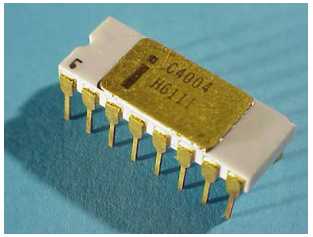
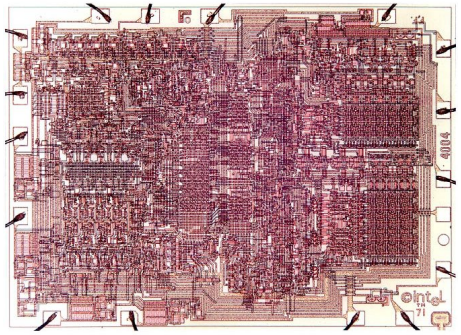
# Introduction aux microprocesseurs

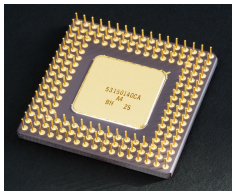
## 1 Historique

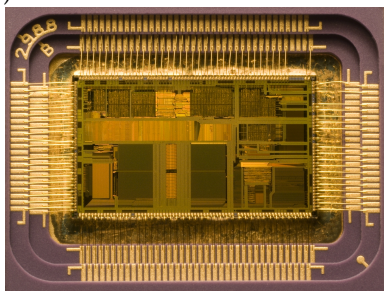
— 1971 : Intel 4004 : premier microprocesseur, 4 bits, 2300 transistors, 108 kHz



— 1978 : Intel 8088/8086, 16 bits, architecture "x86", utilisé dans les "IBM PC"
— 1985 : Intel 80386 (32 bits)
— 1989 : Intel 486 ( $>$ 1 million transistors)

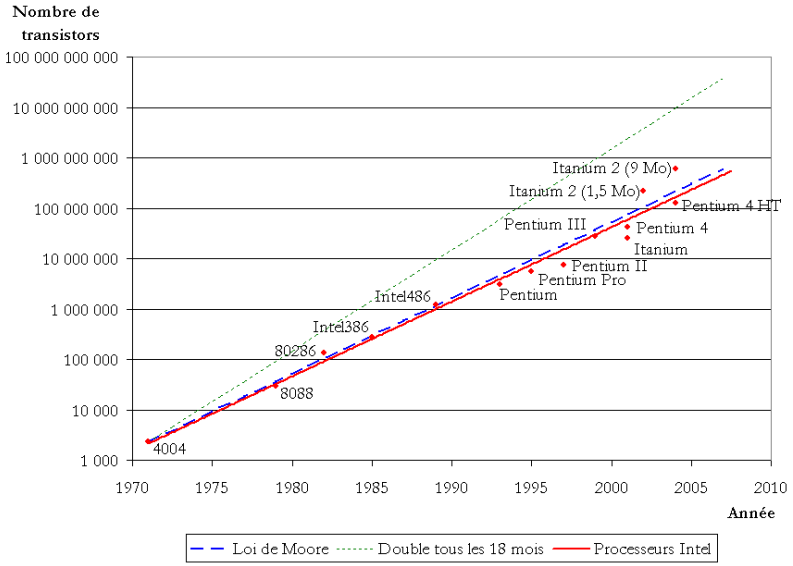




taille réelle : 12 x 7 mm

— 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)** Gordon Moore, cofondateur d'Intel, énonce en 1975 que à l'avenir "*Le nombre de transistors va doubler tous les 2 ans*"




Cette loi empirique s'est révélée étonnament exacte pendant 40 ans, même si aujourd'hui, la croissance de la densité d'intégration ralentit, du fait des difficultés technologiques de la finesse de gravure du Silicium.

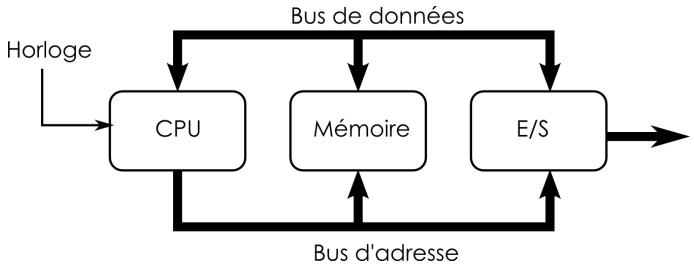
**Evolutions et perspectives** L'augmentation de la puissance de calcul des dernières années s'est faite principalement sur deux axes :

— Augmentation de la fréquence d'horloge. Aujourd'hui : $\sim$ 1GHz.
— Augmentation de la densité d'intégration (finesse de gravure). Aujourd'hui : $<$40 nm

Le problème majeur aujourd'hui est celui de la dissipation de chaleur ($\sim$ 100 W), qui devient trop importante Comme on ne pourra pas repousser indéfiniment ces limites physiques, les constructeurs s'orientent vers des évolutions des architectures (*pipeline* d'exécution, architecture multicœur, etc.)

## 2 Généralités

Tout système $\mu$-informatique contemporain peut-être réduit au diagramme suivant :




On distingue trois éléments fondamentaux :

— le CPU : *Central Processing Unit* ;
— la mémoire, qui contient le programme et les données ;
— les E/S, qui permettent l'interface avec l'extérieur.

Ces trois élements sont interconnectés par deux **bus**, chacun utilisant un certain nombre de connexions, qui permettent de transférer une valeur binaire.

— Le bus d'adresse est unidirectionnel, et piloté par le CPU.
— Le bus de données est bidirectionnel.

En sus (non-représenté ici), il y a un certain nombre de signaux de contrôle, permettant la **synchronisation** des échanges de données entre ces éléments.

## 2.1 Bus de communication

Chacun de ses bus peut avoir une largeur différente.

— 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**, c'est-à-dire combien de "cases mémoire" pourront être adressées (octets).
  — 16 bits $\rightarrow 2^{16}$ = 65 536 octets = 65,5 ko (64 $\times$ 1024)
  — 20 bits $\rightarrow 2^{20}$ = 1 048 576 octets = 1,04 Mo (1024 $\times$ 1024)
  — 32 bits $\rightarrow 2^{32}$ = 4 294 967 296 octets = 4,29 Go

En général, on aura cependant un bus d'adresse au moins égal au bus de données.

**Attention :** La notation 1024 octets = 1 ko est **abusive**, mais parfois encore utilisée en pratique.

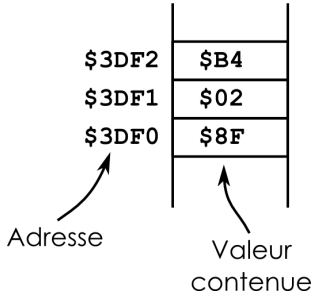
L'humain étant peu à l'aise pour manipuler des nombres exprimés en binaire, il est d'usage d'utiliser l'hexadécimal pour représenter les adresses. Ainsi sur un système à bus d'adresse de 32 bits, l'adresse 1000.0111.1111.0000.1111.1111.0000.0001 s'écrira `$87F0FF01`.

## 2.2 La mémoire

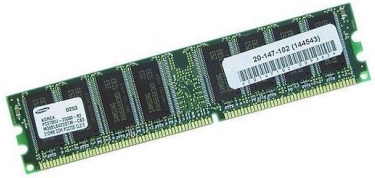
La mémoire est un terme générique désignant tout ce qui a une capacité de mémorisation. A noter que l'on ne parle ici que de **mémoire de travail**, par opposition à la **mémoire de masse** (de type "disque dur") qui est ici considérée comme un périphérique auquel on accède via un controleur dédié.

On distingue l'aspect logique et l'aspect physique.

— D'un point de vue logique, la mémoire peut être vue comme une empilement de "cases mémoires", qui sont toutes numérotées (adresse). La quantité est fonction du bus d'adresse su système considéré. La valeur contenue dans chaque case est (de façon quasi-universelle) toujours un **octet** (8 bits).
  Même sur un système à bus de données de 32 bits, chacun des 4 octets qui composent un mot sur le bus se verra attribué une adresse différente.




— D'un point de vue physique, la mémoire peut être organisée en mots de 8, 16, 32, ... 128 bits, quelle que soit la réalité logique derrière.



Barette de DRAM

### 2.2.1 Aspect physique

On distingue aussi deux types de mémoire :

— la mémoire "vive", dont le contenu est **non persistant**. On parle de RAM (*Random Access Memory*). Ce type peut-être mis en œuvre à travers deux technologies, la DRAM (*Dynamic RAM*), dont l'élement mémoire de base est un condensateur, et la SRAM (*Static RAM*), dont l'élement de base est une bascule (type

RS). Cette dernière est cependant plus chère, et réservée à des usages particuliers.

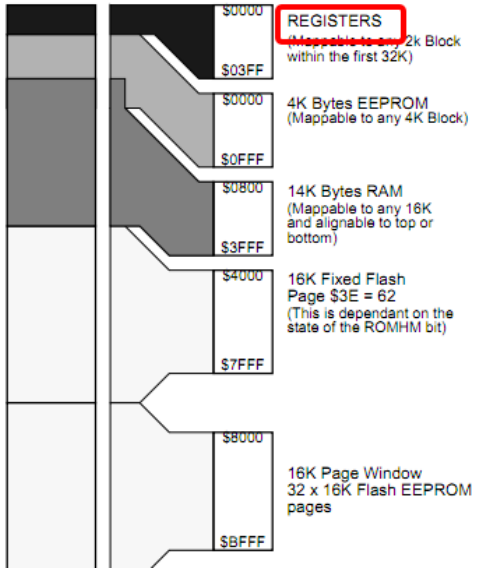
— la mémoire "morte", au contenu **persistant** (ROM : *Read Only Memory*). Elle est la plupart du temps accessible en lecture seule (une opération d'écriture reste sans aucun effet), bien que certains types peuvent être ré-écrites, mais à des vitesses toujours bien inférieures à la RAM.
Il en existe de nombreux types (ROM, PROM, EPROM, EEPROM), mais la plus utilisée aujourd'hui est la **mémoire "flash"**, qui présente le prix à l'octet le plus faible. A noter que ce type de mémoire présente un nombre de cycles d'écritures **limité**.

### 2.2.2 Plan mémoire et E/S

Très fréquemment, les E/S sont souvent "mappées" sur la mémoire : on accède aux registres de contrôles des E/S comme à une case mémoire. Cela réduit d'autant l'espace mémoire adressable.

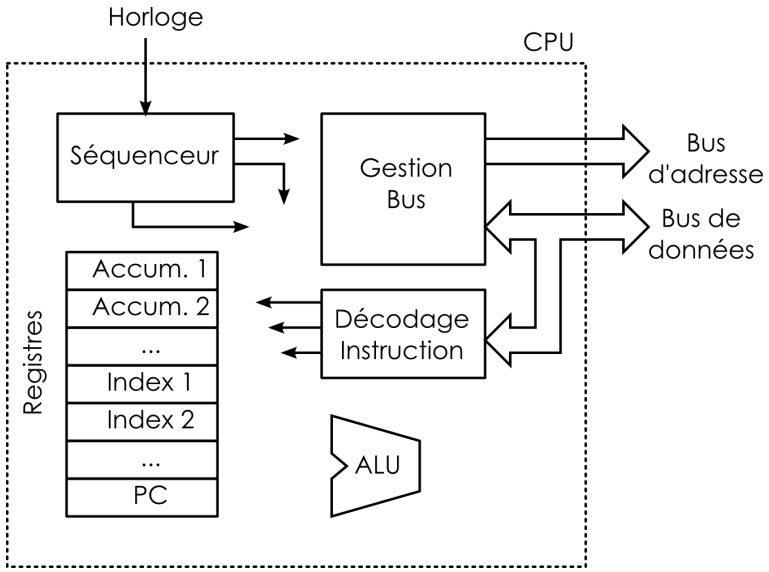
Par exemple, sur une architecture à bus d'adresse de 16 bits, si les E/S sont gérées via un bloc de 1024 adresses, au lieu de pouvoir implanter $2^{16} = 65536$ cases mémoires, on ne pourra en utiliser que $65536 - 1024 = 64512$.

Ci-dessous le plan mémoire d'un processeur à bus d'adresse de 16 bits montrant la nature des différentes zones mémoires, ainsi que les registres d'E/S qui occupent une part de cet espace mémoire.




## 2.3 Structure d'un CPU

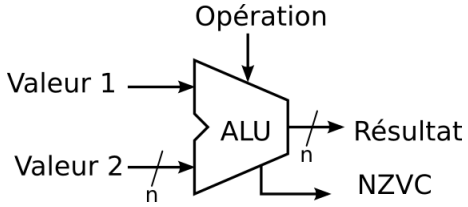
Un CPU réduit à son niveau le plus élementaire est constitué à minima des élements suivants :




— une **ALU** ;
— des registres-CPU ;
— un bloc "gestion des bus", qui se charge de placer l'adresse désirée sur le bus d'adresse, et qui va lire la valeur en mémoire via le bus de données (en cas de lecture en mémoire) ;
— un bloc "décodage" : ce bloc prend en entrée une valeur binaire récupérée depuis la mémoire, va interpréter ce code comme une instruction, et va ensuite configurer les connexions internes de façon à permettre son exécution ;
— un séquenceur, qui va piloter les différentes étapes nécessaires à l'exécution de l'instruction en cours d'exécution.

Les différents élements sont interconnectables.

### 2.3.1 ALU : *Arithmetic and Logic Unit*




La caractéristique fondamentale d'une ALU est la taille des données qu'elle manipule (n : 8, 16, 32, ... bits).

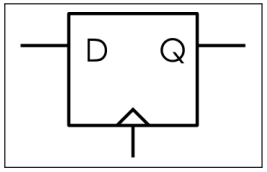
Une entrée supplémentaire permet de sélectionner l'opération à réaliser entre les deux opérandes[1]. Les opérations classiques sont **logiques** (AND, OR, XOR, complément à 1, à 2, décalages). ou **arithmétiques** (+,-,×,/).

Toutes les ALU fournissent en plus du résultat une information sur d'éventuels débordements du calcul, sous la forme de bits d'états. Il sont à minima au nombre de 4 :

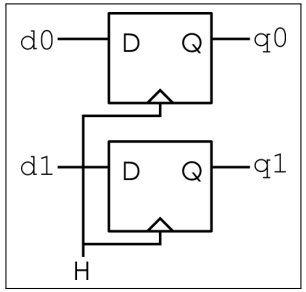
— N (*Negative*) : recopie du bit de poids fort, signale un nombre négatif en arithmétique signée
— Z (*Zero*) : signale un résultat = 0
— V (*oVerflow*) : retenue (arithmétique signée)
— C (*Carry*) : retenue (arithmétique non-signée)

### 2.3.2 Registres / CPU

**Rappel** Le fonctionnement d'une bascule D est décrit par la phrase *"Q recopie D sur le front d'horloge"*. On peut donc considérer cette bascule comme une "mémoire 1 bit".




Si on assemble deux bascules synchronisées par la même horloge, on dispose d'une mémoire 2 bits. Par généralisation, un **registre** $n$ bits est un assemblage de $n$ bascules D synchronisées par un signal commun.




1. A noter que certaines opérations (décalages) ne font intervenir que une seule opérande.

Les registres sont utilisés dans le CPU pour mémoriser des données temporaires, notamment pour les opérandes et les résultats de calculs faits par l'ALU. On distingue

— les **accumulateurs**, utilisés pour les calculs,
— les registres d'**index**, pour gérer des pointeurs (dont la valeur numérique contenue représente une adresse),
— 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.

Le nombre des registres dépend du CPU considéré, typiquement on dispose de 2 à 16 registres utilisateurs sur un CPU 16 bits, plus sur des CPU 32 ou 64 bits.

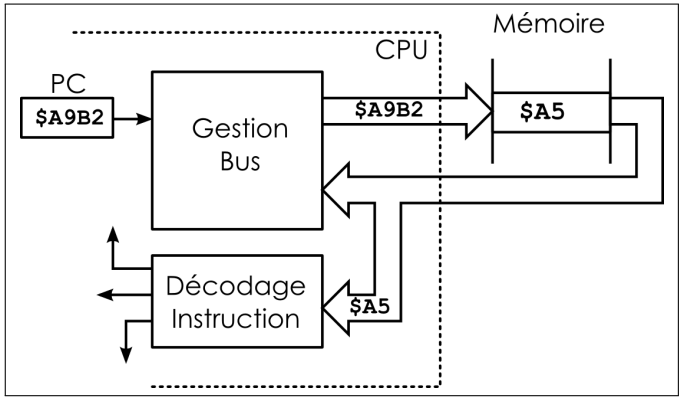
### 2.3.3 Exécution d'une instruction

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, et peut-être variable. Elle s'exprime en **cycles-d'horloge** (3, 4, 5, 6 ... cycles d'horloge, voire plus).

Ces différentes étapes sont :

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 (PC), pour passer à l'exécution de l'instruction suivante.

Ceci peut être représenté par les connexions suivantes montrant le cheminement des données sur les bus :




## 2.4 Jeu d'instructions

Il est défini par le constructeur, et propre à la famille du processeur. Il y a quatre catégories d'instructions :

1. Transfert de données (mémoire↔registre, registre ↔ registre, mémoire ↔ mémoire).
2. Opérations arithmétiques et logiques (impliquant l'ALU).
3. Contrôle de séquence : branchements / sauts (déroutement de l'exécution).
4. Instructions système diverses.

**Attention** Le processeur ne reconnaît que le **code opératoire** : valeur binaire unique qui correspond au code de cette instruction. Pour rendre la lecture d'un programme écrit en assembleur plus lisible, on désigne les instructions par un mot court, qu'on appelle un **mnémonique**.
Exemple : *Load Accumulator A*, noté `ldaa` : charge le registre-accumulateur A avec une valeur.

**Implantation d'une instruction en mémoire** Sur les architectures CISC, le nombre d'octets occupé par une instruction peut être **variable** (1, 2, 3, ... octets). Certaines instructions ont besoin d'une **opérande** (donnée sur laquelle travailler), qui peut être fournie de différentes façons.

Par exemple, une instruction d'incrémentation d'un registre n'a pas besoin d'opérande. A contrario, une instruction de chargement d'un registre avec une valeur ou le contenu d'une adresse mémoire doit connaître la valeur ou l'adresse à considérer. Cette valeur doit donc être **intégrée** dans l'instruction elle même, ce qui occupe donc des octets supplémentaires.

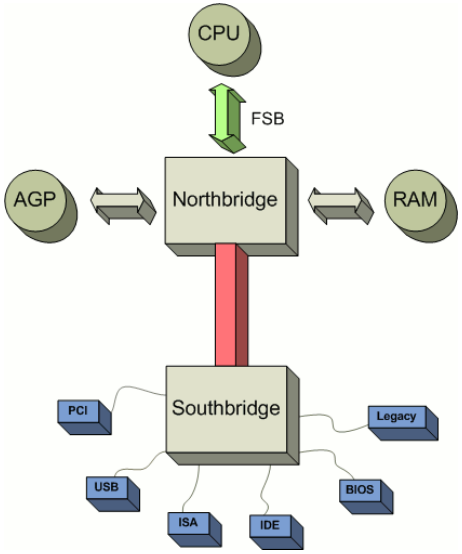
Sur architecture RISC, chaque instruction est en général codée par le même nombre de bits.

## 2.5 Microcontroleur vs. Microprocesseur

Ces deux élements sont proches mais quelques détails les séparent.

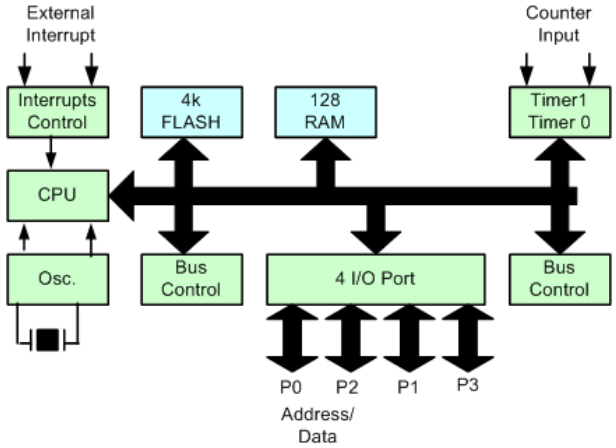
**Microprocesseur** C'est un composant généraliste, optimisé pour la puissance de calcul et la vitesse. Il ne peut fonctionner qu'accompagné de son *chipset*. Sur architecture PC classique, ce chipset est composé de de deux élements principaux :

- le *NorthBridge*, dédié à la communication avec la RAM et les périphériques rapides (carte graphique) ;
- le *SouthBridge*, dédié à la communication avec tous les autres périphériques.




**Microcontrôleur** Il est spécialement adapté aux applications embarquées et est capable de fonctionnement autonome. Il intègre un CPU, plus :

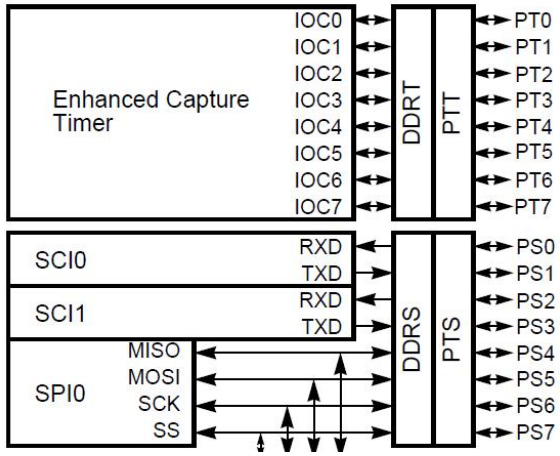
- de la mémoire
  - RAM
  - ROM, EEPROM, Flash
- et des gestionnaire de périphériques :
  - Communication
  - Acquisition (capteur, organe de commande, ...)
  - Pilotage actionneurs et afficheurs




Exemple d'architecture d'un microcontrôleur.

De façon à limiter le nombre de broches, les constructeurs regroupent fréquemment plusieurs fonctions sur une broche, l'utilisateur devra choisir (via le programme) comment il utilise la broche.

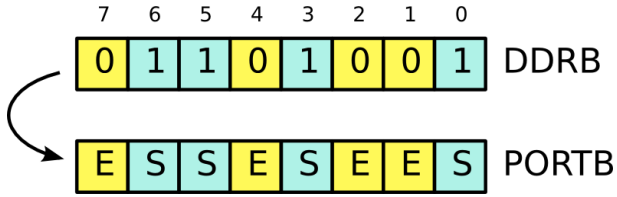
Par exemple (processeur 9s12), les broches peuvent être contrôlées soit un via un *bloc fonctionnel*, soit via un port binaire généraliste (*GPIO*) :




— Si le bloc *Enhanced Capture Timer* (ECT) est activé, les broches PT0 à PT7 seront contrôlées par celui-ci, sinon par les registres DDRT/PTT.
— Si le bloc SCI0 est activé, les broches PS0 et PS1 seront contrôlées par celui-ci, sinon par les bits 0 et 1 de DDRS & PTS.

**Ports GPIO : General Purpose Input Output** Il s'agit de broches regroupées en "port" (de 8 bits le plus souvent) qui peuvent servir aussi bien comme entrée que comme sortie, selon le choix du programmeur. Ce dernier doit écrire une valeur dans un registre particulier pour définir le sens de chaque broche du port. Par exemple, sur 9s12, ce sera pour le port B le registre DDRB (*Data Direction Register port B*).

— 0 : broche en entrée
— 1 : broche en sortie




Pour ensuite fixer un niveau logique sur les broches configurées en sortie, il faudra **écrire** dans le registre du port (sur 9s12, pour le port B, ce registre d'appelle PORTB).

Pour lire le niveau logique présent sur les broches configurées en entrées, il faudra **lire** dans le registre du port.