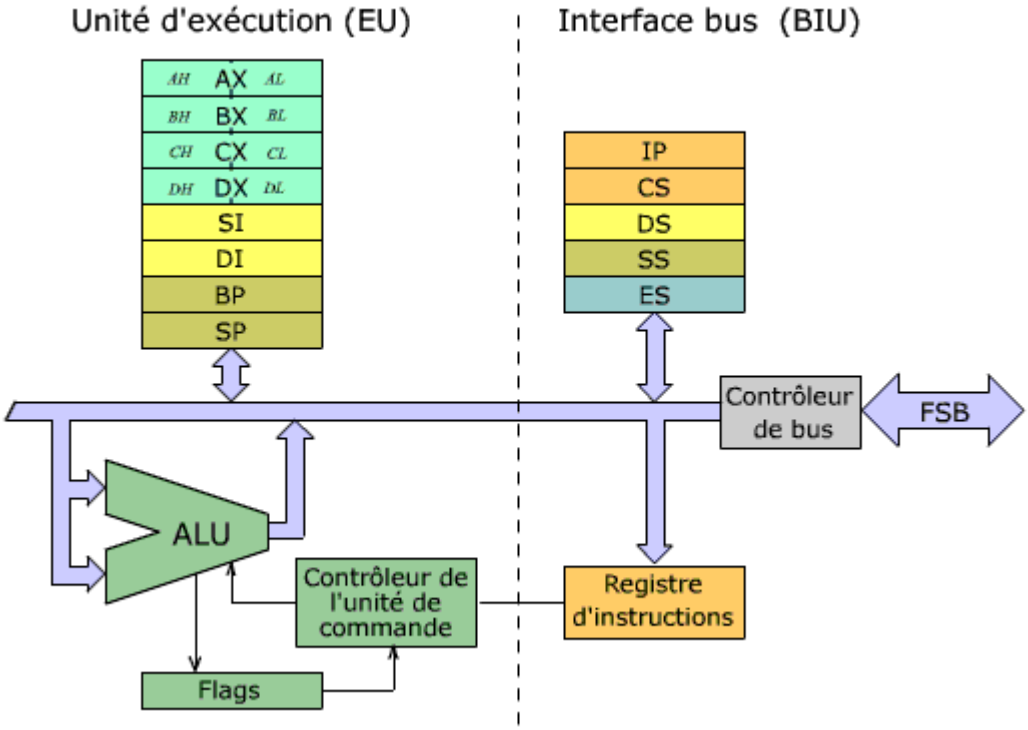
Introduction à l'assembleur

But de ce sujet

L'objectif de ces pages est de donner un aperçu succinct du langage assembleur. Ce langage est dit de "bas niveau » car il est étroitement lié à l'architecture du microprocesseur. « Registres » , « Adresses mémoires », « Interruptions », « Appels système » … voilà une série de notions qui risquent fort de rester théoriques. Ce serait dommage, alors qu'il est si simple de les aborder par la pratique à l'aide du langage assembleur et en visualisant à l'aide d'un debugger les instructions qui s'exécutent et les registres où cela se passe.

Les registres du processeur

Les registres décrits ci-dessous sont les registres du 8086, l'ancêtre des processeurs qui sont au cœur de nos PC actuels. Ces processeurs ont évolué depuis tout en restant compatibles avec leur ancêtre (compatibilité ascendante). Cette description devrait donc suffire se faire une idée de la structure d'un microprocesseur.



Registres de données

AX, BX, CX et DX

Ce sont des registres 16 bits du 8086, ils peuvent chacun être scindés pour y entreposer deux variables d'un octet. Ainsi le registre 16 bits AX peut être considéré comme l'adjonction de deux registres 8 bits AH et AL (H pour *high*, L por *low*)

Certaines instructions dédient ces registres à des rôles spécifiques :

AX "*accumulator*" l'accumulateur est privilégié pour y faire certaines opérations arithmétiques ainsi que les opérations d'entrée/sortie

BX "*base register*" il est appelé registre de base car l'adressage en mémoire peut se faire par son intermédiaire

CX "*count register*"est implicitement le registre compteurs de boucles pour les instructions répétitives. (CL pour les opérations de décalage)

DX "*data register*" sert dans certaines circonstances d'extension à l'accumulateur.

Registres d'adresses

Registres dits d'index :	SI	"*Source index*"
	DI	"*Destination index*"
Registres de base	BP	"*Base pointer*"
	SP	"*Stack pointer*"

Registres de segments

CS	*Code segment*	adresse de base pour le programme
DS	*Data segment*	adresse de base pour les données
ES	*Extra segment*	adresse de base pour d'autres données
SS	*Stack segment*	adresse de base pour la pile

Registre d'états

Ce registre contient les flags, ce sont des bits qui basculent d'un état à l'autre en fonction des résultats de l'exécution d'opérations arithmétiques ou logiques.

- Le flag Carry passe à 1 si une addition donne lieu à un report
- Le flag Signe passe à 1 si le bit le plus significatif du résultat vaut 1.
- Le flag Zéro passe à 1 si le résultat de la dernière opération est nul
- etc.

Ces flags servent entre autre aux instructions de sauts conditionnels.

L'instruction pointeur

IP Aussi appelé compteur ordinal est un registre qui s'incrémente sans cesse. Il contient en permanence l'adresse de la prochaine instruction à exécuter. « Faire un saut » dans un programme revient à inscrire dans le registre IP l'adresse de l'instruction où le programme doit se rendre. L'incrémentation de l'*Instruction Pointer* reprend alors depuis cette nouvelle valeur pour poursuivre la nouvelle séquence d'instructions.

Les adresses mémoire en mode segmenté

Les Pentium et AMD actuels utilisent deux modes de fonctionnement : le mode réel et le mode protégé. Le mode protégé est géré par Windows ou Unix, le mode réel (on devrait dire mode segmenté) est l'état dans lequel on se trouve au démarrage de la machine. La capacité d'adressage est limitée à 1 Mo comme pour les premiers 8086.
Les adresses sont données sous la forme **`Segment:Offset`**

Exemple :

Sous DOS (mais Windows émule parfaitement la chose) l'adresse `B800:0000` est l'adresse de base de la mémoire vidéo en mode texte. Cette adresse est donc celle du byte contenant le code ASCII du caractère situé dans le coin supérieur gauche de l'écran.
Le segment `B800` débute à l'adresse physique `B8000`.
L'adresse physique s'obtient en calculant Segment x 16 + Offset

Manipulation :

Avec l'utilitaire DEBUG, écrivons `41H` = le code ASCII du A majuscule à l'adresse `B800:0400` = adresse physique `B8400` de sorte à faire apparaître ce 'A' au milieu de l'écran.

```
C:\debug
-e B800:400
B800:0400  20.41   07.4E   20.
-q
```

Utilisation des registres pour former des adresses

Les adresses 20 bits sont constituées dans les registres en associant les registres de segment et les registres d'adresses

Les registres d'adresses sont associés par défaut à chacun des registres de segment.

CS est toujours utilisé avec IP pour former l'adresse logique CS:IP

SS est associé au *stack pointer* SP pour former l'adresse du dessus de la pile SS:SP le *base pointer* BP est lui aussi par défaut associé au *stack segment* SS

DS le *Data Segment* sert en principe de segment de base pour les adresses formées avec les registres SI, DI et BX (*source index*, *destination index* et *base register*)

Ecrire en assembleur avec DEBUG

Toujours avec le programme DEBUG, la commande 'a' permet d'encoder des instructions saisies en assembleur.

```
a
mov ax,B800
mov ds,ax
mov cx,4E41
mov di,400
mov [di],cx
ret
```

Utilisez la commande T pour tracer une à une, les instructions de ce bout de programme. Elles placent l'adresse du segment B800 dans le registre DS (*data segment*) Le registre CL reçoit le code ASCII d'un 'A' majuscule (41) (*Ne soyez pas troublés par la valeur 4E inscrite dans le registre CH, elle va joindre des attributs au code ASCII contenu dans CL pour faire ressortir le caractère en jaune sur fond rouge*)

Le nombre placé dans le registre DI indique le déplacement par rapport au début de la mémoire écran.

Les modes d'adressage

Nous n'avons encore utilisé qu'une instruction, l'instruction MOV. L'exemple ci-dessus illustre déjà le fait que les opérandes peuvent être spécifiées selon différentes modes.

Ce sont les modes d'adressage.

MOV DS , AX Adressage de registres , la valeur du registre AX est recopiées dans le registre DS

MOV CX , 4E41 Adressage immédiat , la valeur immédiate 4E41 est recopiée dans CX

MOV [DI],CX Adresse en mémoire , [DI] mis entre crochets signifie "à l'adresse donnée par DI" (cette adresse est l'offset à ajouter au segment de données)

L'instruction MOV attend deux opérandes, le premier indique la destination du déplacement de la donnée, le second opérande indique la source.

- La destination peut être un registre, une adresse mémoire ou un registre de segment (sauf pour le registre CS qui ne peut jamais être une destination)
- La source peut être un registre, une adresse mémoire, un registre de segment ou une valeur immédiate.

La liste ci-dessous inventorie toutes les manières d'utiliser l'instruction MOV :

MOV r1 , r2 r1 *reçoit la valeur identique à celle contenue dans* r2

MOV r , i *Le registre* r *est initialisé avec une valeur immédiate*

MOV m , i *Ecriture d'une valeur immédiate en mémoire*

MOV r , m *Lecture en mémoire à l'adresse m en destination du registre* R

MOV m , r *Ecriture en mémoire à partir du registre* R

Lecture / écriture entre registre de segment et la mémoire :

MOV s , m *Lecture*: s ←m

MOV m , s *Ecriture*: m ←s

Echanges entre registres généraux et registres de segment :

MOV s , r

MOV r , s

Dans cette liste les adresses 'm' sont spécifiées en plaçant entre cochets une valeur de l'offset et/ou des noms de registres. Exemples : [2000] ; [BP] ; [BP + 2000] ; [SI] ; [BP + SI] ; [BP + SI + 2000]

En fait toutes les combinaisons ne sont pas acceptées ! Les combinaisons valides sont celles que l'on forme en ne prenant pas plus d'un élément dans l'une des trois colonnes du tableau :

BX	SI	nombre
BP	DI	

Autrement dit, il ne peut pas il y avoir simultanément dans une adresse
Deux registres de base = dont le nom commence par B
Ou deux registres d'index = dont nom se termine par I

NB. Pour les registres BX, SI et DI le segment par défaut de l'adresse est DS
Dans le cas du Base Pointer BP le segment par défaut est SS

Il est parfois nécessaire de préciser la taille de la valeur à lire. On fait alors précéder l'adresse de la donnée par BYTE PTR ou par WORD PTR selon que l'adresse désigne un ou deux octets.

Aperçu rapide de quelques autres instructions

Les instructions MOV nous ont été utiles pour illustrer les modes d'adressage mais avec les MOV il nous est juste possible de déplacer les données d'un endroit à l'autre. Il nous faut d'autres instructions pour faire des opérations arithmétiques et logiques, des sauts, des appels à des fonctions, à des interruptions etc.

Voici donc d'autres instructions. La liste est loin d'être complète mais elle doit suffire pour se faire une idée de ce que sont les instructions en général.

ADD SUB CMP AND TEST OR OR

Modes d'adressage acceptés: r,m | m,r | r,r | m,i | r,i

MUL IMUL DIV IDIV

Exemples :

MUL BYTE PTR Valeur	AX ← AL x Valeur
MUL WORD PTR Valeur	DX.AX ← AX x Valeur
DIV BYTE PTR Valeur	AL ← AX / Valeur
DIV WORD PTR Valeur	AX ← DX.AX / Valeur

INC DEC NOT NEG

Appel d'un sous-programme : CALL *label*

Sauts inconditionnels : JMP *label*

Sauts conditionnels :

JZ (=JE)	JNZ (= JNE)
JC	JNC
JS	JNS

Assemblage et édition de liens

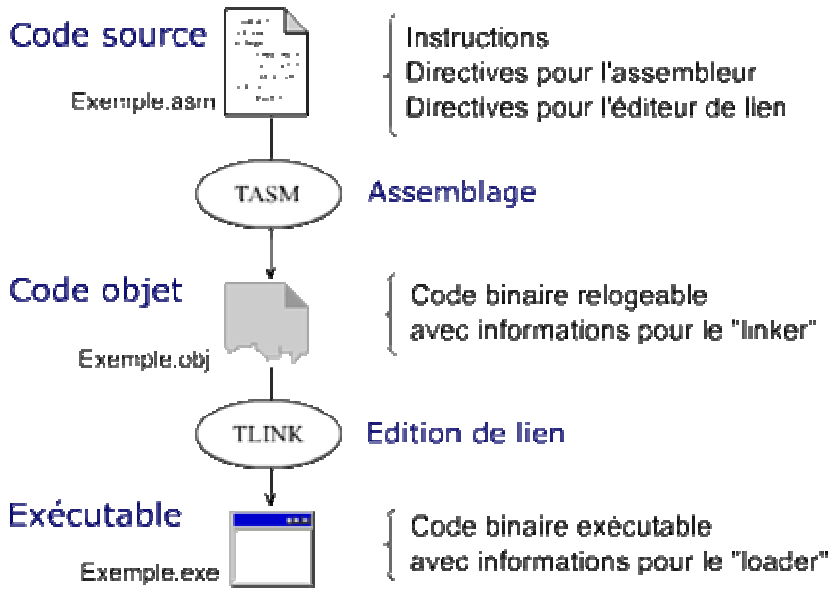
TASM est la commande pour l'assembleur de Borland. Nous l'utilisons avec la commande TLink déjà utilisée avec l'environnement de développement intégré TC ou le compilateur TCC. La vérification de l'exécution se fait avec le Turbo Debugger « TD ».
Tapez simplement TASM pour voir les options disponibles. Ces dernières sont assez rébarbatives aussi une fois que l'on sait celles qui nous conviennent on a tout intérêt à les fixer une fois pour toute dans un fichier de commande.
Voici les fichiers A.bat pour commander l'assemblage et L.bat pour l'édition de liens.

A.bat : `TAsm /zi /c /la %1.asm;`

L.bat : `TLlink /v /k %1.obj;`

Pour rappel, on part d'un code source ".asm" L'assemblage en fait un code objet ".obj" relogeable qui à son tour est traité par l'éditeur de lien pour en faire un code exécutable ".exe"

Accessoirement l'assembleur produit aussi un "listing" qui est un document destiné à être imprimé. On y retrouve côte à côte, le code source et le code binaire correspondant noté en hexadécimal.



L'option /v de la commande TLink demande à l'éditeur de liens d'ajouter à l'exécutable des informations de debug telles que les noms des variables et les numéros de lignes du fichier source. Ces informations sont destinées au Turbo Debugger.

Exemple de fichier source

Outre les instructions, le code source contient des directives aussi appelée "pseudo instruction" pour l'assembleur et l'édition de lien. Ces dernières diffèrent d'un assembleur à l'autre. Nous envisageons les directives reconnues par TASM et MASM. Les utilisateurs de NASM se réfèreront à la documentation de leur assembleur pour voir les correspondances.

Le fichier source SMALL.ASM est l'exemple classique du petit programme qui affiche 'Salut' à l'écran.

Si vous avez l'occasion de lire cet exemple avec la coloration de syntaxe adoptée ci-dessous, vous voyez les instructions en bleu, les directives en vert les commentaires en gris, les nombres en magenta tandis que les noms symboliques de notre cru sont en noir.

```
TITLE Exemple de programme assembleur / Modele small

Pile    segment para public
DB      1024    DUP (?)
DessusPile:
Pile    ENDS
;=====================================

Donnee  SEGMENT PARA PUBLIC

Message db      'Salut',0Ah,0Dh,'$'

Donnee  ENDS
;=====================================

Autre   SEGMENT PARA PUBLIC

Autre   ENDS
;=====================================

Code    SEGMENT PARA PUBLIC

ASSUME  CS:Code, DS:Donnee, ES:Autre, SS:Pile

Start:  MOV     AX,Donnee
        MOV     DS,AX
        MOV     AX,Autre
        MOV     ES,AX
        MOV     AX,Pile
        MOV     SS,AX
        MOV     SP,OFFSET DessusPile

        MOV     DX,OFFSET Message
        MOV     AH,9
        INT     21h

        MOV     AH,4Ch
        MOV     AL,0
        INT     21h

Code    ENDS

        END     Start
```

Les directives

Le fichier source comporte un nombre important de directives. Pour la suite, si vous vous contentez de faire de petits exercices avec uniquement un fichier source, il vous suffira de recopier ces directives en prenant un petit fichier tel que celui-ci comme modèle de base.

TITLE est une directive sans grande importance, elle indique uniquement à l'assembleur le titre que l'on souhaite voir en tête de chaque page du listing.

La paire de directives **SEGMENT** et **ENDS** est autrement plus importante ! Ces directives identifient les segments qui doivent être regroupés quand le code source est éparpillé en plusieurs modules. L'éditeur de lien regroupe tous les fragments de même nom pour former des segments adressables séparément. Les adresses des étiquettes sont alors considérées comme des déplacements (*offset*) par rapport à l'adresse de base du segments dans lequel elles sont définies.
Exemple : `Start` est l'adresse 0 par rapport au segment Code. `DessusPile` représente un déplacement de 1024 par rapport à la base du segment Pile.

Les directives `SEGMENT` et `ENDS` sont destinées à l'éditeur de lien mais il faut aussi indiquer à l'assembleur quels sont les registres de segment sont sensés contenir les adresses de base des segments. C'est le rôle de la directive **ASSUME** :

```
ASSUME  CS:Code, DS:Donnee, ES:Autre, SS:Pile
```

L'assembleur est dès lors capable de décider s'il doit ajouter des préfixes de segment aux adresses quand le segment n'est pas celui par défaut.. Ex : MOV AX,ES:[BP]

Mais ce n'est pas tout, l'éditeur de lien sait quels segments il doit regrouper, l'assembleur sait quels registres sont censés contenir les adresses de base des segments, il faut maintenant donner au CPU les instructions pour qu'il charge les adresse de base des segment dans les registres de segment prévus. Il ne s'agit plus de directives, ce sont bien des instructions qui seront faites après que le programme soit chargé en mémoire pour être exécuté.

```
MOV     AX,Donnee
MOV     DS,AX
MOV     AX,Autre
MOV     ES,AX
MOV     AX,Pile
MOV     SS,AX
```

On pourrait s'étonner que les valeurs immédiates `Donnee`, `Autre` et `Pile` ne puissent pas être directement transférée dans les registres de segments sans passer par l'intermédiaire d'un registre d'usage général tel que AX. Le mode d'adressage immédiat n'est possible qu'avec les registres de l'unité d'exécution, pas avec ceux du BIU *Bus Interface Unit*.

```
END     Start
```

La directive **END** indique la fin du code source. Elle sert aussi à indiquer au loader l'adresse de la première instruction, celle à placer dans les registres CS:IP de suite après le chargement du programme et avant de lui passer la main.

`DB` *Define byte*

La directive DB réserve un ou des emplacement d'un byte.

```
DB      1024    DUP (?)
```

Cette directive réserve tout simplement 1024 bytes comme emplacement libres sur la pile.

```
Message DB      'Salut',0Ah,0Dh,'$'
```

Cette fois les bytes réservés sont initialisés avec des valeurs entrées sous forme d'une chaîne de caractères, ou en hexadécimal. L'étiquette `Message` est l'adresse symbolique de la chaîne de caractères.
Des directives semblables existe pour réserver les mots de deux bytes `DW` (*define word*) , des doubles mots (4 bytes) `DD` (*define double*) et des mots quadruples (8 bytes) `DQ` .

… à suivre