POLYTECH®
DIJON

2024/2025

Architecture interne des ordinateurs

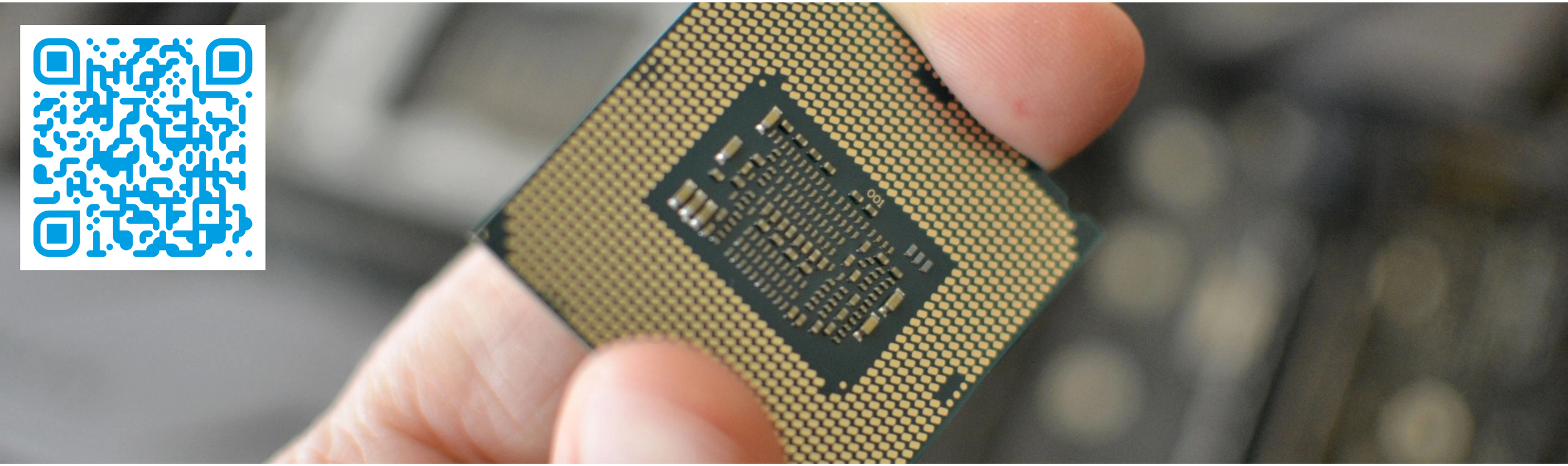
Pr. Dominique Ginhac
dginhac@u-bourgogne.fr

Dernière mise à jour le 04 Dec. 2024 par dG

Chapitre #03

Architecture d’un processeur

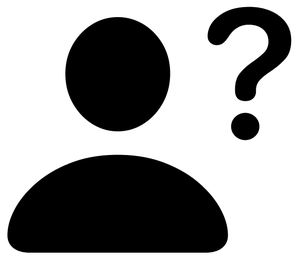
https://ginhac.com/archi/03-architecture.pdf



Découvrir l'architecture des processeurs, c'est plonger au cœur du fonctionnement des technologies numériques.

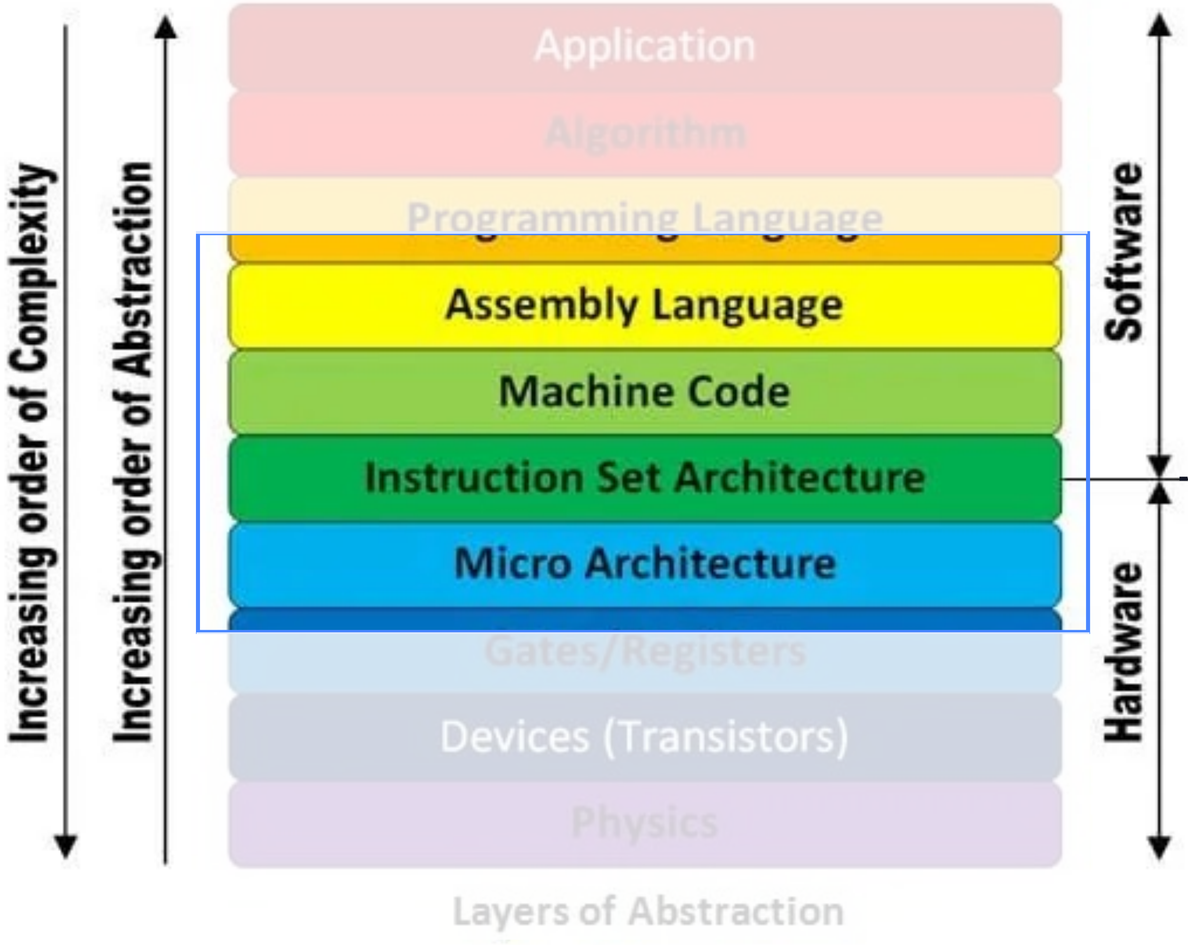
Enjoy!

ARCHITECTURE D'UN PROCESSEUR



C'est quoi ?

Un empilement de couches



Indépendant de la machine

Langages de haut niveau (C, C++, Java, Python, …)

Nécessite une chaine de compilation ou d'interprétation

Dépendant de la machine

Vue abstraite du processeur accessible via le jeu d'instructions (ISA) - Interface entre le matériel et le logiciel

---

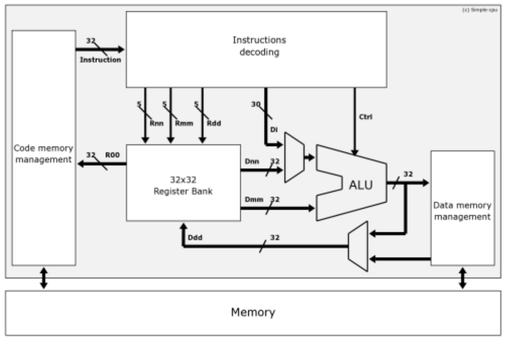
Figé dans le silicium

Micro Architecture matérielle (UAL, opérations, stockage, …)

Portes logiques CMOS réalisant des fonctions combinatoires et/ou séquentielles constituées d'assemblage de transistors

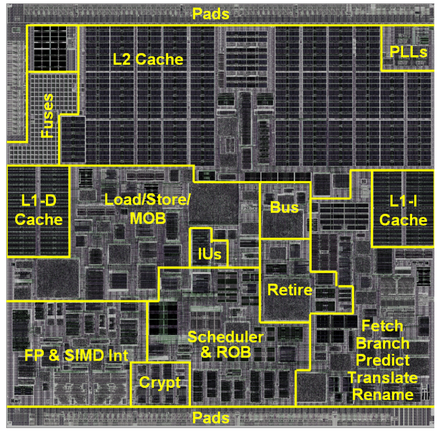
Microélectronique

Architecture vs Microarchitecture ?



Architecture : Vue abstraite d'un processeur donnée au programmeur, accessible par le jeu d'instructions.

Micro Architecture : Ensemble des circuits électroniques constituant un processeur (nombre de transistors, technologie, consommation électrique, fréquence de l'horloge, ...)

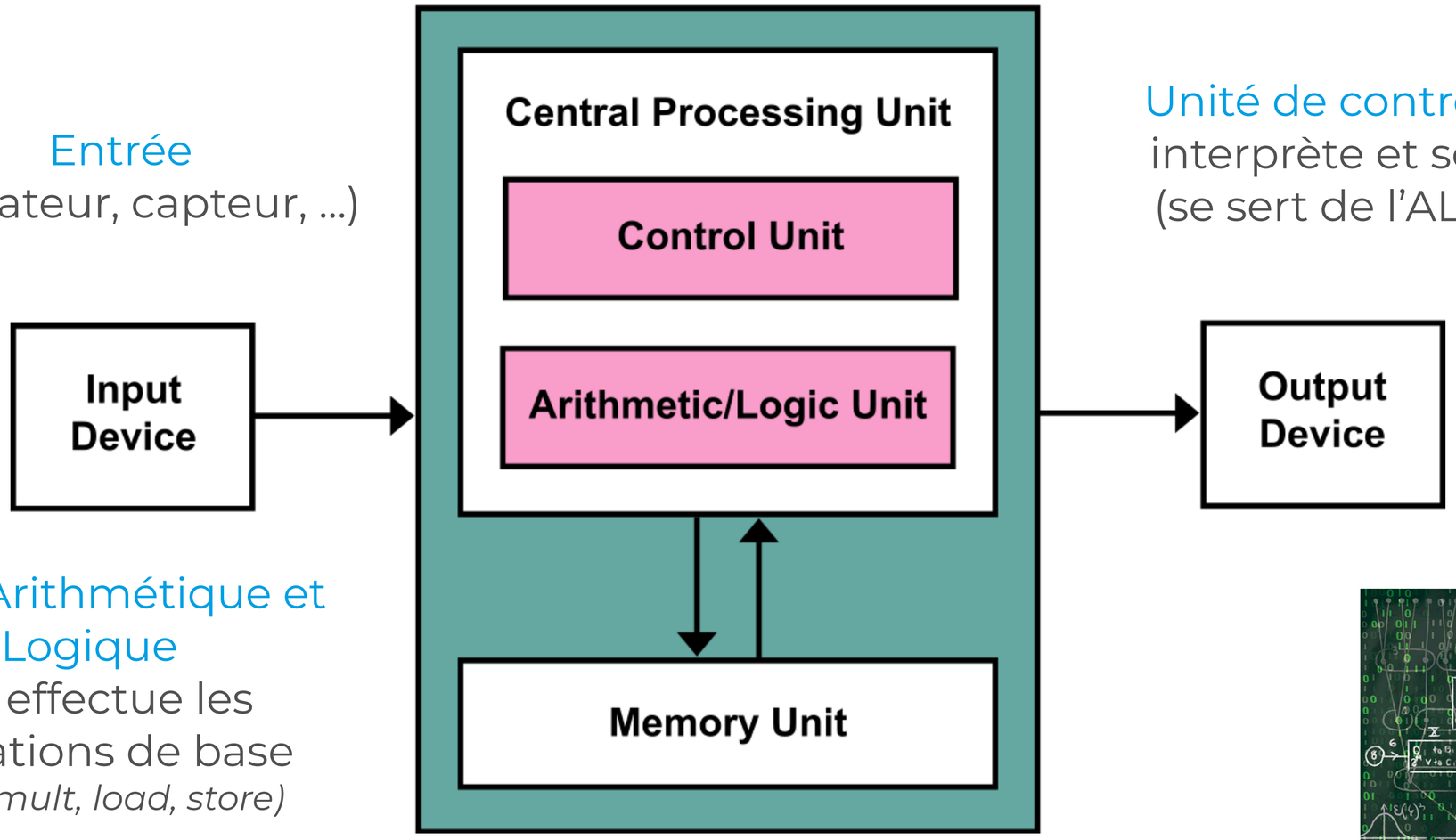


Depuis les années 1970, les processeurs ont conservé globalement la même architecture (type Von Neumann) mais la microarchitecture a énormément changé, en raison des évolutions technologiques.

Architecture Von Neumann (1945)

Entrée
(utilisateur, capteur, ...)

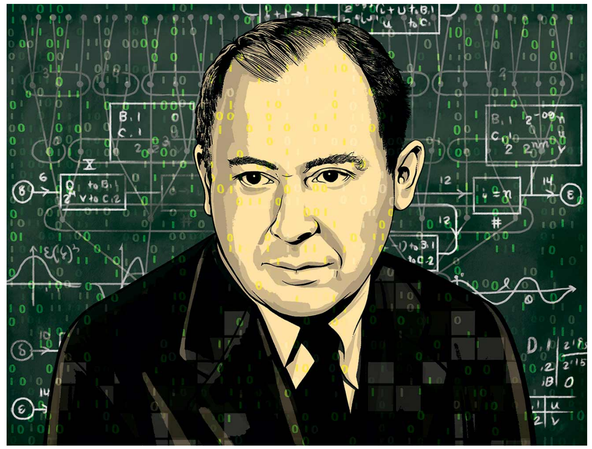
Unité de contrôle (chef d'orchestre) qui interprète et séquence les instructions (se sert de l'ALU pour faire des calculs)



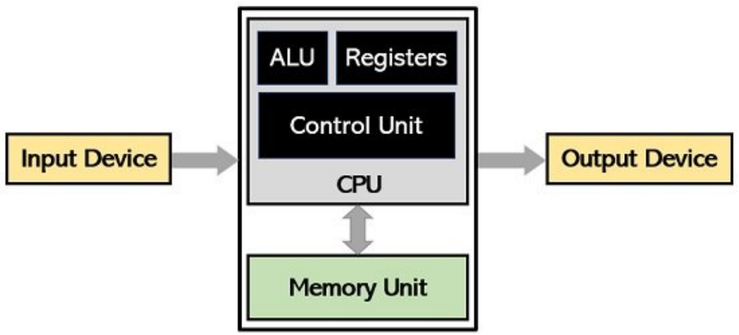
Sortie
(affichage, imprimante, stockage)

Unité Arithmétique et Logique
qui effectue les opérations de base
(add, mult, load, store)

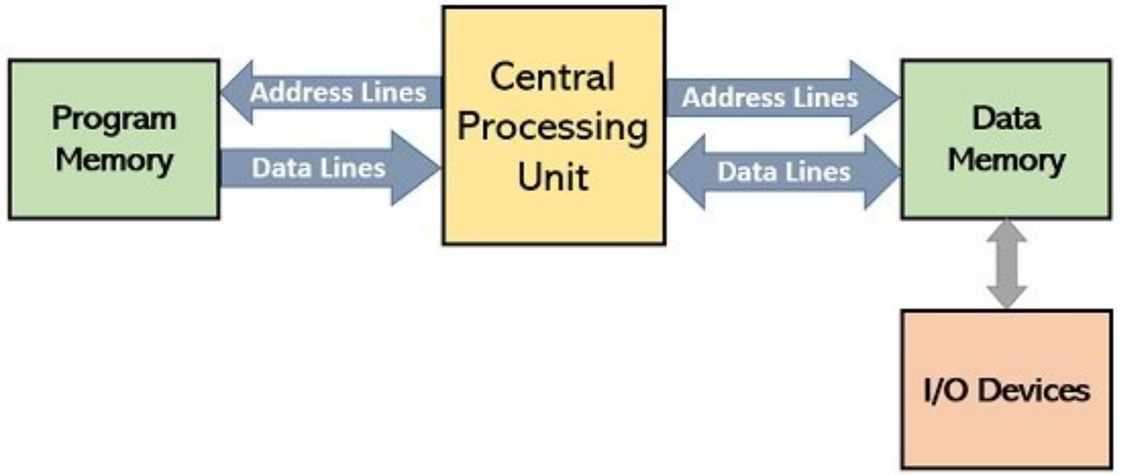
Mémoire qui contient à la fois les données et le programme



Architecture Harvard (1944)



Von Neumann Architecture



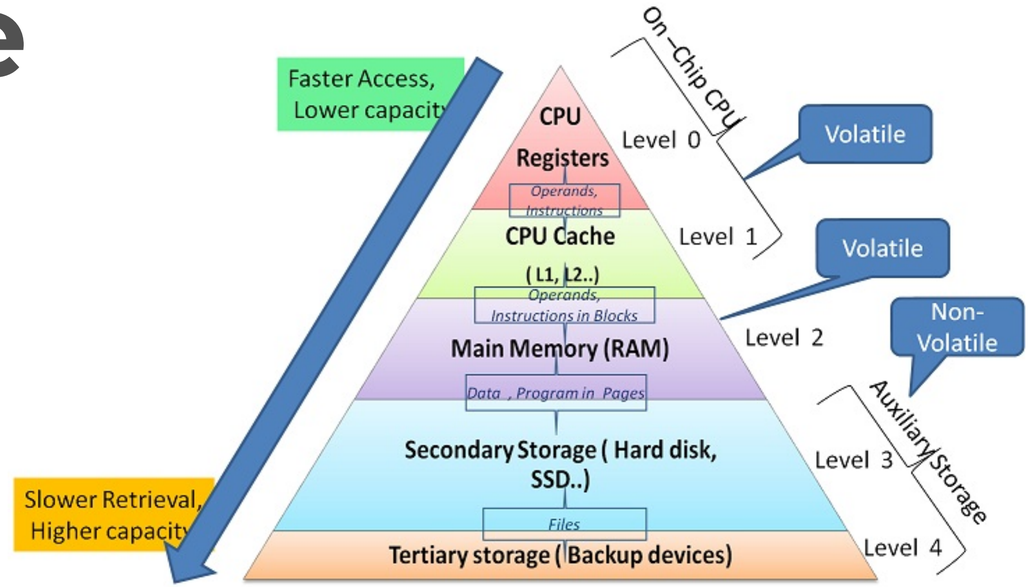
Harvard Architecture

Architecture plus performante mais plus complexe avec séparation de la mémoire programmes et de la mémoire données (accès via des bus distincts)

Inventée à l'Université de Harvard dans les années 40 avec une première implémentation sur le Mark 1 en 1944

Utilisée aujourd'hui dans les micro-contrôleurs (PIC) et DSP

Mémoire



Différentes mémoires

Hiérarchie de mémoire allant du Registre processeur à la Bande d'archivage en passant par différents niveaux de cache avec des caractéristiques diverses :

- Utilisation (mémoire volatile / non volatile)
- Performance (technologie, prix)

Organisation en cellules individuelles

Une cellule stocke une donnée unique (exprimée en nombre de bits) à une adresse unique (exprimée en nombre de bits)

Le type de données à stocker détermine le nombre de cellules élémentaires à utiliser :

- Caractère sur 8 bits,
- Entier sur 16, 32 ou 64 bits
- Réel sur 32 ou 64 bits

Le nombre de bits d'adresse détermine la quantité de mémoire accessible :

- 16 bits = 65 536 adresses = 64 ko
- 32 bits = 4 294 967 295 adresses = 4 Go
- 64 bits = 18 446 744 073 709 551 616 adresses = 16 exa octets = 16 Milliards de Go

Mémoire = Stockage de bits

Nombre prédéfini de bits pour représenter l'adresse

Ici 8 bits pour 256 adresses possibles

Données (mot de 8 bits)

Adresses (mot de 8 bits)

	b7	b6	b5	b4	b3	b2	b1	b0
0x00	0	1	0	1	1	0	1	1
0x01	1	0	0	1	0	0	0	1
0x02	0	1	1	0	1	0	1	1
0x03	0	0	1	1	0	1	0	1
0x04	0	1	0	1	1	0	1	1
...	...	...	...	...	...	...	...	...
0xFF	0	0	1	0	0	1	1	1

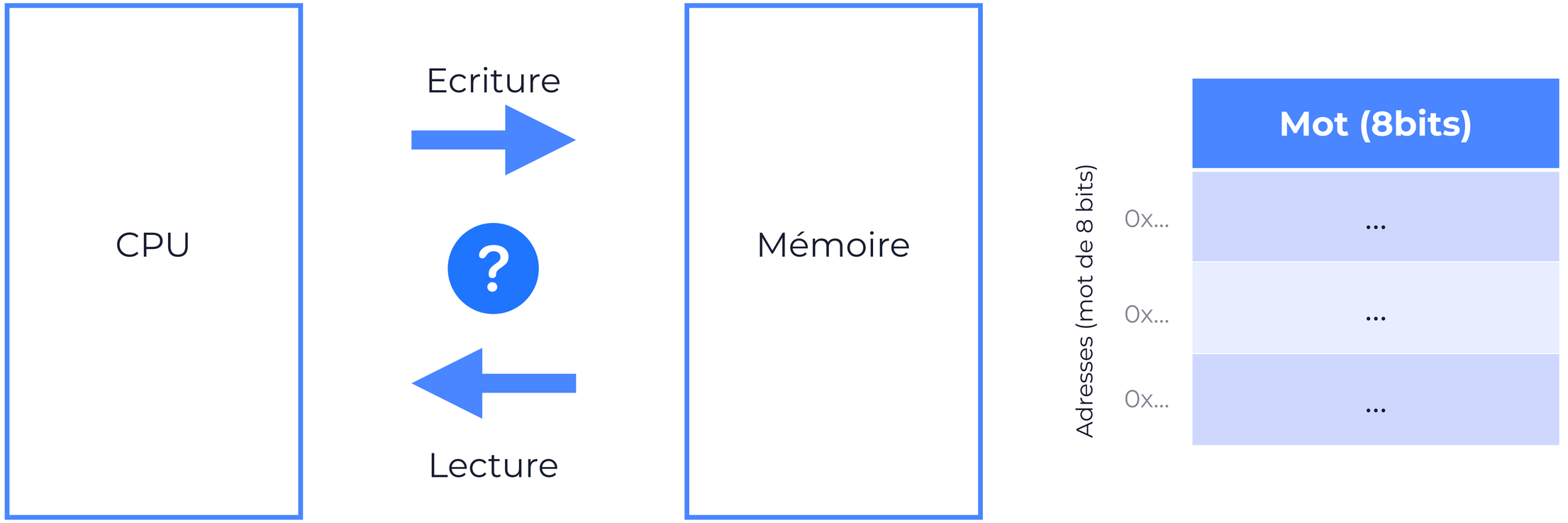
Chaque case contient un bit qui n'est jamais vide (toujours 0 ou 1)

On regroupe les cases pour former des mots (ici de 8 bits)

Ex : dans la case 0x02, la donnée stockée est 107 (= 1+2+8+32+64) si on considère des entiers non signés

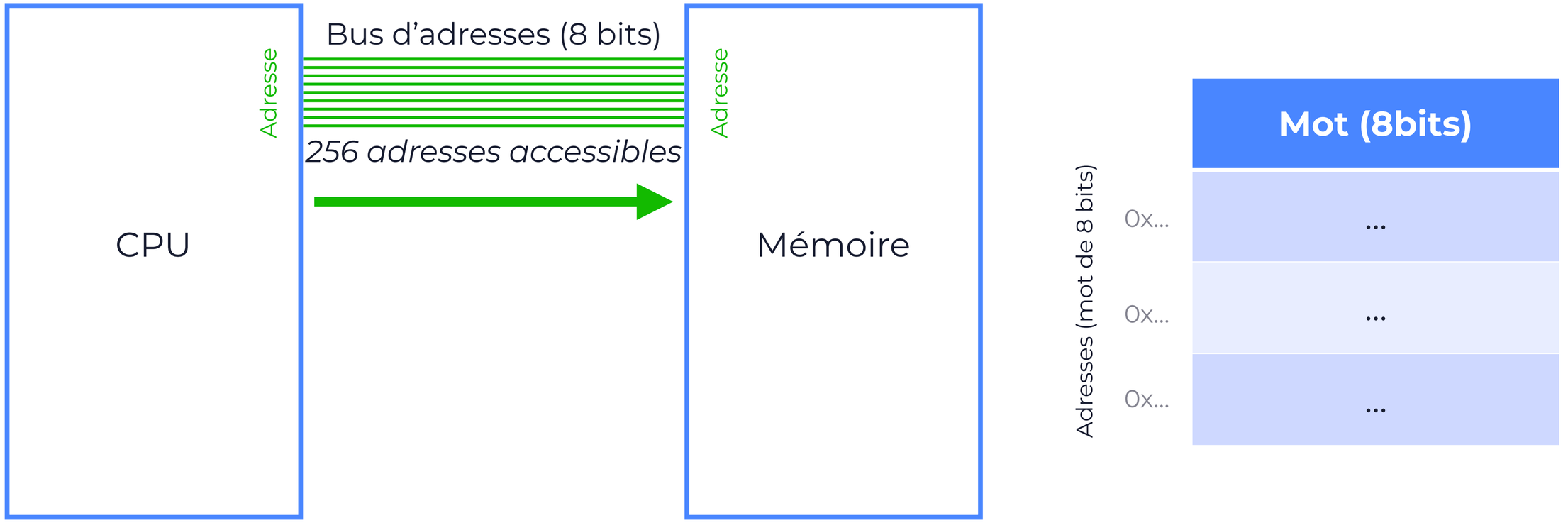
Taille mémoire = Nombre d'adresses possibles x Taille d'un mot
= $2^8 * 1 = 256$ octets

Comment le CPU accède à la mémoire ?



Utilisation d'un bus = groupe de lignes reliant un composant à un autre, chaque ligne transportant un bit d'information à la fois.

Bus d’adresses

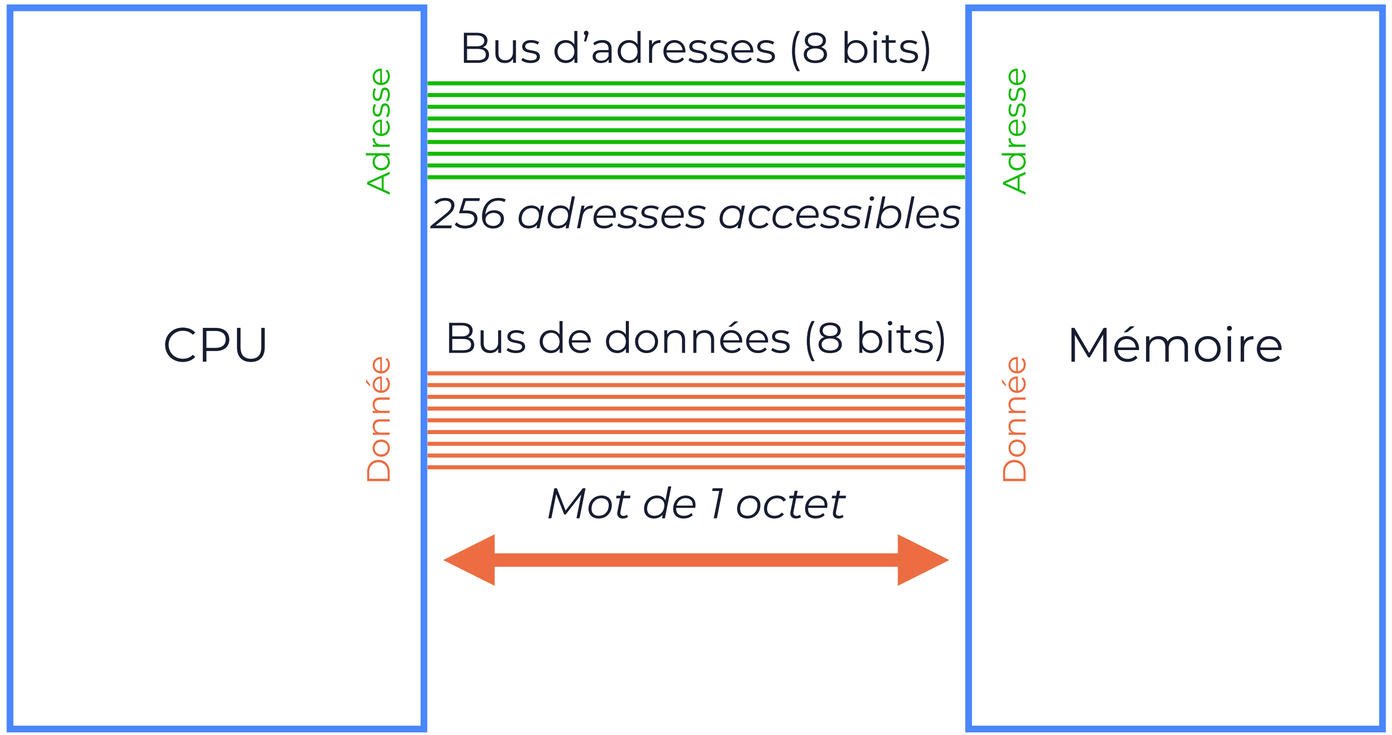


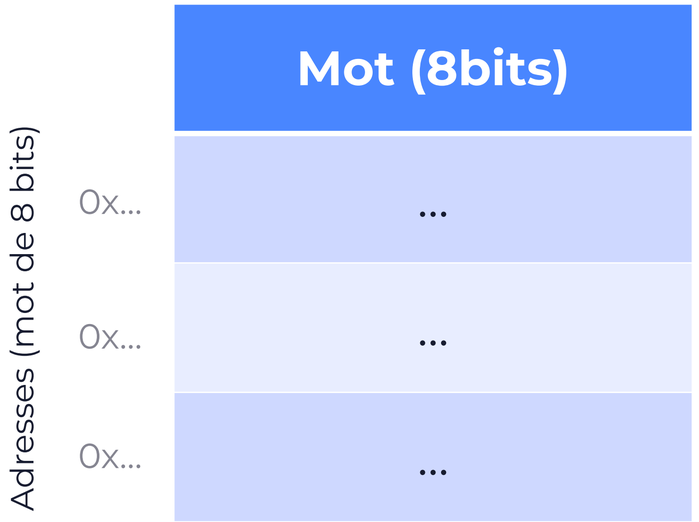
Le bus d’adresse indique l’emplacement de la mémoire à lire ou à écrire.

Le CPU contrôle le choix de l’emplacement, c’est-à-dire qu’il choisit l’adresse de la mémoire.

La taille du bus (ici 8 bits) détermine la quantité de mémoire pouvant être utilisée par le CPU.

Bus de données

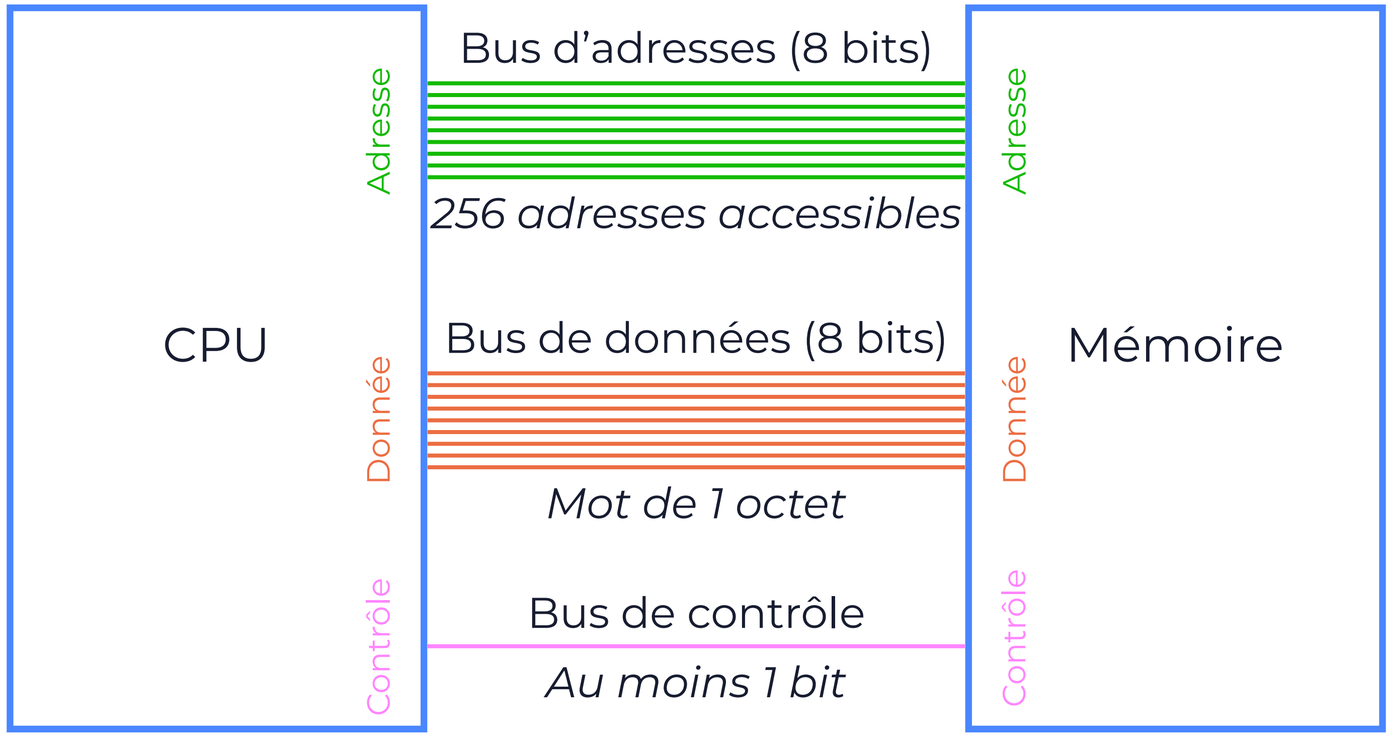


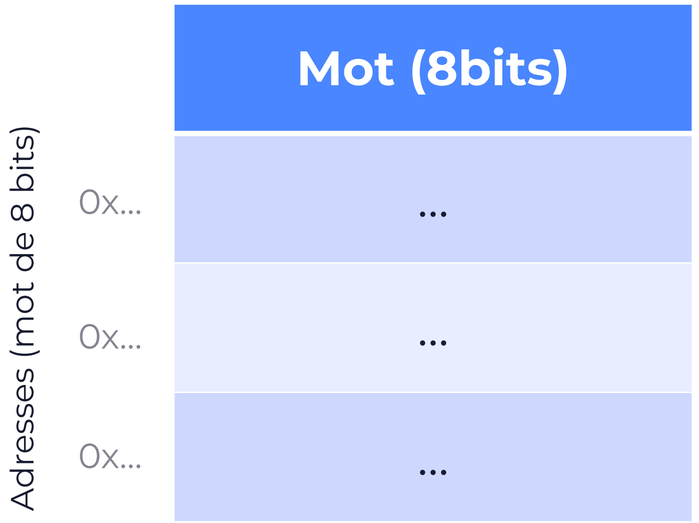


Le bus de données permet le transfert des données du CPU vers la mémoire (écriture) ou de la mémoire vers le CPU (lecture), les données circulant dans un seul sens lors d'un transfert.

La taille du bus de données (ici 8 bits) détermine la grandeur maximale des mots pouvant être transférés en une opération. Pour transférer un mot de 32 bits, il faut 4 opérations successives.[12]

Bus de contrôle



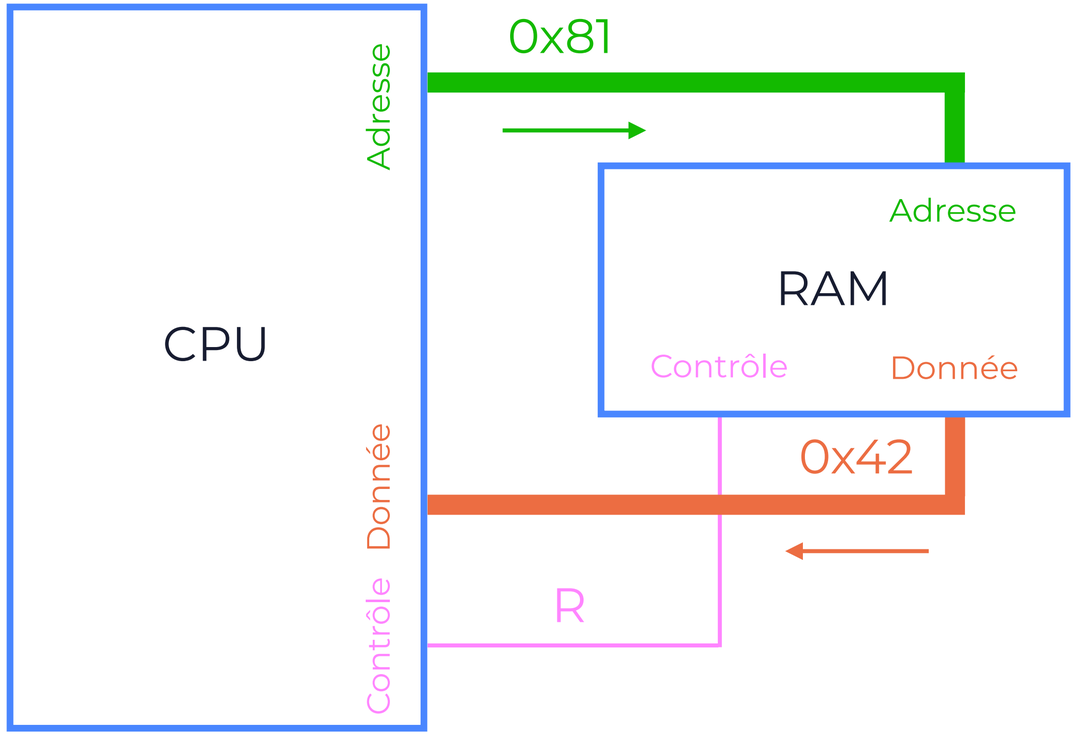


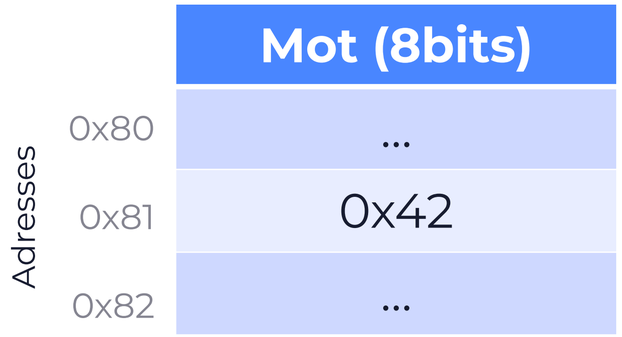
Le bus de contrôle permet de définir le type d'opération (lecture ou écriture).

Il gère la direction des données sur le bus de données.

Il assure aussi la synchronisation entre CPU et Mémoire (horloge, acknowledge, ...).

Exemple de Lecture d'une donnée





Coté CPU :

1. Initie l'action de lecture
2. Place l'adresse 0x81 sur le bus d'adresse
3. Active le bus de controle en lecture

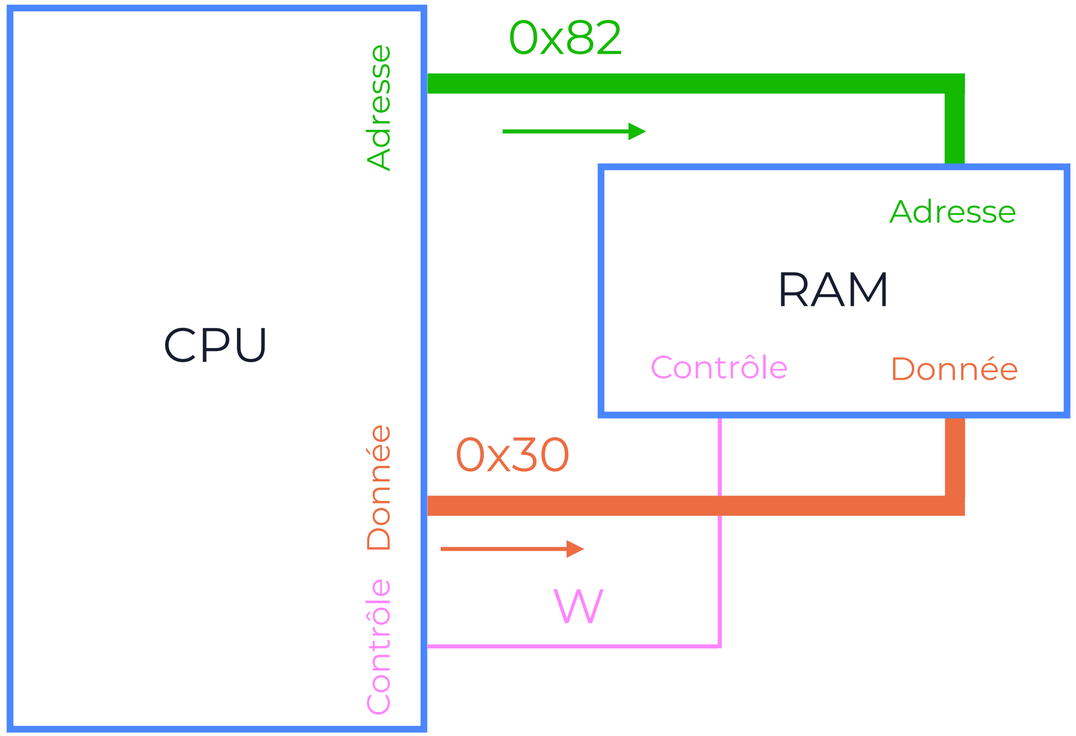
Coté RAM :

1. Place la donnée 0x42 sur le bus de donnée
2. Active le bus de contrôle pour prévenir le CPU que la donnée est disponible

Coté CPU :

1. Lis la valeur 0x42 sur le bus de données et la stocke dans un registre interne du CPU

Exemple d'Ecriture d'une donnée



Adresses	Mot (8bits)
0x80	...
0x81	0x42
0x82	0x30

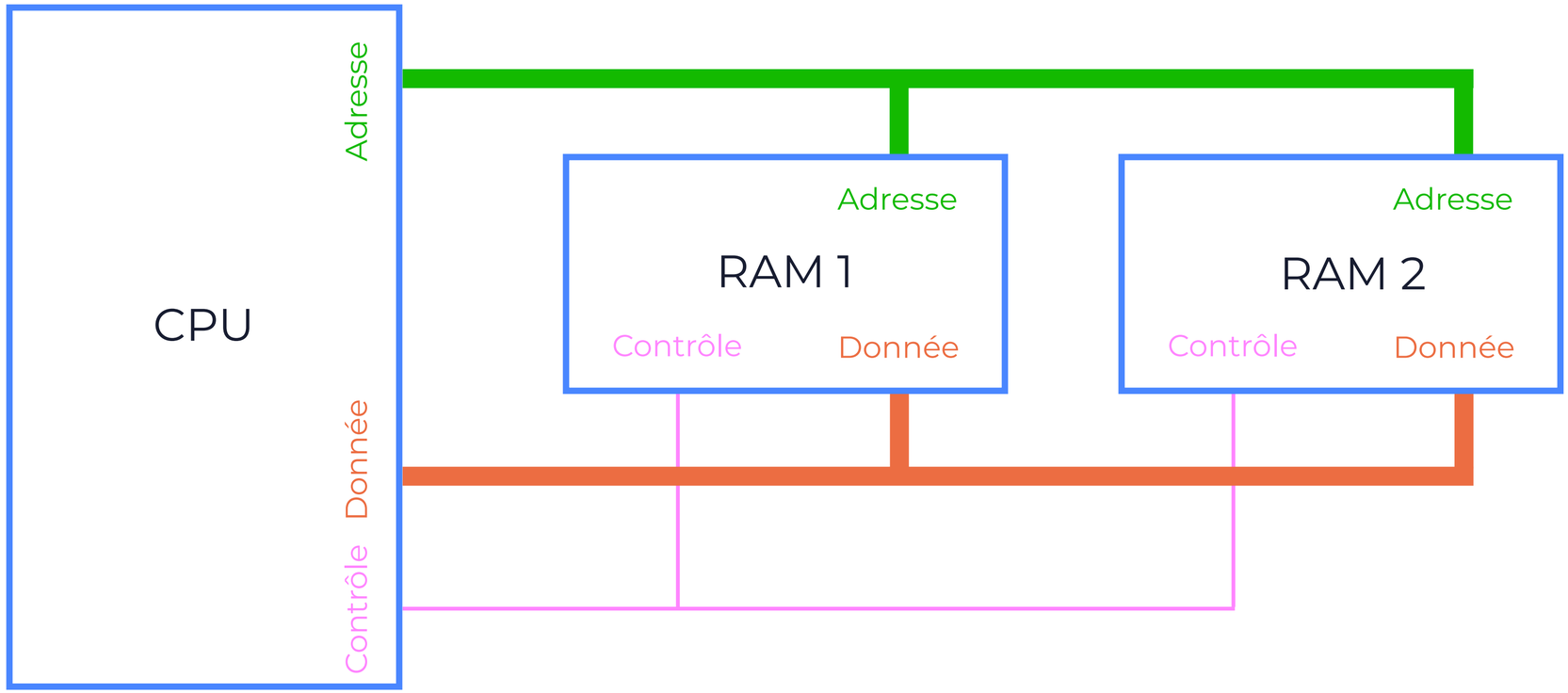
Coté CPU :

1. Initie l'action d'écriture
2. Place l'adresse 0x82 sur le bus d'adresse
3. Place la donnée 0X30 sur le bus de donnée
4. Active le bus de controle en écriture

Coté RAM :

1. Lit la donnée 0x30 sur le bus de donnée
2. Ecrit la donnée à l'adresse fournie par le bus d'adresse
3. Indique sur le bus de contrôle que l'opération d'écriture est terminée

Avec deux mémoires



Comment sélectionner la “bonne” mémoire ?

Ajout d’un décodeur d’adresses

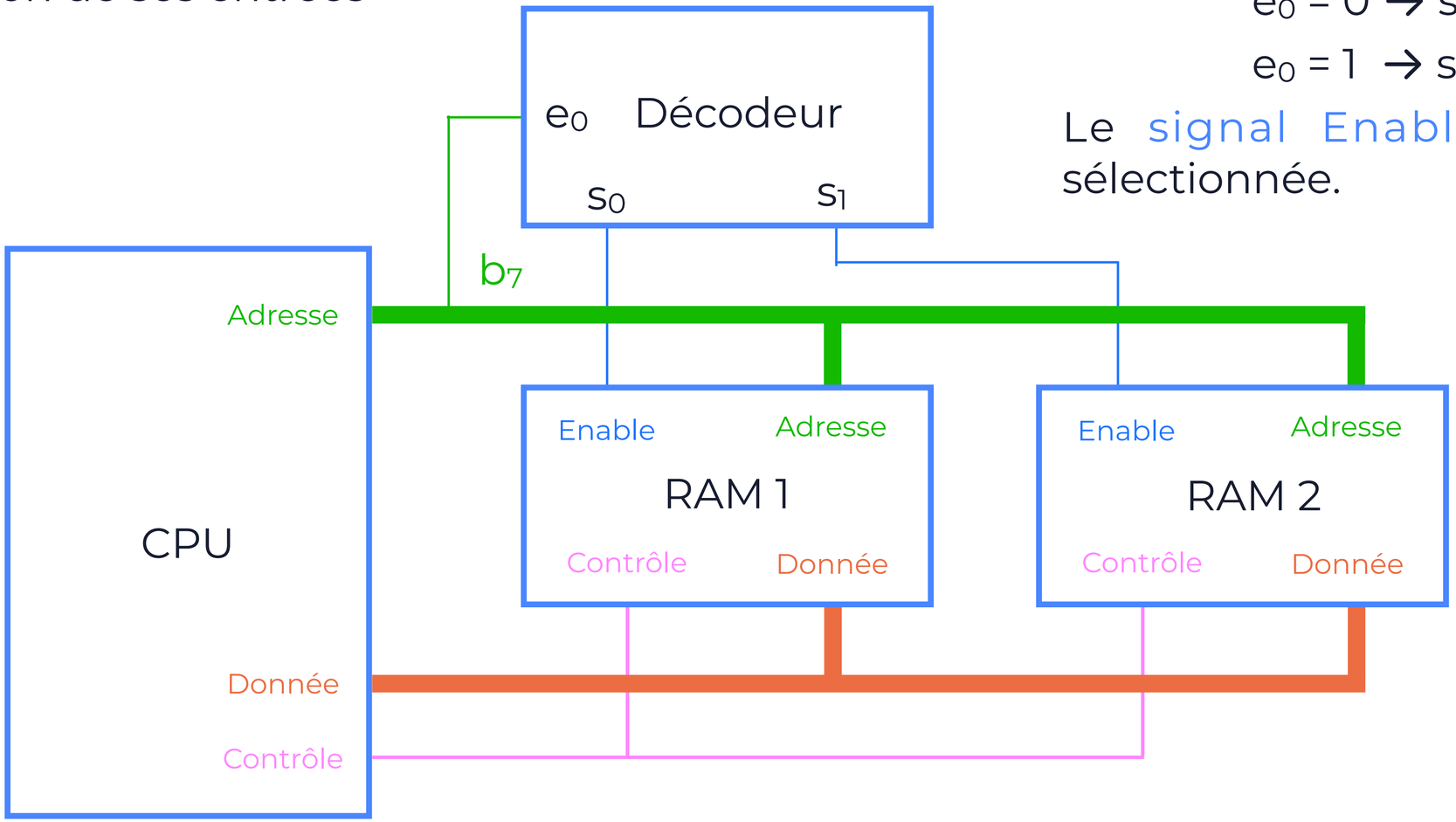
Décodeur chargé de sélectionner une sortie en fonction de ses entrées

Utilisation du bit de poids fort b_7 pour sélectionner la RAM 1 ou la RAM 2

$e_0 = 0 \rightarrow s_0 = 1,\ s_1 = 0 \rightarrow$ RAM 1

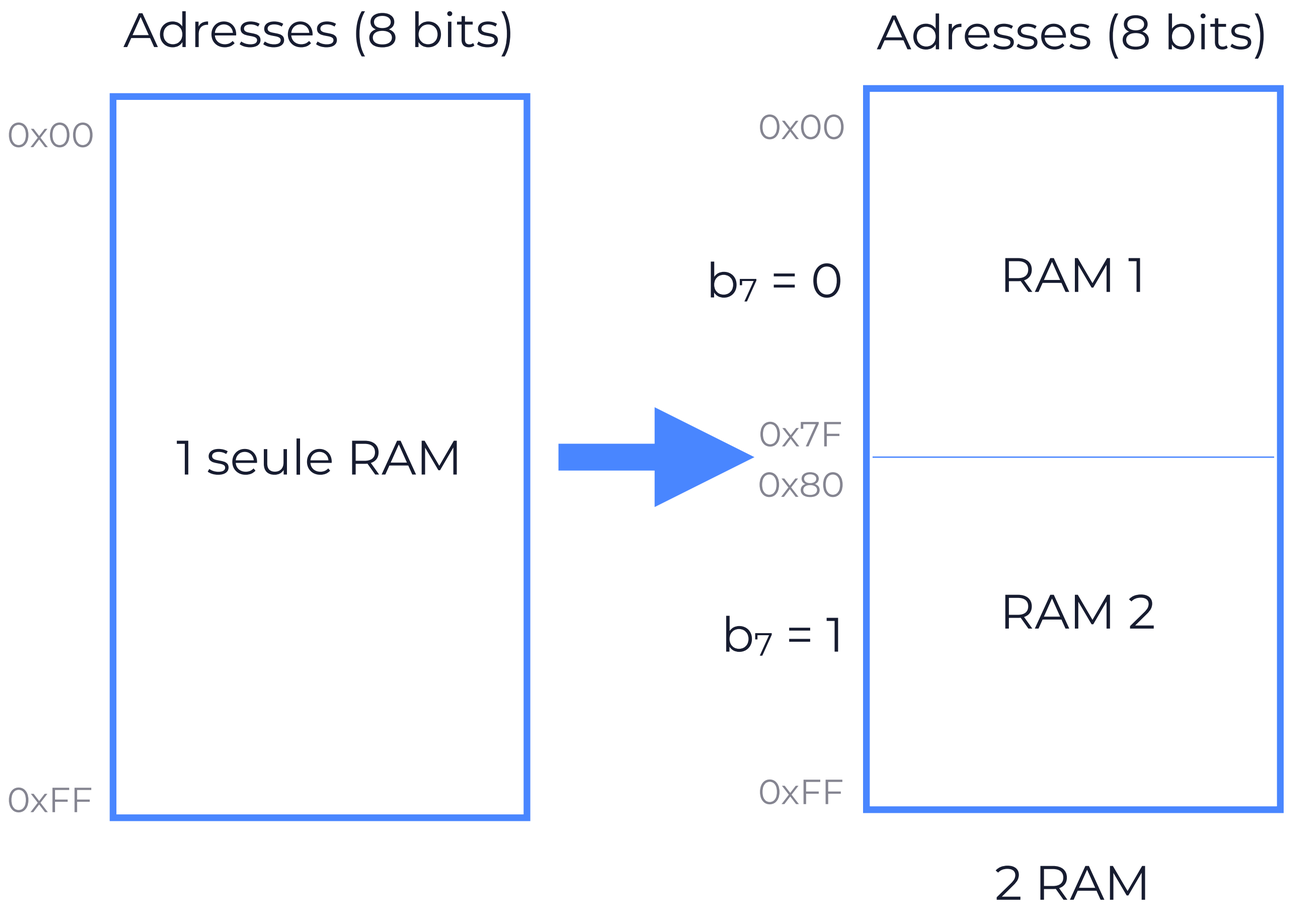
$e_0 = 1 \rightarrow s_0 = 0,\ s_1 = 1 \rightarrow$ RAM 2

Le signal Enable active la mémoire sélectionnée.



Représentation de la RAM vue du CPU

Carte mémoire = Memory map



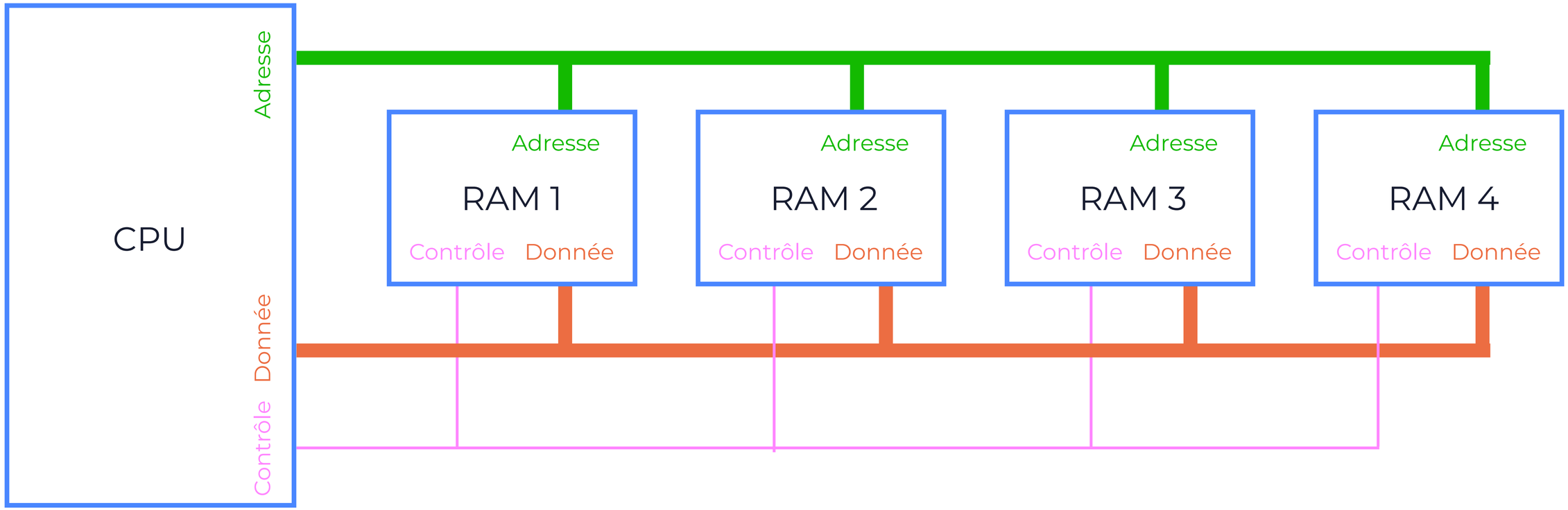
Le CPU voit toujours une seule mémoire allant de 0x00 à 0xFF.

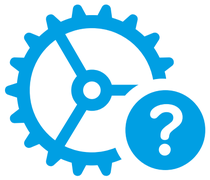
La plage d'adresse du CPU (0x00 à 0xFF) est divisée en deux : 0x00 à 0x7F pour la RAM1 et 0x80 à 0XFF pour la RAM2.

Le bit b7 de poids fort est utilisé par le décodeur d'adresse pour savoir quelle mémoire doit être sélectionnée.

Chaque mémoire possède un signal d'activation (enable) qui indique si la mémoire est sélectionnée et donc connectée au bus de donnée (les autres étant en haute impédance).

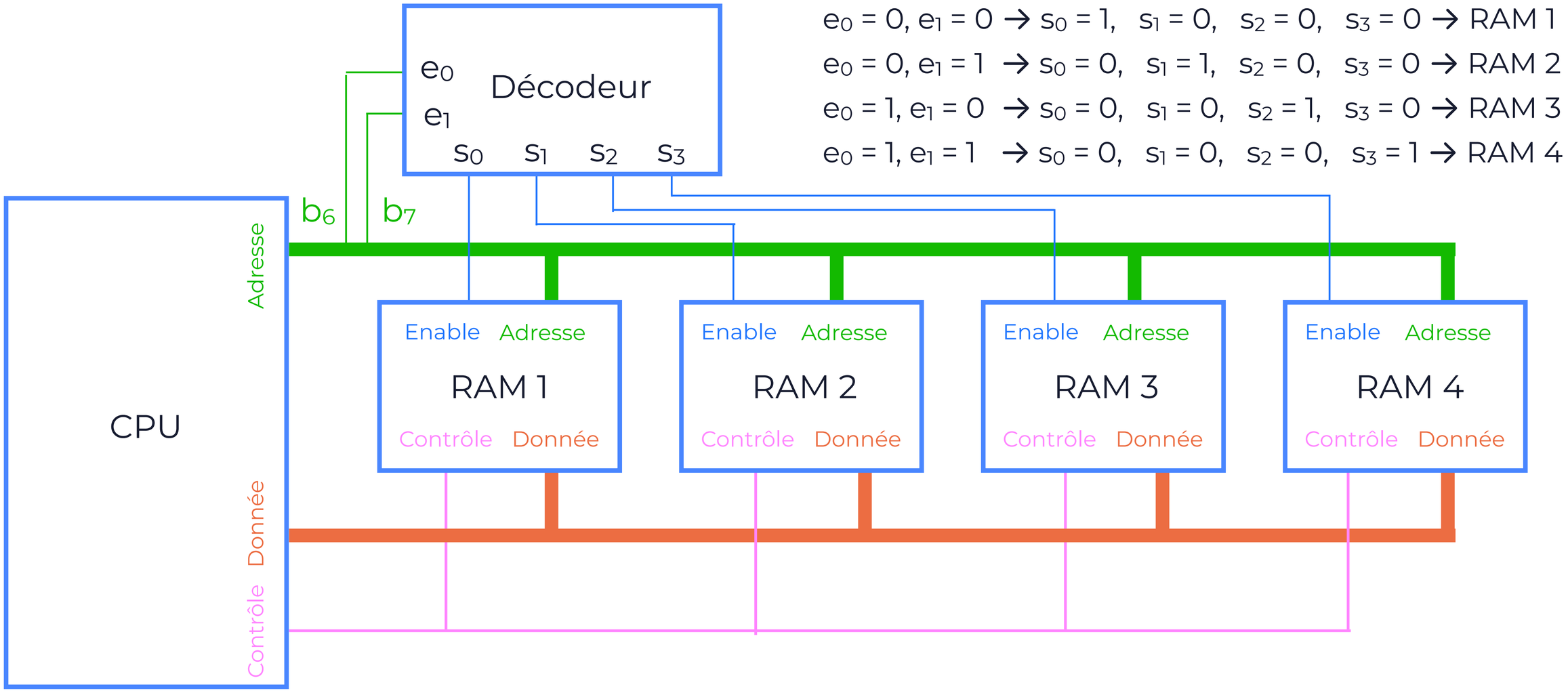
Avec plusieurs mémoires





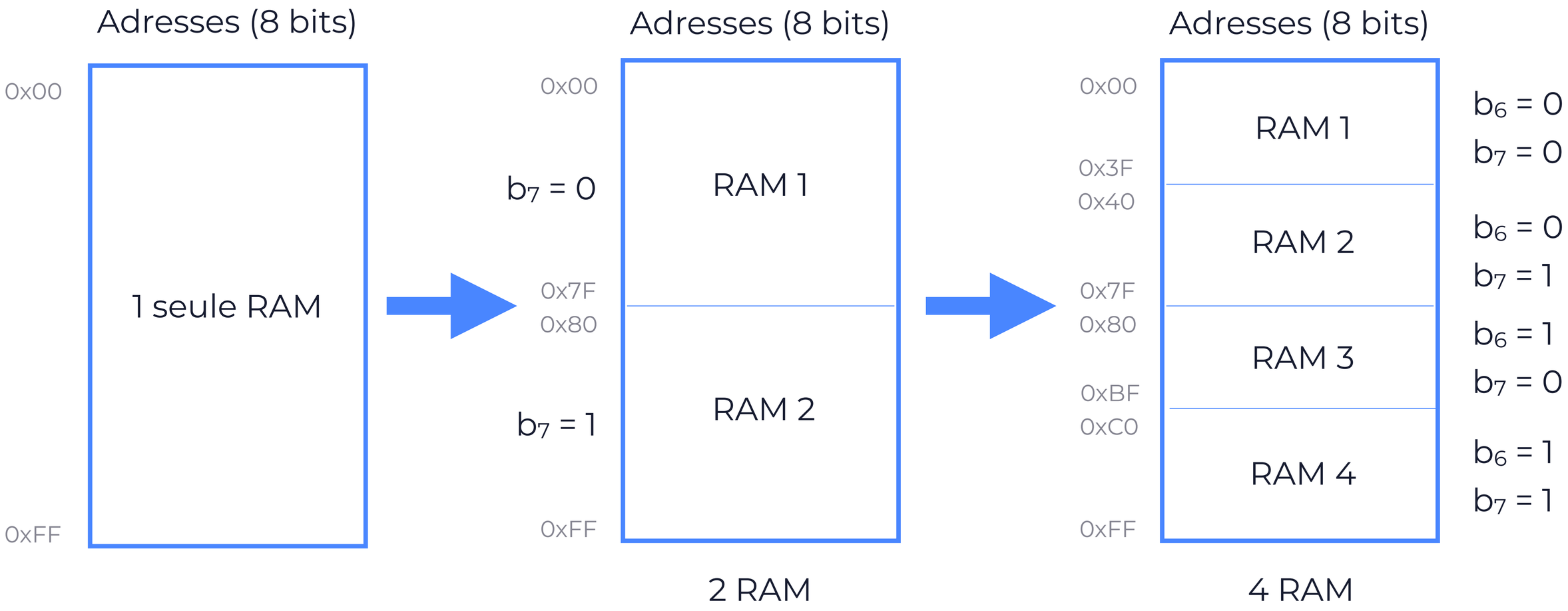
Comment sélectionner la “bonne” mémoire parmi N ?

En utilisant plusieurs bits pour le décodeur



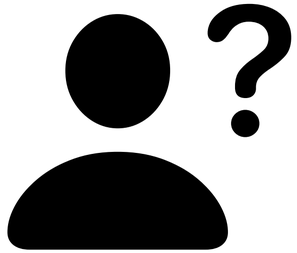
Représentation de la RAM vue du CPU

Carte mémoire = Memory map



Comment **STOCKER** un mot de **4 OCTETS** sachant que chaque **ADRESSE** contient **1 OCTET** ?

Exemple : 0x12 34 56 78 à l'adresse 0x80



Deux solutions valides

0x12 34 56 78 à l'adresse 0x80

	Octet
0x7F	...
0x80	12
0x81	34
0x82	56
0x83	78
0x84	...

Option 1

MSB (octet le plus significatif) en 0x80

	Octet
0x7F	...
0x80	78
0x81	56
0x82	34
0x83	12
0x84	...

Option 2

LSB (octet le moins significatif) en 0x80

Autres représentations

0x12 34 56 78 à l'adresse 0x80

	+0	+1	+2	+3
0xFC	…	…	…	…
0x80	12	34	56	78
0x84	…	…	…	…
0x88	…	…	…	…

Option 1

MSB (octet le plus significatif) en 0x80

	+0	+1	+2	+3
0xFC	…	…	…	…
0x80	78	56	34	12
0x84	…	…	…	…
0x88	…	…	…	…

Option 2

LSB (octet le moins significatif) en 0x80

Endianness : Big endian vs. Little endian

En français → Boutisme (= Ordre des octets) : Gros bout vs. Petit bout

Little Endian

Big Endian

Big Endian : Octet le plus significatif dans la plus petite adresse mémoire

Little Endian : Octet le moins significatif dans la plus petite adresse de la mémoire

En référence à la guerre des oeufs décrite dans les “Voyages de Gulliver” de J. Swift qui oppose les peuples lilliputiens afin de déterminer comment manger correctement les oeufs : Le bout le plus large en haut ou en bas ?

Big endian vs. Little endian

	+0	+1	+2	+3
0xFC	...	...	...	...
0x80	12	34	56	78
0x84	...	...	...	...

Big Endian

MSB en 0x80

	+0	+1	+2	+3
0xFC	...	...	...	...
0x80	78	56	34	12
0x84	...	...	...	...

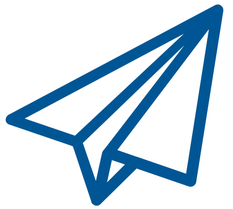
Little Endian

LSB en 0x80

La plupart des CPU modernes (ARM, Intel x86, RISC V, ...) sont Little Endian.

Les systèmes Big Endian ont été très utilisés dans le passé (IBM, Motorola, ...) et le sont moins aujourd'hui.

TAKE HOME MESSAGE



#3

L'architecture **Von Neumann** est l'architecture de processeur majoritairement utilisée.

Le processeur accède à la mémoire (programmes et données) en lecture / écriture via des **bus d'adresses, de données et de contrôle**.

La mémoire stocke les informations sous la forme d'**octets** en mode **Little Endian** (LSB à l'adresse mémoire) ou **Big Endian** (MSB à l'adresse mémoire).

Questions

Photo by Alexander Andrews on Unsplash

Contacts

Pr. Dominique Ginhac

dginhac@u-bourgogne.fr

Retrouvez toutes les infos sur :

https://github.com/dginhac/esirem-archi

POLYTECH®
DIJON