DEBUG

Debug est un utilitaire mis à la disposition des utilisateurs des PC pour la mise au point des programmes. Il convenait parfaitement à cette tâche quand on programmait les premiers PC en assembleur ce qui ne se fait plus que très exceptionnellement aujourd'hui. Le langage assembleur ne sert plus que pour des tâches dites de bas niveau. Entendons par-là les fonctions très proches du hardware.
Actuellement les programmeurs utilisent des programmes de mise au point beaucoup plus performants. Les débogueurs actuels sont des outils graphiques adaptés aux langages de haut niveau et au développement d'applications graphiques.

L'intérêt du programme DEBUG est qu'il est fourni avec tous les systèmes d'exploitations de Microsoft depuis le DOS 2.0 jusqu'aux plus récents comme Windows XP. Ce petit programme est présent dans tous les PC et même sur les disquettes de démarrage. Nous utiliserons DEBUG lors des prochaines séances de laboratoire pour explorer la mémoire et des entrées / sorties ou même pour écrire quelques lignes d'assembleur en espérant acquérir de cette manière une meilleure représentation du cœur du système.

Essai du programme DEBUG

Ouvrez une fenêtre DOS et appelez le programme de DEBUG :

```
C:\>DEBUG ↵
-
```

La réponse de DEBUG est un tiret placé au début de la ligne. C'est « l'invite » de DEBUG. Elle annonce que le programme est prêt à recevoir une commande. Chaque commande se fait en tapant une seule lettre (D pour « Dump », Q pour « Quit ») suivie de paramètres quand c'est nécessaire. La commande la plus pratique est celle pour demander de l'aide, le point d'interrogation. Voici ce que cela donne :

```
-?
assemble (assembler)                A [adresse]
compare (comparer)                  C plage adresse
dump (lister)                       D [plage]
enter (entrer)                      E adresse [liste]
fill (remplir)                      F Plage liste
go (exécuter)                       G [=adresse] [adresses]
hex (hexadécimal)                   H valeur1 valeur2
input (entrer depuis port)          I port
load (charger)                      L [adresse] [lecteur] [secteur] [nbre]
move (déplacer)                     M plage adresse
name (désigner)                     N [nchemin] [listearg]
output (envoyer sur port)           O port val
proceed (avancer)                   P [=adresse] [nbre]
quit (quitter)                      Q
register (registre)                 R [registre]
search (rechercher)                 S plage liste
trace (tracer)                      T [=adresse] [valeur]
unassemble (désassembler)           U [plage]
write (écrire)                      W [adresse] [lecteur] [secteur] [nbre]

allocate expanded memory (allouer EMS)         XA [#pages]
deallocate expanded memory (désallouer EMS)    XD [desc]
map expanded memory pages (affecter)           XM [pageL] [pageP] [desc]
display expanded memory status (état EMS)      XS
```

La commande debug peut être invoquée en précisant le nom d'un fichier à charger en mémoire. Le nom du fichier doit obligatoirement être précisé avec son extension.
Exemple : `DEBUG test.exe ↵`

Les adresses mémoire

L'unité de mémoire est l'octet. Chaque octet a une adresse différente.
Les deux seules opérations possibles en mémoire sont la lecture et l'écriture.
L'adresse mémoire est codée en binaire mais nous la condensons par du code hexadécimal.
Elle se note en deux parties : le segment et l'offset

Adresse physique = Segment * 16 + Offset

Exemple : `1234:0000` = `12340`

On peut donc accéder à la même adresse mémoire à partir de segments distincts
Exemple : les adresses logiques `C000:0010` et `C001:0000` correspondent toutes deux à la même adresse physique `C0010`

Cette répartition de l'adresse en segment et offset est due au fait que les microprocesseurs des premiers PC n'avaient que 16 bits par registre alors qu'ils avaient 20 bits pour former leurs adresses. La plus grande adresse écrite avec 20 bits est donc `FFFFF` soit 1 MB. C'est la taille de la mémoire conventionnelle, la mémoire vue par le DOS.

On peut ainsi diviser la mémoire en un certain nombre de segments. La taille de chaque segment est un multiple de 16 octets (de 16 à 65536)
Les adresses des programmes et des données sont spécifiées relativement à des segments. Les adresses de base de ces segments sont inscrites dans des registres de segment du CPU.
Les registres CS, DS, SS et ES sont des registres de segment. Une fois dans DEBUG vous pouvez voir le contenu des registres avec la commande R.

A La commande « *assemble* » permet d'enregistrer des instructions en langage assembleur.
Ex. Entrez ce petit programme de 4 instructions qui additionne les nombres 1234 et 5678, en hexadécimal.

```
-A CS:100
150F:0100 MOV AX,1234
150F:0103 MOV BX,5678
150F:0106 ADD AX,BX
150F:0108 INT 20
150F:010A
```

Terminez la saisie de ce code par une ligne sans instruction.

D La commande « *Dump* » permet d'afficher le contenu d'une zone mémoire

Ex. Faite avec le programme EDIT un petit fichier texte et appelez-le Toto.txt
Nous chargeons ce texte en mémoire avec la commande DEBUG ensuite nous examinons son code en mémoire :

```
C:\>DEBUG toto.txt
-D
150F:0100  43 65 20 70 65 74 69 74-20 74 65 78 74 65 20 73   Ce petit texte s
150F:0110  65 72 74 20 64 27 65 78-65 6D 70 6C 65 0D 0A 14   ert d'exemple...
150F:0120  07 89 46 FA 0B C0 75 06-46 39 76 FE 77 DA 8A 45   ..F...u.F9v.w..E
```

Chaque ligne affichée par la commande Dump commence par une adresse sous la forme `Segment:Offset` .Viennent ensuite les valeurs de 16 octets successifs affichées en hexadécimal. Au bout de la ligne on retrouve un format texte quand c'est possible.
Les octets qui ne correspondent pas à des codes ASCII imprimables y sont représentés par des points.
La commande Dump accepte des paramètres.

D *offset*	L'offset combiné avec l'adresse de base contenue dans DS indique le premier octet à afficher. Le nombre de byte n'étant pas précisé, le programme affiche 128 bytes par défaut (8 lignes de 16 octets).
D ES:*offset*	Affichage à partir de l'adresse [ES] :*offset* , l'offset étant un nombre sous forme hexadécimale. ES est ici un exemple, les registres de segment DS, SS et CS peuvent aussi être mentionnés de cette manière. Si aucune adresse de base n'est spécifiée, la commande dump considère que le registre DS contient l'adresse de base du segment.
D	La commande D employée seule affiche les 128 suivants
D *base*:*offset*	Base et Offset sont deux nombres de 16 bits écrits en hexadécimal.
D *début fin*	Affichage d'une zone mémoire délimitée entre deux adresses
D *début* L *long*	Idem en indiquant la taille de la zone plutôt que l'adresse où elle se termine

Exemple : La date du BIOS est stockée dans chaque PC à l'adresse FFFF5. Voici comment l'obtenir :

```
C:\>DEBUG
-D FFFF:5 L 8
FFFF:0000 30 37 2F-32 35 2F 30 32 07/25/02
-Q
```

Attention la date est sous la forme américaine mois/jour/année. Le BIOS de cet exemple date donc du 25 juillet 2002.

E

« *Enter* » est la commande pour entrer de nouvelles valeurs en mémoire.
On lui indique l'adresse du début de la zone mémoire à examiner puis les valeurs à y inscrire.

```
-E CS:100 41,42,43,44,45,46,47
```

ou ce qui revient au même puisque (41 est le code ASCII de A, 42 de B etc.)

```
-E CS:100 "ABCDEFGH"
-D CS:100
150F:0100  41 42 43 44 45 46 47 48-20 74 65 78 74 65 20 73   ABCDEFGH texte s
150F:0110  65 72 74 20 64 27 65 78-65 6D 70 6C 65 0D 0A 14   ert d'exemple...
```

La commande E permet aussi d'examiner la mémoire byte par byte et des les modifier un à un. Tapez E puis l'adresse du byte mémoire que l'on veut et ENTER
L'adresse s'affiche à la ligne suivante. Elle est suivie du contenu du byte examiné puis d'un point.

```
-E B800:09F0
B800:00A0 43.41  07.  3A.42  07.
```

Tapez une autre valeur puis un espace pour passer au byte suivant. Si vous voulez passer au byte suivant sans modifier la valeur du byte affiché il suffit de taper de suite un espace (en s'interdisant de faire un retour chariot). La touche 'moins' permet de revenir en arrière. Vous pouvez examiner ainsi plusieurs bytes en les modifiants ou non. Pressez la touche ENTER pour terminer les modifications.

F « *fill* » Rempli une zone mémoire avec le ou les codes spécifiés.
Syntaxe : F *plage liste*
Exemples :
(Attention à bien respecter les adresses indiquées pour ne pas aller jardiner n'importe où !)

```
- F B800:1E0 L A0 6E
```

Remplit la plage mémoire qui part de l'adresse B800 :01E0 et qui a une longueur de 160 bytes avec le ode 6E. (Vous devriez voir la quatrième ligne de votre écran se remplir avec des caractères n en jaune sur fond kaki. L'adresse B8000 était celle du début de la mémoire vidéo à l'époque du DOS. Cette mémoire vidéo est parfaitement simulée par WINDOWS)

```
- F B800:1E0 L A0 41 4E
```

Remplit la quatrième ligne de l'écran avec des A majuscules en jaune sur fond rouge.

G « *Go* » Lance le programme. La commande accepte deux paramètres mais ils sont facultatifs : l'adresse de départ que l'on fait précéder du signe égale et une éventuelle adresse de point d'arrêt.
Exemples :

`- G`	Commande go sans paramètres. Le programme démarre donc depuis l'adresse CS :IP
`- G =100`	Lance le programme à partir de l'adresse `CS:0100`
`- G =100 108`	Lance le programme depuis l'adresse `CS:0100` en plaçant un point d'arrêt à l'adresse `CS:0108`

H « *HEX* » sert de calculatrice hexadécimale. On lui spécifie deux nombres et on obtient la somme et la différence.

```
- H 10 2
0012  000E
-- H 10 20
0030  FFF0
```

I « *Input port* »
Lecture d'un port d'entrée/sortie. Le seul paramètre de la commande indique le numéro du port que l'on désire lire.
`- I  378` → affiche le byte qui est contenu à l'adresse 378 des entrées / sorties.
NB. Cela permet de lire les ports d'entrées mais aussi relire aux adresses des ports de sorties les codes qui y ont été écrits.

L *« Load »*
Chargement en mémoire du fichier dont le nom a déjà été spécifié comme argument de la commande DEBUG ou à l'aide de la commande N
La commande L permet aussi la lecture des secteurs d'un disque sans tenir compte du système de fichier.
Syntaxe : `L` *adresse N°disque N°SecteurDeDépart NombreDeSecteurs*
Exemple : `L 100 2 0 1`
= Charger à l'adresse 100, le secteur n° 0 du disque n° 2 (A=0, …C= 2)
Les registres `BX:CX` donnent sur 32 bits la taille des données chargées en mémoire.

M *« Move »*

Recopie le contenu d'une plage mémoire vers une nouvelle adresse.

```
- M 100 10F 140
```

Copie en DS:0140 le contenu de la plage mémoire allant de `DS:0100` à `DS:010F`

```
- M 100 L 10 140
```

Cette commande a le même effet que la précédente. La seule différence est qu'ici la plage à recopier n'est pas donnée par les adresses de début et de fin (100 et 10F) mais par son adresse de départ et par sa taille (100 et 10)

N *« Name »*

Indiquer le nom du fichier à ouvrir pour le lire ultérieurement avec la commande L (*Load*) ou y écrire avec la commande W (*write*).

O **«** *Output port value »*

Ecriture dans un port de sortie

```
- O 378  FF
```

écrit le code FF dans le port dont l'adresse est 378

Q **«** *Quit »*

Sortir du programme DEBUG.

P « *Proceed »* Exécution du programme pas à pas, un peu de la même manière que la commande T (trace) mais sans entrer dans les routines appelées.
Voir les commandes similaires : T et G

R *« Read »* Commande de visualisation des registres

```
-R
AX=0000  BX=0000  CX=0021  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=150F  ES=150F  SS=150F  CS=150F  IP=0100   NV UP EI PL NZ NA PO NC
150F:0100 41            INC     CX
```

Le registre CS «*Code Segment*» contient l'adresse de base du segment où se trouve le code du programme que l'on teste avec DEBUG. C'est dans l'exemple ci-dessus le segment `150F`. Le registre IP est le registre d'instruction. Il contient ici la valeur `0100`.
La première instruction du programme est donc à l'adresse `150F:0100` ce qui équivaut à l'adresse physique `151F0`. (Segment x 1OH + Offset)

Remarquez que si debug vient d'être appelé avec comme argument le nom d'un fichier, les registres BX :CX contiennent la taille du fichier chargé. 21H dans l'exemple ci-dessus.

La commande R permet aussi de modifier un registre mais il faut alors préciser lequel.

```
-R CX          demander l'affichage du registre CX
CX 0021  CX contient la valeur hexadécimale 0021
:ABCD          les ':' sont une invite à la modification
-
```

T « *Trace »* Exécution des instructions pas à pas en affichant après chacune d'elles l'état des registres et le code de l'instruction suivante.

Exemples :

```
- T          Exécute et trace  l'instruction qui se trouve à l'adresse CS:IP
AX=1234  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=150F  ES=150F  SS=150F  CS=150F  IP=0103   NV UP EI PL ZR NA PE NC
150F:0103 BB7856        MOV     BX,5678
-
```

La commande T accepte deux paramètres :

- l'adresse de départ à faire précéder du signe égale (ex. T =100)
- le nombre d'instruction à tracer (ex. T 3)

Ces paramètre peuvent s'utiliser simultanément (ex. T =100 3)

Commandes similaires : G et P

U « *Unassemble* » Commande de "désassemblage". Affiche le contenu de la mémoire en utilisant les mnémoniques du langage assembleur pour afficher les instructions.

Exemples :

```
-U 100 106      Demande l'affichage des instructions comprises entre les
                adresses CS:0100 et CS:0106
150F:0100 MOV AX,1234
150F:0103 MOV BX,5678
150F:0106 ADD AX,BX
```

Syntaxe : U [plage]

Commande associée : A

W *« write »* Ecriture sur le disque.

Le nom du fichier à écrire est soit le nom du fichier chargé par la commande debug soit un nom donné par la commande N.

La taille de la zone mémoire à enregistrer sur le disque peut être donnée en 32 bits via les registres BX:CX.

L'adresse de départ de la zone mémoire est CS :100 par défaut à moins que vous ne précisiez une autre.

La commande W permet aussi l'écriture d'un disque en précisant uniquement le ou les secteurs à écrire. Inutile de préciser qu'il vaut mieux savoir ce que l'on fait avant d'écrire une telle commande sinon c'est la scoumoune !!!

Syntaxe : `W [adresse] [lecteur] [secteur] [nbr]`