Cours 1 Microprocesseurs

Jalil Boukhobza
LC 206
boukhobza@univ-brest.fr
02 98 01 69 73

But de ce cours

- Comprendre le fonctionnement de base d'un microprocesseur séquentiel simple (non pipeliné? et non superscalaire?):
  - **1ère partie**: conception
    - **Eléments de base** d'un microprocesseur
    - Son **fonctionnement**
  - **2ème partie**: programmation de microprocesseur
  - **3ème partie** µ contrôleur 68HC11

Plan de la première partie

Cours d'aujourd'hui

1. Historique, performances et autre introduction
2. Représentation des nombres
3. Circuits logiques
   1. Combinatoires (et arithmétique)
   2. Séquentiels (et mémorisation)
4. Unités de traitement et unités de contrôle et l'utilisation d'automates
   1. Unité de contrôle câblée
   2. Unité de contrôle microprogrammée

À quoi sert ce cours ?

- Connaître **la base** de l'architecture d'un microprocesseur (pour un informaticien c'est un minimum !).
- Indispensable pour plusieurs UEs de L3 et de M1 (Architecture, Architectures parallèles, système & réseaux, etc.)
- Indispensable pour le *M2 Logiciels pour les Systèmes Embarqués*.




Les microprocesseurs hier, aujourd'hui et …demain ?

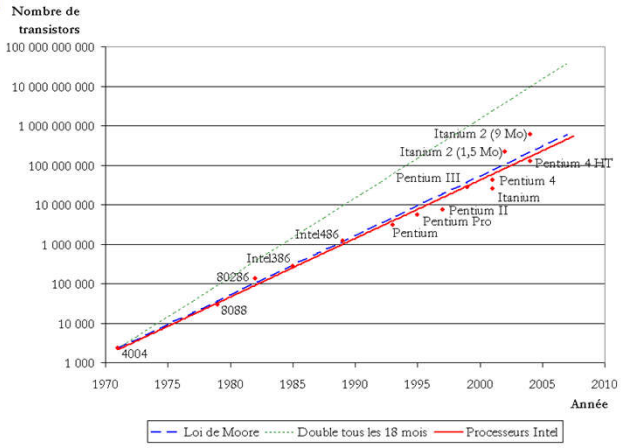
- Au niveau technologique (commutateur):
  Tubes à vide (1904) → transistor (années 20-30) semi-conducteur/65nm → nanotechnologies (1 à 0.1nm)

- Au niveau de la mémoire:
  1 Ko (en 1970) → quelques GO (2^{30})
  Une faune variée: mémoire externe, mémoire centrale, mémoire cache (de plusieurs niveaux différents)
- Au niveau de l'architecture:
  Du modèle Von Neumann → architectures plus parallèles (pipeline?, super scalaire?, multi cœur?)

Performances

- C'est toujours la course aux performances (MIPS, MOPS, FLOPS): plus les machines sont performantes et plus les applications sont gourmandes.
- Parmi ces applications:
  - Traitement d'images, visioconférence, jeux 3D, multimédia;
  - Bioinformatique: décodage du génome humain
  - Modélisation et simulation en climatologie, conception de circuits intégrés, etc.
- Pour accroître les performances, on compte sur l'évolution des ***technologies*** mais aussi les ***architectures*** : parallélisme interne (au sein d'un processeur) ou/et externe (multi processeur)

Loi de Moore



Source wikipedia

Moore (cofondateur d'INTEL) à prédit en 1965 que le nombre de transistor sur une puce doublerait tous les 18 mois et a rectifié sa loi (empirique) vers 1975 à 24 mois.

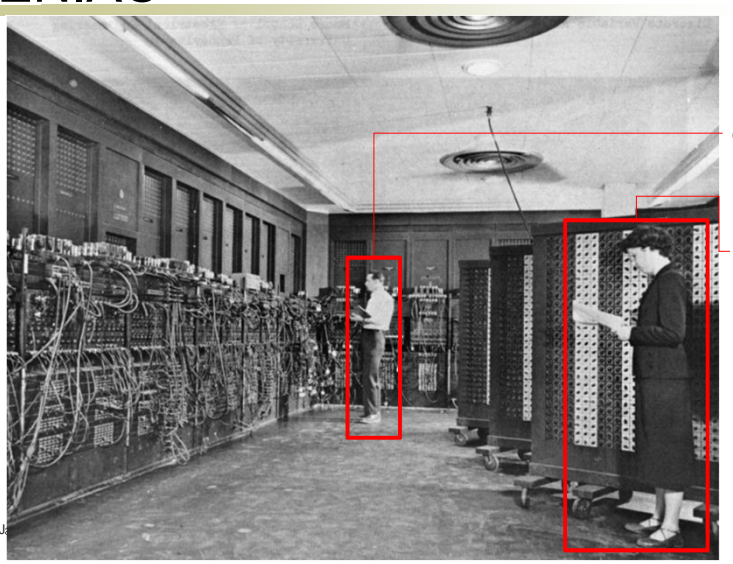
Cette loi est étonnement juste !!!

L'histoire commence en 1943 avec l'ENIAC …

- ***Electronic Numerical Integrator And Computer***: reprogrammable, décimal et utilise des tubes à vide. Conçus à des buts militaires.
- 30 tonnes (500m^2 au sol)
- 18000 tubes à vide et 70000 résistances
- 140 Kwatts/h *(un PC consomme 100 à 200 Watts/h)*
- 5000 additions/seconde/calculateur, il y en a 20 *(un Pentium 4 3.2 GHz effectue 2772~6204 MFLOPS)*.
- À fonctionné jusqu'en 1955 (une dizaine d'année).

Tache principale: réaliser une série de calculs complexes pour tester la faisabilité de la bombe hydrogène.

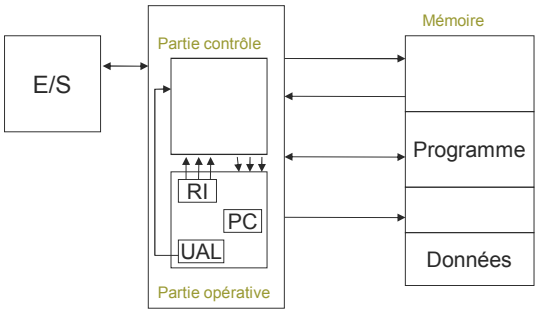
ENIAC



IAS et l'architecture de Von Neumann (1903-1957)

- ***Institute of Advanced Studies*** (Princeton) architecture conçue entre 1946 et 1952.
- **Idée**: représentation du programme sous forme numérique (binaire) **ET** le ranger en mémoire au même titre que les données.
- Caractéristiques:
  - Mémoire principale: **données + instructions**
  - Une unité arithmétique et logique sur **données binaires**
  - Une unité de contrôle (interprétation d'instruction)
  - Dispositif d'E/S
- L'IAS est un modèle d'architecture encore aujourd'hui utilisé.

IAS



… depuis, d'autres modèles ont été définis comme **l'architecture Harvard** (deux mémoire: données et instructions) ainsi que d'autres architectures parallèles

CISC & RISC

- **Constat**: grande majorité des instructions disponibles sont très peu utilisées
- Deux types d'architectures différentes:
  - **CISC** (*Complex instruction Set Computer*): modes d'adressage complexe + plusieurs instructions possibles + nombre de cycles par instruction différent → **Pentium**
  - **RISC** (*Reduced Instruction Set Computer*): peu de modes d'adressage + peu d'instructions possibles + nombre de cycles par instruction constant (ou presque) → **PowerPC**.
  - Actuellement peu de frontière entre les deux …?

Évolution dans la gamme Pentium

- Du 8080 au 80486: microprocesseur 8 bits au 32 bits multitâches (exécution de plusieurs programmes simultanément) avec coprocesseur mathématique.
- Pentium et Pentium Pro: utilisation d'architectures super scalaires (plusieurs instructions en parallèle) et techniques logicielles permettant d'accélérer le traitement:
  - Prédiction de branchement
  - Analyse de flot (enlever le code mort)
  - Exécution spéculative (dans le désordre)
- Pentium II, III et IV: traitement spécifique pour la vidéo, 3D et multimédia.
- Itanium: architecture 64 bits.
- Architectures multi cœurs

Évolution dans la gamme PowerPC

- 801: processeur RISC de Berkeley lancé par IBM (1975).
- System/6000 RISC: super scalaire RISC
- PowerPC: accord entre IBM, Motorolla et Apple
  - 601, 603, 604: processeur 32 bits avec conceptions super scalaires plus avancée.
  - 620 et G3: architecture 64 bits intégrant pour la dernière deux niveaux de cache sur la puce.
  - G4: plus de parallélisme et vitesse interne plus importante.
  - G5 : processeur 64 bits
  - Bientôt Power PC multi cœur
- Cell: play Station 3

Quelques chiffres

	Pentium	Pentium III	Itanium
Date de lancement	1993	1999	2001
Vitesse d'horloge	166 MHz	660 MHz	800 MHz
Nombre de transistors	3,1 Millions	9,5~28 Millions	300 Millions

	601	G4	G5
Date de lancement	1993	1999	2003
Vitesse d'horloge	120 MHz	500 MHz	2 GHz
Nombre de transistors	2,8 Millions	10,5 ~ 33 Millions	58 Millions

Les performances

- La performance d'un microprocesseur pour un utilisateur individuel correspond à la durée s'écoulant entre le début et la terminaison d'une tâche: *temps d'exécution*

$$\text{tempsExec} = \text{nbInstructions} * \left(\frac{\text{nbCycles}}{\text{Instruction}}\right) * \left(\frac{\text{nbSecondes}}{\text{cyclesHorloge}}\right)$$

→ la **fréquence** de fonctionnement **ne suffit pas** pour comparer les performances

→ le nombre de cycles par instruction dépend de la complexité moyenne du jeu d'instructions

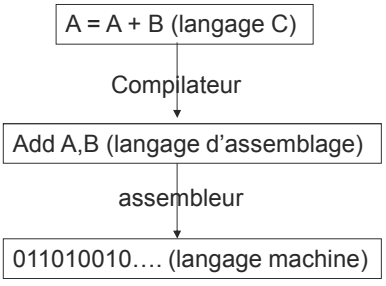
Exemple

- Pour un même jeu d'instructions et deux mises en œuvre différentes avec une machine A et une machine B. Pour un même programme, la A a un cycle d'horloge de 10ns et un CPI (nombre moyen de Cycles Par Instruction) de 2, tandis que la machine B a un cycle d'horloge de 15ns et un CPI de 1,2.

  Quelle est la machine la plus rapide (nombre d'instructions par seconde)?

De l'importance des compilateurs et du logiciel

La programmation de la machine se fait généralement en utilisant un langage de haut niveau (C, Pascal, Fortran,…)



Jeu d'instruction d'une machine

Le jeu d'instruction est dépendant des choix architecturaux de la machine.

une instruction = code instruction + opérande(s)

- le code de l'instruction lu en un ou plusieurs accès mémoire
- le (ou les) opérande(s) peu(ven)t être une donnée ou une adresse

On distingue **quatre catégories** d'instructions:

1. **processeur-mémoire**: transfert mémoire processeur
2. **processeur-E/S**: transfert données vers, ou à partir de périphériques
3. **traitement des données**: opération arithmétique ou logique
4. **contrôle**: branchement

Jeu d'instructions (2)

Le nombre d'opérandes peut varier selon le jeu d'instructions, on parle de jeu d'instructions à 0, 1, 2 voire jusqu'à 4 adresses. Cependant, comme les mots représentant les données (ou adresses) sont de plus en plus longs (32, 64 bits), sont « préférés » les jeux d'instructions avec peu d'adresses.

- Instruction d'addition avec **2 adresses**

  add source destination

  source + destination -> destination

- Dans le cas d'une instruction à **une adresse**, on utilise un registre interne (accumulateur ou acc) pour stocker le premier opérande.

  add source

  acc + source -> acc

Jeu d'instructions (3)

Par exemple, on peut exécuter A = B * (C − D*E) avec

```
LOAD E  (charge dans Acc)
MPY D   (multiplie le contenu de Acc)
STA T1  (range le contenu de Acc)
LOAD C
SUB T1  (soustrait le contenu de Acc)
MPY B
STA A
```


Architectures et instructions

plusieurs stratégies pour l'accès aux données:

- Type **registre-registre** (tendance actuelle):

  ADD R1, R2, R3

  R3 ← R2 + R1
- Type **registre-mémoire**:

  ADD R, @X

  R ← R + @X
- Type **mémoire-mémoire**:

  ADD @X, @Y, @Z

  @Z ← @X + @Y
- Plusieurs types d'adressage … on les verra ultérieurement

Tâches du processeur (modèle 5 phases)

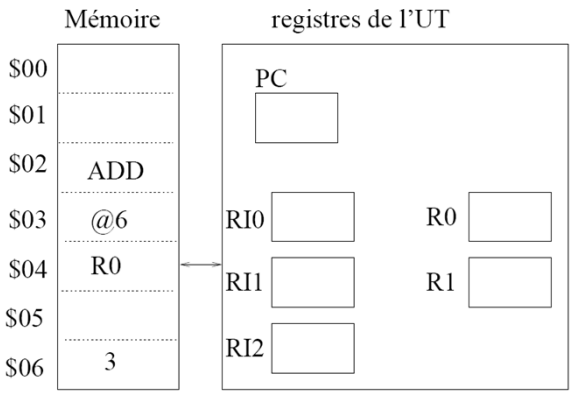
IMPORTANT

Cycle de lecture et d'exécution des instructions:

1. **Lecture (fetch)** de l'instruction à exécuter (charger le registre d'instructions RI avec l'instruction dont l'adresse est dans le PC)
2. **Décodage** de l'instruction (chargement des opérandes des registres si besoin)
3. **Lecture/écriture mémoire**
4. **Exécution**
5. **Écriture** des résultats dans les **registres**

En plus de la sélection de l'instruction suivante (préparer PC pour l'instruction suivante) qui se fait lors de l'opération 3, 4 ou 5.

Illustration



Partie opérative

Cette partie contient :

- Le registre d'instruction **RI**
- Le registre pour le compteur programme (**PC**)
- Les registres internes pour la mémorisation de données ou d'adresses
- **L'UAL** (unité arithmétique et logique) avec deux sorties :
  - résultat du calcul
  - code de conditions arithmétiques (Z= résultat nul, N= résultat négatif,C= retenue, V= dépassement de capacité)

La description de la partie opérative est l'objet du **cours 2**.

Partie contrôle

- La partie contrôle permet l'exécution des instructions. C'est une **machine séquentielle** (automate de contrôle ou séquenceur) permettant d'envoyer des ordres à la partie opérative.
- Les entrées de la partie contrôle sont des informations émanant de la partie opérative (valeur d'un code opération, valeur d'un code condition ...)

Décodage des instructions

Le décodage sert à différencier les types d'instructions
(instruction arithmétique, logique, de transfert, de branchement .)

Décoder = générer d'après le code de l'instruction, les signaux électriques permettant l'exécution de cette instruction

Type de décodage :

- **micro programmé** (utilisation d'une mémoire de microprogramme)
- **câblé** (non reprogrammable)

L'unité de contrôle est l'objet du cours 3&4.

Exemple de réalisation de processeur

Extrait « Architectures logicielles et matérielles »
P.Ambard et al., Dunod 2000

On définit :

- un jeu d'instructions ***réduit***,
- une partie opérative avec un registre de donnée,
- une unité de contrôle permettant l'exécution des différentes instructions sur l'architecture de la partie opérative

Jeu d'instructions

clr	mise à zéro du registre *ACC*
ld #vi	chargement de la valeur immédiate **vi** dans *ACC*
st [ad]	chargement du mot mémoire d'adresse *ad* avec le contenu de *ACC*
jmp ad	saut à l'adresse *ad*
add [ad]	chargement de *ACC* avec la somme du contenu de *ACC* et du mot mémoire d'adresse *ad*

Taille et contenu de la mémoire

Soit le petit programme suivant que l'on désire exécuter:

```
      ld #3
      st [8]
etiq: add [8]
      jmp etiq
```

La taille d'une instruction étant de un ou deux mots, si le programme commence à l'adresse 0 alors le symbole *etiq* est associé à la valeur 4.

Si l'on fixe la taille d'un mot à 4 bits, on fixe la taille de la mémoire à 16 mots sachant que la taille d'une adresse est aussi de 4 bits.

Interprétation des instructions

```
PC := 0
tant que vrai
    selon Mem[PC]
        clr:   Acc :=0; PC := PC + 1
        ld:    Acc := Mem[PC + 1];  PC := PC + 2
        st:    Mem[Mem[PC + 1]] := Acc;  PC := PC + 2
        jmp:   PC := Mem[PC + 1]
        add:   Acc := Acc + Mem[Mem[PC + 1]]; PC := PC + 2
```

Mem[PC] correspond au code de l'instruction tandis que Mem[PC + 1] correspond à l'opérande (donnée ou adresse).

Représentation et précision finie

- Représentation en précision finie (quantité limitée de nombres représentables)
  - Problème de représentation: ex 1/3 n'est pas représentable
  - Les propriétés sur les nombres sont différentes en précision finie

    $(A+B) - C = A + (B-C)$??

    Exemple:
    - ***Loi associative***: a+(b-c) ⇔? (a+b)-c

      Entiers sur 3 chiffres: a=700, b=400, c=300
    - ***Loi distributive***: a*(b-c) ⇔? (a*b)-(a*c)

      Entiers sur 3 chiffres: a=5, b=210, c=100

Les bases utilisées en informatique

- La base 10 bien sûr ! Sauf exception, les programmeurs comptent en base 10 comme le quidam ordinaire (on a bien 10 doigts !)
- La base 2 : les opérateurs de calcul sont souvent prévus pour calculer dans cette base, car leur réalisation en est simplifiée.
- La base 16 ou hexadécimal : elle est utilisée lorsque le programmeur doit résonner en base 2. Les conversions de la base hexadécimal vers la base 2 se calculent très facilement de tête.

Représentation binaire des nombres

- nombre **positif** : (conversion en base 2)

$$A = \sum_{i=0}^{n-1} a_i.2^i$$

- nombre **négatif** :
  - représentation signe-valeur absolue:

$$A = \sum_{i=0}^{n-2} a_i.2^i \; si \quad a_{n-1} = 0$$

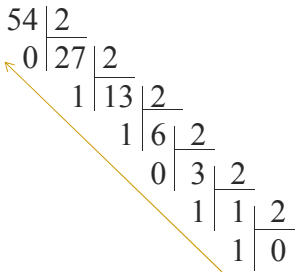
$$A = -\sum_{i=0}^{n-2} a_i.2^i \; si \quad a_{n-1} = 1$$

  - représentation complément à 2 :

$$A = -2^{n-1}.a_{n-1} + \sum_{i=0}^{n-2} a_i.2^i$$

Conversion de la base 10 à une base B

- On divise successivement le nombre à convertir par la base B, tant que le résultat n'est pas nul.
- Les **restes** de chacune des divisions forment les chiffres du nombre converti dans la base d'arrivée, du poids faible au poids fort.



$(54)_{10}=(110110)_2$

Représentation binaire (2)

Représentation décimale	Représentation signe et valeur absolue	Représentation complément à 1	Représentation complément à 2
+4	-	-	-
+3	011	011	011
+2	010	010	010
+1	001	001	001
+0	000	000	000
-0	100	111	000
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4	-	-	100

36

Caractéristiques principales du complément à 2

- Intervalle de représentation: $-2^{n-1} \ldots 2^{n-1}-1$
- Nombre de représentation de zéro: **une seule**
- Négation: $-A = \overline{A} + 1$
- Augmentation du nombre de bits: extension par le bit du signe
- Dépassement de capacité: résultat prend le signe opposé de ce qu'il devrait prendre
- Soustraction: A-B = A + (-B)