

Introduction aux microprocesseurs

module M1103

`Sebastien.Kramm@univ-rouen.fr`

IUT R&T Rouen, site d'Elbeuf

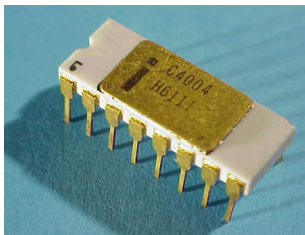
2018-2019

Sommaire

- 1 Historique & évolutions contemporaines
- 2 Mémoire
- 3 Structure d'un système informatique
- 4 Structure d'un CPU
 - Jeu d'instructions
 - Performances et évolution des architectures

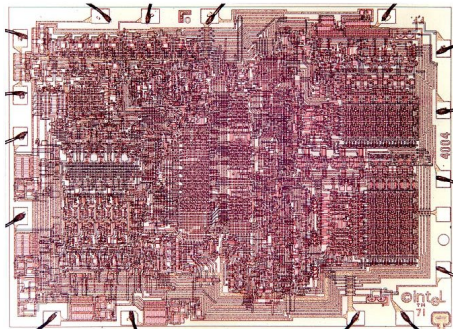
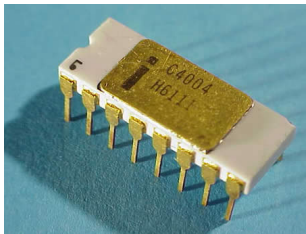
Historique

- 1971 : Intel 4004 : 4 bits, 2300 transistors, 108 kHz



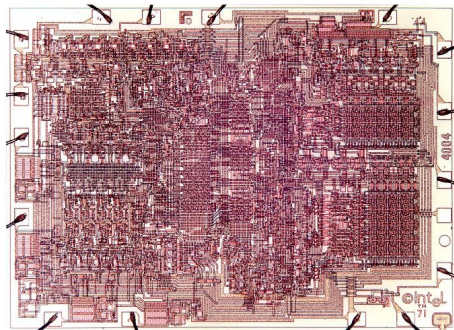
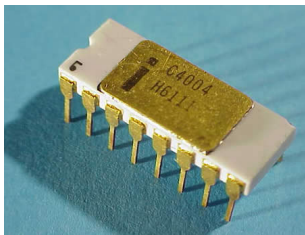
Historique

- 1971 : Intel 4004 : 4 bits, 2300 transistors, 108 kHz



Historique

- 1971 : Intel 4004 : 4 bits, 2300 transistors, 108 kHz

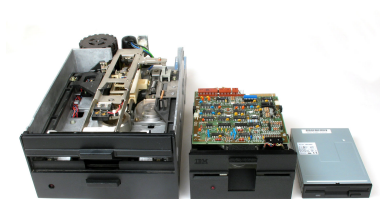


- 1978 : Intel 8088/8086, 16 bits, architecture "x86", utilisé dans les "IBM PC"
- 1985 : Intel 80386 (32 bits)

IBM PC : sorti en 1981



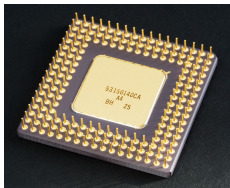
IBM PC : sorti en 1981



source : WP

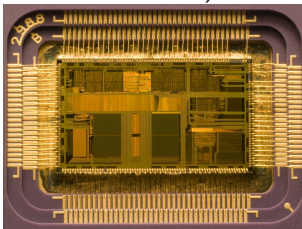
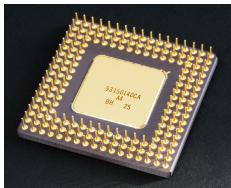
Historique

- 1989 : Intel 486 (> 1 million transistors)



Historique

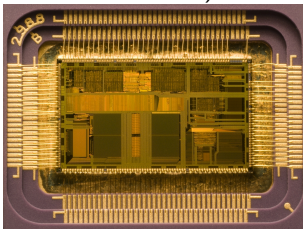
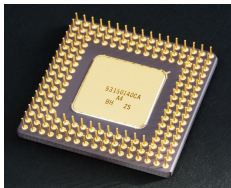
- 1989 : Intel 486 (> 1 million transistors)



taille réelle : 12 x 7 mm

Historique

- 1989 : Intel 486 (> 1 million transistors)

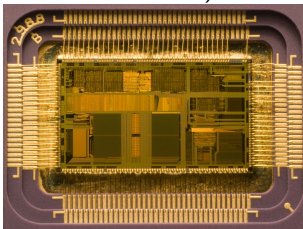
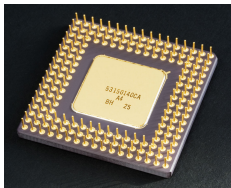


taille réelle : 12 x 7 mm

- 1993 : Intel Pentium
- 2000 : premier processeur x86-64 bits (AMD)
- 2002 : Intel Pentium 4 (Hyper-Threading)

Historique

- 1989 : Intel 486 (> 1 million transistors)



taille réelle : 12 x 7 mm

- 1993 : Intel Pentium
- 2000 : premier processeur x86-64 bits (AMD)
- 2002 : Intel Pentium 4 (Hyper-Threading)
- 2005 : AMD/Athlon 64 X2, premier processeur x86 dual-core.

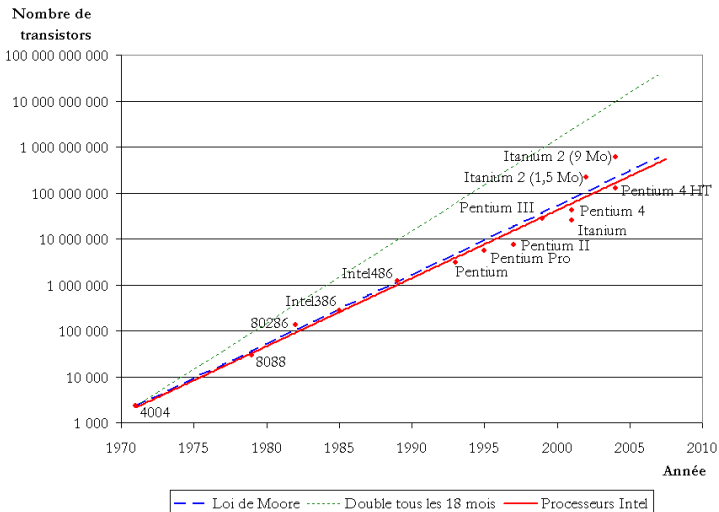


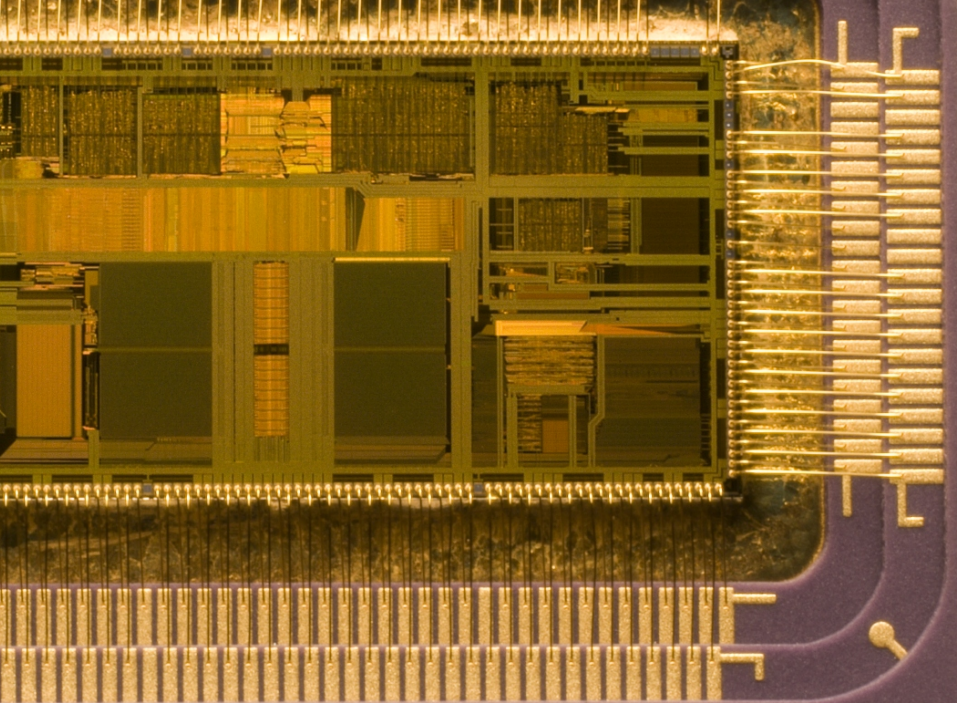
Loi de Moore (1975)

"Le nombre de transistors double tous les 2 ans"

Loi de Moore (1975)

"Le nombre de transistors double tous les 2 ans"

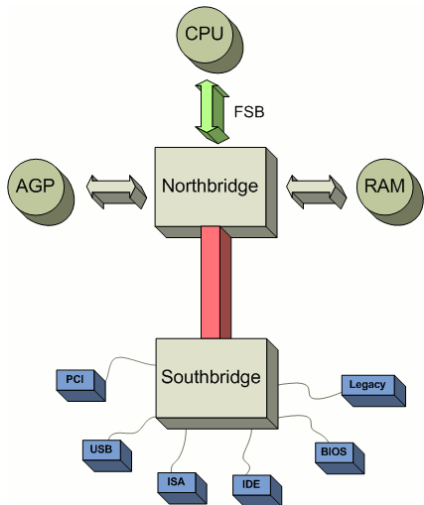




Microcontrôleur vs. Microprocesseur (1)

Microprocesseur :

- composant généraliste, optimisé pour la puissance de calcul et la vitesse,
- ne peut fonctionner qu'accompagné de son *chipset*.

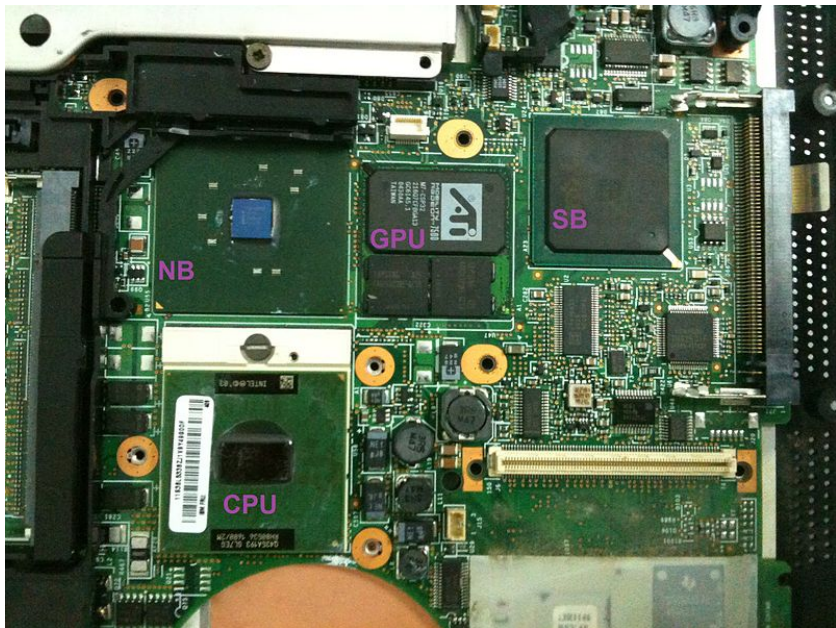


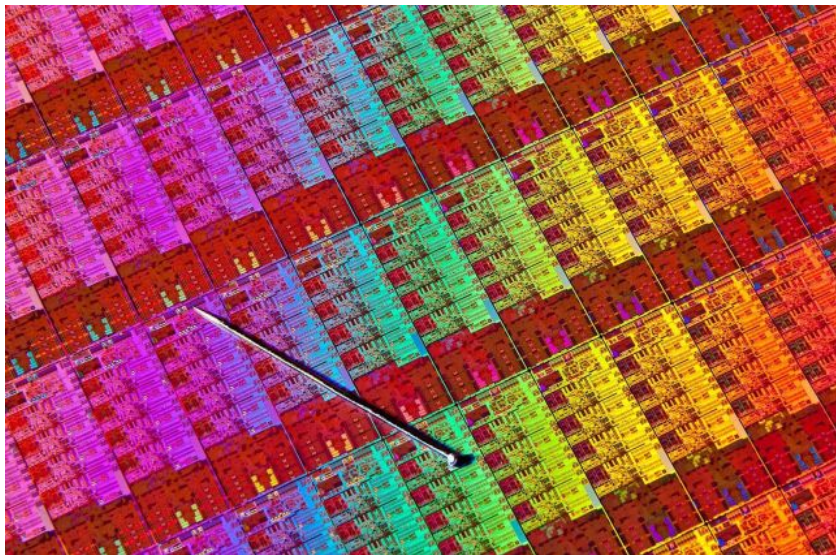
Microcontrôleur vs. Microprocesseur (2)

Microcontrôleur : adapté aux applications embarquées, capable de fonctionnement autonome.

⇒ intègre un microprocesseur, plus :

- Mémoire
 - RAM
 - ROM, EEPROM, Flash
- Gestionnaire périphériques
 - Communication
 - Acquisition (capteur, organe de commande, ...)
 - Pilotage actionneurs et afficheurs





Evolutions et perspectives

- L'augmentation de la puissance de calcul s'est faite principalement sur deux axes :
 - Augmentation de la fréquence d'horloge. Aujourd'hui : $\sim 1\text{GHz}$.
 - Augmentation de la densité d'intégration (finesse de gravure). Aujourd'hui : $< 40\text{ nm}$

Evolutions et perspectives

- L'augmentation de la puissance de calcul s'est faite principalement sur deux axes :
 - Augmentation de la fréquence d'horloge. Aujourd'hui : $\sim 1\text{GHz}$.
 - Augmentation de la densité d'intégration (finesse de gravure). Aujourd'hui : $< 40\text{ nm}$
- Proc. contemporains :
 - 2012 : Intel Core i7 Ivy Bridge : 624 M transistors, 22 nm, 3,5 GHz
 - 2015/08 : Intel SkyLake : 14 nm (évolution à 10 nm en 2018)

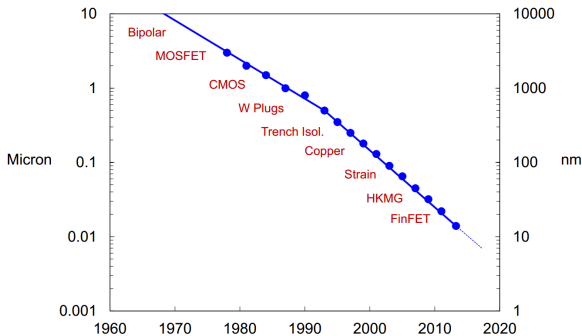
Evolutions et perspectives

- L'augmentation de la puissance de calcul s'est faite principalement sur deux axes :
 - Augmentation de la fréquence d'horloge. Aujourd'hui : $\sim 1\text{GHz}$.
 - Augmentation de la densité d'intégration (finesse de gravure). Aujourd'hui : $< 40\text{ nm}$
- Proc. contemporains :
 - 2012 : Intel Core i7 Ivy Bridge : 624 M transistors, 22 nm, 3,5 GHz
 - 2015/08 : Intel SkyLake : 14 nm (évolution à 10 nm en 2018)
- Problèmes :
 - la dissipation de chaleur ($\sim 100\text{ W}$) devient trop importante ;
 - limites physique sur la finesse de gravure.
- Solutions : architecture interne du CPU :
 - *pipeline* d'exécution,
 - architecture multicœur.

Breaking news !

2015 :

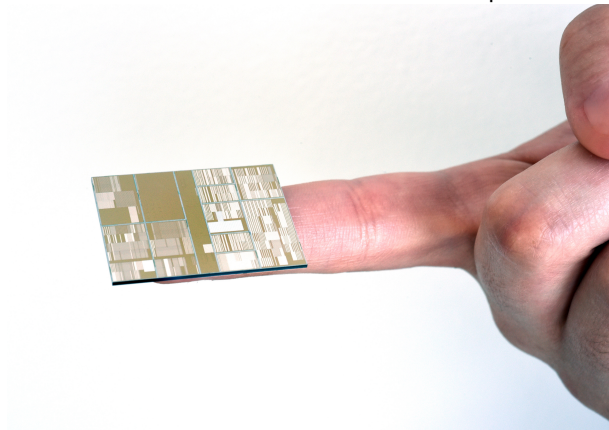
(EP1) Moore's Law Challenges Below 10nm: Technology, Design and Economic Implications



Process/device innovation has always been an indispensable part of scaling

Breaking news !

2017-06 : IBM unveils world's first 5nm chip



Sommaire

- 1 Historique & évolutions contemporaines
- 2 Mémoire**
- 3 Structure d'un système informatique
- 4 Structure d'un CPU
 - Jeu d'instructions
 - Performances et évolution des architectures

Mémoire dans un système informatique

- Un système informatique a besoin de mémoire pour fonctionner.

Mémoire dans un système informatique

- Un système informatique a besoin de mémoire pour fonctionner.
- Deux types :

Mémoire dans un système informatique

- Un système informatique a besoin de mémoire pour fonctionner.
- Deux types :
 - mémoire "vive", ou *Random Access Memory* (RAM) : contenu non rémanent

Mémoire dans un système informatique

- Un système informatique a besoin de mémoire pour fonctionner.
- Deux types :
 - mémoire "vive", ou *Random Access Memory* (RAM) : contenu non rémanent
 - mémoire "morte", ou *Read Only Memory* (ROM) : contenu rémanent

Mémoire dans un système informatique

- Un système informatique a besoin de mémoire pour fonctionner.
- Deux types :
 - mémoire "vive", ou *Random Access Memory* (RAM) : contenu non rémanent
 - mémoire "morte", ou *Read Only Memory* (ROM) : contenu rémanent
- Chaque emplacement mémoire contient une valeur
Exemple : adresse \$f842 → contient la valeur \$4B8

Mémoire dans un système informatique

- Un système informatique a besoin de mémoire pour fonctionner.
- Deux types :
 - mémoire "vive", ou *Random Access Memory* (RAM) : contenu non rémanent
 - mémoire "morte", ou *Read Only Memory* (ROM) : contenu rémanent
- Chaque emplacement mémoire contient une valeur
Exemple : adresse \$f842 → contient la valeur \$4B8

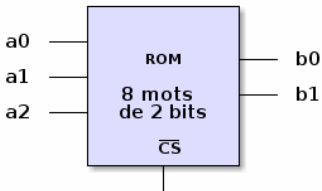
Mémoire dans un système informatique

- Un système informatique a besoin de mémoire pour fonctionner.
- Deux types :
 - mémoire "vive", ou *Random Access Memory* (RAM) : contenu non rémanent
 - mémoire "morte", ou *Read Only Memory* (ROM) : contenu rémanent
- Chaque emplacement mémoire contient une valeur
Exemple : adresse \$f842 → contient la valeur \$4B8

Ne pas confondre

- la vision "système" : à chaque adresse correspond un **octet**
- la vision "composant" (réalité physique) :
 - les puces de mémoire ont **toujours** un nombre d'emplacement égal à 2^n , avec n le nombre de bits du bus d'adresse du composant,
 - chaque emplacement peut contenir 1,2,4,... 8, ..., 16, ... bits.

Exemple de composant mémoire (virtuel)



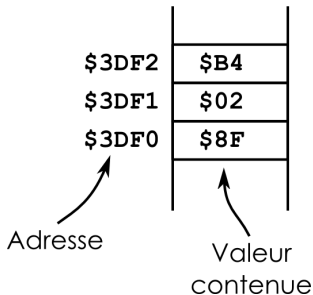
- Exemple de contenu :

	adresse	valeur
0	000	10
1	001	00
2	010	00
3	011	00
4	100	00
5	101	10
6	110	11
7	111	00

- bus d'adresse : 3 bits
⇒ 8 emplacements mémoires (adresses 0 à 7)
- bus de données : 2 bits
- $2^3 = 8$ mots de 2 bits ⇒ Capacité = 16 bits
 - Entrée \overline{CS} (Chip Select) : activation des sorties (actif à 0)

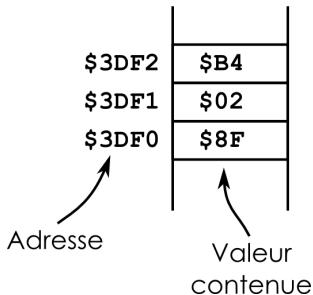
La mémoire

- Aspect logique : empilement de "cases mémoires", numérotées (adresse).
 - quantité : fonction de la taille du bus d'adresse
 - valeur contenue : octet (8 bits) (convention quasi-universelle)



La mémoire

- Aspect logique : empilement de "cases mémoires", numérotées (adresse).
 - quantité : fonction de la taille du bus d'adresse
 - valeur contenue : octet (8 bits) (convention quasi-universelle)



- Aspect physique : organisée en mots de 8, 16, 32, ... 128 bits.



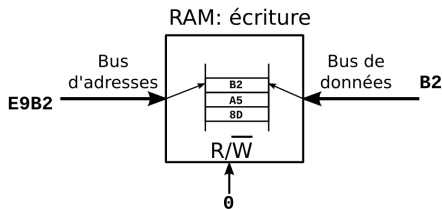
Capacité mémoire

- Une mémoire ayant n bits d'adresses aura une capacité de 2^n "emplacements".
 - Au niveau composant, emplacement : 1, 2, 4 ou 8 bits.
 - Au niveau système : emplacement = octet
- Exemples de capacité mémoire :

n	capacité	Etendue mémoire	
		1ère	dernière
8	256 = \$100	\$00	\$ff
10	1024 = \$400	\$000	\$3ff
14		\$0000	
16		\$0000	
18		\$00000	\$3ffff
20		\$00000	\$fffff

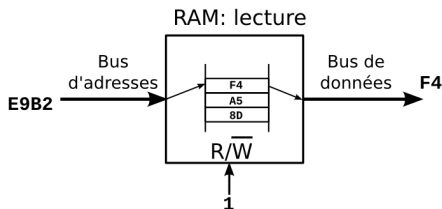
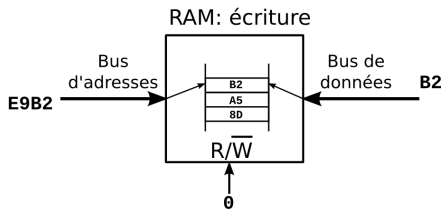
Mémoire RAM : lecture et écriture

- Une mémoire possède deux bus
- Pour la RAM, une entrée permet de sélectionner l'opération : R/\bar{W} (*Read / Write*)



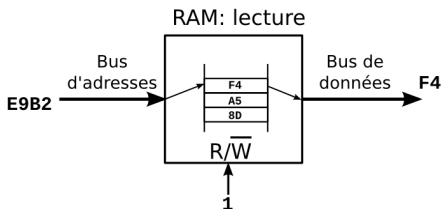
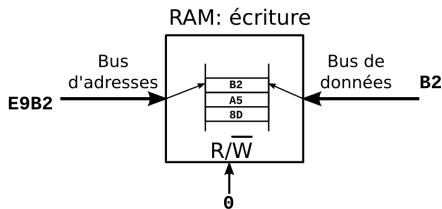
Mémoire RAM : lecture et écriture

- Une mémoire possède deux bus
- Pour la RAM, une entrée permet de sélectionner l'opération : R/\bar{W} (*Read / Write*)



Mémoire RAM : lecture et écriture

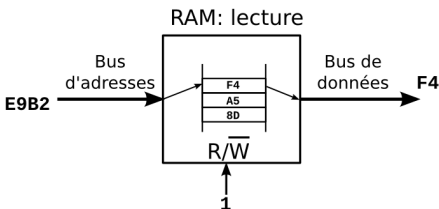
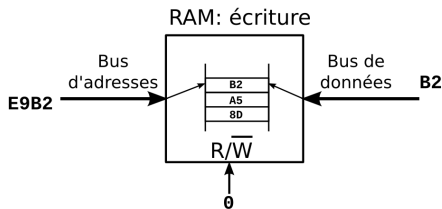
- Une mémoire possède deux bus
- Pour la RAM, une entrée permet de sélectionner l'opération : R/\bar{W} (*Read / Write*)



- Une ROM ne dispose en mode normal que du mode "lecture"

Mémoire RAM : lecture et écriture

- Une mémoire possède deux bus
- Pour la RAM, une entrée permet de sélectionner l'opération : R/\bar{W} (*Read / Write*)

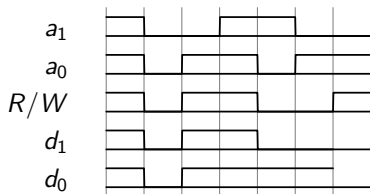


- Une ROM ne dispose en mode normal que du mode "lecture"
- Mais la plupart sont reprogrammable électriquement, sous certaines contraintes.

Mémoire RAM : Exercice

Soit une RAM ayant 2 bits d'adresses (a_1, a_0) et 2 bits de données (d_1, d_0).

- Capacité (en bits) :
- On suppose qu'elle est remplie initialement avec des 1.
- Le CPU génère les signaux suivants :



- Contenu de la RAM avant :

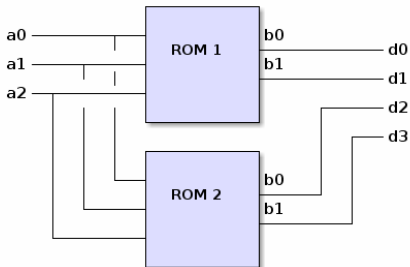
Adr.	Contenu
0	3
1	3
2	3
3	3

- Contenu de la RAM après :

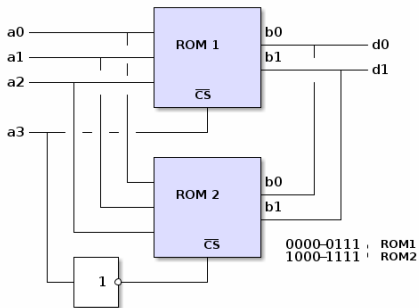
Adr.	Contenu
0	
1	
2	
3	

Assemblage de boîtiers mémoire

- On regroupe ensemble des composants mémoire pour augmenter la capacité.
- Deux approches, qui peuvent être combinées :
 - Augmentation du nombre de bits par adresse,
 - Augmentation du nombre d'adresses.
- Exemple : avec deux ROM de 8 mots (3 bits d'adresse) de 2 bits :
- 8 emplacements de 4 bits :

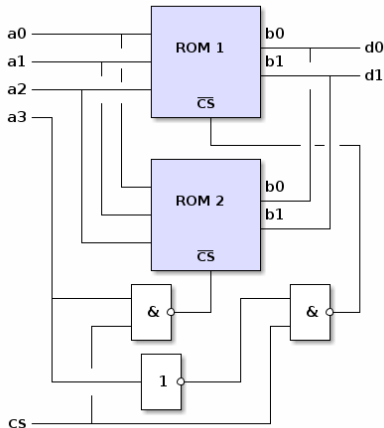
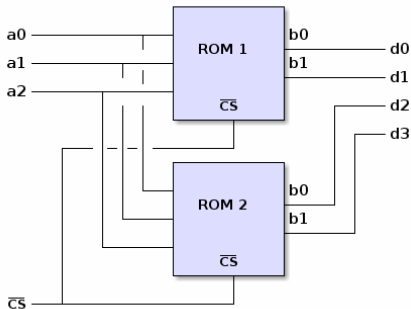


- 16 emplacements de 2 bits :



Assemblage de boîtiers mémoire

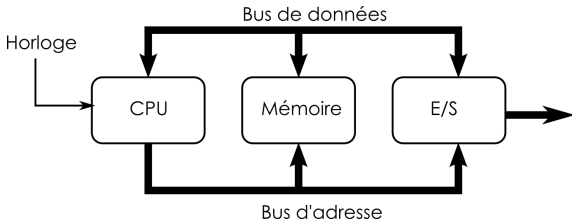
- Attention : dans un vrai système, il ne faudra activer les boîtiers que si c'est ce bloc mémoire qui est activé par le CPU.
- 8 emplacements de 4 bits :
- 16 emplacements de 2 bits :



Sommaire

- 1 Historique & évolutions contemporaines
- 2 Mémoire
- 3 Structure d'un système informatique**
- 4 Structure d'un CPU
 - Jeu d'instructions
 - Performances et évolution des architectures

Système μ -informatique



- 3 éléments fondamentaux :
 - CPU : *Central Processing Unit*
 - Mémoire : contient le programme et les données
 - E/S : interface avec l'extérieur
- Bus : interconnexions entre composants sur 'n' fils.
⇒ permet de transférer une valeur numérique codée en binaire.
 - Le bus d'adresse est unidirectionnel, et piloté par le CPU.
 - Le bus de données est bidirectionnel.
- Un microcontrôleur contient ces 3 sous-ensembles sur la même puce.

Taille des bus

- La taille du bus de données détermine la **catégorie** du processeur (8 bits, 16 bits, 32 bits, ...).
- La taille du bus d'adresse détermine l'**espace mémoire adressable**
⇒ combien de "cases mémoire" pourront être adressés (octets).
 - 16 bits → $2^{16} = 65\,536$ octets = 65,5 ko (64×1024)
 - 20 bits → $2^{20} = 1\,048\,576$ octets = 1,04 Mo (1024×1024)
 - 32 bits → $2^{32} = 4\,294\,967\,296$ octets = 4,29 Go
- L'espace mémoire adressable n'est pas forcément occupé en totalité.

Taille des bus

- La taille du bus de données détermine la **catégorie** du processeur (8 bits, 16 bits, 32 bits, ...).
- La taille du bus d'adresse détermine l'**espace mémoire adressable**
⇒ combien de "cases mémoire" pourront être adressés (octets).
 - 16 bits → $2^{16} = 65\,536$ octets = 65,5 ko (64×1024)
 - 20 bits → $2^{20} = 1\,048\,576$ octets = 1,04 Mo (1024×1024)
 - 32 bits → $2^{32} = 4\,294\,967\,296$ octets = 4,29 Go
- L'espace mémoire adressable n'est pas forcément occupé en totalité.

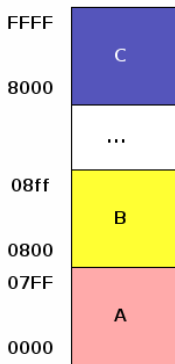
Attention

- 1kB = 1000 octets, 1 MB = 10^6 octets
- 1kiB = 1024 octets, 1MiB = 1 048 576 octets

La notation 1024 octets = 1 ko est **abusive**, mais souvent utilisée en pratique.

Plan mémoire

- Montre la répartition des différents " blocs mémoire" dans l'espace adressable du processeur
- Exemple : soit un " petit" système à bus d'adresse de 16 bits
 - l'espace adressable du processeur est de $2^{16} = 65536 = \$10000$ octets
 - Chaque zone mémoire est placée à un endroit de cette zone via un mécanisme d'adressage



A : [$\$0000 - \$07FF$] = $\$0800 = 2048$ octets

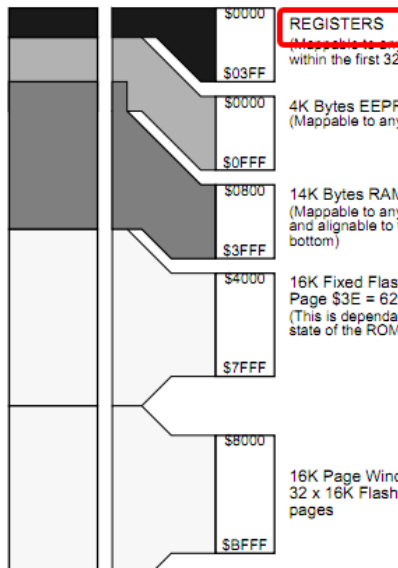
B : [$\$0800 - \$08FF$] = $\$0100 = 256$ octets

C : [$\$8000 - \$FFFF$] = $\$8000 = 32768$ octets

Plan mémoire et E/S

- Sur microcontrôleur, les E/S sont souvent "mappées" sur la mémoire : on accède aux registres de contrôles(*) comme à une case mémoire.

(*) Registres permettant le contrôle des différents blocs d'E/S du microcontrôleur, à distinguer des registres-CPU (voir plus loin).



Exercices

- 1 Soit un circuit intégré de mémoire affiché comme faisant "256kbits".
 - Quelle est sa capacité réelle, en bits? en octets?
 - Si le bus de données de fait 8 bits, combien de bits faudra-t-il sur le bus d'adresse pour accéder à l'ensemble de ces adresses?

Exercices

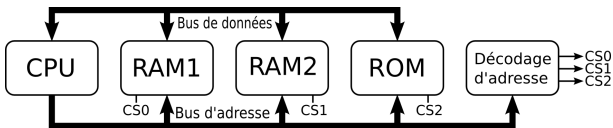
- 1 Soit un circuit intégré de mémoire affiché comme faisant "256kbits".
 - Quelle est sa capacité réelle, en bits? en octets?
 - Si le bus de données de fait 8 bits, combien de bits faudra-t-il sur le bus d'adresse pour accéder à l'ensemble de ces adresses?
- 2 Soit une adresse notée en hexa 0xF2A0. Quelle est la taille du bus d'adresse de ce système?

Exercices

- 1 Soit un circuit intégré de mémoire affiché comme faisant "256kbits".
 - Quelle est sa capacité réelle, en bits? en octets?
 - Si le bus de données de fait 8 bits, combien de bits faudra-t-il sur le bus d'adresse pour accéder à l'ensemble de ces adresses?
- 2 Soit une adresse notée en hexa 0xF2A0. Quelle est la taille du bus d'adresse de ce système?
- 3 Soit un bloc de mémoire dont la première adresse est 0x0AB000 et la dernière 0x0AB7ff
 - Quelle est la taille du bus d'adresse de ce système?
 - Quelle est la taille de ce bloc mémoire, en octets? En bits?

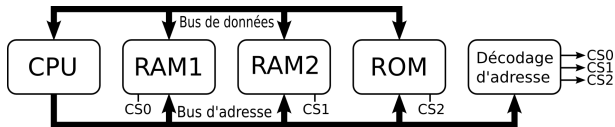
Adressage mémoire

- Les différents blocs (mémoire, périphériques) sont interconnectés, il faut un mécanisme permettant de ne sélectionner **que** celui qui est effectivement adressé par le processeur.



Adressage mémoire

- Les différents blocs (mémoire, périphériques) sont interconnectés, il faut un mécanisme permettant de ne sélectionner **que** celui qui est effectivement adressé par le processeur.



- Ceci se fait sur chaque boîtier physique par un signal de validation (souvent appelé **CS**, pour *Chip Select*). Il est (en général) actif à 0 ce qui fait qu'il est désigné par \overline{CS} .
Si ce signal n'est **pas** activé, alors le boîtier est automatiquement **désactivé**.
- L'électronique associée au CPU doit inclure un circuit d'**adressage mémoire** qui va activer ou non les différents boîtiers de mémoire, selon l'adresse placée sur le bus par le CPU.

Adressage mémoire : exemple

- Exemple : sur le plan mémoire précédent, le bloc de 4KiB d'EEPROM ne doit être activé **que** si l'adresse présente sur le bus a une valeur dans la plage \$0000 à \$0FFF.

Adressage mémoire : exemple

- Exemple : sur le plan mémoire précédent, le bloc de 4KiB d'EEPROM ne doit être activé **que** si l'adresse présente sur le bus a une valeur dans la plage \$0000 à \$0FFF.
- La condition commune sur toute cette plage est que les 4 premiers bits soient à 0 :
"Si les bits $a_{15}, a_{14}, a_{13}, a_{12}$ sont à 0, ALORS, CS doit être à 0"

Adressage mémoire : exemple

- Exemple : sur le plan mémoire précédent, le bloc de 4KiB d'EEPROM ne doit être activé **que** si l'adresse présente sur le bus a une valeur dans la plage \$0000 à \$0FFF.
- La condition commune sur toute cette plage est que les 4 premiers bits soient à 0 :
"Si les bits a_{15} , a_{14} , a_{13} , a_{12} sont à 0, ALORS, CS doit être à 0"

$$\Rightarrow \overline{CS} = \overline{a_{15}} \cdot \overline{a_{14}} \cdot \overline{a_{13}} \cdot \overline{a_{12}}$$

Adressage mémoire : exemple

- Exemple : sur le plan mémoire précédent, le bloc de 4KiB d'EEPROM ne doit être activé **que** si l'adresse présente sur le bus a une valeur dans la plage \$0000 à \$0FFF.
- La condition commune sur toute cette plage est que les 4 premiers bits soient à 0 :
"Si les bits a_{15} , a_{14} , a_{13} , a_{12} sont à 0, ALORS, CS doit être à 0"

$$\Rightarrow \overline{CS} = \overline{a_{15}} \cdot \overline{a_{14}} \cdot \overline{a_{13}} \cdot \overline{a_{12}} \Rightarrow CS = \overline{\overline{\overline{\overline{\overline{a_{15}} \cdot \overline{a_{14}} \cdot \overline{a_{13}} \cdot \overline{a_{12}}}}}}}$$

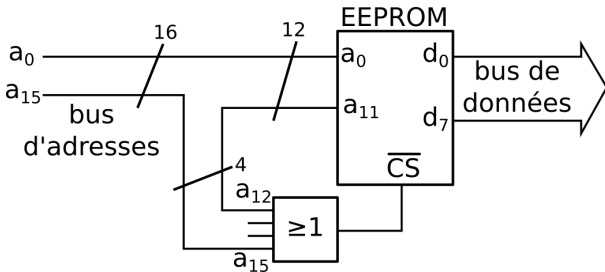
$$\Rightarrow CS = a_{15} + a_{14} + a_{13} + a_{12}$$

Adressage mémoire : exemple

- Exemple : sur le plan mémoire précédent, le bloc de 4KiB d'EEPROM ne doit être activé **que** si l'adresse présente sur le bus a une valeur dans la plage \$0000 à \$0FFF.
- La condition commune sur toute cette plage est que les 4 premiers bits soient à 0 :
"Si les bits a_{15} , a_{14} , a_{13} , a_{12} sont à 0, ALORS, CS doit être à 0"

$$\Rightarrow \overline{CS} = \overline{a_{15}} \cdot \overline{a_{14}} \cdot \overline{a_{13}} \cdot \overline{a_{12}} \Rightarrow CS = \overline{\overline{a_{15}} \cdot \overline{a_{14}} \cdot \overline{a_{13}} \cdot \overline{a_{12}}}$$

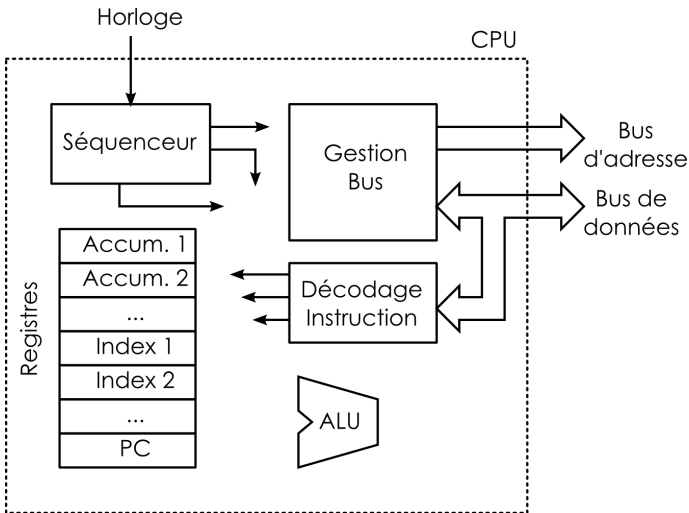
$$\Rightarrow CS = a_{15} + a_{14} + a_{13} + a_{12}$$



Sommaire

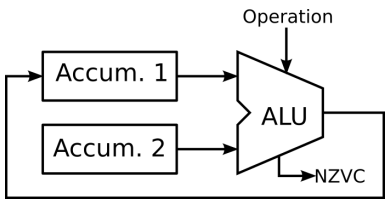
- 1 Historique & évolutions contemporaines
- 2 Mémoire
- 3 Structure d'un système informatique
- 4 Structure d'un CPU
 - Jeu d'instructions
 - Performances et évolution des architectures

CPU : structure



- Les différents éléments sont interconnectables.

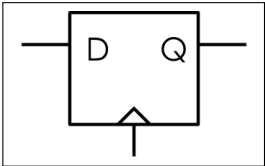
ALU : Arithmetic and Logic Unit



- Caractéristique fondamentale : taille des données (8, 16, 32, ... bits).
- Opérations
 - Logiques : AND, OR, EXOR, complément à 1, à 2, décalages.
 - Arithmétiques
- Bits d'état (NZVC) : fournissent une information sur le résultat.
 - N (*Negative*) : recopie du bit de poids fort, signale un nombre négatif en arithmétique signée
 - Z (*Zero*) : signale un résultat = 0
 - V (*oVerflow*) : retenue (arithmétique signée)
 - C (*Carry*) : retenue (arithmétique non-signée)

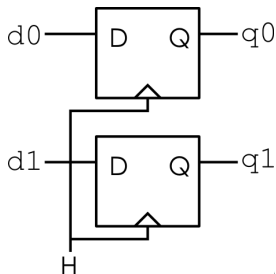
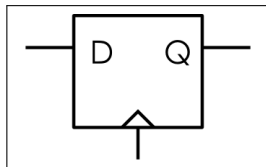
Registres / CPU

- Rappel : bascule D
Fonctionnement : "*Q recopie D sur le front d'horloge*"
⇒ mémoire 1 bit.



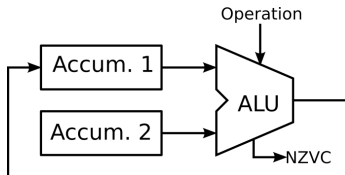
Registres / CPU

- Rappel : bascule D
Fonctionnement : "*Q recopie D sur le front d'horloge*"
⇒ mémoire 1 bit.
- Assemblage de 2 bascules : mémoire 2 bits.
- Un registre 'n' bits est un assemblage de 'n' bascules D, synchronisées par un signal commun.



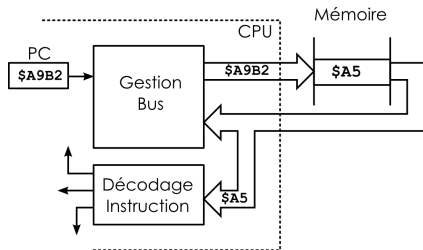
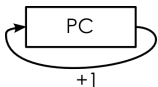
Registres / CPU

- Utilisés pour mémoriser les données utilisées par l'ALU.
- On distingue
 - les **accumulateurs**, utilisés pour les calculs,
 - les registres d'**index**, pour gérer des pointeurs,
 - les registres système :
 - *PC Program Counter* : contient l'adresse de l'instruction en cours d'exécution,
 - Pointeur de Pile (SP : *Stack Pointer*),
 - d'autres registres système.
- Les calculs se font toujours via des registres.
- Par exemple, addition des deux accumulateurs :
 $Acc1 \leftarrow Acc1 + Acc2$



Exécution d'une instruction

- L'exécution d'une instruction se fait en plusieurs cycles d'horloge : le **séquenceur** pilote son déroulement.
 - La durée d'exécution est fonction de la complexité de l'opération (3, 4, 5, 6 ... cycles d'horloge, voire plus).
- 1 lecture en mémoire à l'adresse indiquée par PC du code opératoire de l'instruction,
 - 2 décodage et configuration des interconnexions internes,
 - 3 exécution,
 - 4 incrémentation compteur ordinal.



Sommaire

- 1 Historique & évolutions contemporaines
- 2 Mémoire
- 3 Structure d'un système informatique
- 4 Structure d'un CPU
 - Jeu d'instructions
 - Performances et évolution des architectures

Jeu d'instructions

- Défini par le constructeur, et propre à la famille du processeur.
- 4 catégories d'instructions :
 - ① Transfert de données (mémoire ↔ registre, registre ↔ registre, mémoire ↔ mémoire).
 - ② Opérations arithmétiques et logiques (ALU).
 - ③ Contrôle de séquence : branchements / sauts (déroutement de l'exécution).
 - ④ Instructions système diverses.

Jeu d'instructions

- Défini par le constructeur, et propre à la famille du processeur.
- 4 catégories d'instructions :
 - ① Transfert de données (mémoire ↔ registre, registre ↔ registre, mémoire ↔ mémoire).
 - ② Opérations arithmétiques et logiques (ALU).
 - ③ Contrôle de séquence : branchements / sauts (déroutement de l'exécution).
 - ④ Instructions système diverses.
- Le processeur ne reconnaît que le **code opératoire** : valeur binaire unique qui correspond au code de cette instruction.
- On désigne les instructions par un **mnémonique**.
Exemple : *Load Accumulator A*, noté `ldaa` : charge le registre-accumulateur A avec une valeur.

Implantation d'une instruction en mémoire

- Le nombre d'octets occupé par une instruction est variable.
- Certaines instructions ont besoin d'une **opérande** (donnée sur laquelle travailler), qui peut être fournie de différentes façons
- Par exemple (processeur 9s12) :
 - `inx` : incrémentation du registre X :
⇒ 1 octet : \$08

Implantation d'une instruction en mémoire

- Le nombre d'octets occupé par une instruction est variable.
- Certaines instructions ont besoin d'une **opérande** (donnée sur laquelle travailler), qui peut être fournie de différentes façons
- Par exemple (processeur 9s12) :
 - `inx` : incrémentation du registre X :
⇒ 1 octet : \$08
 - `ldaa #1` : charge le registre A avec la valeur 1 :
⇒ 2 octets (instruction + opérande) : \$86 \$01

Implantation d'une instruction en mémoire

- Le nombre d'octets occupé par une instruction est variable.
- Certaines instructions ont besoin d'une **opérande** (donnée sur laquelle travailler), qui peut être fournie de différentes façons
- Par exemple (processeur 9s12) :
 - `inx` : incrémentation du registre X :
⇒ 1 octet :

\$08

 - `ldaa #1` : charge le registre A avec la valeur 1 :
⇒ 2 octets (instruction + opérande) :

\$86	\$01
------	------
 - `movb $1234,$5678` : copie l'octet situé en \$1234 dans la case-mémoire \$5678 :
⇒ 6 octets : instruction (2) + opérande 1 (2) + opérande 2 (2)

\$18	\$0C	\$12	\$34	\$56	\$78
------	------	------	------	------	------

Sommaire

- 1 Historique & évolutions contemporaines
- 2 Mémoire
- 3 Structure d'un système informatique
- 4 Structure d'un CPU
 - Jeu d'instructions
 - Performances et évolution des architectures

Pipeline d'exécution (1)

- Constat : l'exécution d'une instruction nécessite plusieurs étapes :

(Certaines instructions n'auront pas toutes ces étapes. Par ex :
incrémenter d'un registre)

Pipeline d'exécution (1)

- Constat : l'exécution d'une instruction nécessite plusieurs étapes :
 - ① lecture en mémoire de l'instruction

(Certaines instructions n'auront pas toutes ces étapes. Par ex :
incrémenter d'un registre)

Pipeline d'exécution (1)

- Constat : l'exécution d'une instruction nécessite plusieurs étapes :
 - ① lecture en mémoire de l'instruction
 - ② décodage

(Certaines instructions n'auront pas toutes ces étapes. Par ex :
incrémenter d'un registre)

Pipeline d'exécution (1)

- Constat : l'exécution d'une instruction nécessite plusieurs étapes :
 - ① lecture en mémoire de l'instruction
 - ② décodage
 - ③ configuration des bus internes

(Certaines instructions n'auront pas toutes ces étapes. Par ex :
incrémenter d'un registre)

Pipeline d'exécution (1)

- Constat : l'exécution d'une instruction nécessite plusieurs étapes :
 - ① lecture en mémoire de l'instruction
 - ② décodage
 - ③ configuration des bus internes
 - ④ exécution

(Certaines instructions n'auront pas toutes ces étapes. Par ex :
incrémenter d'un registre)

Pipeline d'exécution (1)

- Constat : l'exécution d'une instruction nécessite plusieurs étapes :
 - ① lecture en mémoire de l'instruction
 - ② décodage
 - ③ configuration des bus internes
 - ④ exécution
 - ⑤ copie du résultat en mémoire

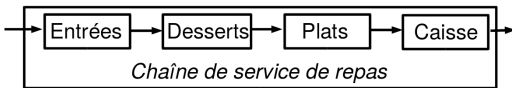
(Certaines instructions n'auront pas toutes ces étapes. Par ex :
incrémenter d'un registre)

Pipeline d'exécution (1)

- Constat : l'exécution d'une instruction nécessite plusieurs étapes :
 - ① lecture en mémoire de l'instruction
 - ② décodage
 - ③ configuration des bus internes
 - ④ exécution
 - ⑤ copie du résultat en mémoire

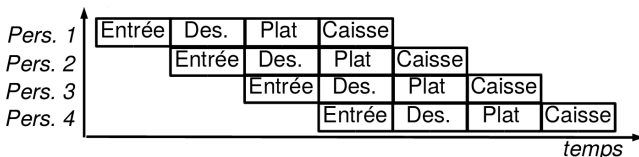
(Certaines instructions n'auront pas toutes ces étapes. Par ex :
incrémenter d'un registre)

- Analogie : restaurant type self



Pipeline d'exécution (2)

- Idée : plutôt que de faire l'ensemble des opérations en entier pour chaque instruction, on exécute les opérations "en parallèle".



source : E. Cariou

Pipeline d'exécution (3)

- Soit un processeur (fictif) où 5 cycles sont nécessaires pour exécuter une instruction :
 - ① IF (*Instruction Fetch*) : charge l'instruction depuis la mémoire,
 - ② ID (*Instruction Decode*) : décode l'instruction,
 - ③ EX : exécute l'instruction (via l'ALU),
 - ④ MEM (*Memory*) : transfert registre ↔ mémoire,
 - ⑤ WB (*Write Back*) : stocke le résultat dans un registre.

Pipeline d'exécution (3)

- Soit un processeur (fictif) où 5 cycles sont nécessaires pour exécuter une instruction :
 - ① IF (*Instruction Fetch*) : charge l'instruction depuis la mémoire,
 - ② ID (*Instruction Decode*) : décode l'instruction,
 - ③ EX : exécute l'instruction (via l'ALU),
 - ④ MEM (*Memory*) : transfert registre ↔ mémoire,
 - ⑤ WB (*Write Back*) : stocke le résultat dans un registre.
- Pour chaque instruction, les différentes étapes vont utiliser 1 cycle d'horloge :



Ex : 3 instructions \Rightarrow 15 cycles d'horloge.

Pipeline d'exécution (4)

- Idée : Si chaque étape correspond à un bloc hardware différent, on peut l'utiliser pour une autre instruction une fois terminée son utilisation pour l'instruction en cours.
- La durée d'exécution globale est raccourcie :

Instr. No.	Pipeline Stage						
	IF	ID	EX	MEM	WB		
1							
2							
3							
4							
5							
Clock Cycle	1	2	3	4	5	6	7

source:wikimedia/Poill



Ex : 3 instructions \Rightarrow 7 cycles d'horloge.

- Inconvénient : l'exécution de certaines instructions dépend parfois du résultat des précédentes.

Conclusion

Les processeurs contemporains sont d'une complexité extrême...



Conclusion

Les processeurs contemporains sont d'une complexité extrême...

