AES. Processeur

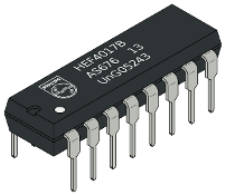
I/ Historique du microprocesseur

Le **processeur** est un composant électronique constitué d'une combinaison de **circuits intégrés**, eux-mêmes composés de **transistors**.

Il est donc naturel que son histoire soit liée à ces avancées technologiques, on peut ainsi établir la frise chronologique suivante :

1906 : triode

1947 : transistor



1958 : circuit intégré

1971 : microprocesseur

L'apparition de la **diode** en 1904 puis de la **triode** (permet notamment l'amplification de courant à HF) est celle du début de l'électronique et marque le développement rapide de la TSF (Télécommunication Sans Fil). A noter que cette technologie participera à signer la première défaite de l'Homme blanc dans une guerre, à savoir la guerre sino-russe de 1905.
Les triodes évolueront en **lampes électroniques** (à cinq « pattes » notamment) et équiperont les premiers ordinateurs entièrement électroniques (*Colossus* britannique, *Eniac* américain).

Les **lampes électroniques**, remplacés majoritairement par les **transistors**, conservent leur place dans certains appareils à grande puissance ou à hautes fréquences (HF) (four à micro-ondes, amplificateurs de sons etc.).

Après la seconde guerre mondiale, des recherches de grande ampleur menées s'ouvrent dans un nombre croissant de laboratoires comme dans celui d'une jeune entreprise : Texas Instrument. Elles permettront la mise au point de modèles de transistors de plus en plus diversifiés, utilisant non plus du germanium mais du silicium. La «transistorisation» des équipements électroniques s'amorce pour les matériels de pointe dès les années 1950. Elle touche le grand public avec les premiers postes de radio à transistor puis de téléviseurs et s'accélérera avec les efforts liés à la Guerre Froide. Elle révèle que la miniaturisation et la capacité de produire ces composants en très grande quantité à des prix de plus en plus faibles est un enjeu industriel majeur.

Un certain Jack Kilby a eu l'idée de relier manuellement plusieurs transistors entre eux : il ne faudra que quelques mois pour miniaturiser et industrialiser le processus et ainsi naîtra le **circuit intégré** fin 1958.
Le **circuit intégré**, aussi appelé **puce électronique**, est un composant électronique, basé sur un semi-conducteur (germanium puis silicium) , reproduisant une ou plusieurs, fonction(s) électronique(s) plus ou moins complexe(s), intégrant souvent plusieurs types de composants électroniques de base dans un volume réduit (sur une petite plaque), rendant le circuit facile à mettre en œuvre. Le circuit intégré a été l'une des découvertes essentielles qui ont permis de rendre possible la **mission Apollo en 1969**.

Jusqu'au début des années 1970, les différents composants électroniques, nécessaires au fonctionnement d'un processeur ne pouvaient pas tenir sur un seul circuit intégré, ce qui nécessitait d'interconnecter de nombreux composants dont plusieurs circuits intégrés. En 1971, la société américaine Intel réussit, grâce à Martial Hoff, pour la première fois, à placer tous les composants qui constituent un processeur sur un seul circuit intégré donnant ainsi naissance au **microprocesseur**.

En savoir plus sur le tube électronique (ou à vide) et ses évolutions ici : https://f5zv.pagesperso-orange.fr/RADIO/RM/RM23/RM23G/RM23G01.HTM

En savoir plus sur Jack Kilby : https://www.webtimemedias.com/article/le-deces-de-jack-kilby-inventeur-du-circuit-integre

En savoir plus : sur la victoire japonaise sur les Russes en 1905 : https://gallica.bnf.fr/html/und/asie/face-lexpansionnisme-japonais?mode=desktop

En savoir plus : vidéo sur l'histoire du Japon de 1860 à nos jours ici : https://www.youtube.com/watch?v=DIOQ0KCeFgg

En savoir plus : une certaine fascination des Français pour la néo-culture japonaise.
Les films d'animation, excellence française (Pixar notamment !) : https://etudiant.lefigaro.fr/les-news/actu/detail/article/quatre-des-dix-meilleurs-ecoles-d-animation-au-monde-sont-francaises-18805/
Les mangas, une passion française : https://www.leparisien.fr/culture-loisirs/livres/au-japon-dans-les-rouages-de-l-industrie-du-manga-08-09-2019-8147736.php#:~:text=Mais%20pour%20l'instant%2C%20rien,du%20march%C3%A9%20de%20la%20BD

2/ Composition et fonctionnement d'un microprocesseur

a) Jeu d'instructions

Rappel : Le microprocesseur est le "cœur" d'un ordinateur : les instructions sont exécutées au niveau du CPU (en anglais *Central Processing Unit*) . Il est schématiquement constitué de 3 parties : les **registres**, **l'unité arithmétique et logique** (UAL ou ALU en anglais), **l'unité de commande** permet d'exécuter les instructions (les programmes).
A noter : reprendre le chapitre précédent pour les détails (ainsi que sur les **bus**).

Un programme écrit dans un langage de haut niveau (plus proche du « langage humain -du moins anglophone 😊 -), Python par exemple, mais éloigné du langage machine -de bas niveau, dépend le moins possible du système d'exploitation et du processeur.

Chaque type de processeur a son langage (appelé **jeu d'instructions**) mais -heureusement- ont beaucoup de structures communes.
Pour être compris, un langage doit être soit « traduit à la volée » c'est-à-dire **interprété** en temps réel (comme Python) ou traduit à l'avance en langage machine à l'aide d'un **compilateur** comme ce (magnifique 😊) langage C.

Exemple : différence des tailles de nombres entiers en fonction de la taille des registres d'un processeur (64 bits pour les récents, 32bits -voire 16 bits- pour les plus anciens) en langage C :

Type	Taille	Borne inférieure	Borne inférieure (formule)	Borne supérieure	Borne supérieure (formule)
`signed char`	≥ 8 bits	-128[2][3]	$-(2^7-1)$	+127	2^7-1
`unsigned char`	≥ 8 bits	0	0	+255	2^8-1
`short`	≥ 16 bits	-32 768	$-(2^{15}-1)$	32 767	$2^{15}-1$
`unsigned short`	≥ 16 bits	0	0	65 535	$2^{16}-1$
`int`	≥ 16 bits (processeur 16 bits) ≥ 32 bits (processeur 32 bits)[4]	-32 768 -2 147 483 648	$-(2^{15})$ $-(2^{31})$	+32 767 +2 147 483 647	$2^{15}-1$ $2^{31}-1$
`unsigned int`	≥ 16 bits (processeur 16 bits) ≥ 32 bits (processeur 32 bits)[4]	0 0	0 0	+65 535 +4 294 967 295	$2^{16}-1$ $2^{32}-1$
`long`	≥ 32 bits	−2 147 483 647	$-(2^{31}-1)$	2 147 483 647	$2^{31}-1$
`unsigned long`	≥ 32 bits	0	0	4 294 967 295	$2^{32}-1$
`long long` *(C99)*	≥ 64 bits	−9 223 372 036 854 775 807	$-(2^{63}-1)$	9 223 372 036 854 775 807	$2^{63}-1$
`unsigned long long` *(C99)*	≥ 64 bits	0	0	18 446 744 073 709 552 000	$2^{64}-1$

Source : Wikipédia

b) Horloge d'un processeur

Le CPU dispose d'une horloge qui cadence l'accomplissement des instructions, l'unité est appelée **cycle**, ainsi, un processeur cadencé à 3 GHz signifie qu'il y a trois milliards de **cycles** par seconde.

Dans une **machine de Turing**, un cycle est composé de quatre actions : lire, écrire, changer d'état et se déplacer. Dans un processeur, un cycle se compose de cinq actions qui sont exécutées :

- Lire l'instruction.
- Décoder l'instruction.
- Exécuter l'instruction.
- Accéder à la mémoire en lecture/écriture.
- Ecrire le résultat dans les registres.

En savoir plus : vidéo sur la fabrication des microprocesseurs : https://www.youtube.com/watch?v=NFr-WyytNfo

En savoir plus : résumé rapide sur l'histoire des microprocesseurs : https://www.gralon.net/articles/materiel-et-consommables/materiel-informatique-et-consommable-informatique/article-le-microprocesseur---une-invention-revolutionnaire-1042.htm

II/ Les langages machine et assembleur

1/ Fonctionnement d'un CPU

Un ordinateur exécute des programmes qui sont des suites d'instructions. Le CPU est incapable d'exécuter directement des programmes écrits, par exemple, en Python. En effet, comme tous les autres constituants d'un ordinateur, le **CPU gère uniquement deux états** (toujours symbolisés par un **"1"** et un **"0"**), les instructions exécutées au niveau du CPU sont donc codées en binaire.

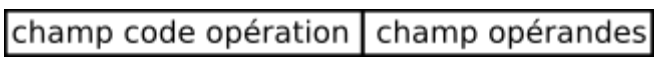
A savoir : l'ensemble des instructions exécutables directement par le microprocesseur constitue ce que l'on appelle le "langage machine".

Dans les débuts de l'informatique, les programmes étaient écrits à l'aide de cartes perforées en langage machine. Le code « 1 » / « 0 » était traduit par le code « trou non perforé » / « trou perforé ». Par ce système, le Colossus britannique parvenait tout de même à effectuer plusieurs milliers d'opérations par seconde.

En savoir plus ici sur les cartes perforées : https://www.cigref.fr/archives/histoire-cigref/blog/a-lorigine-de-linformatique-une-carte-perforee/

Une instruction machine est une chaîne binaire composée principalement de 2 parties :

- Le champ "**code opération**" qui indique au processeur le type de traitement à réaliser. Par exemple le code "00100110" donne l'ordre au CPU d'effectuer une multiplication.
- Le champ "**opérandes**" indique la nature des données sur lesquelles l'opération désignée par le "code opération" doit être effectuée.



Instruction machine

Les **instructions machines** sont basiques, en voici quelques exemples :

- Les **instructions arithmétiques et logiques** comme les additions, soustractions, comparaisons, et/ou etc. . On peut avoir comme instruction « additionne la valeur 125 avec celle contenue dans le registre 0 et range cela dans le registre 1 ».
- Les **instructions de transfert de données** comme : « va chercher l'instruction contenue à l'adresse mémoire XXX et mets la dans le registre 2 ». Il ne faut pas oublier que le processeur travaille avec les données contenues dans ses registres.
- Les **instructions de saut (ou branchement)** qui correspondent aux instructions conditionnelles et boucles. On peut avoir comme instruction « si la valeur contenue dans le registre 0 est inférieure à la valeur 125 alors va chercher l'instruction dans l'adresse mémoire 255 et exécute-la sinon va chercher l'instruction dans l'adresse mémoire 1300 et exécute-la ».
  A noter : Il s'agit en fait du futur fameux « GOTO » présent en langage BASIC.

A savoir : **Il faut également indiquer au processeur la nature des données (opérandes).**
Il y a trois grands types d'opérandes :

- **Adressage immédiat** : une **valeur constante**, les opérations sont effectuées directement.
- **Adressage direct** : une **valeur** stockée dans un **registre**. L'opérande indique dans quel registre elle se trouve (R0, R1 etc.).
- **Adressage indirect** : une **valeur** stockée dans la **mémoire vive**. Son adresse est indiquée dans l'opérande, il faut d'abord la transférer dans un registre pour pouvoir l'utiliser.

A savoir : **parmi les registres d'un CPU, il en existe particulier, appelé pointeur d'instruction** *(instruction pointer* en anglais) **ou compteur ordinal** (*program counter* en anglais) **qui a pour rôle de pointer vers la prochaine instruction à exécuter**

Les exemples précédents sont écrits ici en langage formel mais sont bien une succession de « 1 » et de « 0 » en réalité, y compris pour les instructions.

Exemple : La chaîne « 11100010100000100001000001111101 » signifie « ajoute la constante 125 avec la valeur stockée dans R0 et met le tout dans R1 » dans un système 8 bits (donc bien loin des 64 bits actuels) !
- 11100010 est l'addition.
- 10000010 signifie « mets le résultat de l'addition dans R1 ».
- 00010000 signifie « va chercher la valeur stockée dans R0 ».
- 11111101 est le codage de 125 en binaire.

En savoir plus sur le langage machine ici (de niveau L1/L2 mais plutôt clair) : https://lipn.univ-paris13.fr/~poinsot/save/L2%20Archi/Cours/Cours%203%20-%20Print.pdf

Chaque famille de CPU dispose ainsi d'un **jeu d'instructions**, il est en fonction de sa version (8 bits, 16 bits, 32 bits, 64 bits), c'est pour cela qu'avant d'installer un logiciel, on demande souvent la version du CPU (32 bits ou 64 bits actuellement) : par exemple, un logiciel prévu pour une version 64 bits ne fonctionnera pas sur une version 32 bits.

Ce jeu d'instructions dépend aussi du constructeur, ce qui a d'ailleurs alimenté la lutte entre Intel et AMD depuis plusieurs décennies.
Heureusement que beaucoup d'instructions sont communes, sinon la bonne exécution d'un logiciel dépendrait du processeur !

En savoir plus : AMD vs Intel en 2019, ici https://www.celside-magazine.com/fr-fr/processeurs-amd-intel-computex-2019/

2/ Le langage assembleur

Il est parfois utile de programmer au plus bas de la machine pour augmenter les performances, cela est toutefois moins vrai de nos jours du fait de la puissance des machines actuelles. On utilise un langage dit « **d'assemblage** » plus lisible pour l'être humain que le langage machine.

Par souci de simplification, on utilisera le système décimal pour désigner les nombres et les adresses mémoires. Les registres seront nommés R0, R1 etc.

On décrira par la suite quelques instructions écrites en **langage assembleur** adapté au site de Peter Higginson http://www.peterhigginson.co.uk/AQA/ que l'on utilisera.

a) Transfert de données

LDR *reg*, *adrRAM*
Effet : transfère la valeur contenue à l'adresse RAM dans le registre.
Rappel : le CPU ne peut utiliser directement que des valeurs contenues dans un de ses registres.

STR *reg*, *adrRAM*
Effet : transfère la valeur contenue dans le registre à l'adresse RAM.

MOV *reg1*, *reg2*
Effet : transfère la valeur contenue dans le registre 2 dans le registre 1

MOV *reg1*, *#nbr*
Effet : crée le nombre ayant pour valeur nbr et le place dans le registre 1
Rappel : #nbr représente un nombre constant, c'est un adressage immédiat.

b) Opérations arithmétiques

ADD *reg1*, *reg2*, *reg3*
Effet : additionne les valeurs contenues dans les registres 2 et 3 et place le résultat dans le registre 1.

ADD *reg1*, *reg2*, *#nbr* (*nbr* représente un nombre entier)
Effet : additionne la valeur contenue dans le registre 2 avec *nbr* et place le résultat dans le registre 1.

SUB *reg1*, *reg2*, *reg3*
Effet : comme ADD mais pour le calcul d'une différence.

CMP *reg1*, *reg2*
Effet : compare la valeur dans le registre 1 et celle dans le registre 2

CMP *reg1*, *#nbr*
Effet : compare la valeur dans le registre avec *nbr*.

Remarque : Cette instruction CMP doit précéder une instruction de branchement conditionnel BEQ, BNE, BGT, BLT (voir ci-dessous)

c) Instructions conditionnelles de saut

Un **label** (ou étiquette) est une adresse RAM dans laquelle y est stockée la prochaine instruction à exécuter. Cela évite à utiliser directement l'adresse en question.

L'instruction est celle-ci :

B *myLabel*
Effet : va à l'adresse *myLabel*, y lire et exécuter l'instruction.

On compare dans un premier temps deux valeurs avec l'instruction **CMP** *reg1*,*reg2*. Les deux valeurs peuvent être égales, différentes, la première peut être supérieure (ou inférieure) à la seconde.

BEQ *myLabel*
Effet : comme l'instruction **B** mais seulement si les deux valeurs comparées sont égales.

BNE *myLabel*
Effet : comme l'instruction **B** mais seulement si les deux valeurs comparées sont différentes.

BGT *myLabel*
Effet : comme l'instruction **B** mais seulement si la valeur dans *reg1* est supérieure à celle dans *reg2*.

BLT *myLabel*
Effet : comme l'instruction **B** mais seulement si la valeur dans *reg1* est inférieure à celle dans *reg2*.

Et enfin, une instruction indiquant que le **programme** (ou une **série d'instructions**) **s'arrête** : **HALT**. Il ne faut pas l'oublier sinon il y aura une erreur ou pire, certaines parties du programme non souhaitées vont s'exécuter.

Exemple :

Programme	*Explications*
MOV R0,#20	Place la valeur « 20 » dans le registre 0 (noté R0).
CMP R0,#18	Compare la valeur dans R0 avec la valeur « 18 ».
BGT etiq	Si la valeur dans R0 est supérieure à 18, aller au label « etiq » (C'est bien le cas ici).
MOV R0,#14	Non exécuté.
HALT	Non exécuté.
etiq:	
ADD R0,R0,#1	Ajoute 1 à la valeur dans R0 et place le résultat dans R0.
STR R0,75	Transfère la valeur dans R0 à l'adresse RAM « 75 ».
HALT	Stoppe le programme.

En savoir plus : John Carmack, l'inventeur des FPS et du multijoueur ici, le grand « gourou » du langage C et de l'assembleur : https://blog.materiel.net/id-software-avec-du-doom-quake-et-rage-dedans/