Le Zen de l'assembleur pécé

« L'initiation à l'assembleur la plus rapide de toute l'histoire de la micro-informatique »

par Ice

N.B. Le but de cette initiation est de permettre un accès facile, rapide et « dédramatisé » à d'autres sources d'informations plus complètes, pas de les supplanter en cinq pages !

L'assembleur est un langage *trés* facile à apprendre pour la simple raison qu'il n'est capable d'effectuer que quelques opérations très élémentaires: placer ou déplacer une donnée en mémoire après l'avoir éventuellement modifiée. C'est tout. La difficulté provient de ce côté rudimentaire qui rend difficile l'écriture de longs programmes.

Cette simplicité est due au fait que le processeur ne manipule que des bits, et l'ensemble des représentations que nous avons (une image n'est qu'une suite de bits, un son également) se code de façon binaire, tout comme les instruction de l'assembleur. Mais l'assembleur fournit une façon plus rationnelle d'écrire une instruction: au lieu d'écrire 10010000 l'assembleur permet d'écrire simplement: NOP et se charge de le traduire à notre place en 10010000. Plutôt pratique.

De même, pour simplifier l'écriture de nombres aussi simples que 00111101, on utilise la base hexadécimale. Le tableau qui suit permet de convertir tous les nombres:

Binaire	Decimal	Hexa
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Prenons l'octet 00111101: 0011 correspond à 3 et 1101 à D , 00111101 donne donc 3D en hexadécimal.

Par convention, les bits d'un octet sont numérotés ainsi: 7 6 5 4 3 2 1 0 , les premiers bits(0,1...) sont appelés bits de poids faible et les derniers bits de poids fort.

La mémoire est composée de suite d'octets repérables par une ADRESSE. La mémoire est elle-lême divisée en zones de 65536 octets maximum que l'on appelle SEGMENTS. Cette partie est la plus embrouillée de l'architecture pécé. Pour bien la comprendre, et pour comprendre bien d'autres choses, il faut insister sur la représentation binaire. Un octet, qui est une suite de 8 bits (8=2 élevé à la puissance 3) peut se rrpésenter par un groupe de 8 symboles 0 ou 1; il peut donc prendre (2 élevé à la puissance 8=) 256 valeurs

soit un nombre compris entre 0 et 255 (ou de -128 à +127, dans ce cas le signe est défini par le bit le plus à gauche: s'il est égal à 1 le nombre est négatif, sinon il est positif). En effet:

00000001=1 en décimal
00000010=2
etc...
10000000=128
etc...
11111111=255

Un groupe de 16 bits (soit deux octets) appelé mot peut donc représenter un nombre de 0 à 65535 (ou de -32768 à +32767), un long mot de 32 bits (soit deux mots ou quatre octets) de 0 à 4294967295, etc...

Pour ce qui nous intéresse, les processeurs Intel, jusqu'au Pentium, traînent comme un boulet leur « compatibilité » avec un vieux modèle 16 bits, ce qui explique que les segments mémoire ne puissent contenir que 65536 octets ou 64 Ko (1 kilo-octet =1024 octets donc 64 x 1024=65536). Pour accéder à une mémoire située au-dela de ces 64 Ko, on a recours à une adresse de base au début du segment sur 16 bits et à un déplacement (OFFSET) sur 16 bits également. L'adresse effective se calcule en multipliant l'adresse du segment par 16 (10h en notation hexa) et en ajoutant le déplacement: l'adresse 40:17 (on note toujours segment:offset) correspond donc à 40h x 10h + 17h ce qui donne 417h.

Il y a beaucoup d'autres choses à dire sur la mémoire et sur sa gestion: distinguer la mémoire RAM de la mémoire ROM, la mémoire vidéo, la cartographie mémoire, la distinction entre le mode réel et le mode protégé, mais nous verrons tout cela en temps utile.

Les registres

Les registres du processeur sont des petites cellules mémoire très rapides: ils ne font pas partie de la mémoire, n'ont pas d'adresse et sont réservés à certaines opérations spécialisées telles que stocker temporairement une valeur, une adresse mémoire, etc. L'architecture Intel 8086 en comporte quatorze de 16 bits chacun. Sur les processeur supérieurs, ils sont étendus (et leur nom est précédé de « E » pour « Extended » par exemple EAX au lieu de AX, mais pour le moment nous nous contenterons pour nos explications du modèle 8086, à charge pour le lecteur de l' « étendre » à 32 bits).

Quatre registres généraux: AX, BX, CX, DX qui peuvent se décomposer en deux registres de huits bits (AH et AL: High pour les bits 8 à 15 et Low pour les bits 0 à 7).

Deux registre d'index: SI et DI

Quatre registres de segments: CS (Code Segment), DS (Data Segment), ES (Extra Segment), SS (Stack Segment). Ils permettent de calculer l'adressage des segments comme nous l'avons vu plus haut.

Trois registres pointeurs: BP (Base Pointer), SP (Stack Pointer (PILE), IP (Instruction Pointer) Ils permettent d'adresser les données contenues dans la pile qui est une simple zone mémoire réservée pour y ranger temporairement certaines opérations, nous en parlerons plus longuement un jour). Le pointeur IP indique la prochaine instruction à exécuter.

Un registre de drapeaux (FLAGS) : ce sont de simples bits qui donnent l'état du processeur à un certain moment ou qui sont testés par certaines instructions de sauts conditionnels que nous verrons une autre fois.

Syntaxe du langage assembleur

Une ligne assembleur se présente ainsi, chaque champ étant séparé par au moins un espace:

identificateur (facultatif) opcode opérandes(s) commentaire(facultatif)

Exemple:

```
Debut:      MOV     AX, 3       ;place 3 dans le registre AX
```

Un identificateur sert à identifier une adresse contenant une variable ou un label. Une variable est une cellule de mémoire que l'on peut lire ou modifier à volonté. Exemple:

```
texte:      DB    'bonjour'     ; chaîne contenant un ou plusieurs éléments tenant sur un octet (B= Byte/octet)
score       DW    5             ; sur un mot (Word)
bidon       DD    ?             ; sur un double mot (?= pas fixé)
```

Les modes d'adressage

Les modes d'adressage permettent d'accéder de différentes manières aux données situées en mémoire avant d'effectuer leur traitement. Leur bonne connaissance est fondamentale. Sur pécé le format se lit de droite à gauche : destination, source.

L'adressage immédiat: par exemple l'instruction MOV AX,3 va placer 3 dans le registre AX

L'adressage de registre: MOV BX,CX va transférer dans BX la valeur contenue dans CX (si CX= 3 alors BX = 3)

L'adressage direct:
```
            MOV  AL,[score]   ; si l'emplacement mémoire score = 5 (voir plus haut) alors AL=5
            MOV  [score], AL  ; place la valeur contenue dans le registre AL dans l'emplacement mémoire score.
```

L'adressage indirect de registre: c'est une combinaison de deux précédents.

```
            MOV  AL,[BX]      ; si BX contient l'adresse (offset) 10 et qu'à l'adresse 10 il y a 5 alors on place 5 dans AL.
```

L'adressage indexé direct: l'adresse source est calculée en ajoutant un offset à un registre DI ou SI, la destinationest un registre (ou vice-versa)

```
MOV  AL, [score+DI]        ;place en AL la valeur située à l'adresse score+ le contenu de DI.
```

L'adressage indexé de base: comme le précédent mais avec le registre BP en plus.

```
MOV  AL, [score+BP+DI]
```

L'adressage de base: `MOV  AL, [BX+2]` ; contenu de BX+2... (si BX=offset 10 c'est 12 !)

Les interruptions

Les interruptions sont des petits programmes (des routines) fournies par le DOS ou le BIOS. Il y a 256 interruptions qu'on peut diviser en deux catégories: matérielles et logicielles. Nous nous intéresserons ici uniquement aux interruptions logicielles et notamment l'interruption portant le numéro 21h qui permet d'accéder à de très nombreuses fonctions fondamentales.

Voilà nous en savons assez pour écrire notre premier programme.

Prenons un éditeur de texte (edit de DOS suffit) et tapons le programme suivant (les mots après « ; » sont facultatifs):

```
.model tiny                     ; petit modèle

    .data                       ; segment données

texte:  db    'Hello','$'       ; le texte qu'on veut afficher

    .code                       ; début segment instruction (code)

    org   100h                  ; débute à l'adresse 100 hexa (.COM du DOS)

prog:                           ; identificateur de début

    mov   dx,offset texte       ; place l'offset texte dans le registre DX
    mov   ah,09h                ; numéro de fonction 9 dans AH
    int   21h                   ; appelle de l'INTerruption N° 21h

    int   20h                   ; fin du programme (retour au DOS) par int 20h

    end   prog                  ; fin de notre programme (pour l'assembleur)
```

Sauvons ce texte (sans erreurs...) sous le nom de hello.asm, tapons sous DOS:

```
tasm   hello.asm
tlink -t hello.obj
```

Lançons hello.com, ça marche (normalement)

Sous DOS nous avons deux types de fichiers exécutables: les .COM qui ne contiennent qu'un seul segment de 64 Ko maximum et les .EXE qui peuvent contenir plusieurs segments de 64 Ko. L'indicateur « -t » permet d'indiquer à turbo assembleur (tasm) que nous désirons un exécutable .com.

Voilà, fin de cette première initiation, nous pouvons passer au dépouillement des volumineuses documentations Intel MMX et à la création de mondes 3D multmédia virtuels interactifs. Hum.