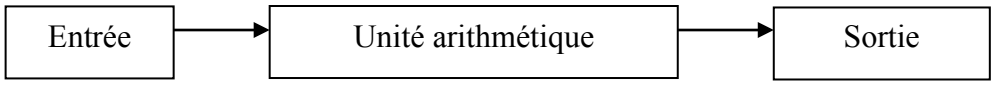
# Architecture.

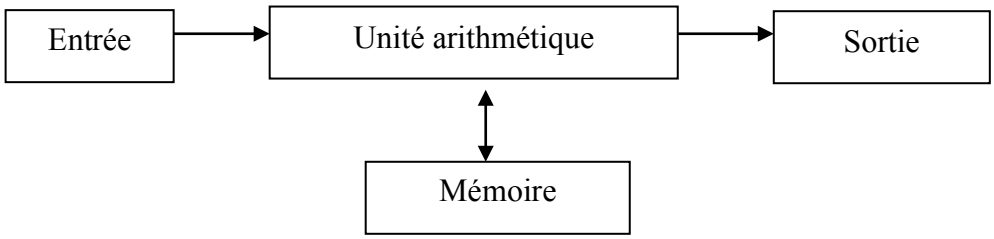
1. Genèse et histoire.

Entre le XIe et le XVIIe siècle, l'ordinateur désigne celui qui est chargé de « régler les affaires publiques ».
Le sens actuel a été proposé en 1955 en réponse à une demande du service publicitaire d'IBM France.
La définition a beaucoup évolué en 60 ans. Celle de Wikipedia est aboutie : « machine électronique qui fonctionne par la lecture séquentielle d'un ensemble d'instructions qui lui font exécuter des opérations logiques et arithmétiques sur des chiffres binaires ». Nous avons déjà vu l'aspect « lecture séquentielle d'instructions », en écrivant des programmes, et l'aspect binaire dans le cours sur le codage. Nous allons dans ce chapitre voir comment l'architecture permet d'effectuer les « opérations arithmétiques et logiques ».

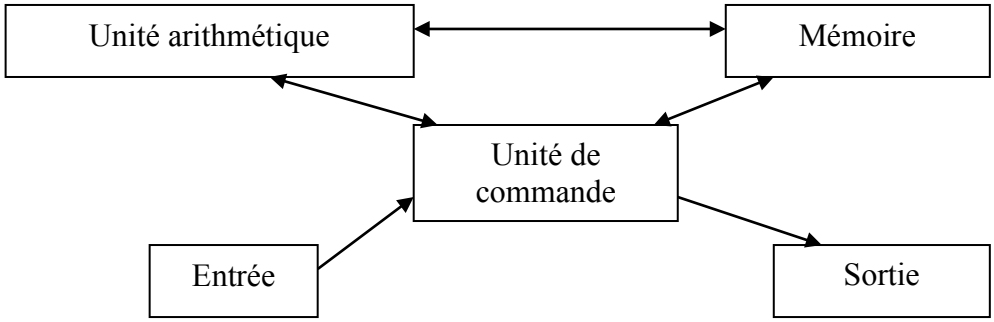
Un peu d'histoire.
Les premières machines à calculer datent du XVIIe siècle. La pascaline est la plus connue, c'est en fait la seule dont on soit sûr de l'existence puisqu'il en existe toujours une dizaine. Elle ne pouvait faire que des additions, mais Blaise Pascal avait trouvé la méthode du complément à 10 (sur le principe du complément à 2 vu dans le dernier chapitre) pour faire des soustractions.
On a le schéma suivant :




Leibniz, en 1673, rajoute la mémorisation des résultats intermédiaires : sa machine pouvait faire ainsi des multiplications et des divisions.
D'où le schéma :

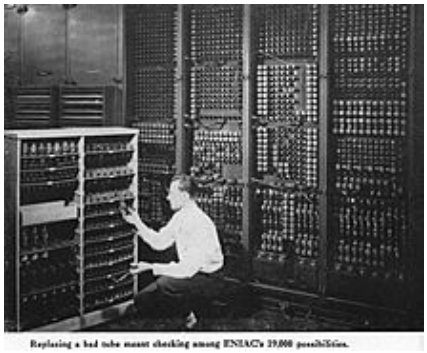



Leibniz évoque de plus la représentation binaire grâce… à la Chine. En effet, il connaît le « Yi-King », traité de divination, dont les tirages divinatoires se font en fait en binaire. Il communique sur le binaire à l'Académie des Sciences en 1703.
150 ans plus tard, en 1847, Georges Boole fonde l'algèbre de Boole, que l'on verra au prochain chapitre. Cette algèbre est basée sur les valeurs de vérité Vrai ou Faux, et permet notamment de fabriquer des circuits électroniques compacts.
À peine plus tôt, en 1834, Charles Babbage à l'idée d'incorporer dans la machine à calculer des cartes perforées, pour donner la suite d'instructions à exécuter. Il ne put jamais réaliser sa machine la plus performante, faute de fonds suffisants. Ada Lovelace, qui travailla avec Babbage, écrivit le premier véritable programme informatique de l'histoire. Son nom a été donné à un langage de programmation, et son portrait figure sur les hologrammes d'authentification des produits Microsoft.
Depuis cette époque, le schéma qui va servir de référence est celui-ci :



Le premier calculateur utilisant l'électricité et le binaire fut le Z1, conçu en 1936 par l'ingénieur allemand Konrad Zuse, qui crée aussi le premier langage de haut niveau. Ce langage fut oublié, faute de machine capable de le supporter, jusqu'en 1972. Il n'a été implémenté qu'en 2000, à titre historique. Quant à l'ordinateur, dont la troisième version, le Z3, fut détruite par les alliés à la fin de la guerre, il semblerait qu'il était bien plus avancé que ses concurrents directs, dont l'ENIAC (cf. ci-dessous).

C'est Claude Shannon, spécialiste de la théorie de l'information, qui expose en 1938 comment utiliser l'algèbre de Boole pour construire des machines à calculer basées sur des commutateurs et des relais.


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Le premier ordinateur « moderne », l'ENIAC, est mis en service en 1946. Einstein et Gödel (vous connaissez le premier, le deuxième est un génie de la logique) estimaient que cette coûteuse réalisation n'apporterait aucune contribution à la science… Cet ordinateur de première génération (1946-1956) fonctionnait avec des lampes à vide (toujours utilisées dans les amplificateurs audio de grande qualité), de durée de vie très limitée. La durée moyenne entre deux pannes était de quelques heures. Une fausse origine du terme « bug » est l'une des causes de pannes : un insecte (bug) qui, se posant sur un tube, brûle et cause la rupture du tube. En fait ce terme est bien plus ancien, il désigne depuis le XIXe siècle les disfonctionnements dans des éléments mécaniques.

En 1947 apparaît le transistor, petit, fiable, et peu coûteux. Il remplace les lampes à vide dans les ordinateurs de deuxième génération (1956-1963).

Enfin les circuits intégrés formés de quelques dizaines à plusieurs centaines de millions de transistors voient le jour en 1958. Ils sont au cœur des ordinateurs de troisième génération (1963-1971).

Nous sommes dans la quatrième génération d'ordinateurs, celle qui utilise des microprocesseurs, qui ont permis la naissance des micro-ordinateurs. Selon certaines sources, le premier micro-ordinateur, serait français : le Micral, crée en 1972.

Selon la loi de Moore, la densité des composants électroniques sur une puce suit une croissance exponentielle (suite géométrique), en doublant tous les deux ans. Le premier microprocesseur, un Intel était une unité de calcul sur 4 bits, tournant à 108 kHz et intégrant 2300 transistors. Un Intel actuel (un des i7) comporte 3,2 milliards de transistors, tourne à 3,7 Ghz, et calcule sur 64 bits. La loi de Moore s'est vérifiée jusqu'en 2010, mais la miniaturisation devient telle que des limites physiques infranchissables se dressent.

Les recherches se portent actuellement sur le long terme, avec des ordinateurs quantiques, où un bit peut superposer en même temps les valeurs 0 et 1…

2. L'architecture de la Machine de Von Neumann

L'architecture générale des ordinateurs a été fournie par John Von Neumann en 1945. Cette architecture est identique du nano-processeur qui équipe la machine à laver jusqu'au superordinateur.

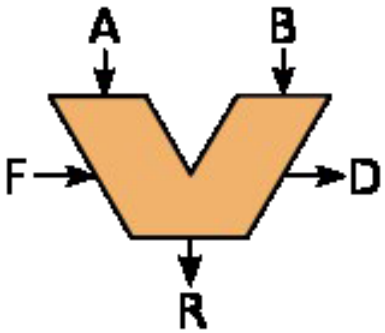
On retiendra les principes suivants :

- La fonction de calcul est réalisée par *l'UAL* (*unité arithmétique et logique*) ;
- La fonction d'enregistrement est réalisée par la *mémoire* ;
- Le déroulement séquentiel est réalisé par *l'unité de commande*.
- Les *registres* stockent temporairement des données.

L'UAL et l'unité de commande sont regroupés dans le processeur, qui communique avec la mémoire par un bus.

Ces composants constituent l'unité centrale, située sur la carte mère de votre pc. L'ensemble est rythmé par une horloge, c'est la fréquence du microprocesseur. Exemple : 3,4 gHz, soit 3 milliards 400 000 000 cycles d'horloge par seconde pour un Core i7-2600 K.

a. UAL




L'unité arithmétique et logique est le cœur de l'ordinateur.
On la représente habituellement par ce schéma :

- A et B sont les opérandes (les entrées)
- R est le résultat
- F est une fonction binaire (l'opération à effectuer)
- D est un drapeau indiquant un résultat secondaire de la fonction (signe, erreur, division par zéro, « supérieur »,…)

Les opérations que peuvent réaliser une UAL de base sont :

- Les opérations arithmétiques usuelles (+, - , × , ÷ ) ;
- Les opérations logiques (ET, OU, NON) ;
- Les opérations de comparaison ( > , < , =)

Certaines UAL sont spécialisées (calcul sur les nombres en virgule flottante, cartes graphiques, cartes sons, …). Un même processeur peut comporter plusieurs UAL. Dans certaines architectures, les UAL sont spécialisées et ne fonctionnent pas simultanément, dans d'autres les UAL fonctionnent en parallèle (architectures multi-cœur). Les architectures multi-cœur permettent d'augmenter la puissance de calcul sans se heurter aux problèmes de miniaturisation (cf. limites sur la loi de Moore).
Toues les fonctions réalisées par les UAL le sont grâce à des circuits électroniques effectuant des fonctions binaires, nous en verrons quelques exemples dans le chapitre sur le calcul booléen.

b. Mémoire

Dans sa forme de base, c'est un ensemble de cellules organisées en tableau.
Chaque cellule possède :

- Une adresse, qui est l'indice de la cellule dans le tableau : X
- Un contenu, qui est la valeur de l'élément dans le tableau : M[X]

| Adresse | | Contenu (exemple) |
|---|---|---|
| Décimal | Binaire | |
| 0 | 0 | 00101100 |
| 1 | 1 | 11001011 |
| 2 | 10 | … |
| 3 | 11 | … |

Les cellules servent à la fois à stocker les données (ou variables), et le programme.
Remarque : un virus, c'est un programme qui se camoufle en données pour pouvoir s'exécuter en douce…

On distingue plusieurs types de mémoire :

- Interne vive : la RAM, les barrettes que vous pouvez rajouter
- Interne morte : la ROM, indispensable pour le démarrage de l'ordinateur
- Externe (de masse) : disques durs, cartes flash des appareils photo (qui ne sont réinscriptibles que 100 000 fois au plus, ce sont un type particulier de ROM) , clefs USB, etc…

Pour la suite du cours, on considère que la mémoire utilisée est la RAM.

c. Le langage machine.

Les langages de programmation les plus proches du codage de « bas niveau », en 0 et 1, sont appelés « Assembleurs ». Il y a presque autant d'Assembleurs que de microprocesseurs. Un programme écrit en Assembleur ne tournera que sur une seule machine, contrairement à un programme écrit dans un langage de haut niveau.
En Assembleur, dans sa forme la plus simple, une instruction est stockée dans une cellule de mémoire désignée par une adresse.
Elle est constituée de 2 parties :

- Un code opération (CO) qui indique quelle opération doit être effectuée

- Une adresse (AD) désignant l'opérande de cette opération

Ainsi, par exemple, un algorithme de calcul d'une addition décrit par l'expression arithmétique z = x + y donnera naissance a` une suite de 3 instructions telles que :

- LDA X Chargement (load) du contenu de la cellule mémoire X dans une cellule appelée accumulateur (c'est un des *registres*). On note ce registre A ici.
- ADD Y Addition du contenu de la cellule mémoire Y avec le contenu de l'accumulateur
- STO Z Stockage (sto pour… store) du contenu de l'accumulateur dans la cellule mémoire Z

LDA, ADD, STO étant des codes opération ; X, Y, Z les adresses des opérandes.

Toute expression arithmétique ou logique, quelle que soit sa complexité, va être décomposée en une suite d'instructions de cette nature. L'unité de commande va alors se charger d'enchaîner séquentiellement la suite de ces instructions calculant cette expression.

Cependant il est bien évident que ce seul mécanisme est insuffisant et ne peut réaliser les instructions itératives ou conditionnelles des langages de haut niveau.
Pour cela, il est nécessaire d'introduire des instructions spécifiques appelées des « ruptures de séquence ». Celles si peuvent être simples ou conditionnelles.
Une instruction de rupture de séquence simple est de la forme :

- JMP X Saut Inconditionnel (jump) indique que le programme se poursuit à partir de l'adresse X

Une instruction de rupture de séquence conditionnelle est de la forme :

- JMPc X Saut Si *c* indique que le programme se poursuit à partir de l'adresse X si la condition *c* est réalisée.

Si la mémoire d'adresse X contient 10, MUN est l'adresse d'une cellule contenant la valeur -1, et « + » signifie le résultat est strictement positif, que va faire le programme suivant ?

```
        CHA   X
ADR     ADD   MUN
        JMP+  ADR
        STO   X
```

Remarque : il est particulièrement simple de tester si un entier est positif. Comme nous l'avons vu dans le chapitre précédent sur le codage, son premier bit vaut 0. Les opérations réalisables par les premières UAL étaient des opérations particulièrement simples. Rappelez-vous comment on fait pour multiplier par 2/diviser par 2 par exemple…

d. L'unité de commande (approfondissement).

Si l'UAL est le cœur de l'ordinateur, l'unité de commande en est le cerveau : comme son nom l'indique, c'est elle qui donne les ordres à toutes les autres parties de l'ordinateur.
Elle est principalement de quatre registres :

- CO : le compteur ordinal, qui contient l'adresse de la prochaine instruction à executer ;
- RI : le registre d'instruction qui contient l'instruction en cours d'execution ;
- ACC : l'accumulateur chargé de stocker des opérandes intermédiaires ;
- CC : le code condition, utilisé pour les instructions de rupture conditionnelle (saut).

Son algorithme de fonctionnement est le suivant :

**Répéter (indéfiniment)**

*L'instruction désignée par le compteur ordinal est chargée dans le registre d'instructions, soit :*

RI ← M[CO]

*Le compteur ordinal est incrémenté pour désigner l'instruction suivante, soit :*

CO ← CO + 1

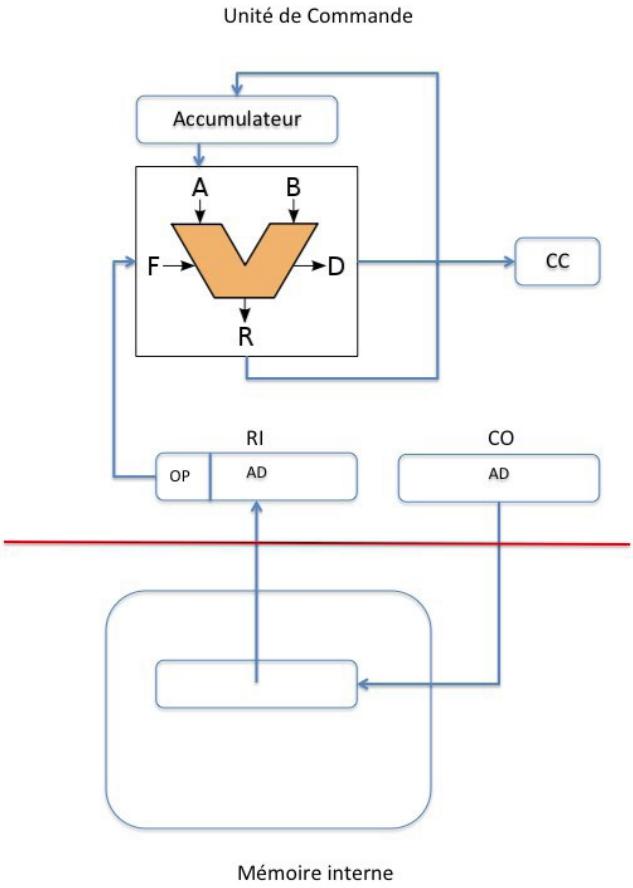
*L'opération chargée dans RI se décompose en deux parties : la fonction proprement dite désignée par OP, et l'opérande sur lequel cette fonction est exécutée, désignée par AD. L'opération désignée par OP est exécutée avec l'opérande AD, soit :*

RI.OP (RI.AD)

*En cas de rupture conditionnelle, le contenu du compteur ordinal CO est changé par l'adresse du saut.*

**Fin répéter**

Le schéma de fonctionnement en est le suivant, vous pouvez constater que l'entrée A de l'UAL correspond au registre accumulateur ACC.




3. <u>Périphériques</u>

On distingue :

- Les organes d'*entrée/sortie* permettant l'échange d'informations entre le système et le milieu extérieur (son environnement). Les principaux organes d'entrée sont le clavier et la souris ; les principaux organes de sortie sont l'écran et l'imprimante.
- Les *mémoires auxiliaires* qui sont des supports de stockage de grande capacité.

  Les principaux supports de stockage sont les disquettes, les disques durs et les CDROM.

  L'information est stockée dans ces différents supports sous forme de fichiers.

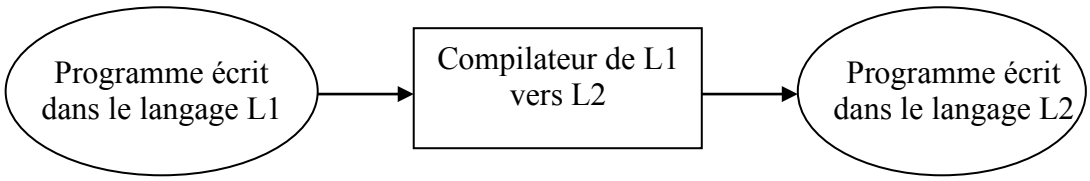
4. Compléments : compilation et systèmes d'exploitations.

Programmer en Assembleur n'est ni très simple ni très agréable. Aussi, de nouveaux langages de programmation, dit de haut niveau, ont peu à peu été définis pour faciliter la tâche des programmeurs. En inventer un nouveau, cela signifie :

- Définir un langage permettant d'exprimer n'importe quel algorithme ;
- Définir une correspondance, une méthode de traduction (exprimable par un algorithme !) entre ce nouveau langage et un langage d'assemblage.

La difficulté vient plutôt de la seconde condition que de la première. En effet, toutes les langues humaines sont en principe capables d'exprimer toute méthode de calcul, tout algorithme, de façon plus ou moins informelle. Mais les ordinateurs ne comprennent rien aux langues humaines. De plus, les langues humaines sont beaucoup trop imprécises et ambiguës pour être traduisibles directement en Assembleur. Le problème est donc de définir une langue qui sera traduisible en Assembleur, de telle sorte que cette traduction elle-même puisse s'exprimer par un algorithme.

Ce rôle de traduction est joué par ce qu'on appelle les *compilateurs*. Un compilateur est un logiciel capable de transformer un programme écrit dans un langage de programmation donné L1, appelé code *source*, en un programme réalisant le même traitement mais écrit dans un autre langage L2, appelé code *compilé*, en général un Assembleur, comme le montre le schéma de la figure suivante :




Les compilateurs étant des programmes, ils sont eux-mêmes écrits dans un certain langage de programmation (et s'ils ne sont pas écrits en Assembleur, ils doivent eux-mêmes être compilés...).
La manière la plus simple de compiler un programme est de traduire les instructions une par une. Cependant, les compilateurs optimisent la traduction afin de rendre les programmes plus efficaces. Par exemple, conserver une donnée très souvent utilisée dans un registre, plutôt que d'aller la rechercher en mémoire systématiquement, ce qui peut prendre cent fois plus de temps.

Enfin, votre expérience de l'usage des ordinateurs diffère quelque peu de ce que l'on a vu dans ce chapitre : vous n'aviez jamais écrit une instruction en assembleur jusqu'à ce jour... C'est qu'un programme tourne en permanence sur l'ordinateur, le système d'exploitation (Windows 8, OsX, Linux, etc...). Ce programme est chargé au démarrage de l'ordinateur, parfois il plante : vous avez vu des écrans en ligne de commande en cas d'erreur majeure sur l'*os* (opérating system). Les rôles principaux du système d'exploitation sont les suivants :

- Fournir une «interface» entre l'ordinateur et l'utilisateur pour permettre à ce dernier de donner des ordres à la machine (par exemple : lire ou écrire des informations dans la mémoire, lancer une impression...) ou pour lui signaler les erreurs d'exécution ; cette interface prend soit la forme d'un langage de commande (comme «MS-DOS», Shell) soit la forme d'objets graphiques à manipuler (fenêtres, menus...) ;
- Gérer les «ressources» de l'ordinateur, à savoir ses mémoires, son microprocesseur et ses périphériques : les systèmes d'exploitation actuels, en effet, sont «multitâches» ; cela signifie qu'ils permettent à plusieurs programmes de s'exécuter en même temps, et se chargent de répartir l'occupation des ressources utilisées par chacun d'eux (par exemple si deux programmes P1 et P2 sont lancés en même temps, le système d'exploitation permettra à un petit bout de P1 de s'exécuter, puis laissera la place à un petit bout de P2, puis de nouveau à un petit bout de P1, etc., de sorte que l'utilisateur aura l'impression que P1 et P2 sont exécutés en parallèle, alors que le processeur est toujours unique et séquentiel) ;
- Être indépendant du matériel, masquer les particularités de la machine en substituant aux ressources physiques des abstractions (par exemple, la notion de fichier, sur laquelle nous reviendrons, est une notion abstraite, indépendante de la nature du support sur lequel les données de ce fichier sont réellement stockées) ;
- Contrôler les usagers en leur donnant des droits différents selon leur statut (associés par exemple à différents mots de passe).

# Exercices architecture.

Les exercices qui suivent s'appuient sur une structure ultra simplifiée d'un ordinateur totalement imaginaire dont voici les caractéristiques :

La mémoire centrale est constituée d'un tableau de 32 cellules
Chaque cellule est constituée d'un octet
Chaque instruction est constituée
. d'un code opération sur 3 bits
. d'une partie adresse sur 5 bits

Le jeu d'instructions est constitué des 8 instructions suivantes :

| Instruction | Nom | Effet (version simple) | Effet (version moins simple) | Code opération |
|---|---|---|---|---|
| LD X | Charger | Charge dans l'accumulateur (l'entrée A de l'UAL) le contenu situé à l'adresse X | ACC ← M [X] | 000 |
| STO X | Stocker | Stocke à l'adresse X le contenu de l'accumulateur (le résultat du calcul précédent) | M [X] ← ACC | 001 |
| ADD X | Additionner | Ajoute à l'accumulateur (entrée A de l'UAL) le contenu situé à l'adresse X | ACC ← ACC+M [X] | 010 |
| SUB X | Soustraire | Soustrait à l'accumulateur (entrée A de l'UAL) le contenu situé à l'adresse X | ACC ← ACC-M [X] | 011 |
| JMP ADR | Saut | Saut à l'adresse ADR | CO ← A | 100 |
| JMPZ ADR | Saut si = 0 | Idem, si résultat du calcul à la ligne d'avant (stocké dans l'accumulateur) est égal à 0 | si CC = 0 alors CO ← A | 101 |
| JMPP ADR | Saut si > 0 | Idem, si résultat du calcul dans l'accumulateur est supérieur à 0 | si CC > 0 alors CO ← A | 110 |
| JMPN ADR | Saut si < 0 | Idem, si résultat du calcul est inférieur à 0 | si CC < 0 alors CO ← A | 111 |

On considérera qu'un programme commence à l'adresse 8 (1000 en binaire), et que les adresses 0 à 7 seront utilisées pour stocker les données. On notera END pour la fin de programme

Ex 1.
Que font les programmes ci-dessous ? Version facile en code, version pénible en binaire.

| Adresse | Contenu |
|---|---|
| 0 | 41 |
| 1 | 54 |
| … | |
| 8 | LD 0 |
| 9 | ADD 1 |
| 10 | STO 2 |

| Adresse | Contenu |
|---|---|
| 0 | 00100011 |
| 1 | 01000011 |
| … | |
| 1000 | 000000001 |
| 1001 | 01100000 |
| 1010 | 00100010 |

Ex 2.
Écrire un programme vérifiant les conditions suivantes :

- Soit *x* une valeur écrite dans la case mémoire 0.
- Lire le nombre rangé dans la case mémoire 1.
- Si ce nombre est plus grand ou égal à *x*, remplacer le contenu de la case 3 par ce nombre.

Ex 3.
Même exercice que le précédent, mais maintenant, on écrit dans la case mémoire 3 :

- 0 si le contenu de la case 1 est strictement inférieur à celui de la case mémoire 2
- 1 sinon.

Ex 4 : Écrire un programme qui effectue la multiplication de deux entiers positifs ou nuls.

Ex 5 : Écrire un programme qui effectue la division euclidienne d'un entier *a* par un entier *b*. On stockera le quotient et le reste.