```
40052a:       48 83 ec 20             sub    rsp,0x20
40052e:       89 7d ec                mov    DWORD PTR [rbp-0x14],edi
400531:       48 89 75 e0             mov    QWORD PTR [rbp-0x20],rsi
400535:       c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
40053c:       eb 0e                   jmp    40054c <main+0x26>
40053e:       bf e4 05 40 00          mov    edi,0x4005e4
400543:       e8 b8 fe ff ff          call   400400 <puts@plt>
400548:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
40054c:       83 7d fc 09             cmp    DWORD PTR [rbp-0x4],0x9
```

Le langage machine

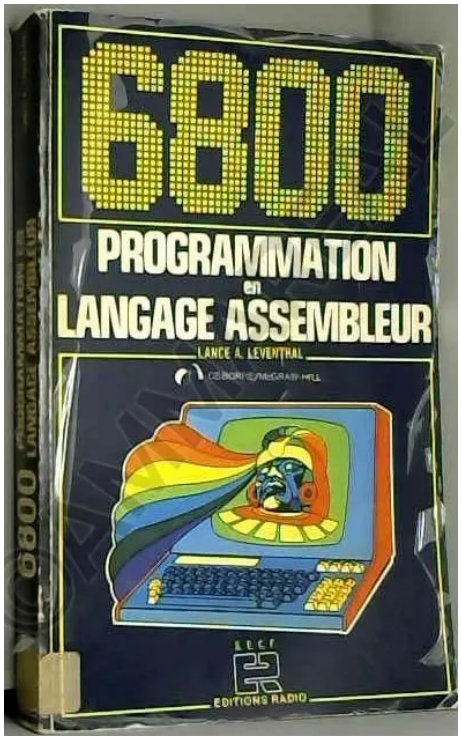
Le langage machine, ou code machine, est la suite de bits qui est interprétée par le processeur d'un ordinateur exécutant un programme informatique. C'est le langage natif d'un processeur, c'est-à-dire le seul qu'il puisse traiter. Il est composé d'instructions et de données à traiter codées en binaire.

Chaque instruction commence par un nombre appelé `opcode` (ou code opération) qui détermine la nature de l'instruction.
Par exemple, pour les ordinateurs d'architecture `x86`, l'opcode `0xB0` (en binaire 10110000) correspond à l'instruction écrire dans le registre `AL`.
Par conséquent, l'instruction 0xB0 0x14 (10110000 00010100) correspond à écrire dans le registre `AL` la valeur 0x14 (ou 20 en décimal).

Alors que le langage machine était le seul disponible à l'aube des ordinateurs, il est aujourd'hui très long et fastidieux de développer en binaire : il faut passer par au moins un langage intermédiaire.

L'assembleur

Le langage machine étant très long et fastidieux de développer en binaire, il faut passer par au moins un langage intermédiaire. Le langage le plus facile à convertir en code machine est l'**assembleur** car il possède quasiment les mêmes instructions. L'assembleur (ou langage assembleur) diffère d'un processeur à l'autre, bien que les instructions soient au bout du compte très semblables.



Exemple

Pour un processeur de la famille `x86` reconnaît l'instruction :

`« écrire le nombre 97 (la valeur est donnée en hexadécimal : $61_{16} = 97_{10}$) dans le registre AL »`

Cette instruction en opcode :

`10110000 01100001`

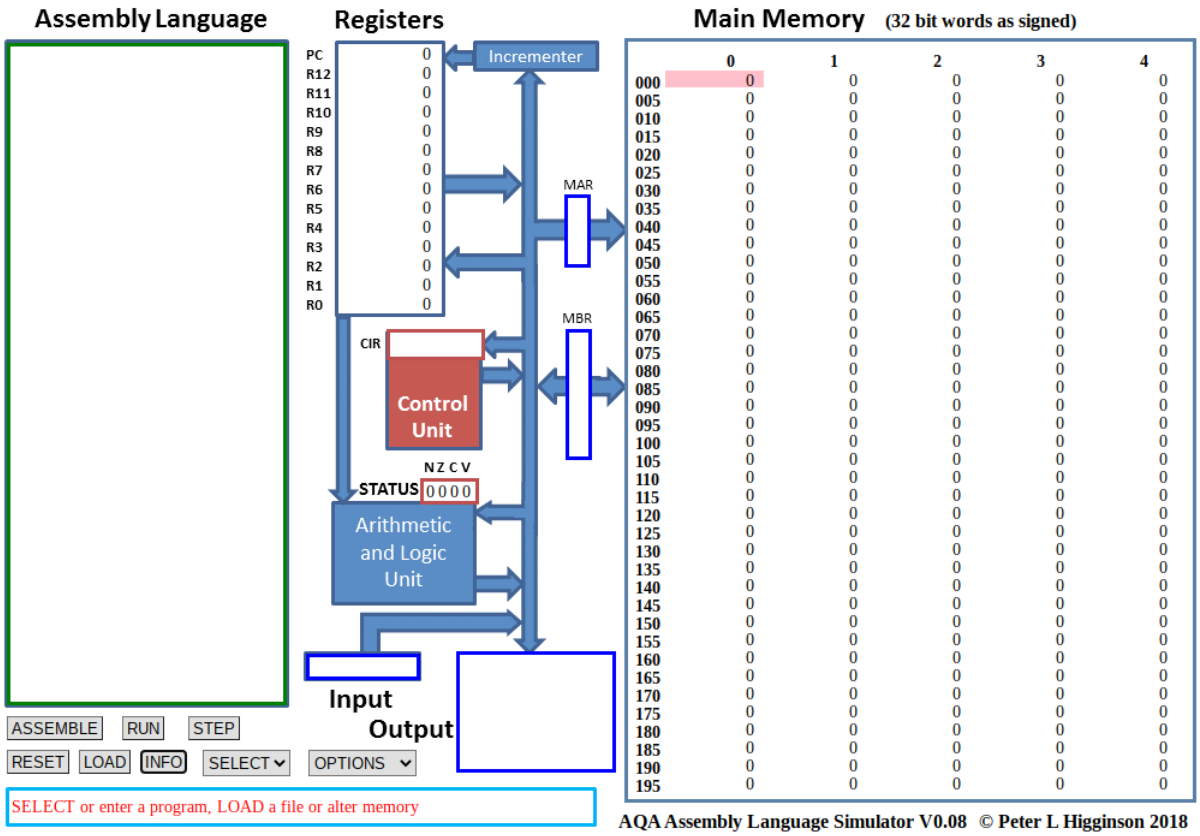
En assembleur x86

```
movb $0x61,%al
```

Les instructions

Dans le cadre de ce cours nous allons utiliser la simulation de Peter Higginson

http ://www.peterhigginson.co.uk/AQA/



?AIDE : Voici une traduction de la documentation proposée à partir du bouton INFO

— Les registres sont numérotés de R0 à R12. Le contenu du régistre n est noté Rn.

— Une **constante** notée `#nnn` .
Exemple : `#4` : la valeur décimale 12.

```
MOV R0, #4
```

Ce qui correspond à :
`Mettre le nombre 4 dans le registre R0`

— Les **opérations** peuvent être effectuées sur deux types de valeurs (notées <opérande2> dans la suite) :

Exemple :

```
MOV R0, #4
ADD R1, R0, #11
```

Ce qui correspond à :
`Mettre le nombre 4 dans le registre R0`
`Ajouter le nombre 11 à la valeur du registre R0 et le stocker dans le registre R1`

— On peut ajouter des **étiquettes** dans le programme (notées <label>) ; il suffit de noter le nom de l'étiquette de son choix et de le faire suivre de deux points.

Exemple :

```
    MOV R0, #4
    CMP R0, #10
    BNE else
    MOV R0, #11
    B endif
else:
    ADD R0, R0, #1
endif:
    STR R0,30
    HALT
```

Ce qui correspond à :

```
Mettre 4 dans le registre R0
Comparer R0 à 10
Si R0 est égale
Mettre 11 dans le registre R0
Sinon
Ajouter 1 à la valeur du registre R0 et
le placer dans le registre R0
Fin si
Stocker la valeur du registre R0 dans
l'emplacement mémoire 30
```

Toute référence à cette étiquette fera "sauter" le programme à la case mémoire de l'étiquette et exécutera donc les instructions qui suivent.

— les **commentaires** commencent par au moins un /.

Exercice 1 ★

Faites en sorte que dans la mémoire 100, il y ait le résultat de la somme de 42 et 54.

Exercice 2 ★

Ecrire un programme qui calcule la somme des 3 nombres 50, 42 et 54, en n'utilisant que les deux registres R0 et R1.

Exercice 3 ★★

Ecrire un programme qui demande à l'utilisateur de saisir successivement 2 nombres et qui renvoie en sortie leur somme.
On utilisera les instructions `INP` et `OUT` :

Exemples :

— `INP R0,2` : stocke l'entrée `utilisateur` dans le registre `R0`

— `OUT R0,4` : Affiche la valeur du registre `R0`

Exercice 4 ★

Ecrire un programme qui calcule la somme de $1+2+\cdots+100$

Affectations	Description
LDR Rd, <adresse mémoire>	Charge la valeur stockée dans l'emplacement de mémoire spécifié par <adresse mémoire> dans le registre d.
STR Rd, <adresse mémoire>	Stocke la valeur qui se trouve dans le registre d dans l'emplacement de mémoire spécifié par <adresse mémoire>.
MOV Rd, <opérande2>	Copiez la valeur spécifiée par <opérande 2>dans le registre d.
Opérations	Description
ADD Rd, Rn, <opérande2>	Ajouter la valeur spécifiée dans <opérande 2>à la valeur du registre n et stocker le résultat dans le registre d.
SUB Rd, Rn, <opérande2>	Soustrayez la valeur spécifiée par <opérande 2>de la valeur du registre n et stockez le résultat dans le registre d.
Comparaison	Description
CMP Rn, <opérande2>	Comparez la valeur stockée dans le registre n avec la valeur spécifiée par <opérande 2>
B <condition> <label>	Connectez conditionnellement l'instruction à la position <label> dans le programme si la dernière comparaison a répondu aux critères spécifiés par la <condition>. Les valeurs possibles pour <condition> et leur signification sont les suivantes :
<condition> EQ	égal à,
<condition> NE	différent de,
<condition> GT	supérieur à,
<condition> LT	inférieur à.
Fin du programme	Description
HALT :	Arrête l'exécution du programme.
Opérateur	Description
AND Rd, Rn, <opérande2> :	Effectue une opération ET logique au niveau du bit entre la valeur du registre n et la valeur spécifiée par <opérande 2> et stocke le résultat dans le registre d.
ORR Rd, Rn, <opérande2> :	Effectue une opération OU logique au niveau des bits entre la valeur du registre n et la valeur spécifiée par <opérande 2>et stocke le résultat dans le registre d.
EOR Rd, Rn, <opérande2> :	Effectue une opération logique ou binaire exclusive au sens des bits entre la valeur du registre n et la valeur spécifiée par <opérande 2>et stocke le résultat dans le registre d.
MVN Rd, <opérande2> :	Exécutez une opération NOT logique au niveau des bits sur la valeur spécifiée par <opérande 2>et stockez le résultat dans le registre d.
LSL Rd, Rn, <opérande2> :	Décalez logiquement vers la gauche la valeur stockée dans le registre n du nombre de bits spécifié par <opérande 2> et stockez le résultat dans le registre d.
LSR Rd, Rn, <opérande2> :	Décalez logiquement à droite la valeur stockée dans le registre n du nombre de bits spécifié par <opérande 2> et stockez le résultat dans le registre d.