# M99 : l'ordinateur en papier

**ENS-Rennes**

**Objectifs pédagogiques:**

- Comprendre le principe de base d'un processeur informatique
- Comprendre l'encodage des opcodes dans la mémoire, et le cycle fetch/decode/execute (ex 1)
- Modifier un programme en assembleur, entrevoir le travail d'un compilateur (ex 2)

Le M99 (page séparée) est doté de 100 cases mémoire (la grille en haut), et d'un processeur (en bas).

La mémoire est composée de 100 mots mémoire de 3 chiffres (valeur de 000 à 999). Chaque mot mémoire peut être désigné par une adresse codée sur deux chiffres. Cette mémoire va contenir données et instructions.

**Unité Arithmétique et Logique (ALU).** Le processeur dispose de trois registres directement utilisables: A et B sont utilisés pour les opérandes des opérations tandis que R est pour le résultat. Ces registres sont de 3 chiffres, mais contrairement à la mémoire, ils ont un signe. Leurs valeurs sont donc comprises entre -999 et 999.

Le processeur dispose aussi d'un quatrième registre nommé PC (Program Counter). C'est le pointeur d'instruction, contenant l'adresse mémoire de la prochaine instruction à exécuter. Lorsqu'on utilise le M99, on peut noter le numéro de l'instruction à exécuter dans la case prévue à cette effet, mais en pratique, il est plus simple de le matérialiser avec un "pion" situé sur une des cases de la grille mémoire, ou même de suivre avec son doigt.

**Unité de commande.** Elle pilote l'ordinateur. Son cycle de fonctionnement comporte 3 étapes :

1. *(fetch)* charger l'instruction depuis la case mémoire pointée par PC dans la case `instr` (sous le PC).
2. *(decode)* identifier l'opération à réaliser à partir des 3 chiffres la codant et grâce au pense-bête de droite.
3. *(execute)* exécuter cette instruction, puis incrémenter ensuite le PC.

**Premier jeu d'instructions.** D'autres instructions pourront être rajoutées au fil des exercices.

| *Code* | *Mnémonique* | *Instruction à réaliser* |
|---|---|---|
| 0 x y | `STR xy` | Copie le contenu du registre R dans le mot mémoire d'adresse *xy* |
| 1 x y | `LDA xy` | Copie le mot mémoire d'adresse *xy* dans le registre A |
| 2 x y | `LDB xy` | Copie le mot mémoire d'adresse *xy* dans le registre B |
| 3 x y | `MOV x y` | Copie registre Rx dans Ry (R0: R; R1: A; R2: B) |
| 4 0 0 | `ADD` | Ajoute les valeurs des registres A et B, produit le résultat dans R |
| 4 0 1 | `SUB` | Soustrait la valeur du registre B à celle du registre A, produit le résultat dans R |
| 4 x y | etc | (d'autres opérations arithmétiques et logiques peuvent être encodées ainsi) |
| 5 x y | `JMP x y` | Branche (saute) en *xy* (PC reçoit la valeur *xy*) |
| 6 x y | `JPP x y` | Branche en *xy* si la valeur du registre R est positive |
| 7 x y | `JEQ x y` | Saute une case (PC++) si la valeur du registre R est égale à *xy* |
| 8 x y | `JNE x y` | Saute une case (PC++) si la valeur du registre R est différent de *xy* |

PC est incrémenté deux fois si la condition de JEQ ou JNE est vraie: par l'unité de commande et par l'opération. On peut utiliser le mnemotechnique `DAT x y z` pour stocker une valeur numérique arbitraire *xyz* en mémoire. Les dépassements de capacité *(overflow)* sont gérés de manière cyclique: 999 + 1 = 0; 0 - 10 = 990.

**Boot et arrêt.** La machine démarre avec la valeur nulle comme pointeur d'instruction (`PC=0`) et elle s'arrête si le pointeur d'instruction vaut 99. On peut donc utiliser le mnémonique `HLT` comme synonyme de `JMP 99`.

En cas de faute (comme par exemple une division par zéro), le programme est interrompu (`PC := 99`).

**Entrées/sorties** Les entrées/sortries sont "mappées" en mémoire: Écrire le mot mémoire 99 écrit sur le terminal, tandis que les valeurs saisies sur le terminal seront lues dans le mot mémoire 99.

★ **Exercice 1:** Prise en main, opcode et cycle fetch/decode/execute.

▷ **Question 1:** Que fait le programme chargé à l'adresse 0 ?

▷ **Question 2:** Que fait le programme débutant à l'adresse 13?

▷ **Question 3:** Écrire à partir de l'adresse 20 un programme affichant le minimum de deux entrées clavier.

★ **Exercice 2:** Premiers programmes en assembleur.

▷ **Question 1:** Que fait le programme débutant à l'adresse 40 (pour les entrées 5 et 2)?

▷ **Question 2:** Raccourcissez ce programme pour l'écrire avec aussi peu d'instructions que possible.

▷ **Question 3:** Corrigez ce programme quand la seconde entrée vaut 0.

**À propos du M99.** Cette activité de découverte a été inventée par le groupe InfoSansOrdi.
Plus d'informations et de ressources en ligne: `https://github.com/InfoSansOrdi/M999`

★ **Réalisme du M99.** Cet ordinateur en papier est assez simple à utiliser avec seulement un crayon, mais il a été pensé pour être relativement réaliste par rapport aux vrais ordinateurs.

▷ **Mémoire.** La mémoire d'un vrai ordinateur est également découpée en mots mémoires, chacun étant doté d'une adresse unique. En général, les vrais ordinateurs utilisent des mots de 1 octet (8 bits). Les ordinateurs 32 bits peuvent avoir jusqu'à $2^{32}$ mots (soit un peu plus de 4 Go de mémoire) tandis que les ordinateurs 64 bits peuvent en avoir jusqu'à $2^{64}$ en théorie (18 Exaoctets, $18 \cdot 10^{18}$ octets).

▷ **Registres et caches.** Les vrais processeurs ont également des registres afin de gérer au mieux le problème de la barrière mémoire. Ils ont également des caches pour optimiser les échanges entre la mémoire et le CPU. Là où lire en mémoire peut demander une centaine de cycles CPU, lire en cache prend entre 10 et 30 cycles. Le M99 a un nombre très limité de registres (et aucun cache) pour simplifier.

▷ **Opcodes.** Les vrais programmes sont également écrits sous forme d'opcodes en mémoire des vrais ordinateurs, avec le préfixe indiquant l'opération tandis que le suffixe indique les opérandes. Le jeu d'opérations élémentaires disponibles varie beaucoup d'une famille de processeurs à l'autre.

Pour le M99, nous avons choisi d'utiliser des mots mémoires de trois positions décimales, ce qui contraint fortement le nombre d'instructions disponibles. Ces contraintes sont parfaitement réalistes de celles que doivent résoudre les fabriquants de CPU. Ajouter des instructions simplifie l'écriture de programmes efficaces, mais complique grandement le processeur, qui devient plus cher et plus énergivore.

Les processeurs de la famille RISC (reduced instruction set CPU) visent la simplicité et n'offrent que peu d'instructions tandis que ceux de la famille CISC (complex instruction set CPU) offrent des opérations optimisées plus spécialisées, comme des opérations vectorielles.

Il serait faux de dire que l'une des familles est vraiment préférable à l'autre. Il s'agit plutôt de deux compromis différents entre complexité du processeur et complexité des programmes. Les processeurs des téléphones portables sont souvent des RISC (par exemple du constructeur ARM) tandis que ceux des ordinateurs sont souvent des CISC (par exemple des constructeurs Intel ou AMD).

▷ **Dépassements de capacité.** De nombreux processeurs ne spécifient pas leur comportement en cas de problème arithmétique. Le calcul (équivalent à) $999 + 1$ est réputé absurde sur un vrai processeur, qui est alors libre de renvoyer n'importe quelle valeur. Suite à cela, on peut même avoir $42 > 24$. Ces cas aux comportements indéfinis *(Undefined Behavior)* sont fréquents au coeur de l'ordinateur pour éviter de sur-contraindre le design des composants. Charge aux utilisateurs de ne jamais faire de dépassement de capacité !

▷ **Entrées/sorties.** Gérer les entrées/sorties au travers d'adresses particulières de l'espace d'adressage du bus mémoire est parfaitement réaliste. En revanche, il est rare d'avoir plusieurs périphériques à la même adresse et on aurait pu séparer les lectures du clavier et les écritures à l'écran dans des zones mémoire différentes. De plus, nous avons ignoré toute la synchronisation qu'un vrai processeur doit faire pour échanger avec les périphériques, souvent bien plus lents que le processeur.

▷ **Pile et fonctions.** La plus grande limitation du M99 est certainement l'absence de pile d'exécution, ce qui empêche l'écriture de fonctions. L'algorithme récursif de la factoriel est par exemple impossible à écrire pour l'instant. On peut étendre le M99 en ajoutant un nouveau registre `SB` (Stack Base – initialisé à 98) et les opérations suivantes (où `Rx` désigne le registre n°$x$. `R1` est le registre A):

| *Code* | *Mnémo* | *Instruction à réaliser* |
|---|---|---|
| 48 x | `PSH x` | `mem(SB):=Rx; SB--` (Rx est écrit dans la case pointée par SB, qui est ensuite décrémenté) |
| 49 x | `POP x` | `SB++; Rx:=mem(SB)` (Incrément de SB puis le contenu de mem(SB) est écrit dans Rx) |
| 9 xy | `CAL xy` | `mem(SB):=PC; SB--; PC:=xy` (PC est mis sur la pile, puis l'exécution saute en $xy$) |
| 409 | `RET` | `PC:=mem(SB); SB--` (on dépile l'adresse où reprendre l'exécution) |

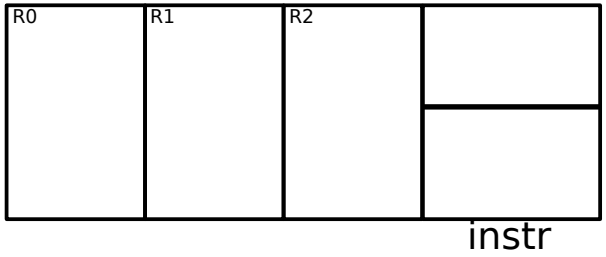
Écrivez le code de la factoriel sous sa forme récursive en utilisant cette extension. Vous pourrez soit modifier le programme déjà écrit, ou ajouter une opération `MUL` de code 402 pour stocker le produit de `A` et `B` dans `R`.

| | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 1 0<br>LDA 10 | 0 4 2 | | | 1 9 9<br>LDA 99 | 1 6 0<br>LDA 60 | 0 0 2<br>DAT 2 | | | | 0 |
| 1 | 2 1 1 | 0 1 0 | | | 3 1 0<br>MOV A R | 2 6 1<br>LDB 61 | 0 0 1<br>DAT 1 | | | | 1 |
| 2 | 4 0 1<br>SUB | | | | 0 5 9<br>STR 59 | 4 0 1<br>SUB | 0 0 0<br>DAT 0 | | | | 2 |
| 3 | 6 0 7 | 1 9 9 | | | 2 9 9<br>LDB 99 | 0 6 0<br>STR 60 | | | | | 3 |
| 4 | 3 1 0 | 3 1 0 | | | 3 2 0<br>MOV B R | 6 4 6<br>JPP 46 | | | | | 4 |
| 5 | 0 9 9 | 0 1 0 | | | 0 6 0<br>STR 60 | 1 6 2<br>LDA 62 | | | | | 5 |
| 6 | 5 9 9 | 1 9 9 | | | 1 6 2<br>LDA 62 | 3 1 0<br>MOV A R | | | | | 6 |
| 7 | 3 2 0 | 3 1 0 | | | 2 5 9<br>LDB 59 | 0 9 9<br>STR 99 | | | | | 7 |
| 8 | 0 9 9 | 0 1 1 | | | 4 0 0<br>ADD | 5 9 9<br>JMP 99 | | | | | 8 |
| 9 | 5 9 9 | 5 0 0 | | | 0 6 2<br>STR 62 | 0 0 3<br>DAT 3 | | | | INPUT<br>OUTPUT | 9 |
| | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | |

# M99

v200908

ordinateur en papier




| Code | Mnémo | Signification |
|---|---|---|
| 0 x y | STR xy | R → mem(xy) |
| 1 x y | LDA xy | A ← mem(xy) |
| 2 x y | LDB xy | B ← mem(xy) |
| 3 x y | MOV x y | Copie le registre Rx dans Ry |

310 (MOV A R) R := A

| | | |
|---|---|---|
| 4 0 0 | ADD | R := A + B |
| 4 0 1 | SUB | R := A − B |

| | | |
|---|---|---|
| 5 x y | JMP xy | PC := xy |
| 6 x y | JPP xy | R > 0 ⇒ PC :=xy |
| 7 x y | JEQ xy | R=xy ⇒ PC+=2 |
| 8 x y | JNE xy | R≠xy ⇒ PC+=2 |