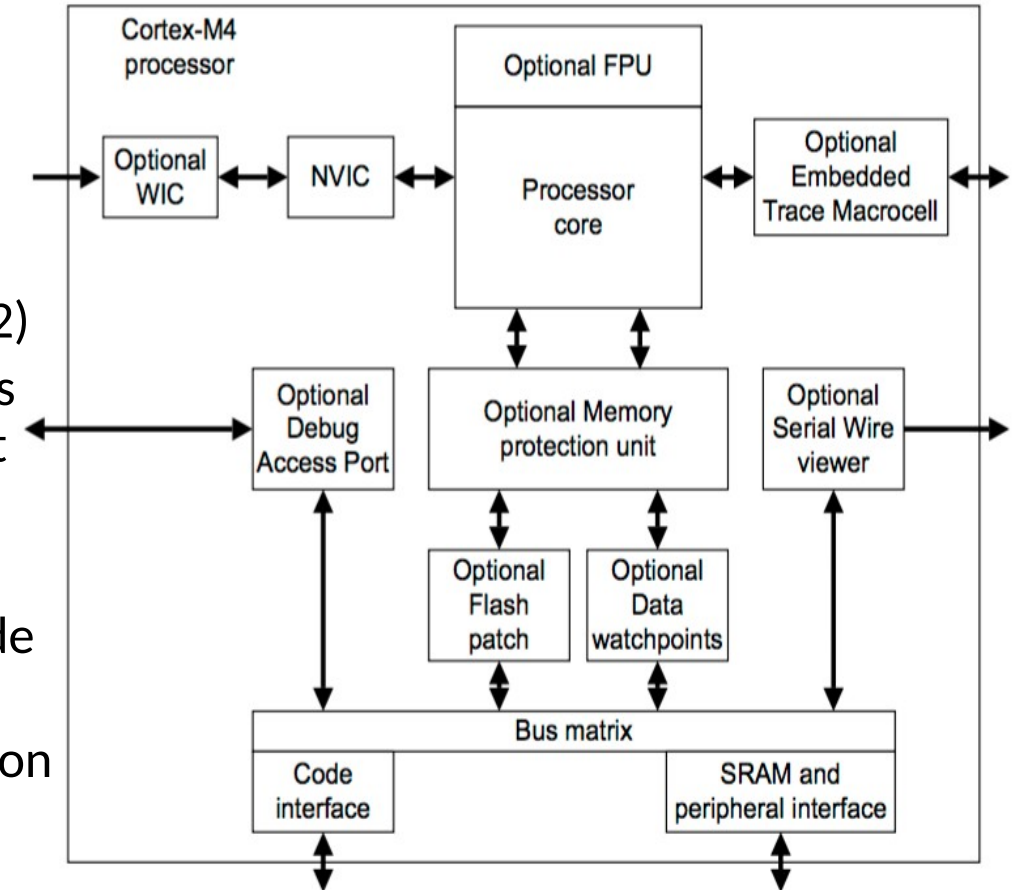


<http://www.emcu.it/>

3. Cortex M4

Cortex-M4

- Architecture de processeur RISC 32 bits
- Architecture de Harvard (bus de données et d'instructions ≠),
- 3 étages de pipeline
- 2 Jeux d'instructions sur 32/16 bits (ARM/Thumb2)
- Cortex-M4F : contient une unité de traitement des nombres flottants (FPU) + capacités de traitement (DSP)
- Contrôleur d'interruption (NVIC) à faible latence de traitement
- Une unité de protection de mémoire (MPU) -option
- Une solution faible coût de debug



□ Cortex-M4 : Pipeline 3 étages

- Rappel : 3 étapes pour exécuter une instruction (Fetch/Decode/execute)
 - **Fetch** : recherche de l'instruction en mémoire
 - **Decode** : décodage de l'instruction
 - **Execute** : exécution de l'instruction (+ rangement du résultat en mémoire)
- Quels avantages par rapport à une architecture séquentielle ? Combien de temps pour exécuter 2 instructions ? Quels problèmes (branchement, rupture de pipeline) ?

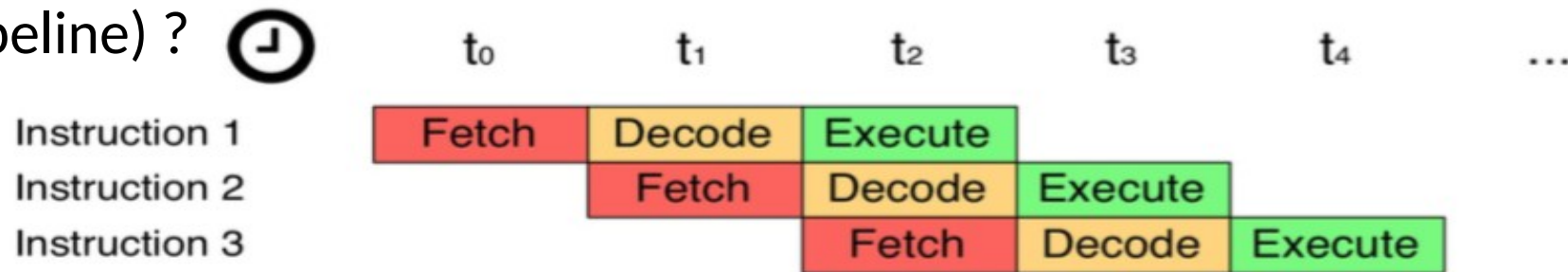
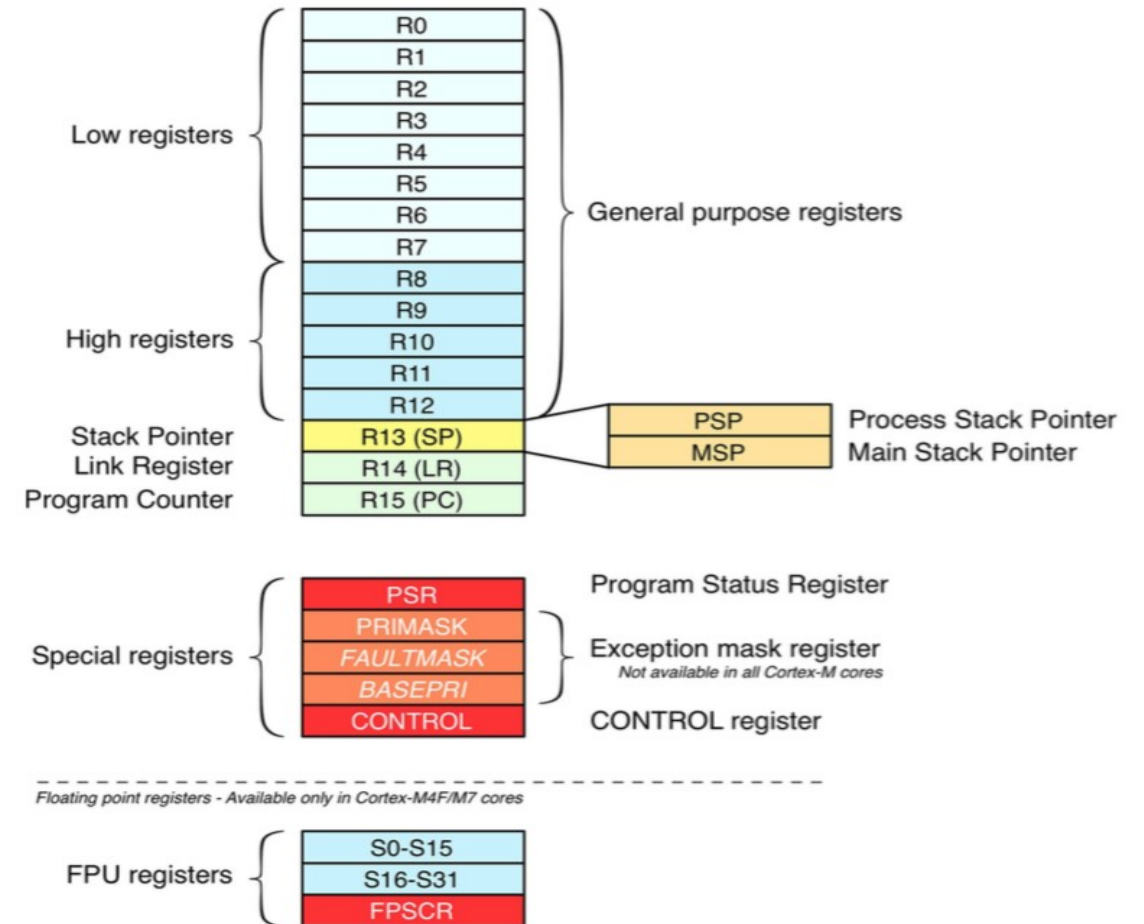


Figure 8: Three stage instruction pipeline

Les registres du Cortex-M4

- 13 registres généraux 32 bits –R0 à R12
- Des registres spécifiques
 - Compteur Programme (PC)
Contient l'@ de l'instruction suivante à exécuter
Au reset, initialisé avec l'adresse de base du *reset handler*
 - Pile (SP)
Pointeur de pile contenant l'adresse du sommet (Pile généralement initialisée en SRAM)
 - Link Register (LR) :
Contient l'@ de retour lors d'appel à sous-programme
- Registres spéciaux :
 - PSR : registre de *status* (flags d'ALU, gestion + flags d'interruption et d'exception)
 - Control : utilisation de FPU,...

[Source : Mastering STM32, Carmine Noviello]

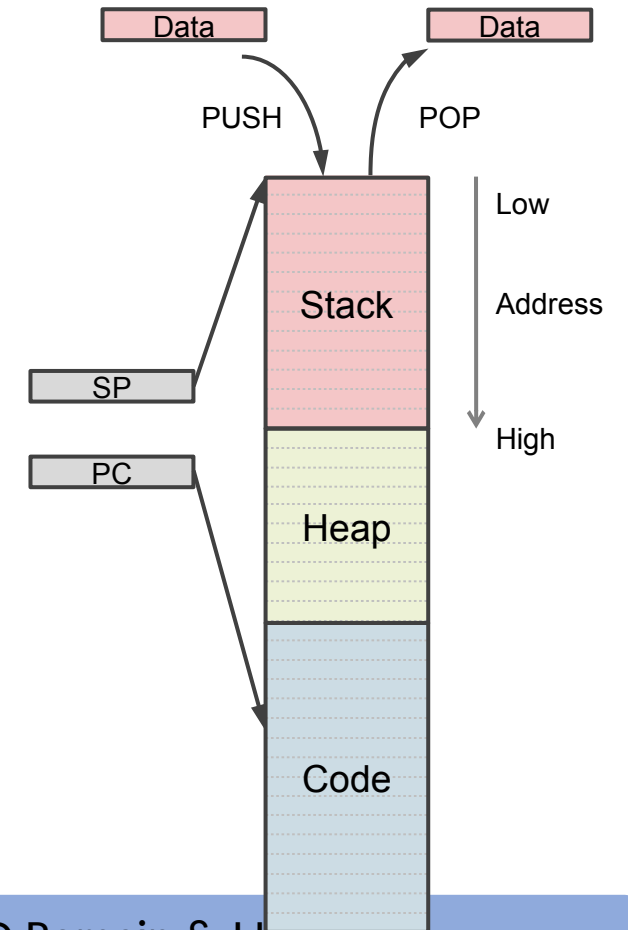


□ La pile et le pointeur de Pile SP (R13)

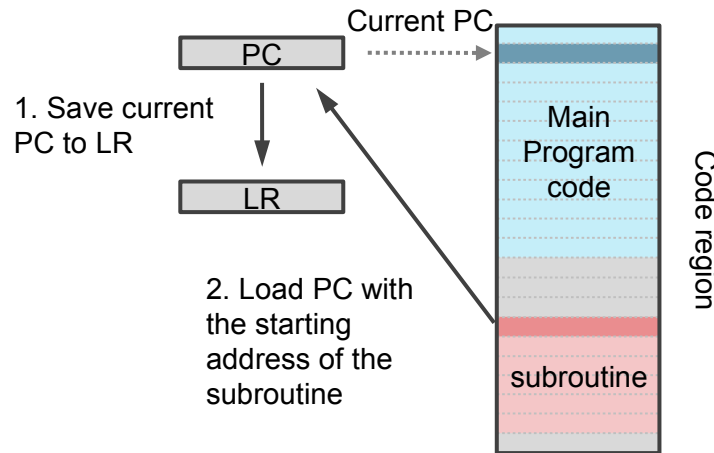
- Contient l'adresse courante du sommet de la pile
- Utiliser pour préserver le contexte d'un programme lors de changement de tâches (appels sous programme)
- Cortex M-4 a deux SPs : Main SP est utilisé dans des applications qui ont un accès privilégié (OS etc) et Process SP est utilisé dans des applications n'utilisant pas de gestionnaire exception.

□ Program Counter (PC)

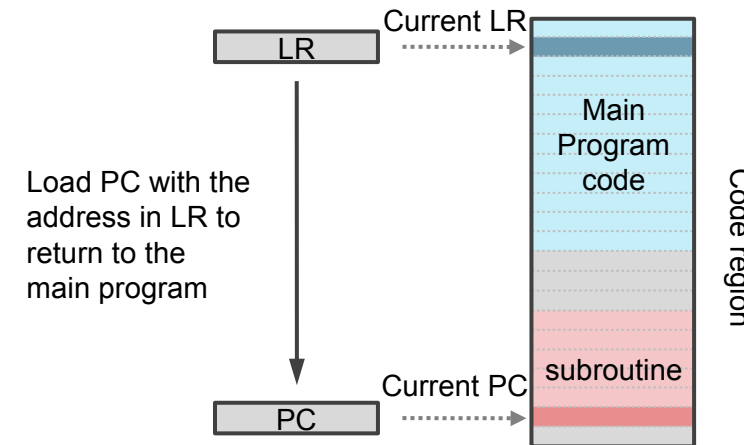
- Contient l'adresse de la prochaine instruction à exécuter
- Est automatiquement incrémenté de 4 à chaque opération (instructions de 32 bits) à l'exception des branchements
- Une opération de branchement (appels de fonctions,...) modifie la valeur de PC tandis que l'adresse courante est sauvegardé dans le Link Register (LR)



- R14: Link Register (LR)
 - Le registre de lien permet de stocker l'adresse de retour lors d'appel d'un sous programme ou d'une fonction
 - Le compteur programme (PC) récupère la valeur du LR une fois la fonction terminée



Call a subroutine



Return from a subroutine to the main program

Le registre de *status*

- Program Status Register

Combine 3
sous-registres
exclusifs

Application Program Status Register (APSR)
Interrupt Program Status Register (IPSR)
Execution Program Status Register (EPSR)

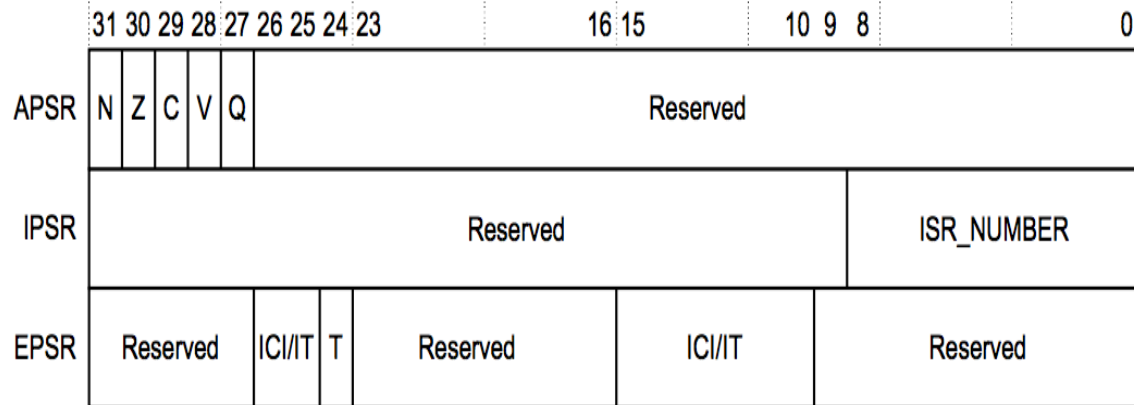


Table 2.4. APSR bit assignments

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	DSP overflow and saturation flag
[26:20]	-	Reserved
[19:16]	GE[3:0]	Greater than or Equal flags. See SEL for more information.
[15:0]	-	Reserved

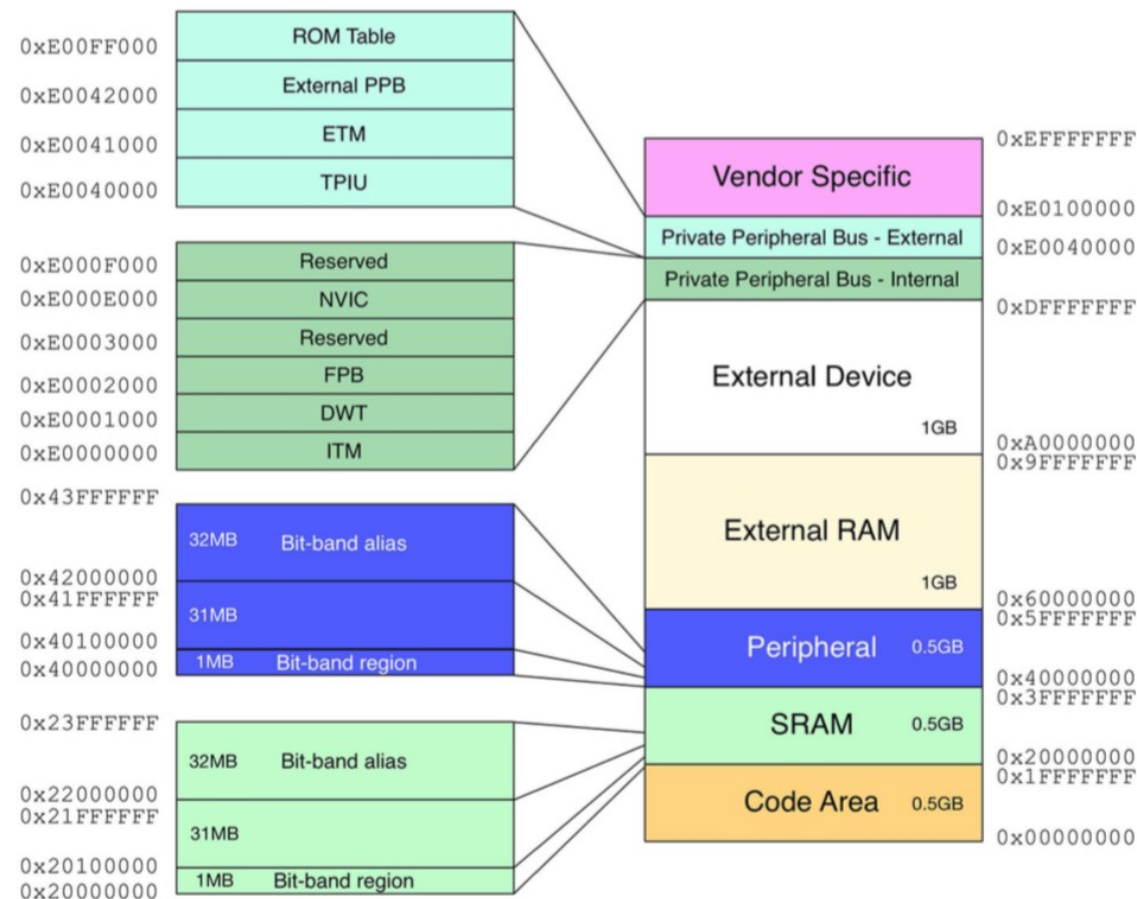
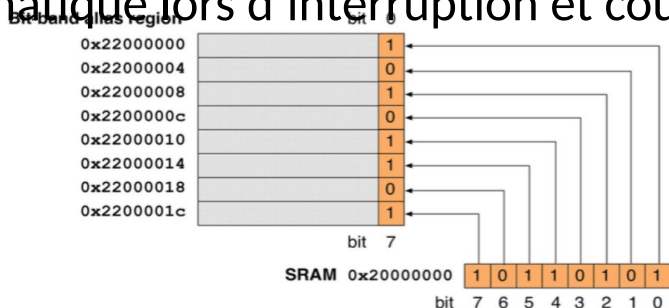
- ISR_Number : contient le numéro de l'exception
 - 0 -> Thread Mode, 3 -> HardFault, 16 -> IRQ0, ..
- ICI/IT : bits indiquant l'exécution d'instruction multiples (Load ou store multiples) lors d'interruption (*Interruptible-Continuable (ICI) et If-Then (IT)*)

3. Cortex M4

[Source : Mastering STM32, Carmine Noviello]

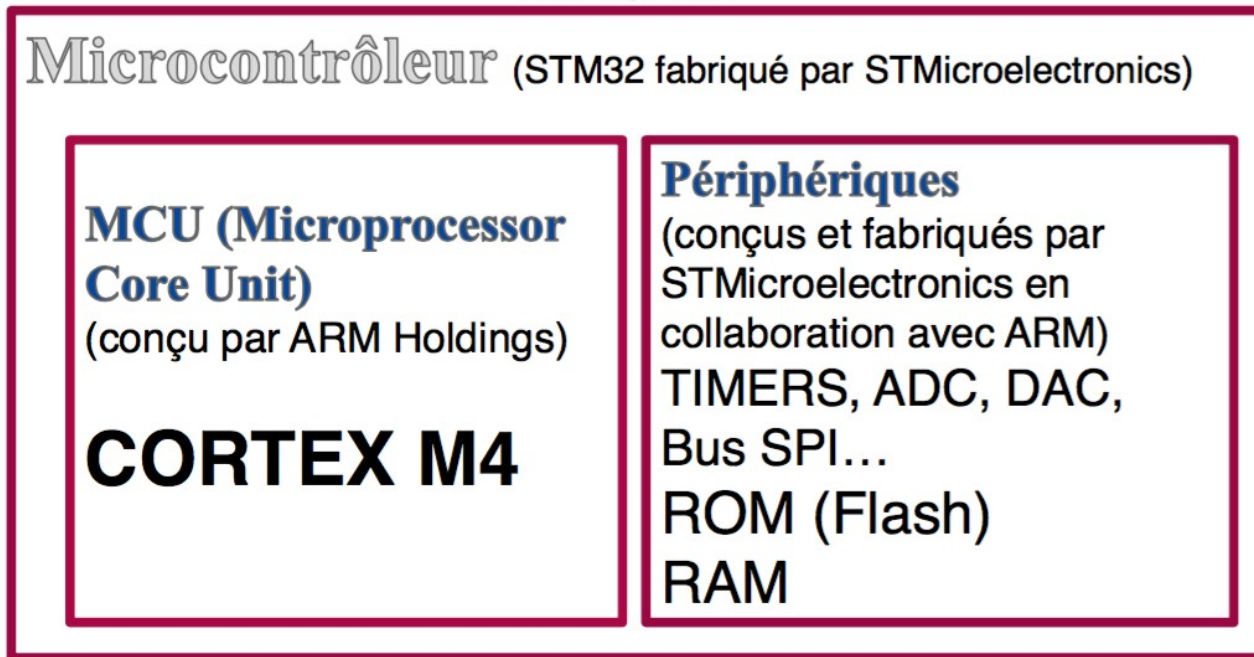
Mapping Mémoire

- 4 Gio de mémoire totale adressable pour tous les Cortex-M (portabilité du code de différents fabricants)
- Spécification physiques par chaque fabricant
- Bit-band region and alias region
 - Permet d'écriture/lire la valeur d'un bit en 1 cycle machine
 - Evite le procédé classique de lecture d'un octet / modification avec masque / écriture (3 cycles) qui est problématique lors d'interruption et couteux



3. STM32F4xx

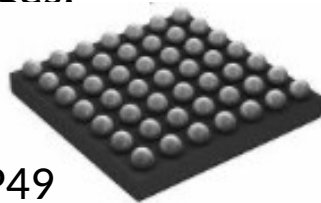
□ Positionnement du Core ARM dans le microcontrôleur STM32



Principales caractéristiques du STM32F401RE

- Core ARM Cortex-M4: Fréquence maximale de 84MHz
- Mémoires : up to 512Ko Flash // up to 96Ko SRAM
- Operating Voltage : 1.7 to 3.6V
- Plusieurs sources d'horloge :
 - 4-26 MHz crystal oscillator
 - Internal 16MHz RC (default CPU clock on reset)
 - 32kHz, ...
- Consommation: Run (146uA/MHz), StandBy (2.4uA) + Deep power down mode
- Up to 11 Timers : génération de PWM,...
- Up to 81 I/O ports (5V-tolerant)
- Interface de communication (I2C, USART, SPI, USB...)
- Les types de packages:

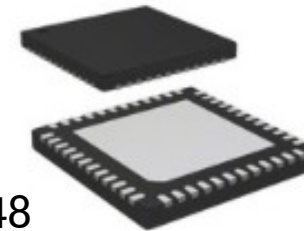
WLCSP49



LQFP64/100



UFQFPN48



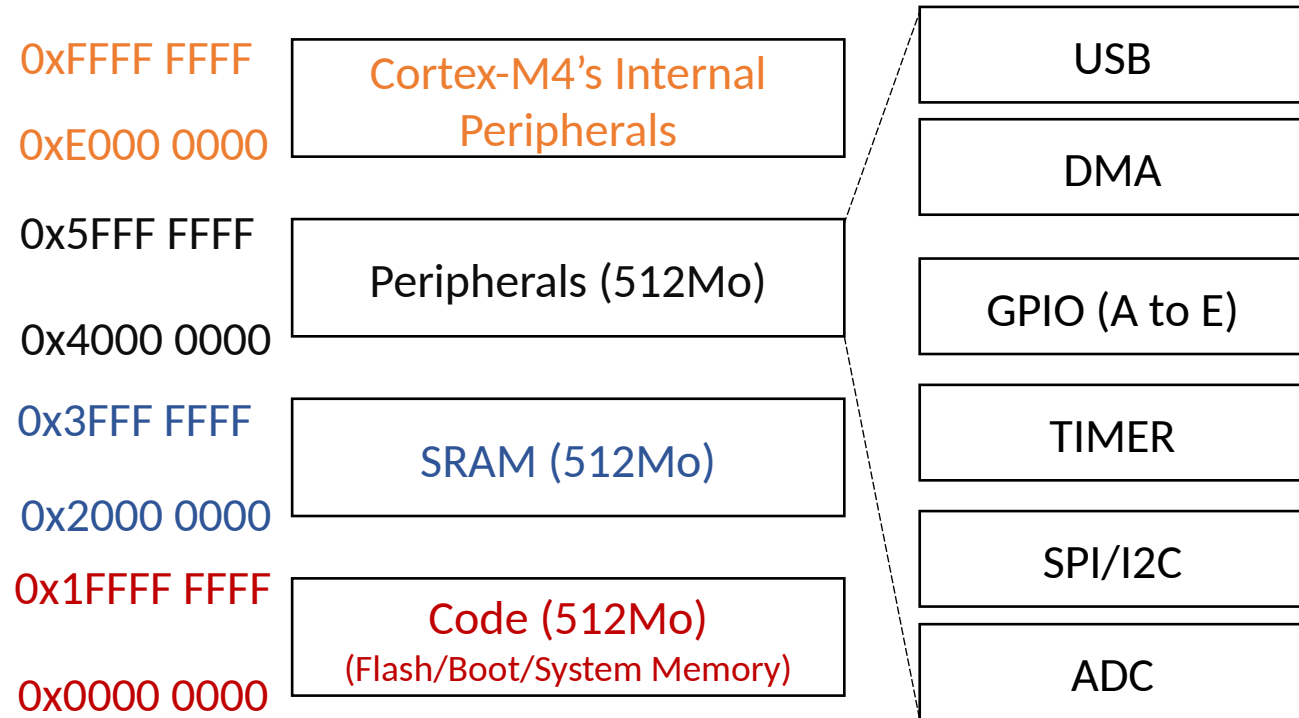
UFBGA100



[Source : STM32F401xD STM32F401xE datasheet p51/52]

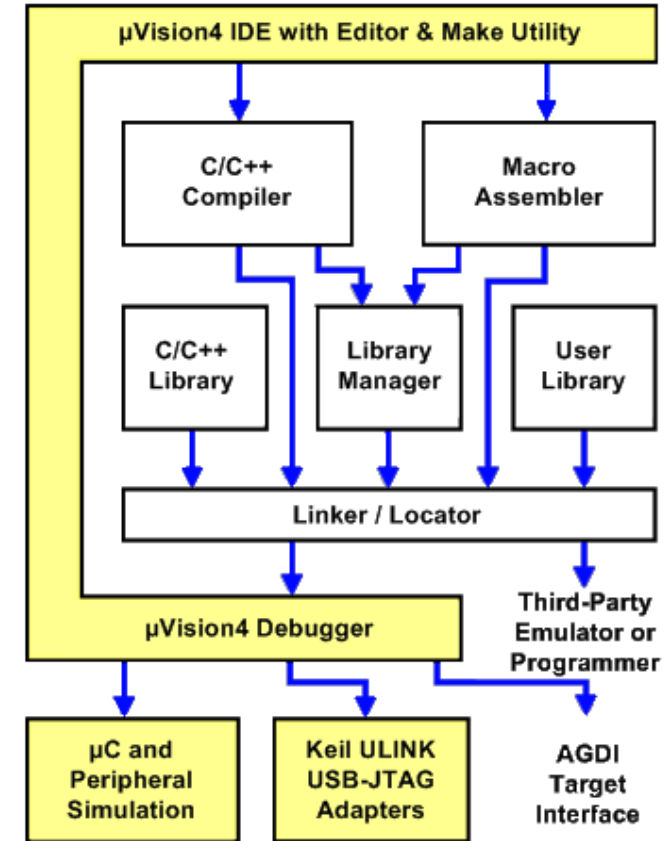
Mapping Mémoire du STM32F4xx

- Spécifications des tailles des zones mémoires par chaque fabricant (ici ST)
- Les périphériques sont contrôlés et échangent avec l'uc via une zone mémoire dédiée
 - Contrôle par lecture/écriture de registres dédiés



□ Cycle de développement logiciel

- L'IDE μ Vision intègre:
 - Project manager (configuration wizard..)
 - μ Vision Editor (*completion*, vérification de syntaxe dynamique..)
- C/C++ Compilateur et assembleur de macro
 - Fichiers sources -> fichiers objets (code machine)
- Library manager
 - Permet la création de fichiers objets
- Linker / Editeur de lien
 - Création de l'exécutable (.hex) à partir des fichiers objets et des bibliothèques pré-compilées
 - code fixé en mémoire / cible précise
- Debugger
 - Incluant un simulateur (microprocesseur + périphériques)



keil.com

O.Romain & J.Lorandel
Màj: F.Ghaffari & S.Zuckerman - 2019

IDE

The screenshot shows the Keil IDE interface with the following components labeled:

- Project Name:** LPCWithSFRs - µVision
- Execution Profiler:** Performance Analyzer window showing a call stack with 'main' at 183.898 ms.
- Text Editor:** Disassembly window showing assembly code for 'main' with instructions like 'LSL R0, R0, #0' and 'MOV R5, R14'.
- Logic Analyzer:** Logic Analyzer window showing a signal trace graph.
- Register Window:** Registers window showing the current state of registers R0 through R15.
- Command Line:** Command window at the bottom left with assembly code for button configuration.
- Command Window:** Command window at the bottom left showing 'LA 'AIN0''.
- Call Stack & Memory & Locals Window:** Call Stack window at the bottom right showing the current function 'main' at address 0x00000210.
- Symbols Window:** Symbols window at the bottom right showing the symbol table.
- Commands Available:** Command window at the bottom left showing available commands like 'ASSIGN BreakDisable BreakEnable'.
- Peripheral Window:** Peripheral window at the bottom left showing GPIO0 configuration.
- Instruction Trace:** Instruction Trace window showing a list of instructions with addresses and opcodes.
- Status Bar:** Status bar at the bottom right showing 't1: 0.18391577 sec'.

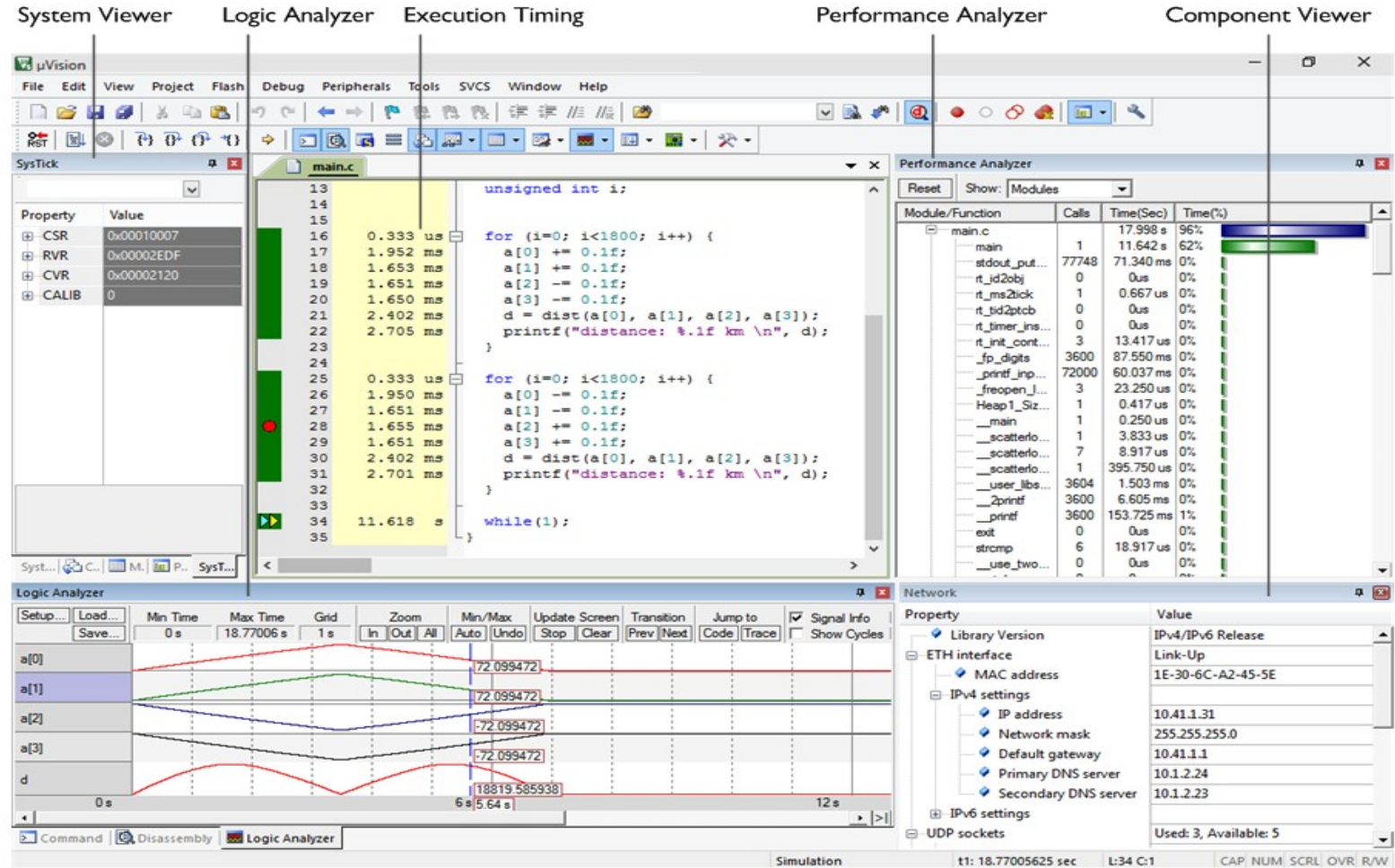
keil.com

O.Romain & J.Lorandel
Maj: F.Ghaffari & S.Zuckerman - 2019

3. Outils de développement SW

Le debugger

- Permet de tester, vérifier et optimiser le code de l'application
- 2 modes de fonctionnement :
 - **Simulator Mode** : vérification purement logicielle -> Pas besoin de cible matérielle
 - **Target Mode** : vérification via connexion avec la cible matérielle
- Breakpoints/Watch supportés



The screenshot displays the IAR Embedded Workbench IDE with several tool windows open:

- System Viewer:** Shows hardware registers like CSR, RVR, CVR, and CALIB with their current values.
- Logic Analyzer:** Shows a timing diagram for variables a[0], a[1], a[2], a[3], and d, with values like 72.099472 and 18819.585938.
- Execution Timing:** Shows a list of code lines with their execution times, such as 0.333 us for line 16 and 11.618 s for line 34.
- Performance Analyzer:** Shows a table of module/function calls and their execution times, with 'main' accounting for 96% of the time.
- Component Viewer:** Shows network settings for the ETH interface, including IP address (10.41.1.31), network mask (255.255.255.0), and default gateway (10.41.1.1).

□ En résumé

- Evolution considérable des microprocesseurs depuis les dernières décennies
- Omniprésence du cœur ARM dans les microcontrôleurs de nouvelle génération
 - Utilisation en TP du μ C STM32F401RE basé sur le cœur ARM Cortex-M4
 - Utilisation de l'environnement de développement μ Vision de Kiel
- Pour les séances suivantes, on abordera le mécanisme d'interruption ainsi que le contrôle de périphériques essentiels d'un microcontrôleur, pour mettre un œuvre un objet connecté

□ Pour commencer à jouer : *STM32CubeMX tool*

- Utilitaire graphique (gratuit) permettant de générer le code d'initialisation du processeur, des périphériques (internes/externes), de l'arbre d'horloge...
- Il génère aussi le projet, incluant la librairie de drivers HAL (Hardware Abstraction Layer)

