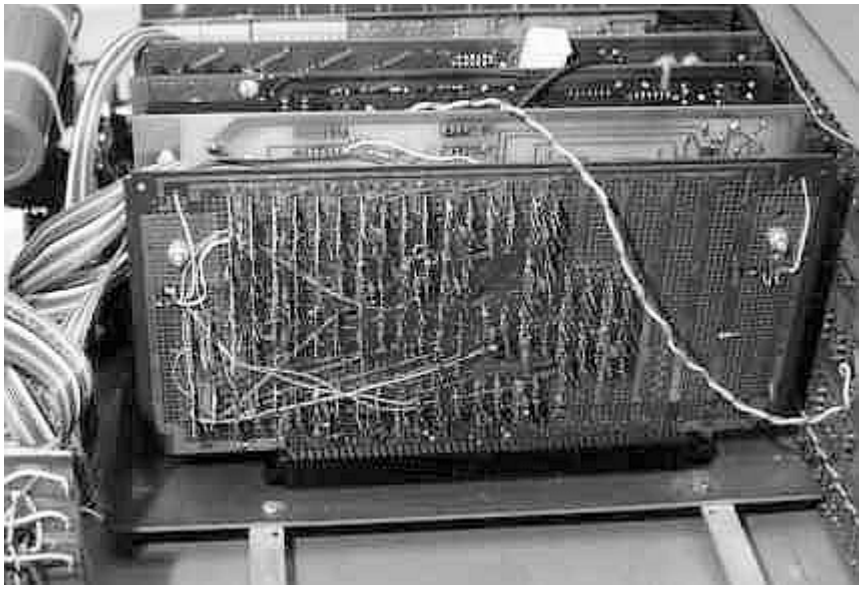
L'ASSEMBLEUR

SUR PC



LE PROCESSEUR DU PC

Les registres

On ne s'intéressera qu'aux registres de 16 bits :

La mémoire est divisée en 4 segments de 64K octets chacun désigné par l'un des registres de segment CS , DS , SS et ES.

CS désigne le segment contenant le code du programme c'est donc dans ce segment que sont recherchées les instructions
DS désigne le segment contenant les opérandes
SS désigne le segment associé à la pile dont le sommet est pointé par le registre SP. La pile est utilisée pour les adresses de retour et les paramètres de sous programme ainsi que pour sauvegarder et restituer les contextes lors d'interruptions.
ES désigne un segment supplémentaire de données.

Ces 4 segments peuvent se chevaucher et même être confondus. Les opérandes et les instructions sont référencées par un déplacement à l'intérieur de l'un de ces 4 segments. L'adresse réelle en mémoire est obtenue en ajoutant à ce déplacement le contenu du registre de segment multiplié par 16. Cet artifice permet d'accéder à lM octet (2^{20}) de mémoire.

La mémoire est accessible octet par octet ou par mots de 16 bits. Toutefois l'accès à un mot de 16 bits se fait en 2 transferts lorsque l'adresse du mot est impaire. Il est donc utile de veiller à "l'alignement" en adresses paires des opérandes 16 bits si l'on ne veut pas voir les performances chuter de façon importante.

AX accumulateur accessible en 2 fois 8 bits AH et AL il sert aux opérations arithmétiques et d'entrées/sorties
DX registre de données accessible en 2 fois 8 bits DH et DL, il est utilisé comme AX et pour les adresses d'entrées/sorties.
CX compteur accessible en 2 fois 8 bits CH et CL, il sert de comptage lors des instructions répétitives.
BX base accessible en 2 fois 8 bits BH et BL, il est registre de base dans le segment de données (DS).
BP registre de base dans le segment de pile (SS).
SI index associé au segment de données (DS).
DI index associé au segment de données (DS). Certaines instructions l'associent au segment ES.
SP pointeur de pile associé au segment de pile (SS).
CO compteur ordinal associé au segment de code (CS)
SR registre d'état contenant les indicateurs de l'UAL (signe, retenue, débordement, résultat nul, parité) ainsi que les bits associés aux interruptions et aux instructions répétitives.

Les opérandes

Les opérandes traités sont les suivants :

- Entiers naturels sur 8 et 16 bits
- Entiers relatifs sur 8 et 16 bits en complément à 2
- Caractères en ASCII sur 8 bits
- Chaînes de caractères de 1 à 64K codes ASCII
- Décimaux en représentation DCB sur 8 bits (1 ou 2 chiffres)
- Pointeurs constitués soit d'un déplacement sur 16 bits soit, sur 32 bits, d'un segment et d'un déplacement.

JEU D'INSTRUCTIONS

Notation : pour simplifier les descriptions ci-dessous on choisira de désigner par :

RG un registre quelconque autre qu'un registre de segment
RS un registre de segment
Val une valeur immédiate
Mem un opérande en mémoire désigné en mode direct ou indirect (voir plus loin pour la désignation des opérandes)

Déplacement de données

MOV dest,source *opération* : dest ← source.

source \ dest	RG	RS	Mem
RG	X	X	X
RS	X		
Mem	X	X	
Val	X		X

XCHG op1 , op2 *opération* : échange op1 et op2.

op2 \ op1	RG	Mem
RG	X	X
Mem	X	

LEA RG , Mem *opération* : RG ← adresse de l'opérande désigné en mémoire.

LDS RG , Mem *opération* : Mem est un opérande sur 2 mots de 16 bits.
Le premier est placé dans RG et le suivant dans DS.

LES RG , Mem *opération* : Mem est un opérande sur 2 mots de 16 bits.
Le premier est placé dans RG et le suivant dans ES.

XLAT *opération* : place dans AL l'octet dont l'adresse est obtenue
en ajoutant à BX le contenu de AL considéré comme un entier non signé.
AL est donc considéré comme un indice de la table désignée par BX et
récupère l'élément correspondant de cette table.

Arithmétique

ADD op1 , op2 *opération* : op1 ← op1 + op2.

ADC op1 , op2 *opération* : op1 ← op1 + op2 + retenue de l'opération précédente.

SUB op1 , op2 *opération* : op1 ← op1 - op2.

SBB op1 , op2 *opération* : op1 ← op1 - op2 - retenue de l'opération précédente.

op2 \ op1	RG	Mem
RG	X	X
Mem	X	
Val	X	X

MUL oper *opération* :
si oper est un octet : AX ← AL * oper
si oper est un mot : (DX ‖ AX) ← AX * op.
L'opérande peut être RG ou Mem et est considéré comme un entier naturel.

IMUL oper *opération* :
si oper est un octet : AX ← AL * oper
si oper est un mot : (DX ‖ AX) ← AX * oper.
L'opérande peut être RG ou Mem et est considéré comme un entier relatif.

DIV oper *opération* :
si oper est un octet : AL ← quotient de AX / oper
AH ← reste de AX / oper
si oper est un mot : AX ← quotient de (DX ‖ AX) /oper
DX ← reste de (DX ‖ AX) /oper
L'opérande peut être RG ou Mem et est considéré comme un entier naturel.

IDIV oper *opération* :

si oper est un octet : AL ← quotient de AX / oper
AH ← reste de AX / oper
si oper est un mot : AX ← quotient de (DX ‖ AX) /oper
DX ← reste de (DX ‖ AX) /oper
L'opérande peut être RG ou Mem et est considéré comme un entier relatif.

INC oper — *opération* : oper ← oper + 1.
L'opérande peut être RG ou Mem.

DEC oper — *opération* : oper ← oper - 1.
L'opérande peut être RG ou Mem.

NEG oper — *opération* : oper ← - (oper).
L'opérande peut être RG ou Mem.

AAA — *opération* : instruction utilisée après une instruction d'addition sur AL ne mettant en jeu que des chiffres inférieurs ou égaux à 9 pour effectuer un ajustement de AX afin que le chiffre des dizaines du résultat soit dans AH et celui des unités dans AL (cette représentation en base 10 permet d'obtenir le nombre sous forme de 2 caractères ASCII par une opération de OU avec 00110000).

DAA — *opération* : instruction utilisée après une instruction d'addition sur AL entre 2 nombres en DCB pour effectuer un ajustement de AL afin que le résultat soit lui aussi en DCB avec le premier chiffre dans les 4 bits de gauche de AL et le second dans les 4 de droite.

AAS — *opération* : comme AAA après une soustraction

DAS — *opération* : comme DAA après une soustraction

AAM — *opération* : instruction utilisée après une instruction de multiplication entre 2 chiffres en DCB dont le résultat (dans AX) ne dépasse pas 99. AAM place dans AH le chiffre des dizaines et dans AL celui des unités (cette représentation en base 10 permet d'obtenir le nombre sous forme de 2 caractères ASCII par une opération de OU avec 00110000)..

AAD — *opération* : met sous forme DCB sur un octet (dizaines dans les 4 bits de gauche, unités dans les 4 bits de droite) un nombre en décimal dont le chiffre des dizaines est dans AH et celui des unités dans AL.

CBW — *opération* : Etend à AX entier l'entier relatif contenu dans AL (passage d'une représentation sur 8 bits à une représentation sur 16 bits).

CWD — *opération* : Etend à DX concaténé à AX entier l'entier relatif contenu dans AX (passage d'une représentation sur 32 bits).

Logique

OR op1 , op2 — *opération* : oper ← oper OU op2.

AND op1 , op2 — *opération* : oper ← oper ET op2.

XOR op1 , op2 — *opération* : oper ← oper OU EXCLUSIF op2.

NOT oper — *opération* : oper ← COMPLEMENT(oper).
L'opérande doit être RG ou Mem.

op2 \ op1	RG	Mem
RG	X	X
Mem	X	
Val	X	X

Comparaison

CMP op1 , op2 — *opération* : compare op1 à op2.

TEST op1 , op2 — *opération* : réalise un ET logique entre op1 et op2 sans produire de résultat autre que celui de positionner les indicateurs permettant de faire un branchement si nul ou non nul.

op2 \ op1	RG	Mem
RG	X	X
Mem	X	
Val	X	X

Décalages

SHR op1 , op2 — *opération* : op1 est décalé op2 fois à droite (décalage logique). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

SHL op1 , op2 *opération* : op1 est décalé op2 fois à gaucheb (décalage logique). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

SAR op1 , op2 *opération* : op1 est décalé op2 fois à droite (décalage arithmétique). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

SAL op1 , op2 *opération* : op1 est décalé op2 fois à gauche (décalage arithmétique). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

ROR op1 , op2 *opération* : op1 est décalé op2 fois à droite (décalage cyclique). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

ROL op1 , op2 *opération* : op1 est décalé op2 fois à gauche (décalage cyclique). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

RCL op1 , op2 *opération* : op1 est décalé op2 fois à gauche (décalage cyclique incluant le bit de retenue de registre d'état comme s'il était le bit de poids fort de op1). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

RCR op1 , op2 *opération* : op1 est décalé op2 fois à droite (décalage cyclique incluant le bit de retenue de registre d'état comme s'il était le bit de poids faible de op1). op1 doit être RG ou Mem. op2 doit être le registre CL ou la valeur 1.

Branchements

JMP oper *opération* : branchement inconditionnel.
oper peut être :
- une étiquette (instruction où aller)
- Mem (variable qui contient l'adresse où aller)
- RG (registre qui contient l'adresse où aller).

Jx oper *opération* : branchement conditionnel.
oper doit être une étiquette (instruction où aller).

Se référer à la table ci-dessous pour les différentes conditions de branchement :

condition	comparaison d'entiers naturels	comparaison d'entiers relatifs
égal	JE ou JZ	JE ou JZ
différent	JNE ou JNZ	JNE ou JNZ
inférieur	JB ou JNAE	JL ou JNGE
inférieur ou égal	JBE ou JNA	JLE ou JNG
supérieur	JA ou JNBE	JG ou JNLE
supérieur ou égal	JAE ou JNB	JGE ou JNL
débordement	JC	JO
non débordement	JNC	JNO
positif		JNS
négatif		JS

JP ou JPE oper *opération* : branchement si parité paire (la parité est évaluéepar une instruction TEST). oper doit être une étiquette (instruction où aller).

JNP ou JPO oper *opération* : branchement si parité impaire (la parité est évaluée par une instruction TEST). oper doit être une étiquette (instruction où aller).

JCXZ oper *opération* : branchement si CX est nul. oper doit être une étiquette (instruction où aller).

Les instructions qui suivent permettent de programmer des boucles avec comptage (registre CX) et une éventuelle condition de sortie autre que la valeur du compteur (résultat d'une opération de comparaison ou de test) :

LOOP	oper	*opération* : décrémente CX et effectue le branchement à l'étiquette oper si CX est non nul.
LOOPE	oper	*opération* : décrémente CX et effectue le branchement à l'étiquette oper si égalité (l'instruction LOOP doit être précédée d'une instruction de comparaison) et si CX est non nul.
LOOPNE	oper	*opération* : décrémente CX et effectue le branchement à l'étiquette oper si non égalité (l'instruction LOOP doit être précédée d'une instruction de comparaison) et si CX est non nul.

Pile

PUSH	oper	*opération* : empile oper. L'opérande peut être RG , RS ou Mem sur 16 bits.
POP	oper	*opération* : dépile oper. L'opérande peut être RG , RS ou Mem sur 16 bits.
PUSHF		*opération* : empile le registre d'état.
POPF		*opération* : dépile le registre d'état.

Sous programmes et interruptions

CALL	oper	*opération* : appel de sous-programme référencé par oper. oper peut être : une étiquette (instruction où aller) Mem (variable qui contient l'adresse où aller) RG (registre qui contient l'adresse où aller).
RET	Val	*opération* : retour de sous-programme. La valeur en opérande est facultative. Si elle est précisée, elle indique le nombre d'octets en sommet de pile à enlever (ignorer) après le retour. Ceci correspond à des paramètres mis dans la pile avant l'appel du sous-programme et que l'on désire enlever à la fin.
INT	Val	*opération* : appel de l'exception de numéro Val Les indicateurs, CS et IP sont empilés puis la procédure l'interruption désignée par Val est lancée.
IRET		*opération* : retour de procédure d'interruption Les indicateurs , CS et IP sont dépilés.
INTO		*opération* : appel de l'exception relative au débordement. Cette instruction est équivalente à INT 4 mais n'est exécutée que si l'indicateur de débordement est positionné.

Instructions de contrôle

WAIT	*opération* : Le processeur s'arrête jusqu'à ce que la ligne physique TEST soit active. Si une interruption physique se produit, elle est prise en compte puis le processeur retourne dans l'état d'attente.
HLT	*opération* : arrêt du processeur jusqu'à l'arrivée d'une interruption ou une réinitialisation.
LOCK	*opération* : Ce n'est pas une vraie instruction mais un préfixe qui est mis avant une instruction. LOCK permet de demander au processeur de vérouiller les accès à ses bus pendant toute la durée de l'instruction qu'il préfixe. Ceci permet la mise en place de mécanismes d'exclusion mutuelle sur des systèmes où plusieurs processeurs peuvent accéder à la mémoire centrale (machines multiprocesseurs, DMA ...).
ESC	*opération* : aucune, permet seulement des échanges avec d'autres processeurs (co-processeurs).

Instructions concernant le registre d'état

LAHF *opération* : transfère le registre d'état dans AH
SAHF *opération* : transfère AH dans le registre d'état

CLC *opération* : mise à zéro de l'indicateur de retenue
STC *opération* : mise à un de l'indicateur de retenue
CMC *opération* : inversion de l'indicateur de retenue
CLD *opération* : mise à zéro de l'indicateur de direction.
Les instructions de traitement de chaînes se feront en incrémentant les index.
STD *opération* : mise à un de l'indicateur de direction.
Les instructions de traitement de chaînes se feront en décrémentant les index.
CLI *opération* : Interdit la prise en compte des interruptions physiques
STI *opération* : Autorise la prise en compte des interruptions physiques

Entrées / Sorties

IN reg,port *opération* : transfère un octet ou un mot selon que reg est AL ou AX depuis un contrôleur de périphérique désigné par son n°. La désignation du contrôleur peut être une valeur (0 à 255) ou le registre DX (0 à 65535).
OUT port,reg *opération* : transfère un octet ou un mot selon que reg est AL ou AX vers un contrôleur de périphérique désigné par son n°. La désignation du contrôleur peut être une valeur (0 à 255) ou le registre DX (0 à 65535).

Traitement de chaînes

LODSB *opération* : place dans AL l'octet pointé par DS:SI puis SI est incrémenté ou décrémenté selon la valeur du bit de direction (voir CLD/STD).
LODSW *opération* : place dans AX le mot de 16 bits pointé par DS:SI puis SI est augmenté ou diminué de 2 selon la valeur du bit de direction (voir CLD/STD).
STOSB *opération* : place AL dans l'octet pointé par ES:DI puis DI est incrémenté ou décrémenté selon la valeur du bit de direction (voir CLD/STD).
STOSW *opération* : place AX dans le mot de 16 bits pointé par ES:DI puis DI est augmenté ou diminué de 2 selon la valeur du bit de direction (voir CLD/STD).
MOVSB *opération* : copie l'octet pointé par DS:SI dans celui pointé par ES:DI puis, SI et DI sont incrémentés ou décrémentés selon la valeur du bit de direction (voir CLD/STD).
MOVSW *opération* : copie le mot de 16 bits pointé par DS:SI dans celui pointé par ES:DI Puis, SI et DI sont augmentés ou diminués de 2 selon la valeur du bit de direction (voir CLD/STD).

Remarque : Chacune des instructions ci-dessus peut être précédée du préfixe REP qui repète CX fois l'instruction. Après chaque exécution de l'instruction, CX est décrémenté et la répétition cesse lorsqu'il est nul.
Ainsi REP MOVSB copie la zone de CX octets pointée par DS:SI vers la zone pointée par ES:DI. Aprés chaque transfert d'octet, SI et DI sont incrémentés ou décrémentés selon la valeur du bit de direction.

SCASB *opération* : compare AL à l'octet pointé par ES:DI puis SI est incrémenté ou décrémenté selon la valeur du bit de direction (voir CLD/STD).
SCASW *opération* : compare AX au mot de 16 bits pointé par ES:DI puis SI est augmenté ou diminué de 2 selon la valeur du bit de direction (voir CLD/STD).
CMPSB *opération* : compare l'octet pointé par ES:DI à celui pointé par DS:SI puis, SI et DI sont incrémentés ou décrémentés selon la valeur du bit de direction (voir CLD/STD).
CMPSW *opération* : compare le mot de 16 bits pointé par ES:DI à

celui pointé par DS:SI Puis, SI et DI sont augmentés ou diminués de 2 selon la valeur du bit de direction (voir CLD/STD).

Remarque : Chacune des instructions ci-dessus peut être précédée du préfixe REPE ou du préfixe REPNE:

REPE répète l'instruction tant qu'il y a égalité. Après chaque exécution de l'instruction, CX est décrémenté et la répétition cesse lorsqu'il est nul ou lorsque la condition d'égalité n'est pas remplie.

REPNE répète l'instruction tant qu'il n'y a pas égalité. Après chaque exécution de l'instruction, CX est décrémenté et la répétition cesse lorsqu'il est nul ou lorsque la condition d'égalité est remplie.

Ainsi REPE CMPSB compare la zone de CX octets pointée par DS:SI à la zone pointée par ES:DI. Après chaque comparaison d'octet, SI et DI sont incrémentés ou décrémentés selon la valeur du bit de direction La comparaison s'arrête lorsque CX est nul (fin de zone atteinte sans rencontrer de différence) ou dès la première différence rencontrée (DS:SI et ES:DI pointent sur l'octet qui constitue cette différence)

L'ASSEMBLEUR

La syntaxe d'une ligne d'assembleur est :

[etiquette **:**] COP [operande1 [**,** operande2]] [**;** commentaire]
ou **;** commentaire

Désignation des opérandes dans les instructions autres que de branchement :

Registre : On met le nom du registre qui contient l'opérande
exemple : MOV ds,ax

Valeur immédiate : On met la valeur de l'opérande selon la forme :

décimal	123	-4	
hexadécimal	1ah	0ffa3h	
binaire	01110010b		
ASCII	'a'	'Bonjour'	

exemples : MOV ax,0ff03h
MOV ax,'Aa'

Remarque : afin qu'il ne puisse pas y avoir de confusion entre un nom de variable et une constante exprimée en hexadécimal, il est nécessaire qu'une constante commence par un caractère numérique. Ainsi, 1Ah est correct alors que B80h doit s'écrire 0B80h.

Direct : On met le nom de la variable
exemples : MOV ax,var1
MOV var2,-5

Indirect : 4 types sont possibles :

- par registre : On met le nom du registre entre crochets.
[registre] désigne l'objet pointé par ce registre le registre ne peut être que BX BP SI ou DI
exemples : MOV ax,[bx]
MOV [si],12h

- par base : On met le nom du registre de base suivi d'un signe + ou - et de la valeur du déplacement entre crochets.

[base+déplacement] désigne l'objet pointé par ce registre de base augmenté du déplacement. Le registre de base ne peut être que BX ou BP.
Le déplacement est un entier relatif sur 16 bits ou un nom de variable. Dans ce dernier cas l'élément désigné est le $BX^{ème}$ octet de cette variable.

exemples :
```
MOV ax,[bp+5]
MOV [bx+var],12h
```

- par index : On met le nom la variable suivi d'un signe + et du nom du registre d'index
[variable+index] désigne le $index^{ème}$ octet de la variable désignée.
Le registre index ne peut être que SI ou DI.

exemples :
```
MOV ax,[var+si]
MOV [var+di],01110001b
```

- par base et index : C'est la réunion des 2 cas précédents.
[base+variable+index] désigne le $base+index^{ème}$ octet de la variable.
[base+index+déplacement] désigne l'objet pointé par ce registre de base augmenté du registre d'index et du déplacement.
Le registre d'index ne peut être que SI ou DI
Le registre de base ne peut être que BP ou BX.
Le déplacement est un entier relatif sur 16 bits.

exemples :
```
MOV ax,[bx+var+si]
MOV [bx+di-300],01110001b
```

Remarque : Le segment utilisé est toujours DS sauf lorsque le registre BP est mentionné. Dans ce cas c'est le segment SS qui est pris en considération.
On peut toutefois imposer l'utilisation du segment de son choix en faisant précéder l'opérande de **RS:**

exemples :
```
MOV ax,ES:[bx+si]
```

Désignation des opérandes dans les instructions de branchement :

Direct : On met l'étiquette qui désigne l'instruction à laquelle on doit aller.

exemple :
```
JNE debut
```

C'est le seul mode utilisable avec les branchements conditionnels.

Registre : On met le nom du registre qui contient l'adresse de l'instruction à laquelle on doit aller.
Les registres utilisables sont AX BX CX DX BP SI et DI

exemple :
```
JMP ax
```

Indirect : Les 4 types décrits ci-dessus sont possibles.
L'opérande en mémoire désigné contient l'adresse de l'instruction à laquelle on doit aller.

exemples :
```
CALL [bx+si+2]
JMP [var+si]
```

Taille des opérandes :

Dans certains cas la taille des opérandes ne peut pas être déduite de l'instruction et il est nécessaire de l'indiquer à l'assembleur par la directive PTR de la façon suivante :
taille **PTR** [opérande] où taille désigne l'un des mots clés :
BYTE WORD DWORD ou **QWORD** (1,2,4 ou 8 octets)

exemples :
```
        MOV  ax,[bx]   ; on sait que bx pointe sur un mot puisque l'on fait référence à ax.
mais    MOV  [bx],0    ; on ne peut pas savoir si l'on doit mettre à 0 un octet, un mot ...
on fera alors:
        MOV  BYTE PTR [bx],0      ; si c'est un octet
        MOV  WORD PTR [bx],0      ; si c'est un mot etc.
```

La segmentation

Le segment de pile est défini par la directive **.STACK** taille où taille indique la taille en octets réservée pour la pile.
On définit les autres segments comme suit :

code : **.CODE**
données : **.DATA**

Lors du chargement du programme les registres CS, SS et SP seront initialisés dans les segments de pile et de code.
Par contre, le registre DS doit être initialisé par le programme grâce à la séquence :

```
mov ax,@DATA
mov ds,ax
```

placée en tout début de programme.

Pour que le programme obtenu fonctionne sous DOS, il faut le commencer par les directives :

DOSSEG
.MODEL small

Constantes et variables

Constantes : Elles sont définies par :

nom **EQU** valeur

Variables : Elles sont déclarées et initialisées par :

nom taille liste de valeurs
ou nom taille nombre DUP (valeur)

Taille est l'un des mots suivants :

- **DB** qui désigne 1 octet
- **DW** qui désigne un mot
- **DD** qui désigne un double mot
- **DQ** qui désigne un quadruple mot

La liste de valeurs est constituée d'une suite d'éléments séparés par des virgules. Chaque élément peut être :

- une valeur numérique
- un nom de variable (on prendra alors l'adresse de cette variable)
- une chaîne de caractères délimitée par des '
- **?** pour indiquer qu'il n'y a pas de valeur initiale

nombre **DUP** désigne le facteur de répétition (nombre de fois valeur)

exemples :

var1 DB 12,0ffh,'ABC'	var1 est sur 5 octets et contient : 12 (0ch) dans le premier octet 0ffh dans le deuxième 65 (code ASCII de A) dans le troisième 66 (code ASCII de B) dans le quatrième 67 (code ASCII de C) dans le cinquième
var2 DW -1,var1,?	var2 est sur 3 mots et contient : -1 (0ffffh) dans le premier mot l'adresse de var1 dans le deuxième n'importe quoi dans le troisième
var3 DB 12 DUP (0)	var3 est sur 12 octets tous initialisés à 0
var4 DW 4 DUP (?)	var4 occupe 4 mots non initialisés.

Procédures

Procédures : Une procédure est définie par :

nom **PROC**
code de la procédure terminé par RET
nom **ENDP**

Squelette d'un programme en assembleur

```
            DOSSEG
            .MODEL small
            .STACK      200h
            .DATA
                        définition des variables et constantes
            .CODE
debut:          mov     ax,@DATA
                mov     ds,ax
            ..... suite du code
                dernière instruction (en général un retour au système)
sp1             PROC
            ..... corps de la procédure 1
sp1             ENDP
.....
spn             PROC
            ..... corps de la procédure n
spn             ENDP
                END     debut
```

La dernière ligne indique à l'assembleur :
que le programme est terminé
que le point de lancement du programme est désigné par l'étiquette debut

Constitution d'un programme executable

1°) Taper le source : nomfich.asm à l'aide d'un éditeur de textes quelconque (EDIT par exemple).

2°) Appeler l'assembleur par la commande :
TASM nomfich
Le suffixe .asm du fichier est sous-entendu.
Un fichier nomfich.obj est obtenu.

3°) Faire l'édition de liens par la commande :
TLINK nomfich
Un fichier exécutable nomfich.exe est obtenu.

Compilation séparée

Il est possible d'écrire des programmes en assembleur constitués de modules compilés séparément. Le lien entre ces modules se fait alors lors de l'édition de liens TLINK liste de noms de fichiers).

Les étiquettes (nom de variable , de constante ou de procédure) contenues dans un module et devant pouvoir être vues depuis un autre seront repérées dans le module où elles sont déclarées par la directive:
PUBLIC etq1 , etq2 , ...

Elles seront réintroduites dans chacun des modules qui les utiliseront par la directive :
EXTRN etq1 : type1 , etq2 : type2 , ...

Les types sont : BYTE , WORD , DWORD , QWORD pour les variables
ABS pour les constantes
PROC pour les procédures

exemple :

; dans le segment de données du module 1 on trouvera
PUBLIC octet,tableau,taille

```
            ; pour que ces variables soient accessibles de l'extérieur
octet       DB    ?
taille      EQU   10
tableau     DW    taille DUP (?)

; dans son segment de code on trouvera :
            PUBLIC      calcul
            ; pour que cette procédure soit accessible de l'extérieur
calcul      PROC
            .....
calcul      ENDP

; dans le segment de données du module 2 on trouvera
            EXTRN       octet:BYTE, tableau:WORD, taille:ABS
            ; pour accéder aux variables du module 1

; dans son segment de code on trouvera
            EXTRN       calcul:PROC
            ; pour utiliser la procédure du module 1
```

UTILISATION EN ASSEMBLEUR DES FONCTIONS DU SYSTEME MDSOS

Appel à des fonctions du système MSDOS :

Elles se font par des appels à des interruptions selon la séquence suivante :

préparation des paramètres d'appel
INT numéro du service demandé
récupération des résultats en retour

Entrées Sorties

Entrées au clavier :

Test de présence d'un caractère au clavier :
```
mov     ah,11
int     21h
```
On récupère le caractère dans al et 0 si aucun caractère n'a été entré.

Saisie d'un caractère avec écho à l'écran :
```
mov     ah,1
int     21h
```
Le caractère est récupéré dans al.

Saisie d'un caractère sans écho à l'écran :
```
mov     ah,8
int     21h
```
Le caractère est récupéré dans al.

Saisie d'une chaîne de caractères :
```
lea     dx,chaine
mov     ah,10
int     21h
```
Le premier octet de la variable chaine doit contenir le nombre maximum de caractères devant être entrés (incluant le caractère retour chariot de fin de saisie).
Un deuxième octet doit être réservé pour que MSDOS puisse y mettre le nombre de caractères réellement entrés.

La variable chaine peut être déclarée de la façon suivante :
```
lgmax   equ     10      ; pour une chaîne d'au plus 10 caractères
chaine  db      lgmax   ; taille maximale de la chaîne
```

```
                db      ?       ; taille réelle mise par MSDOS
                db      lgmax dup (?)  ; place réservée pour les caractères
```

Sorties à l'écran :

D'un caractère :
```
                mov     dl,caractère
                mov     ah,2
                int     21h
```

D'une chaîne de caractères :
```
                lea     dx,chaine
                mov     ah,9
                int     21h
```

La variable chaine contient des caractères codés en ASCII et est terminée par le caractère $. Elle peut par exemple être déclarée de la façon suivante :

```
        chaine  DB      'En voilà une jolie chaîne',0ah,0dh,'$'
```

Les 2 valeurs 0a et 0d (line feed et retour chariot) permettent de passer à la ligne lors de l'affichage.

Fonctions d'utilisation de fichiers

MSDOS offre, grâce aux fonctions de l'interruption 21h, la possibilité de manipuler des fichiers.

Création d'un fichier nouveau :
```
                mov     ah,5bh
                mov     cx,attribut
                lea     dx,nom
                int     21h
```

attribut : 0 création d'un fichier normal
2 création d'un fichier caché
10h création d'un répertoire

nom : est une chaîne de caractères contenant le nom du fichier et terminée par 0 (le nom peut être de la forme : A:rep1\rep2\fichier.moi)

On récupère dans ax le descripteur du fichier.
Si une erreur se produit, le bit de retenue est mis à 1 (on peut utiliser une instruction JC après le int 21h)

Ouverture d'un fichier :
```
                mov     ah,3dh
                mov     al,mode
                lea     dx,nom
                int     21h
```

mode : est un octet dont les 4 bits de gauche définissent les règles de partage du fichier et les 4 de droite les règles d'accès.

4 bits de gauche
1 toute autre ouverture de ce fichier est interdite
2 toute autre ouverture de ce fichier en écriture est interdite
3 toute autre ouverture de ce fichier en lecture est interdite
4 autres ouvertures autorisées

4 bits de droite
0 accès en lecture seulement
1 accès en écriture seulement
2 accès en lecture et écriture

nom : est une chaîne de caractères contenant le nom du fichier et terminée par 0 (le nom peut être de la forme : A:rep1\rep2\fichier.moi)

On récupère dans ax le descripteur du fichier.
Si une erreur se produit, le bit de retenue est mis à 1 (on peut utiliser une instruction JC après le int 21h)

Fermeture d'un fichier :
```
                mov     ah,3eh
                mov     bx,descripteur
```

```
                int     21h
```

descripteur : c'est la valeur récupérée lors d'une ouverture ou d'une création du fichier.

Lecture dans un fichier :
```
        mov     ah,3fh
        mov     bx,descripteur
        lea     dx,tampon
        mov     cx,taille
        int     21h
```

descripteur : c'est la valeur récupérée lors d'une ouverture ou d'une création du fichier.
tampon : c'est la zone dans laquelle seront copiées les valeurs lues dans le fichier
taille : Nombre d'octets à lire dans le fichier. Cette taille doit être inférieure ou égale à celle de tampon mais peut dépasser la taille réelle du fichier.

On récupère dans ax le nombre d'octets réellement lus dans le fichier qui peut être inférieur à la valeur demandée si on est arrivé en fin de fichier.
Si une erreur se produit, le bit de retenue est mis à 1 (on peut utiliser une instruction JC après le int 21h)
La lecture se fait à partir du pointeur courant du fichier qui est placé en début de fichier lors de l'ouverture. A chaque lecture ou écriture ce pointeur est déplacé.

Ecriture dans un fichier :
```
        mov     ah,40h
        mov     bx,descripteur
        lea     dx,tampon
        mov     cx,taille
        int     21h
```

descripteur : c'est la valeur récupérée lors d'une ouverture ou d'une création du fichier.
tampon : c'est la zone dans laquelle sont placées les valeurs à écrire dans le fichier
taille : Nombre d'octets à écrire dans le fichier. Cette taille doit être inférieure ou égale à celle de tampon.

On récupère dans ax le nombre d'octets réellement écrits dans le fichier qui peut être inférieur à la valeur demandée si l'unité de stockage est pleine.
Si une erreur se produit, le bit de retenue est mis à 1 (on peut utiliser une instruction JC après le int 21h)
L'écriture se fait à partir du pointeur courant du fichier qui est placé en début de fichier lors de l'ouverture.
A chaque lecture ou écriture ce pointeur est déplacé.

Déplacement du pointeur :
```
        mov     ah,42h
        mov     bx,descripteur
        mov     cx,depl_fort
        mov     dx,depl_faible
        mov     al,methode
        int     21h
```

descripteur : c'est la valeur récupérée lors d'une ouverture ou d'une création du fichier.
depl_fort , depl_faible : constitue un mot de 64 bits considéré comme un entier relatif compris entre $-(2^{63}-1)$ et $2^{63}-1$ qui contient, en nombre d'octets, le déplacement du pointeur de fichier à effectuer.
methode : indique à partir de quel point du fichier doit être mesuré le déplacement du pointeur.
- 0 : Déplacement mesuré à partir du début du fichier. Un déplacement de 0 octets dans ce mode positionne le pointeur en début de fichier.
- 1 : Déplacement mesuré à partir de la position actuelle du pointeur
- 2 : Déplacement mesuré à partir de la fin du fichier. Un déplacement de 0 octets dans ce mode positionne le pointeur en fin de fichier.

Les fichiers textes sont terminés par un caractère spécial, il est donc nécessaire, si l'on veut écrire en fin d'un tel fichier de faire un déplacement de -1 de façon à effacer cette marque de fin de fichier que l'on ré-écrira ensuite.
On récupère dans dx et ax la nouvelle position du pointeur mesurée à partir du début du fichier (dx contient le mot de fort poids et ax celui de faible poids).
Si une erreur se produit, le bit de retenue est mis à 1 (on peut utiliser une instruction JC après le int 21h)

Remarque : la séquence suivante :

```
mov     ah,42h
mov     bx,descripteur
mov     cx,0
mov     dx,0
mov     al,2
int     21h
```

place le pointeur en fin de fichier et retourne dans dx||ax la longueur du fichier.

Effacement d'un fichier :

```
mov     ah,41h
lea     dx,nom
int     21h
```

nom : est une chaîne de caractères contenant le nom du fichier et terminée par 0
(le nom peut être de la forme : A:rep1\rep2\fichier.moi)

Le fichier sera détruit s'il n'est ouvert par aucun programme.
Si une erreur se produit, le bit de retenue est mis à 1 (on peut utiliser une instruction JC après le int 21h)

Retour à MSDOS et fin de programme

Tout programme assembleur dans un environnement MSDOS doit se terminer par la séquence suivante:

```
mov     ah,4ch
int     21h
```

Autres fonctions

Des quantités d'autres fonctions sont accessibles (gestion des disques, de la mémoire, des lignes série, du graphique ...).
Les documentations MSDOS contiennent la description des fonctions accessibles.
Il convient cependant d'être très prudent lors de l'utilisation de certaines d'entre elles car les effets en cas d'erreur peuvent aller du simple 'plantage' du PC à des pertes irrémédiables de données sur disque ou disquettes.

RELATIONS ASSEMBLEUR / C

On peut appeler des fonctions écrites en assembleur depuis un programme écrit en Turbo C (le contraire est aussi possible mais présente peu d'intérêt en pratique).

Règles d'écriture de l'Assembleur et du C

1°) Le programme en assembleur utilise la pile de C

On supprime donc la directive **.STACK taille**

Il n'y a plus d'étiquette après le **END** de fin du programme en assembleur puisque ce programme ne doit pas démarrer mais est constitué de fonctions appelées par C.

2°) Les noms des fonctions devant être connues par C doivent être explicitement indiquées dans le source en assembleur

A l'intérieur du segment de code on déclarera les noms des fonctions devant être vues par C de la façon suivante :

PUBLIC C nom de la fonction

C'est la même directive que pour la compilation séparée de modules écrits en assembleur mais le mot clé C permet d'avoir des étiquettes compatibles avec C.

3°) Les procédures en assembleur ne doivent modifier ni DS ni BP ni SS ni SI.

On prendra soin de sauvegarder leur valeur initiale s'il faut les modifier et de la restituer avant de retourner au C.

4°) Les entêtes des fonctions écrites en assembleur doivent être déclarées dans le programme en C

On déclarera chaque fonction de la façon suivante :

type EXTERN nom(paramètres);

L'entête est exactement celle que l'on aurait mise si on avait écrit la procédure ou la fonction en C. Le mot clé **EXTERN** indique que le code est à rechercher ailleurs que dans le programme en C.

5°) Retour de fonction assembleur

Les fonctions écrites en assembleur doivent utiliser l'instruction de retour **RET**.

La valeur en retour d'une fonction sera placée dans :

AL si la fonction est d'un type codé sur un octet
AX si la fonction est d'un type codé sur un mot
DX||AX si la fonction est d'un type codé sur un double mot
(fort poids dans DX, faible poids dans AX) ou, s'il s'agit d'un pointeur long,
(segment dans DX, adresse dans AX)

Passage des paramètres de C vers l'Assembleur

C place dans la pile les paramètres en commençant par le dernier :
Un paramètre de type octet (caractère, booléen ...) sera placé dans la partie faible poids du mot empilé.

Représentation en mémoire des variables de C :

type	représentation
int ou short	mot de 16 bits signé (WORD)
long	double mot signé (fort poids dans le premier et faible poids dans le second) (DWORD)
char	octet (codage ASCII) (BYTE)
double	quadruple mot (QWORD)
pointeur	mot : contenant l'adresse (WORD)
long double	10 octets
float	double mot (DWORD)
chaîne de caractères	tableau d'octets terminé par 0.

Squelette de source écrit en Assembleur pour être utilisé depuis un programme écrit en C

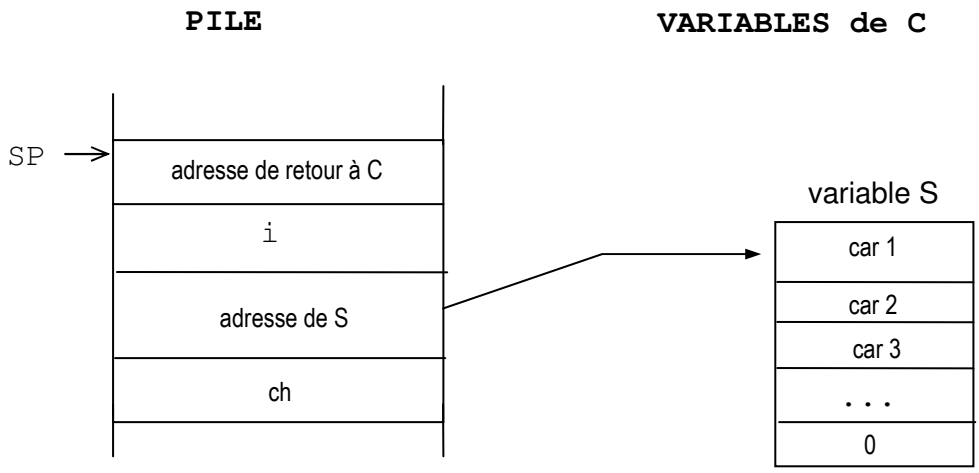
```
        DOSSEG
        .MODEL      small
        .DATA  ; c'est le même segment de données que le programme en C
        ; mettre ici les déclarations de variables et de constantes

        .CODE  ; c'est le même segment de code que le programme en C
        PUBLIC C    truc    ; fonction accessible depuis C
truc    PROC
        push    bp      ; sauvegarder bp
        mov     bp,sp   ; pour pouvoir accéder aux paramètres dans la pile
        ...             ; corps de la procédure
        pop     bp      ; restituer le bp de C
        ret
truc    ENDP

 ; autres procédures

        END
```

Exemple de pile après appel d'une fonction par C

Supposons avoir défini la fonction suivante :

```
int extern test (int i, char *s, char ch);
```

Lorsque cette fonction sera appelée par le programme en C elle trouvera dans la pile les éléments suivants :



La fonction en assembleur commençant par une instruction de sauvegarde de BP (PUSH BP) puis une initialisation de BP à partir du pointeur de pile
(MOV BP,SP), la désignation des paramètres dans l'assembleur se fera de la façon suivante :

[BP+4] désigne le mot contenant la valeur de i
[BP+6] désigne le pointeur permettant d'accéder à s
[BP+8] désigne le mot contenant en faible poids la valeur de ch

La fonction en assembleur se terminera par RET et la valeur entière à retourner sera placée dans AX.

Compilation

1°) Pour chaque fichier contenant des fonctions en assembleur, la commande à utiliser est :

Z:\TASM\TASM /mx *aaa.asm*

dans laquelle *aaa.asm* est le nom du fichier source en assembleur et Z celui du disque qui contient TASM.

Cette commande générera un fichier objet appelé *aaa.obj*.

2°) Pour le fichier contenant le programme en C, la commande à utiliser est :

TCC -r- -IZ:\turboc\include -LZ:\turboc\lib -e*xxx aaa.c b1.obj b2.obj .... bn.obj*

dans laquelle :

Z est le nom du disque qui contient turboc
xxx est le nom du fichier exécutable à obtenir
aaa.c est le nom du fichier source en C
b1.obj, b2.obj ... bn.obj sont des programmes objets obtenus par la commande TASM à partir de sources en assembleur (voir 1°))

UTILISATION DE DEBUG POUR EXECUTER UN PROGRAMME SOUS CONTROLE

Présentation :

L'utilitaire MSDOS DEBUG permet de :

- charger un programme exécutable dans la mémoire
- lire les instructions d'un programme et les modifier
- voir le contenu des registres du processeur et les modifier
- voir et modifier le contenu de la mémoire
- exécuter un programme en pas à pas ou en continu

Ce type d'utilitaire qui existe sur la plupart des machines permet de tester et de mettre au point des programmes écrits en assembleur ou dont on ne dispose pas du code source.

Appel :

DEBUG nomfich.exe

Commandes :

Les commandes de DEBUG sont constituées d'une lettre suivie d'éventuels paramètres. DEBUG ne distingue pas majuscules et minuscules de sorte qu'une commande de la forme D 0 L 4 peut indifféremment s'écrire d 0 l 4.

Paramètres des commandes : on distingue 3 types de paramètres essentiels :

octet : constitué de 2 chiffres hexadécimaux
valeur : constituée de 1 à 4 chiffres hexadécimaux
adresse : elle peut revêtir 3 formes :

valeur:valeur	la première valeur désigne le segment, la seconde l'adresse dans ce segment. Exemple b800:4 désigne l'adresse 4 du segment b800 (soit l'adresse réelle en mémoire B8004)
regseg:valeur	regseg est l'un des registres de segment (CS DS SS ou ES), la valeur désigne l'adresse dans ce segment. Exemple ds:4 désigne l'adresse 4 du segment pointé par ds
valeur	La valeur désigne une adresse dans un segment. Ce segment dépend de la commande (CS pour les commandes relatives au programme et DS pour les autres).

Commandes relatives à la mémoire :

Affichage D adresse [L valeur]

Permet d'afficher en hexadécimal le nombre d'octets spécifié par la valeur qui suit le L contenus dans la mémoire à partir de l'adresse indiquée. Par défaut le segment choisi sera ds.

Si l'on omet le second paramètre, 128 octets seront affichés.

exemple : La commande d 15 L 6 provoquera un affichage de la forme :

2D11:0010 29 1B 28-0A 1A 01

On lit de gauche à droite l'adresse du segment désigné par DS, l'adresse du bloc de 16 octets contenant le 1er octet affiché puis le contenu en hexadécimal des octets demandés.

Modification E adresse

Affiche en hexadécimal le contenu de l'octet désigné par l'adresse. Par défaut le segment choisi sera ds. On peut ensuite :

entrer une nouvelle valeur pour cet octet
passer à l'octet d'adresse suivante (touche espace)
passer à l'octet d'adresse précédente (touche -)
arrêter (touche Entrée)

Commande relative aux registres:

R [nom_de _registre]
Affiche en hexadécimal le contenu du registre du processeur désigné. On peut alors:

entrer une nouvelle valeur pour ce registre
ne pas modifier ce registre (touche Entrée)

Si le nom de registre n'est pas précisé, tous les registres du processeur sont affichés ainsi que l'instruction actuellement désignée par le compteur ordinal (IP) mais aucune modification n'est possible.

Exemple : La commande r proquera un affichage de la forme :

```
AX=0480  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000
DI=0000  DS=2D1   ES=A411  SS=132D  CS=2AEE  IP=0100
 NV UP EI PL NZ NA PO NC
2AEE:0100 96        XCHG  SI,AX
```

Les indicateurs du registre d'état sont affichés sous la forme de 2 caractères dont la signification est la suivante :

OV débordement des entires relatifs	NV non débordement des ER
NG signe négatif	PL signe positif
ZR égalité	NZ non égalité
PE parité paire	PO parité impaire
CY débordement des entiers naturels	NC non débordement des EN

DN et UP sont utilisés dans les instructions répétitives
EI et DI sont utilisés par les interruptions physiques
AC et NA sont des indicateurs de retenue auxiliaire utilisés par quelques instructions

Commandes relatives à l'exécution du programme :

Affichage U adrdeb [adrfin]
Permet d'afficher en clair le morceau de programme qui commence à l'adresse indiquée par adrdeb et qui va jusqu'à celle indiquée par adrfin. Par défaut le segment choisi sera cs. Si le paramètre adrfin n'est pas spécifié, 32 octets de programme seront pris en compte par la commande.
Il faut remarquer que l'instruction JE (branchement si égalité) est appelée JZ alors que JNE (branchement si non égalité) est appelée JNZ

Exemple : La commande : u 0 9 provoquera un affichage de la forme :

```
2AEE:0000 B8F22A MOV   AX,2AF2
2AEE:0003 8ED8   MOV   DS,AX
2AEE:0005 A00000 MOV   AL,[0000]
2AEE:0008 26             ES:
2AEE:0009 A22C07 MOV   [072C],AL
```

On distingue de gauche à droite :

l'adresse du segment (contenu de CS)
l'adresse de l'instruction
le codage de l'instruction (COP et opérandes)
l'instruction en clair

On remarquera qu'une instruction MOV ES:AL,var est affichée comme étant composée d'un préfixe sur un octet ES: suivi d'une instruction normale
MOV AL,var

Ecriture A adr

Permet d'écrire des instructions qui seront rangées dans la mémoire à partir de l'adresse désignée. Par défaut le segment choisi sera cs.
Les instructions s'écrivent selon la syntaxe de l'assembleur processeur et les valeurs numériques doivent être exprimées en hexadécimal.
Il faut toutefois utiliser une syntaxe particulière pour l'écriture des opérandes placés dans un segment non implicite. Ainsi, on écrira MOV ax,[124A] pour accéder à l'opérande placé à l'adresse 124Ah du segment désigné par DS mais il faudra écrire ES: puis MOV ax,[124A] pour accéder à l'opérande placé à cette même adresse dans le segment désigné par ES (voie exemple ci-dessous).

Exemple : la commande A 10 permettra la saisie d'instructions sous la forme:

2AEE:0010 es:
2AEE:0011 mov ax,[5b85]
2AEE:0014 <Entrée>

Les caractères en gras indiquent ce qui est affiché par DEBUG.

Lancement du programme G =adresse

Lance l'exécution du programme à partir de l'adresse précisée. Le segment est CS.

Pas à pasT

Exécute l'instruction courante puis s'arrête en affichant les registres comme le fait la commande R.

Trace P nbr

Exécute nbr instructions à partir de l'instruction courante (pointée par IP) dans le segment cs comme si l'on effectuait nbr fois la commande T

Sortie de DEBUG :

Q