

Le Microprocesseur 80 X86

Le microprocesseur **80 X86** a été créé par la société **INTEL**.

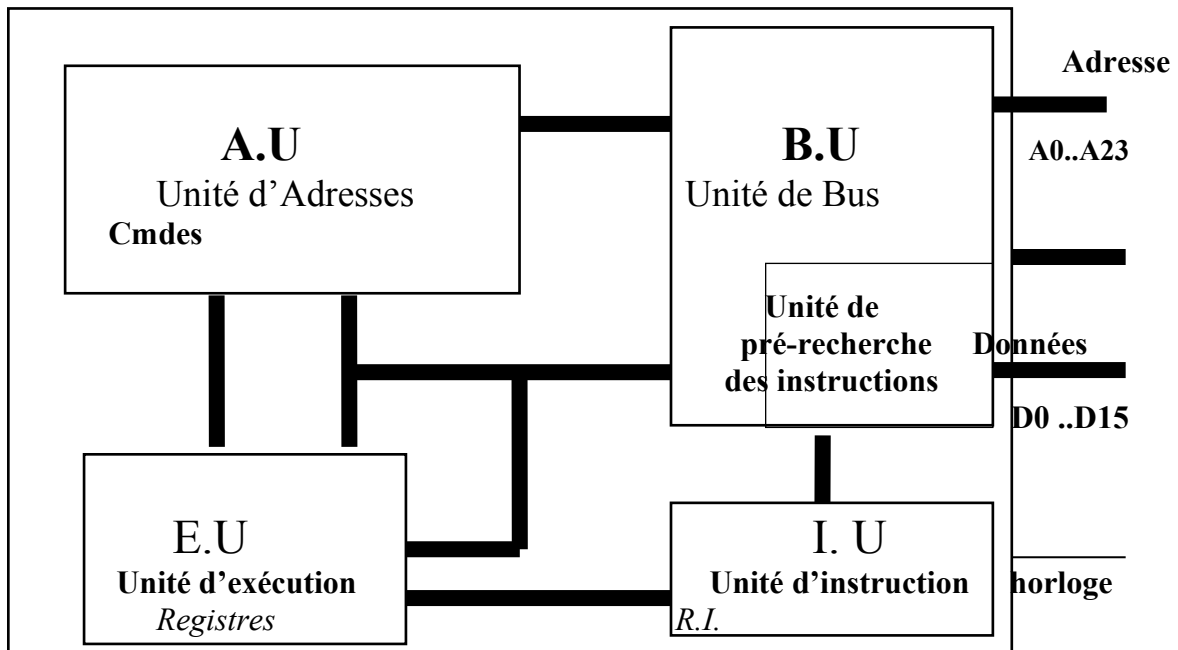
Le premier microprocesseur est un quatre (4) bits : **4004** a été lancé en **1971**, ensuite a donné naissance aux huit bits **8080** et **8085**.

La micro-informatique a pris une grande dimension à la création du premier seize (16) bits **80 86**, suivi du **80 88** (un 16 bits ayant un bus de donnée de 8 bits).

Ensuite le **80286** qui aussi un seize (16) bits est apparu au environ de 1985. Il est plus puissant que les microprocesseurs précédents. Sa particularité, en plus du mode réel qui assure la compatibilité des microprocesseurs précédent, qui est par défaut, un nouveau mode est introduit: c'est le **mode protégé**.

1°)- Architecture

le 80 X86 est structure en quatre unités fonctionnelles travaillant en parallèle :



SCHEMA FONCTIONNEL DU MICROPROCESSEUR

Le Fonctionnement :

- L'Unité de Bus gère toutes les opérations relatives aux Bus. Elle engendre les signaux nécessaires pour les adresses, les données et les commandes, et contrôle également l'organisation du système lorsque des processus ou des extensions sont connectées aux bus. Lorsqu'elle n'est pas occupée, pré-recherche les instructions suivantes dans la mémoire et les introduit dans une file d'attente.
- L'Unité d'adresse gère la mémoire, les protections et calcule les adresses physiques à partir des adresses logiques.

- L'Unité d'instruction reçoit les instructions de la file d'attente de L'Unité de Bus, les décode et place le résultat dans une autre file d'attente de l'Unité d'Exécution.
- L'Unité d'Exécution les exécute et utilise les bus pour transférer des données vers les mémoires ou les entrées/sorties.

2°)- Les Registres internes :

Un registre est:

- mémoire spécialisée
- de capacité limitée
- affectée à des fonctions particulières pour concourir à l'exécution des opérations dans un traitement.

Lorsqu'on *programme*, on se sert des registres et ce de façon *implicite* ou *explicite*.

2.1- Registres généraux

Huit (8) Registres généraux sur huit (8) bits chacun.

Sauvegarde des informations utilisées fréquemment ou des résultats intermédiaires.

ces registres sont:

AL	AH
BL	BH
CL	CH
DL	DH

Combinés deux à deux, ils donnent quatre (4) Registres généraux de seize (16) bits.

AH	AL		AX
BH	BL		BX
CH	CL		CX
DH	DL		DX

En plus d'être général, les registres de 16 bits ont des rôles particuliers avec certaines opérations.

- **Le registre AX** : L'Accumulateur pour les opérations arithmétiques (multiplication et division ...) et d'entrées/sorties.
- **Le registre BX** : Sert aussi de Registre de Base dans le calcul de l'adresse effective d'un objet en mémoire.
- **Le registre CX** : Sert de Compteur (boucle).
- **Le registre DX** : Extension de l'accumulateur à 32 bits, intervient toujours dans les instructions de multiplication et division et les opérations d'entrées/sorties.

2.2- Registres qui servent essentiellement à calculer des adresses en mémoire (16 bits)

- 2 Registres d'index :

SI	Source index
DI	Destination index

parcourir des tables ou manipuler des chaînes de caractères.

- 2 Registres pointeurs de la pile :

BP Base pointer : registre d'accès aux éléments de la pile

SP Stack pointer : pointeur du sommet de pile LIFO (Last In First Out)

- 1 registre pointeur programme :

IP Instruction Program : pointeur de l'instruction courante. Il ne peut être modifié que par le séquenceur.

2.3- 4 Registres Sélecteurs de Segments :

Chaque registre donne l'adresse mémoire du segment associé.

- **CS** Sélecteur Segment de Code (Code Segment)
- **DS** Sélecteur Segment de données (Data Segment)
- **SS** Sélecteur Segment de Pile (Stack Segment)
- **ES** Sélecteur Segment supplémentaire (Extra Segment)

2.4- Registre d'état :

	NT	IOPL		OF	DF	IF	TF	SF	ZF		AF		PF		CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

pos. du bit	parfois marqué	Nom	fonction
0	C	CF	Mis à 1 si retenue, sinon 0
2	P	PF	Parité: mis à 1 si parité pair
4	A	AF	Retenue sur quarté de faible poids du registre AL
6	Z	ZF	Indicateur de zéro : 1 si résultat nul
7	S	SF	signe : identique au premier bit du fort poids
8	T	TF	si à 1 : déclenche un IT après l'instruction suivante puis revient à 0
9	I	IF	autorisation d'ITs masquables
10	D	DF	1 décrementés, 0 incrémentés les index visés
11	O	OF	Dépassement sur résultat signé, mise alors à 1
12/13		IOPL	niveau de privilège d'E/S (mode protégé)
14		NT	Emboîtements des taches (mode protégé)

3°) ORGANISATION DE LA MEMOIRE ET LE CALCUL D'ADRESSE

- La plus petite unité adressable est l'octet.
- En mode réel, la taille de la mémoire est de 2^{20} structurée en segment.
- Chaque segment peut avoir une longueur maximale de 2^{16} et commence à une adresse physique (adresse réelle) multiple de 16.
- Un emplacement mémoire est spécifié par une adresse logique qui est constituée d'un numéro de segment et du déplacement dans ce segment.
- L'adresse logique est transformée en adresse effective (adresse physique) par l'Unité d'Adresse de la manière suivante :

Adresse physique = numéro de segment * 16 + déplacement

Remarque:

- Un emplacement mémoire a plusieurs adresses logiques mais une seule adresse physique (il appartient à plusieurs segments).

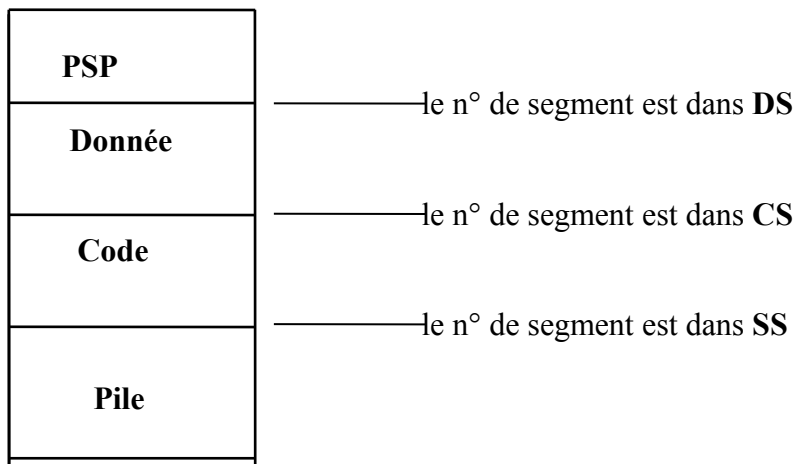
Exemple le 17^{ième} octet du 1^{er} segment est le 1^{er} octet du 2^{ième} segment

adresse physique	mémoire	numéro de segment	adresse logique et le calcul d'adresse
0		0	
1			
2			
3			
4			
5			
6			
7			
8			
9			
A			
B			
C			
D			
E			
F			
10		1	(0,11) ou (1,1) $\Rightarrow 0*10+11=1*10+1=11$
11			
12			
i*10		i	(i,3) ou (i-1,13) ou (i-2,23) ect... \Rightarrow $i*10+3=(i-1)*10+13=(i-2)*10+23$
i*10+1			
i*10+2			
i*10+3			

4°) STRUCTURE INTERNE D'UN PROGRAMME

Un programme est constitué de 3 segments et d'une entête et éventuellement d'un segment supplémentaire:

- **entête : PSP** (Préfix status program) : les informations nécessaires et suffisantes pour l'exécution du programme
- **segment de données** : zone mémoire qui contient les données
- **segment de code**: zone mémoire qui contient les instructions
- **segment de pile** : zone mémoire associée aux opérations de pile
- **extra segment** : zone mémoire qui contient des données spécifiques facultatifs



L'ordre des segments est indépendant, selon la définition de l'utilisateur.

Les différents segments peuvent avoir le même numéro de segment en mémoire (le calcul d'adresse physique se fait à partir du même numéro de segment).

5°) FORMAT D'UNE INSTRUCTION MACHINE :

COP	MOD	DEPLACEMENT	OPERANDE IMMEDIAT
-----	-----	-------------	-------------------

Si le COP est toujours présent, les autres champs dépendent de la nature de l'instruction.

5.1 LE CODE OPERATION : COP sur un octet

XXXX XXXX	XXXREGXXX
XXXX XREG	XXXX WREG
XXXX XXDW	

Avec:

D=1 \Rightarrow REG est destinataire

D=0 \Rightarrow REG est source

REG : 3 bits

Et

W=1 \Rightarrow donnée sur 16 bits

W=0 \Rightarrow donnée sur 08 bits

	Registre de 8 bits	Registres de 16 bits
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

	Reg segment
00	ES
01	CS
10	SS
11	DS

5.2 LE MODE D'ADRESSAGE : MOD sur un octet

MOD	REG	REG/M
-----	-----	-------

MOD : sur 2 bits

valeur	SIGNIFICATION
00	Pas de déplacement si REG/M=110 alors adresse effective sur 16 bis
01	déplacement sur 1 octet
10	déplacement sur 2 octets
11	REG/M est registre

REG/M: sur 3 bits

Valeur	MOD=00	MOD=01	MOD=10
000	(BX)+ (SI)	Idem que MOD=00 + déplacement avec extension du signe sur 8 bits	Idem que MOD=00
001	(BX)+ (DI)		+ déplacement
010	(BP)+ (SI)		avec extension
011	(BP)+ (DI)		du signe
100	(SI)		sur 16 bits
101	(DI)		
110	dépl. sur 16 bits	(BP) + dépl. sur 8 bits	(BP) + dépl. sur 16 bits
111	(BX)	(BX) + dépl. sur 8 bits	(BX) + dépl. sur 16 bits

DEPLACEMENT est sur 8 ou 16 bits selon le mode

OPERANDE IMMEDIAT (VALEUR) est sur 8 ou 16 bits selon le second opérande

REMARQUE:

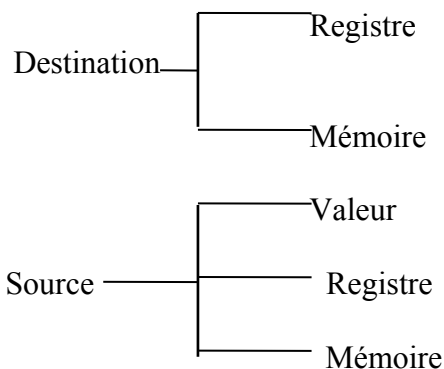
? La longueur de l'instruction est de 7 octets (1 octet de préfixe), 80-386 peut aller jusqu'à 14 octets

? A partir du 80-386 le déplacement et valeur peuvent être sur 32 bits et ceci est indiqué par un octet de préfixe.

6°) SYNTAXE D'UNE INSTRUCTION ASSEMBLEUR:

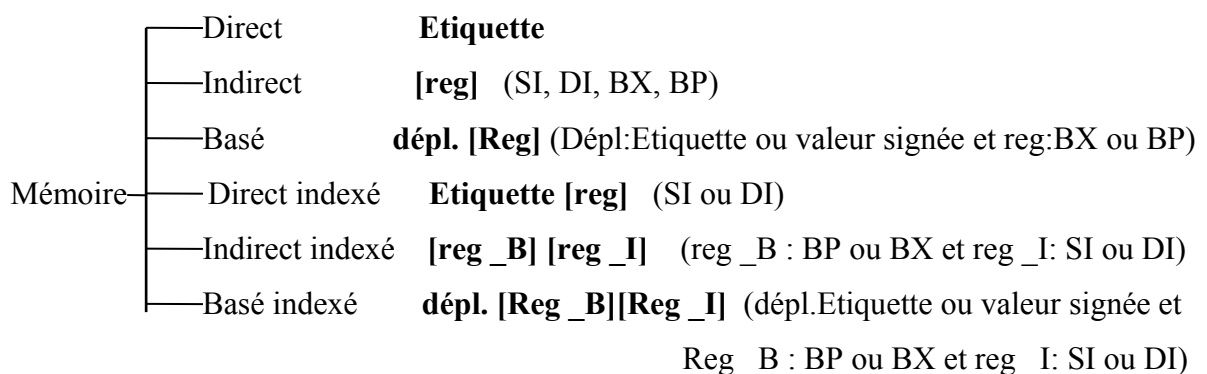
|< Etiquette >| Cmde |< Opérande >|
Avec |< ...>| : Facultatif

Opérande : Destination, |<Source >|



Registre: peut être soit :

- un registre de 8 bits (AL - BH)
- un registre de 16 bits (AX - DI ou registre segment)
- un registre segment (ES - DS)



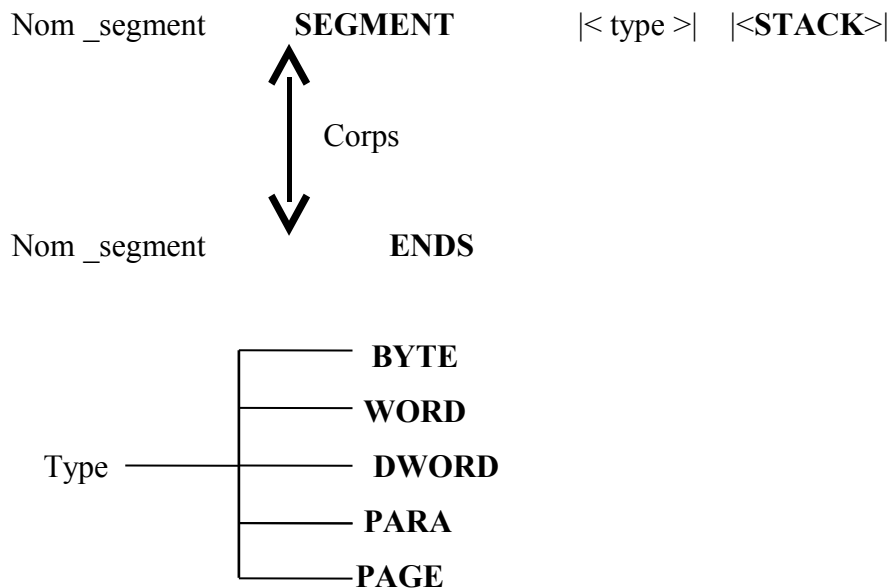
Reg: registre

Reg_B: registre de base

Reg_I: registre d'index

Dépl. : Valeur signée ou Etiquette

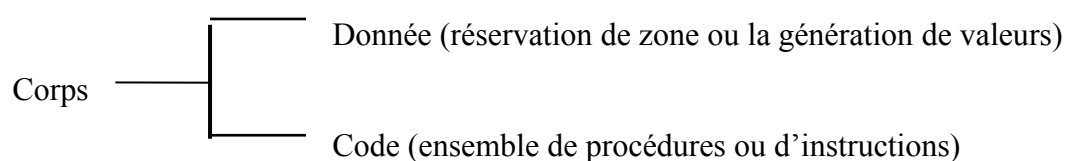
6.1 DEFINITION D'UN SEGMENT :



1. **BYTE** : le segment commence juste au premier octet libre, c.a.d. juste après le segment précédemment défini.
2. **WORD** : le segment commence juste au premier mot libre (adresse physique paire)
3. **DWORD** : le segment commence juste au premier double mot libre (adresse physique multiple de 4)
4. **PARA** : est l'option par défaut, le segment commence au début d'un segment physique (adresse physique est multiple de 16).
5. **PAGE** : le segment commence aune adresse physique multiple de 256

Lorsque le type est **BYTE** ou **WORD** ou **DWORD** le numéro associé à ce segment est le **numéro de segment physique où se trouve le premier octet ou le mot ou le double mot libre.**

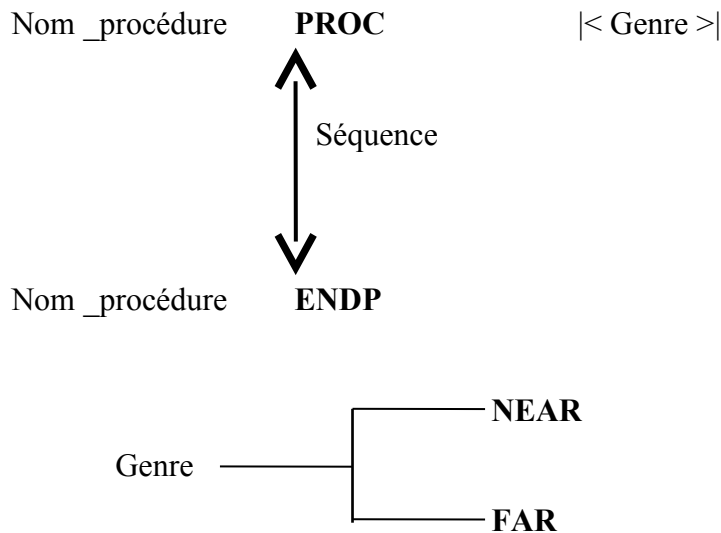
STACK : indique que c'est le segment de pile



REMARQUE :

Dans un segment, il peut y avoir des données et du code en même temps. Mais de préférence, séparer les données du code.

6.2 DEFINITION D'UNE PROCEDURE :



NEAR : la procédure est local au segment, elle doit être dans le même segment que la procédure appelante.

FAR : la procédure peut être dans autre segment que le segment appelant.

Séquence : est une suite d'instructions

6.3 SCHEMA DE PROGRAMME :

Un programme est constitué de trois segments :

- Un segment de PILE
- Un segment de DONNEE
- Un segment de CODE

Le numéro de chaque segment est dans le registre correspondant (donnée dans DS, code dans CS et pile dans SS); mais on peut avoir les 3 segments ayant le même numéro.

6.3.1 STRUCTURE D'UN PROGRAMME

Il est recommandé de structurer tout programme comme suit :

<Segment de pile >
Segment de donnée
Segment de code
END Etiquette

Etiquette : indique au système la première instruction à exécuter pour ce programme.

Remarque :

1. Si le segment de pile est omis, une pile (zone mémoire bien déterminée) est associée au programme ayant une taille de 64K - longueur du programme (donnée + code)
2. on peut avoir la pile, les données et le code dans un seul segment.

7°) FORMAT DES DONNEES:

Les différents types de données utilisées sont :

7.1 NOMBRE :

1. Binaire : est une suite de 0 et de 1 qui se termine par la lettre **B**;
2. Octal : est suite de chiffres de valeur inférieur à 8 qui se termine par la lettre **O**;
3. Décimal : est la base par défaut, est une suite de chiffres décimaux qui se termine par la lettre **D**;
4. Hexadécimal : est suite de chiffres appartenant à l'intervalle [0. . F] qui commence obligatoirement par un chiffre appartenant à l'intervalle [0.. 9] et se termine par la lettre **H**.

Les nombres ayant une longueur supérieur à un octet sont représentés en mémoire comme suit :

- L'octet de faible poids est rangés à l'adresse faible du champs et l'octet de FORT POIDS à l'octet FORTE.

Exemple : La valeur 125H représenté sur un mot mémoire est rangée comme suit

adresse	mémoire
a	25
a+1	01

7.2 CARACTERE :

Est une suite de caractères alphanumérique et caractères spéciaux encadrée par des quotes ('). Chaque caractère est représenté par sa valeur ASCII.

Si le caractère quote est dans la suite, alors elle est doublée.

Registre pour calcul de l'adresse physique :

Référence mémoire	Registre Segment		Offset (Décalage)
	par défaut	remplacement possible	
recherche d'une instruction	CS	Aucun	IP
Opération sur Pile	SS	Aucun	SP
Variables (les exceptions suivent)	DS	CS,SS,ES	adresse effective
Source pour une chaîne	DS	CS,SS,ES	SI
Destination pour une chaîne	ES	Aucun	DI
Registre BP remplaçant le registre de base	SS	CS,DS,ES	adresse effective
Registre BX remplaçant le registre de base	DS	CS,SS,ES	adresse effective

INITIALISATION DES REGISTRES :

Lorsque' on lance l'exécution d'un programme, le DOS lit tout d'abord le PSP et initialise les registres du microprocesseur selon des règles rigoureuses.

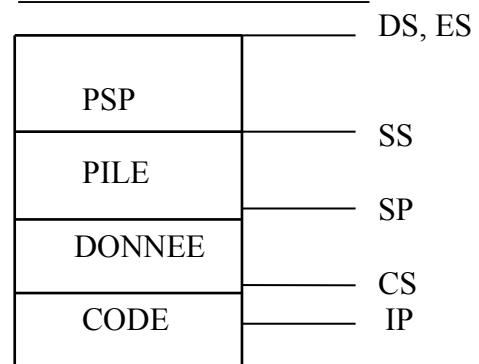
STRUCTURE DU PROGRAMME

Segment de Pile

Segment de données

Segment de code

STRUCTURE INTERNE



STRUCTURE DU PROGRAMME

Segment de données

Segment de code

STRUCTURE INTERNE

PSP	DS, ES
DONNEE	SS
CODE	CS IP
PILE	SP

- Le registre IP contient l'adresse du début d'exécution (valeur associée au symbole qui suit le END).
- Le registre SP contient l'adresse du dernier octet de la pile si elle définie, sinon la valeur 0 ce qui indique le dernier octet du segment (2^{16}) sachant que opération sur le registre SP est décrémenté de 2 qui nous donne $-2 = 0FFFEH$ qui sera pris comme adresse (valeur non signée).
- les registres sélecteurs sont initialisés comme suit :
 - Les Registres DS et ES pointent le segment du PSP.
 - Le Registre CS pointe le segment qui contient la première instruction à exécuter.
 - le registre SS pointe le segment de pile si il est défini (segment de type STACK), sinon le premier segment du programme (juste après le PSP)

Les DIRECTIVES :

Une directive, contrairement à une instruction est interprétée par l'assembleur (traducteur). Elle permet d'introduire des paramètres et de générer des données. Par contre une instruction est traduite pour être exécuté

- **ASSUME** : Affecte un registre segment à un segment.

Syntaxe :

ASSUME Reg_sélecteur:Nom_de_segment|< Reg_selecteur:Nom_de_segment,...>|

Signifie le registre segment spécifié assume la charge de contenir l'adresse (numéro) du départ du segment

- **EQU** : Equivalent

Syntaxe :

Étiquette EQU Expression

Affecte la valeur de l'expression à l'Etiquette.

- **=** « signe égal » sert à redéfinir les constantes. Est semblable à EQU mais permet de redéfinir des affectation en cours de route.

Syntaxe :

Label=Expression

- **EVEN** : pair Impose à IP une adresse paire

Syntaxe :

EVEN

- **ORG** : Origine Spécifie l'adresse de départ du code

Syntaxe :

ORG Expression

- **%out texte** : liste le « texte sur écran pendant l'assemblage, ce qui permet de suivre sa traduction.

- **;** ce qui suit est commentaire jusqu'à la fin de ligne.

- **;;** le commentaire qui suit n'apparaît pas sur dans le listing.

- **.RADIX** Sert à définir une fois pour toute la base de conversion.

Syntaxe :

.RADIX Base

- **END** : indique le du programme à l'assembleur.

Syntaxe :

END |<Etiquette>|

Où Etiquette indique la 1^{ère} instruction exécutable du programme.

- AUTRES :

SEGMENT, ENDS, PARA, WORD, BYTE, PAGE : voir définition de segment

PROC, ENDP, FAR, NEAR : voir définition de procédure

SCHEMA D'UN PROGRAMME :

; Définition du segment de pile

S _pile SEGMENT

DB lgueur DUP (?)

S _pile ENDS

; Définition du segment de données

S _donnée SEGMENT

; Déclarations de valeurs et

; Génération de zone de travail

S _donnée ENDS

; Définition du segment code (programme)

S _code SEGMENT

ASSUME CS: s _code, DS: s _donnée

ASSUME SS: s _pile

Début

PROC ; _____

MOV AX, s _donnée ; initialisation du reg segment DS |

MOV DS, AX ; _____ |

;

; Corps de la procédure principale |

;

MOV AH, 4CH ; _____

INT 21H ; indique la fin du programme |

Début

ENDP ; _____ |

ENDS

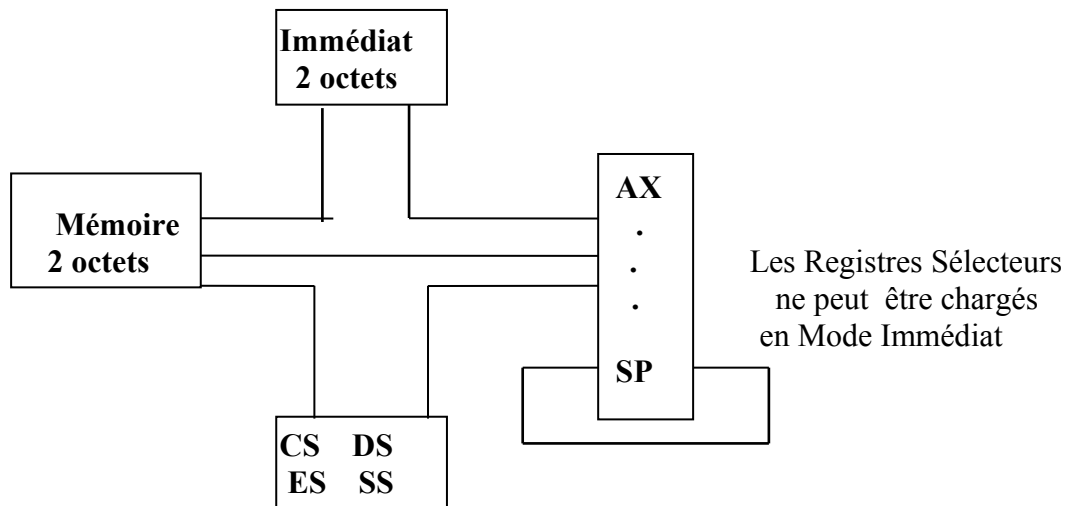
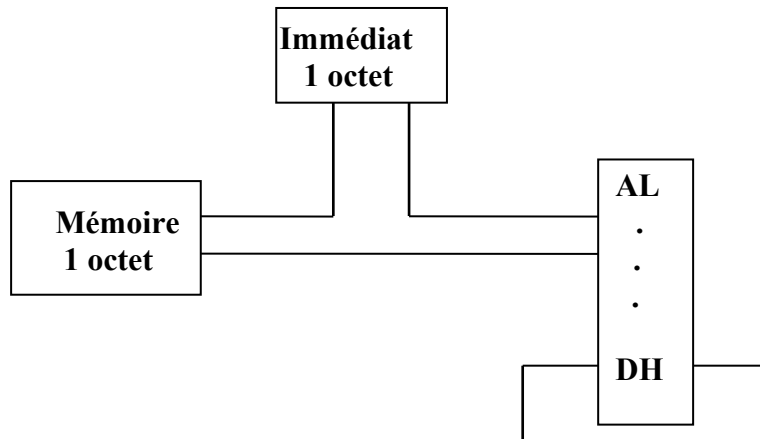
END début

LES INSTRUCTIONS

1°) - Les instructions de transferts

- Instruction MOV**

[étiquette] MOV Destination, Source
Transfère une donnée (1 ou 2 octets (s)) de la **source** vers la **destination**.
Les indicateurs du registre d'état (**F**) : inchangés



- Instruction XCHG :**

[Étiquette] XCHG Destination, Source
Échange, permute les contenus des opérandes source et destination (octet ou mot).

destination et source sont soient

Registre et mémoire
Registre et Registre

Remarque : Sont **Exclus** les registres **Sélecteurs**
les indicateurs du registre d'état (**F**) : inchangés

- **Instruction LEA**

[etiquette] LEA Destination,Source

Destination : Obligatoirement est un registre seize (16) bits
source : Opérande en mémoire

Chargement de l'adresse effective de l'opérande dans le registre.
Adresse effective est le déplacement dans le segment

les indicateurs du registre d'état (**F**) : inchangés

LEA Destination, Source
est équivalent à
MOV Destination, **OFFSET** source

- **Instruction LDS**

[etiquette] LDS Destination, Source

Destination : Obligatoirement est un registre seize (16) bits
source : Opérande en mémoire

Transfert un **pointeur de 32 bits** contenant l'adresse d'un segment et son déplacement
dans **DS** et un **Reg**.

Le pointeur est en mémoire dans double mot.

REG _{16 bits} reçoit le contenu de source et **DS** reçoit le contenu de (source +2)

les indicateurs du registre d'état (**F**) : inchangés

1°) - Les instructions Arithmétiques

Remarque:

Les Opérandes peuvent être aussi bien sur un octet que sur un mot (2 octets)
Les registres Sélecteurs sont exclus pour toutes les opérations arithmétiques

- **Instruction ADD**

[etiquette] ADD Destination,Source

Destination = Destination + Source

- **Instruction ADC**

[etiquette] ADC Destination,Source

$\text{Destination} = \text{Destination} + \text{Source} + \text{CF}$

Destination peut être registre ou mémoire

Source peut être registre ou mémoire ou mémoire

Les indicateurs affectés sont : AF, CF, OF, PF, SF, ZF

- **Instruction SUB**

[etiquette] SUB Destination, Source

$\text{Destination} = \text{Destination} - \text{Source}$

- **Instruction SBB**

[etiquette] SBB Destination, Source

$\text{Destination} = \text{Destination} - \text{Source} - \text{CF}$

La soustraction est identique à ADD et ADC

- **Instruction INC**

[etiquette] INC Destination

$\text{Destination} = \text{Destination} + 1$

- **Instruction DEC**

[etiquette] DEC Destination

$\text{Destination} = \text{Destination} - 1$

- **Instruction NEG**

[etiquette] NEG Destination

$\text{Destination} = 0 - \text{Destination}$

Destination peut être registre ou mémoire

Les indicateurs affectés sont : AF, CF, OF, PF, SF, ZF

3°) - L'instruction de Comparaison :

- **Instruction CMP**

[etiquette] CMP Destination, Source

Comparaison de deux opérandes (1 octet Ou 1 mot)

Destination peut être registre ou mémoire

Source peut être registre ou mémoire ou mémoire

N.B. : **la comparaison se fait par la soustraction de Destination - Source**, mais le résultat n'est pas conservé et les opérandes inchangés

Les indicateurs affectés sont : AF, CF, OF, PF, SF, ZF

— Reg _multiplicande, Valeur _immédiate

Multiplication signé avec le mode immédiat (à partir du 80 286)

Multiplicande : Registre ou mémoire

Reg _résultat ou Reg _Multiplicande : registre 8 ou 16 bits (Registres sélecteurs exclus)

Multiplicande*valeur immédiate, le résultat est dans **Reg _résultat**

Reg_ Multiplicande* valeur immédiate, le résultat est dans **Reg_ Multiplicande**

Indicateurs du Registre F : $OF = CF = 0$ Résultat est correcte

(registre suffit pour le résultat)

$OF = CF = 1$ Résultat est incorrecte

Les Autres : *Indéfinis*

- **Instruction DIV**

[etiquette] DIV Diviseur

Division non signé du contenu de l'accumulateur AX ou DX|AX par le diviseur.

Le quotient est dans AL ou AX et le reste est dans AH ou DX en fonction du Diviseur.

Le Diviseur peut être un registre ou mémoire (les registres sélecteurs sont exclus).

Diviseur 1 octet ——— $AX / Diviseur$ $\left\{ \begin{array}{l} \text{— AL : Quotient} \\ \text{— AH : Reste} \end{array} \right.$

Diviseur 1 mot ——— $DX|AX / Diviseur$ $\left\{ \begin{array}{l} \text{— AX : Quotient} \\ \text{— DX : Reste} \end{array} \right.$

Indicateurs du Registre F modifié et indéfini

Remarque : Une It système est déclenché (arrêt du pgme) si le diviseur est égal à 0 ou le quotient est FFH ou FFFFH

- **Instruction IDIV**

[etiquette] IDIV Diviseur

Division signée du contenu de l'accumulateur AX ou DX|AX par le diviseur.

Le quotient est dans AL ou AX et le reste est dans AH ou DX en fonction du Diviseur.

Le Diviseur peut être un registre ou mémoire (les registres sélecteurs sont exclus).

Diviseur 1 octet ——— $AX / Diviseur$ $\left\{ \begin{array}{l} \text{— AL : Quotient} \\ \text{— AH : Reste} \end{array} \right.$

$\left\{ \begin{array}{l} \text{— AX : Quotient} \end{array} \right.$

Diviseur 1 mot ———— **DX|AX / Diviseur** —**DX** : Reste

Indicateurs du Registre F modifié et indéfini

Remarque : Une It système est déclenché (arrêt du pgme) si le diviseur est égal à 0 ou le quotient est 7FH ou 7FFFH

- **Instruction CBW**

[etiquette] CBW

Conversion d'octet en mot (AL en AX)
si AL contient sxxx xxxx alors AH reçoit ssss ssss

- **Instruction CWD**

[etiquette] CWD

Conversion de mot en double mot (AX en DX|AX)
si AX contient sxxx xxxx xxxx xxxx alors DX reçoit ssss ssss ssss ssss

les indicateurs du registre d'état (**F**) : inchangés

5°) - Les instructions de boucles :

- **Instruction LOOP**

[etiquette] LOOP Cible

CX: contient le nombre d'itération

Fonctionnement : CX ——— CX -1
Si CX # 0 alors aller à Cible
Sinon continuer en séquence

Mode d'adressage est le mode relative court (déplacement est sur un octet signé)

Remarque : le nombre d'octet entre l'instruction qui suit le Loop et l'instruction Cible ne doit pas dépassé 127 octet (déplacement est compris entre -128 et 127).

- **Instruction LOOPE
LOOPZ**

[etiquette] LOOPE Cible
LOOPZ

Identique a LOOP sauf que le branchement a cible se fait sur la condition CX # 0 et ZF=1

- **Instruction LOOPNE
LOOPNZ**

[etiquette] LOOPNE Cible
LOOPNZ

Identique a LOOP sauf que le branchement a cible se fait sur la condition CX # 0 et ZF=0

les indicateurs du registre d'état (**F**) : inchangés

5°) - Les instructions de branchements :

- Les instructions de branchements conditionnels**

Le mode d'adressage des instructions de branchement conditionnels est le mode relatif court (déplacement sur un octet signé). Dans un segment, le déplacement doit être compris entre -128 et +127 à partir de l'adresse de l'instruction qui suit le branchement.

- Instruction *J* Condition**

[etiquette] J_{Condition} Cible

Condition peut être :

A: Above >

B: Below <

E : Equal =

G : Greater >

L: Less <

N_{Condition} : Not_{Condition} Non condition

Instructions conditionnelles	Indicateurs					Commentaires	BRANCHEMENT SI :
	OF	CF	ZF	PF	SF		
JE /JZ	X	X	1	X	X		égal / zéro
JP /JPE	X	X	X	1	X		parité paire
JO	1	X	X	X	X		débordement
JS	X	X	X	X	1		signe est à 1
JNE /JNZ	X	X	0	X	X		non égal / <>0
JNP /JPO	X	X	X	0	X		parité impaire
JNO	0	X	X	X	X		non débordement
JNS	X	X	X	X	0		Signe est à 0
arithmétique signée							
JL /JNGE	A	X	X	X	B	A < B	inférieur /non supérieur ou
JLE /JNG	A	X	1	X	B	ZF=1 OU A < B	inférieur ou égal / non
JNL /JGE	A	X	X	X	B	A = B	non inférieur /supérieur ou
JNLE /JG	A	X	0	X	B	ZF=0 OU A = B	non inférieur ou égal
arithmétique non signé							
JB /JNAE /JC	X	1	X	X	X		inférieur /non supérieur ou
JBE /JNA	X	1	1	X	X	CF =1 OU ZF =1	inférieur ou égal / non
JNB /JAE /JNC	X	0	X	X	X		non inférieur /supérieur ou
JNBE /JA	X	0	0	X	X	CF =0 ET ZF =0	non inférieur ou égal

- Les instructions de branchements inconditionnels**

- Instruction *JMP***

[etiquette] JMP Cible

Cible peut être :

Le mode d'adressage des instructions de branchement inconditionnel est :

- le mode relatif court, le déplacement est compris entre -128 et +127 à partir de l'adresse de l'instruction qui suit le branchement,
- le mode relatif long, le déplacement est compris entre -32768 et +32767 à partir de l'adresse de l'instruction qui suit le branchement,
- Indirecte Registre général de 16 bits,
- indirecte mémoire.

Modes :

- Intra-segment relatif court ou long,
- Inter-segment direct : champs adresse de l'instruction est sur 32 bits 2^{ème} et 3^{ème} octet valeur de IP et 4^{ème} 5^{ème} valeur de CS,
- Inter-segment indirect : mémoire uniquement (pointeur à 2 mots), le premier mot contient la valeur de IP et le second mot contient la valeur de CS.
- Intra-segment indirect : par registre ou mémoire.

6°) - Les instructions de pile :

La pile se trouve dans le segment de Pile où son premier mot est pointé par le pointeur SS:SP.

• Instruction **PUSH**

[etiquette] PUSH Source

Source peut être :

Registre 16 bits,
Registre sélecteur (registre segment),
Mot mémoire,
Valeur immédiate (sur 16 bits).

Destination : Pile

Fonctionnement :

	SP	———	SP-1
	Octet Fort poids Source est empilé à l'emplacement pointé par SS:SP		
	SP	———	SP-1
	Octet Faible poids Source est empilé		

• Instruction **POP**

[etiquette] POP Source

Source : Pile

Destination peut être :

Registre 16 bits,
Registre sélecteur ((registre segment),

Mot mémoire,

Fonctionnement :

	Octet faible poids est dépilé à partir de l'emplacement pointé par SS:SP
	SP ——— SP + 1
	Octet Fort poids est empilé
	SP ——— SP + 1

Le mot se trouvant au sommet de pile est transféré dans la destination, SP est décrémenté de 2.

- **Instruction PUSHA**

[etiquette] PUSHA

Empilement de tous les
registres généraux dans
l'ordre suivants

DI	SP
SI	
BP	
SP	
BX	
DX	
CX	
AX	

- **Instruction POP**

[etiquette] POP

Dépilement : les registres généraux sont restaurés dans l'ordre général:
DI, SI, BP, SP, BX, DX, CX, AX.

Remarque : le registre SP contient la valeur du sommet de pile et **non** la valeur dépilée.

- **Instruction PUSHF**

[etiquette] PUSHF

Sauvegarde du registre d'indicateurs F dans la pile,
SP est décrémenté de 2.

les indicateurs du registre d'état (**F**) : inchangés pour toutes les instructions de piles.

- **Instruction POPF**

[etiquette] POPF

Le registre d'indicateurs F est restauré à partir du sommet de pile,
SP est incrémenté de 2.

6°) - Les instructions de procédures :

Définition d'une procédure :

Nom_ procédure	PROC	Type
	Corps	
Nom_ procédure	ENDP	

L'appel d'une Procédure

- **Instruction CALL**

[etiquette]	CALL	<table border="0"> <tr><td>—</td><td>Nom_ procédure</td></tr> <tr><td>—</td><td>Registre (16 bits)</td></tr> <tr><td>—</td><td>Mémoire</td></tr> </table>	—	Nom_ procédure	—	Registre (16 bits)	—	Mémoire
—	Nom_ procédure							
—	Registre (16 bits)							
—	Mémoire							

Mode d'adressage : direct Nom_ procédure
Indirect par Registre ou mémoire

Fonctionnement :

Procédure **NEAR**

* Empilement de IP
(SP — SP-2)
* Chargement de IP
Avec l'AE de la procédure

Procédure **FAR**

* Empilement de CS puis IP
(SP — SP-4)
* Chargement de IP (2^{ieme} et 3^{ieme} oct)
* Chargement de CS (4^{ieme} et 5^{ieme} oct)

L'exécution se poursuit à partir de la nouvelle adresse (CS:IP)

Le retour d'une Procédure

- **Instruction RET**

[etiquette] RET { *n_immédiat* }

Fonctionnement :

Dépilement de IP
SP ——— SP+2
{Eventuellement, Dépilement de CS si RETF
SP ——— SP+2
SP ——— SP+n_ immédiat si n est présent}.

L'exécution se poursuit à partir de l'adresse (CS:IP), c.a.d. après l'instruction d'appel.

7°) - Les instructions des bits :

Les instructions logiques

Les instructions logiques (AND, OR, XOR, NOT, TEST)

les indicateurs du registre d'état (**F**) sont inchangés pour l'instruction NOT

Sinon

OF=CF=0

SF, ZF, PF en fonction du résultat

AF : indéfini

- **Instruction AND**

[etiquette] AND Destination,Source

Destination = Destination et Source

- **Instruction OR**

[etiquette] OR Destination,Source

Destination = Destination OU Source

- **Instruction XOR**

[etiquette] XOR Destination,Source

Destination = Destination OUEX Source

- **Instruction NOT**

[etiquette] NOT Destination

Destination = NON Destination (complément à 1)

- **Instruction TEST**

[etiquette] TEST Destination,Source

la Destination et la Source sont inchangés, mais les indicateurs sont positionnés en fonction du résultat du AND Destination,Source.

Les instructions de décalages

[etiquette] type_ décalage Destination,nb_pos.

Nb_pos peut le **registre CL de 8 bits** ou un **entier** positif et ceci à partir 80 286

les indicateurs du registre d'état (**F**) sont positionnés :

AF : indéfini

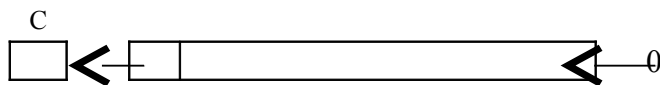
SF,ZF,PF et CF en fonction du résultat

OF est positionné en fonction du dernier et de l'avant dernier bit

Décalages logiques (SHL, SHR)

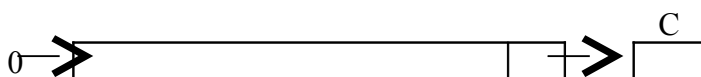
- **Instruction SHL**

[etiquette] SHL Destination,nb_pos



- **Instruction SHR**

[etiquette] SHR Destination,nb_pos

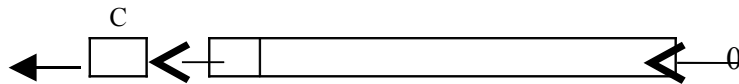


Décalages arithmétiques (SAL, SAR)

- **Instruction SAL**

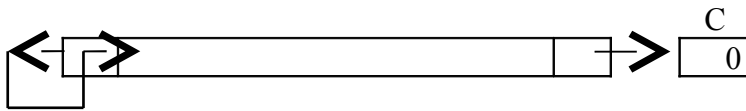
[etiquette] SAL Destination,nb_pos

nb_pos peut le *registre de 8 bits CL* ou un *entier* positif et ceci à partir 80 286



- **Instruction SAR**

[etiquette] SAR Destination,nb_pos



Décalages circulaires (ROL , ROR)

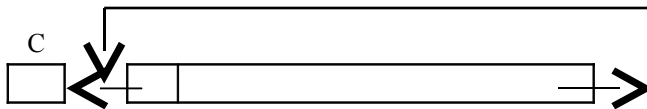
les indicateurs du registre d'état (**F**) sont positionnés :

CF en fonction du résultat

OF est positionné en fonction du dernier et de l'avant dernier bit

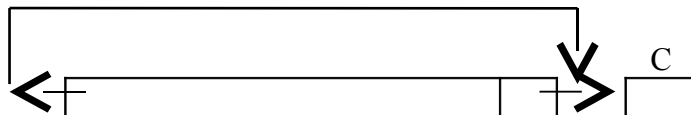
- **Instruction ROL**

[etiquette] ROL Destination,nb_pos



- **Instruction ROR**

[etiquette] ROR Destination,nb_pos



Décalages circulaires avec retenue (RCL , RCR)

les indicateurs du registre d'état (**F**) sont positionnés :

CF en fonction du résultat

OF est positionné en fonction du dernier et de l'avant dernier bit

- **Instruction RCL**

Le type Octet ou Mot est déterminé par l'assembleur en fonction de Destination,Source
Transfert Mémoire Mémoire : l'octet ou le mot pointé par DS:SI est transféré à l'adresse ES:DI
SI et DI sont incrémenté ou décrémenté de 1 ou 2 en fonction de la valeur de DF.

La Destination et source sont désignées par ES:DI et DS:SI et non par l'Opérande de l'instruction (Destination,Source) qui ne sert uniquement à déterminer le type (octet ou Mot) de transfert.

les indicateurs du registre d'état (**F**) : inchangés

Ou bien |< Etiquette>| MOVSB ⇒ transfert octet
 |< Etiquette>| MOVSW ⇒ transfert mot

• Instruction **LODS**

|< Etiquette>| **LODS** Source ;Chargement d'un octet ou d'un mot dans l'accumulateur
Le type Octet ou Mot est déterminé par l'assembleur en fonction de Source
SI est incrémenté ou décrémenté de 1 ou 2 en fonction de la valeur de DF.

Chargement de l'octet pointé par DS:SI dans AL

ou Chargement du mot pointé par DS:SI dans AX

La source est désignée par DS:SI et non par l'Opérande de l'instruction (Source) qui ne sert uniquement à déterminer le type (octet ou Mot) du chargement.

les indicateurs du registre d'état (**F**) : inchangés

Ou bien |< Etiquette>| LODSB ⇒ Chargement octet dans l'accumulateur AL
 |< Etiquette>| LODSW ⇒ Chargement mot dans l'accumulateur AX

• Instruction **STOS**

|< Etiquette>| **STOS** Destination ;
Rangement d'un octet ou d'un mot de l'accumulateur dans une chaîne
Le type Octet ou Mot est déterminé par l'assembleur en fonction de destination
DI est incrémenté ou décrémenté de 1 ou 2 en fonction de la valeur de DF.

Rangement de AL dans l'octet mémoire pointé par ES:DI

Ou Rangement de AX dans le mot pointé par ES:DI

La Destination est désignée par ES:DI et non par l'Opérande de l'instruction (Destination) qui ne sert uniquement à déterminer le type (octet ou Mot) du Rangement.

les indicateurs du registre d'état (**F**) : inchangés

Ou bien |< Etiquette>| STOSB ⇒ Rangement de AL dans ES:DI
 |< Etiquette>| STOSW ⇒ Rangement AX Dans ES:DI

Remarque : Les instructions MOVS, LODS et STOS peuvent être préfixés par **REP**

• Instruction **CMPS**

|< Etiquette>| **CMPS** Destination,Source ;Comparaison d'octet ou de mot

Le type Octet ou Mot est déterminé par l'assembleur en fonction de Destination, Source
 Comparaison Mémoire Mémoire : Identique à CMP, mais La comparaison se fait par la soustraction à source référencée par DS:SI de Destination référencée par ES:DI (**Source-Destination**)
 SI et DI sont incrémenté ou décrémenté de 1 ou 2 en fonction de la valeur de DF.
La Destination et source sont désignées par ES:DI et DS:SI et non par l'Opérande de l'instruction (Destination, Source) qui ne sert uniquement à déterminer le type (octet ou Mot) de transfert.

les indicateurs du registre d'état (**F**) : OF, SF, ZF, AF, PF, CF sont positionnés

Ou bien	< Etiquette> CMPSB	⇒	comparaison octet
	< Etiquette> CMPSW	⇒	comparaison mot

- **Instruction SCAS**

< Etiquette>| SCAS Destination ; Comparaison d'octet ou de mot avec l'accumulateur

Le type Octet ou Mot est déterminé par l'assembleur en fonction de Destination
 Comparaison Mémoire avec l'accumulateur AL ou AX : La comparaison se fait par la soustraction à AL ou AX de Destination référencée par ES:DI
 AL - Destination si type octet Ou bien AX- destination si type mot sans que les opérandes de la comparaison soient modifiés.

DI est incrémenté ou décrémenté de 1 ou 2 en fonction de la valeur de DF.

La Destination est désignée par ES:DI et non par l'Opérande de l'instruction (Destination) qui ne sert uniquement à déterminer le type (octet ou Mot) de transfert.

les indicateurs du registre d'état (**F**) : OF, SF, ZF, AF, PF, CF sont positionnés

Ou bien	< Etiquette> SCASB	⇒	comparaison octet avec AL
	< Etiquette> SCASW	⇒	comparaison mot avec AX

Remarque : Les instructions CMPS et SCAS peuvent être préfixés par **REP, REPZ OU REPNZ**

Les Préfixes permettant la répétition d'une opération de base.
 Cela implique l'utilisation implicite de CX comme compteur.

- **REP** ; répétition inconditionnelle
 l'exécution de l'instruction préfixée est répétée jusqu'à ce que CX atteigne 0;
- **REPE**
- **REPZ** ; répétition conditionnelle
 l'exécution de l'instruction préfixée est répétée jusqu'à ce que CX atteigne 0 ou ZF=0
- **REPNE**
- **REPNZ** ; répétition inconditionnelle
 l'exécution de l'instruction préfixée est répétée jusqu'à ce que CX atteigne 0 ou ZF=1

Chaîne

X	X		X	\$		
---	---	--	---	----	--	--

LES ASSEMBLEURS

I- Définitions

Le langage d'assemblage est le langage symbolique d'une machine ou le langage autocode (c.a.d le codop et les adresses des opérandes sont des identificateurs), ce langage est appelé le **LANGAGE ASSEMBLEUR** et le traducteur de celui ci est un **ASSEMBLEUR**.

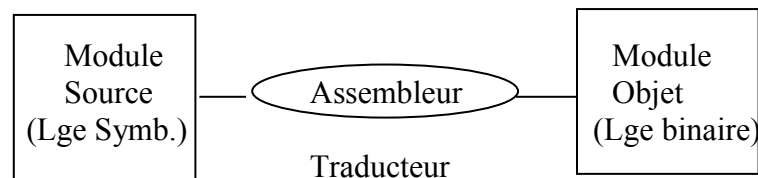
Caractéristiques des langages symboliques

1. les instructions de base sont caractérisées par le *code opération* appelé **MNEMONIQUE** (MOV, ADD, INC, ...) et à chaque instruction de base correspond une instruction machine et une seule.
2. Les *emplacements* des instructions et des données peuvent être référencés par des *adresses symboliques* (Nom, Etiquette, ...)
3. L'Assembleur dispose
 - du même jeu d'instructions que le langage machine (binaire),
 - de pseudo-instruction ou Directives,
 - de Macro-instructions (séquences pré définies d'instructions).

Nécessité des Assembleurs

Un *ordinateur* ne peut *interpréter* que des *instructions* écrites en langage machine *binaire*, il est donc nécessaire de *transformer* le *programme source* (écrit en langage autocode) en un **programme équivalent** en langage *binaire* (langage objet).

Cette *traduction* appelée **Assemblage** est réalisée par un *programme de service* du système : **ASSEMBLEUR**.



Le rôle d'un Assembleur se limite (se concentre sur) à *deux fonctions essentielles*.

1. *Transposer* les codes opérations mnémorique dans leur forme binaire et décoder les différentes parties de l'instruction.
2. Remplacer les adresses Symboliques (rencontrées dans le module source) par des adresses relatives dans le module objet.

Ces fonctions ne s'exécutent pas toujours de façon simultanée. Pour réaliser ces fonctions, il parfois nécessaire de faire plusieurs passages du module source

Un passage = Une lecture du programme source = Une passe

- Le programme source est assemblé dans une **Zone de Travail**.
- Les adresses relatives sont donc des déplacements par rapport au début de la zone de travail.
- Pour parcourir la zone de travail, l'Assembleur utilise un compteur d'emplacement (CE) qui indique le prochain emplacement libre.
- C'est seulement lorsque le chargeur implante le programme en mémoire pour l'exécution que les adresses relatives sont remplacées par des adresses réelles.

II- Description des tables nécessaires :

Certains programmes font appel à d'autres programmes (sous programmes macro-instructions), donc il est nécessaire d'établir des correspondances entre eux.

les étiquettes qui sont propres aux sous programmes en cours d'assemblage sont dites : étiquettes locales ou internes, et les étiquettes qui sont communes à un ensemble de sous programmes sont dites : étiquettes globales ou externes.

En effet : il ne faut pas confondre les identificateurs (étiquettes, adresse symbolique) entre les différents programmes à assembler.

D'où pour gérer les différentes informations au processus de traduction, l'assembleur se sert de trois tables principales :

- table des codes opération : TCO
- table des identificateurs internes : TI
- table des identificateurs externes : TE

1. table des codes opérations : TCO

La partie principale d'une ligne source (pseudo-instruction: PI, macro-instruction : MI , instruction : I) est le code opération

C'est le code opération qui définit la procédure à exécuter.

L'ensemble des codes opérations du langage autocode sont réunis dans une table : la TCO.

Dans cette table on fait correspondre à chaque code opération (mnémonique son code binaire.

En générale, l'identification du code opération déclenche l'exécution d'un sous programme de traduction

remarque : Toutes les instructions appartenant à une même catégorie sont traduites par un même sous programme.

Donc une entrée de la TCO comporte :

- le mnémonique
- code binaire
- l'adresse du sous programme de traitement correspondant

Mnémonique	code	sous programme
mov(reg/mémoire)	100010dw	adr s/pgme1
jmp (court)	11101001	adr s/pgme2
inc (reg)	01000reg	adr s/pgme3

2. Table des identificateurs locaux : TI

- Elle va contenir l'ensemble des étiquettes et adresses symboliques locales du programme en cours d'assemblage
- A chaque identificateur, l'assembleur fera correspondre l'adresse numérique relative de l'emplacement qu'il définit
- Il se sert pour cela du CE (Compteur d'Emplacement) qui spécifie le début de cette emplacement.

La TI a la structure suivante :

Chaque entrée comporte :

- Nom de l'identificateur

- l'adresse relative du début de l'emplacement associé à l'identificateur
- un indicateur qui indique si l'identificateur est défini (a une adresse relative) ou non.

identificateur	adresse relative	I
eti	15	d
debut	20	d
fin	?	i

Cette table n'est plus utilisée à la fin de l'assemblage de son programme, donc elle être effacée pour récupérer l'espace mémoire qu'elle occupe pour l'assemblage d'autre programme.

2. Table des identificateurs non locaux (ou externes) :TE

la TE peut être répartie en deux sous tables

- TRE : table de références externes
- TDE :table de définition externes

Les identificateurs externes sont des étiquettes déclarées comme :

Points d'entrées (définitions externes) : étiquettes définies dans un programme et pouvant être référencée par d'autres programmes ou sous programmes .

Ces identificateurs sont rangées dans la TDE et a la même structure que la TI en général.

Externes : étiquettes référencées dans un programme définies dans d'autres programmes ou sous programmes.

Ces identificateurs sont rangées dans la TRE et peut avoir la même structure que la TI ou bien chaque entrée de la table n'a que l'identificateur.

identificateur

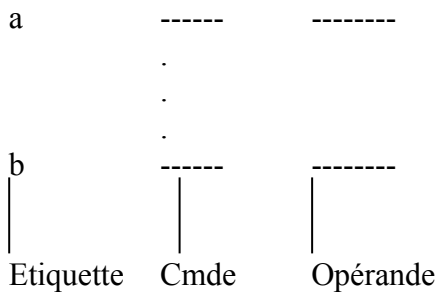
N.B. :Une étiquette ne peut être définie que dans un seule programme comme point d'entrée mais peut être référencée par plusieurs autres programmes.

une étiquette externe référencée par un programme doit être définie dans autre programme et un seul.

La TE est conservée a la fin de l'assemblage du programme ou le sous programme, car elle utilisée à l'Edition de lien et au chargement.

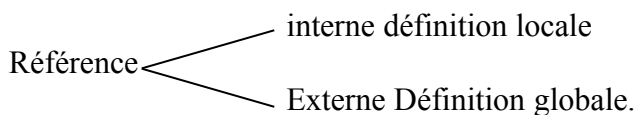
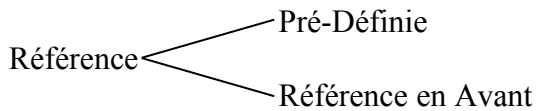
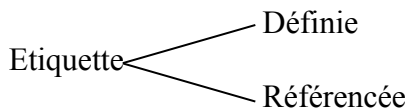
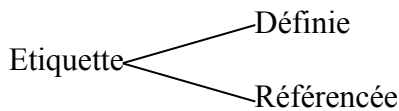
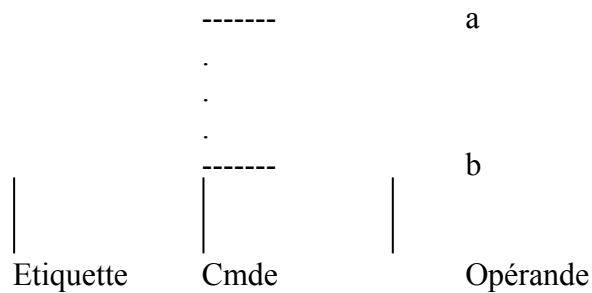
Programme P1

PUBLIC a,b



Programme P2

EXTRN a,b



Principales Techniques d'assemblage

Pour des raisons de simplicité, on considère que les données et le code forme un seul segment. l'espace adressable est linéaire (elle est constituée d'un seul bloc ou Segment)

I. Assembleurs a deux passes

cette technique d'assemblage nécessite deux lectures du programme source.
la lecture des lignes sources est séquentielles.

I.1. Première passe: Traitement des Identificateurs

- Tous les identificateurs rencontrés dans le champs Etiquette du module source sont rangés dans la TI avec leur adresse numérique relative au début du programme.
- A ce niveau, il faut également décoder le contenu du champs commande (I, PI, MI) et éventuellement le champs opérande tout en Analysant la syntaxe de toute l'instruction, car cela influence la progression du compteur d'emplacement (CE est incrémenté de la longueur de l'instruction décodée)

Adresse numérique = CE à la rencontre de l'étiquette

Algorithme de la première Passe.

CE est initialisé à 0

lire la ligne courante tout en ignorant les lignes commentaire

Tant qu'on a pas rencontré le END

Faire

Si Etiquette présent dans le champs Etiquette

Alors

S'il n'existe dans aucune des Tables (TI, TDE ,TRE)

Alors insérer l'Etiquette dans la TI avec comme adresse CE et I défini

Sinon S'il existe dans TI ou TDE avec I indéfini

Alors mise à jour de adresse avec CE i défini

Sinon « ERREUR »

Finsi

Finsi

Si Etiquette présent dans le champs opérande

Alors S'il n'existe dans aucune des tables

Alors

- insérer dans TDE si Cmde indique la définition Externe avec I indéfini
- insérer dans TRE
- insérer dans la TI dans les Cas avec I indéfini

Finsi

Finsi

 Analyser l'instruction pour déterminer la valeur d'incrémentation du CE

 Incrementer le CE si nécessaire

 Lire la ligne courante tout en ignorant les lignes commentaire

Finfaire

Remarque : Les différentes Erreurs possibles peuvent être détectées dès la fin de la Première Passe

- Toutes les étiquettes qui ne sont définies;
- les opérations qui n'appartiennent pas à la TCO;
- les différentes autres erreurs syntaxiques ou du mode d'adressage;
- ect...

I.2. deuxième Passe : génération du code binaire (module Objet)

En Entrée : le module Source

 les différentes tables (TCO, TI, TE)

 les différentes bibliothèques (macro-instructions, sous-programmes référencés dans le module Source)

En Sortie : Le module Objet (code binaire + les informations nécessaires pour générer le code exécutable + TE)

 Eventuellement en option le listing

Le Compteur d'Emplacement CE est ré-initialisé à 0 et indique à l'assembleur le prochain emplacement de libre

pour chaque ligne du programme source on fait le traitement suivant :

cas :

Instruction (traduite en binaire)

- Le code opération est mis sous forme binaire et rangé à l'emplacement indiqué par le CE
- les codes complémentaires (w, d, mode d'adressage, registre, ...) sont également traduits et mis en place
- les adresses symboliques des opérandes sont remplacées par leurs adresses relatives trouvées dans les différentes tables (TI,TE)

pseudo-instruction (Exécuté)

Elle est exécutée par l'assembleur selon son type.

- Réservation : Progression du CE de la longueur de la Zone
- Génération de Données : les Données sont traduites (converties en binaires) dans le code indiqué et placées à l'emplacement(s) indiqué(s) par le CE et le CE progresse.
- Spéciales : En fonction de la spécification

Exemples: PUBLIC étiquette dans la TE

END Fin du programme

ORG CE initialisé à la valeur spécifiée

Ect....

- Macro-instruction : On va voir ultérieurement le traitement des macro instructions.

I.3. Le Listing du Programme

Il comporte :

le programme source,
le Code généré à l'assemblage,
les messages d'erreurs de l'assemblage s'ils existent
les TI et TE

II. **Assembleur à une passe** : (Module Objet complètement est en mémoire à cause du retour arrière).

On cherche à éviter la seconde lecture et donc éviter de conserver en mémoire le programme source.

But : Faire le maximum de travail (générer le code) durant la première passe et ne pas faire des retours arrières seulement pour compléter les instructions qui n'ont pu être totalement assemblées, cela est dû aux références avancées (identificateur référencé avant d'être défini).

Rencontre d'une référence :

1. **Référence prédéfinie** (adresse relative est la TI ou TE) :

Une instruction qui renferme une référence prédéfinies peut être assemblée complètement dès sa première rencontre par l'assembleur (adresse déjà définie ou référence externe).

2. **Référence en avant** (adresse relative n'est pas encore définie):

Il faut faire de telle sorte que lorsque cette référence sera définie, on est capable de mettre à jour les champs adresses des toutes les instructions qui font référence à cette étiquette. Pour cela on chaîne toutes les instructions qui la référencent de la manière suivante :

- **Première rencontre de la référence** :

- * L'instruction est codée (code opération, code complémentaire,...) sauf la référence en avant qui sera remplacée par un « NIL » qui indique la fin de chaîne (LISTE)
- * la référence est insérer dans la TI (éventuellement dans la TE s'il s'agit d'une définition externe) avec comme adresse relative, l'adresse de l'instruction courante et comme indicateur, indéfinie
- *1^{ière} rencontre de la même référence :*
 - * L'instruction est codée (code opération, code complémentaire,...) sauf la référence en avant qui sera remplacée par l'adresse de l'instruction qui la référencée précédemment, existant dans la TI ,
 - * et l'adresse de cette instruction remplace l'adresse relative qui est recopiée dans le champs adresse de l'instruction dans la TI (insertion en tête de LISTE).

Exemple

ident.	adr	I
x	c	i

Code		
ADR	Opérations	référence
a	op1	NIL
b	op2	a
c	op3	b

Nous avons utilisé une technique de chaînage pour relier toutes les instructions qui font référence à la même étiquette .

Rencontre d'une définition d'étiquette :

1. *rencontre d'une étiquette existante dans la TI :*
 - ⇒ étiquette définie (indicateur est à définie) : dans ce cas ERREUR « Double définition »
 - ⇒ étiquette non définie : déjà référencée avant, dans ce cas :
 - * on parcourt les instructions qui font référence à cette étiquette à l'aide du chaînage déjà existant où la tête de liste est dans la TI (parcourir la liste jusqu'à ce qu'on rencontre le NIL;
 - * mettre à jour l'adresse relative et l'indicateur à définie
2. *rencontre d'une étiquette inexistante dans la TI :*
 - ⇒ Insérer dans la TI cette étiquette avec son adresse relative et son indicateur à définie

Remarque :

- La technique de génération du code objet est le même que l'assembleur à deux passe, sauf que il y a un traitement particulier aux étiquettes référencées avant d'être définies
- Le principe est le même aussi pour les références externes (définition ou références)

Après la définition de l'étiquette x avec comme adresse relative d :

ident.	adr	I
x	d	déf

Code		
ADR	Opérations	référence
a	op1	d
b	op2	d
c	op3	d

III . Assemblage des macro-instructions

Au moment de l'assemblage, il y a génération automatique d'une séquence d'instructions particulier à chaque que la macro instruction est appelée.

On peut considérer ces macro instructions comme des petits sous programmes (ouverts), écrit à part puis insérés à l'endroit voulu (endroit d'appel).

Elle permettent l'emploi des paramètres formels qui seront remplacés par les paramètres effectifs lors de l'assemblage.

1. Traitement des Macro Instructions Standards :

- ⇒ Les macro instructions Standards existent dans le système,
- ⇒ Elles sont déjà définies,
- ⇒ Les noms de ces macro instructions existent soit dans la TCO, soit dans une autre table connue du système.

- * Le traitement de la macro instruction se fait de la même manière qu'une instruction, sauf qu'on génère en mémoire toute une suite d'instructions (celle de la macro instruction) à la place de l'appel.
- * La macro instruction peut posséder ses propres adresses relatives qui sont donc exprimées en adresses relatives par rapport à son propre début.

Le Rôle de l'assembleur est

- * Décaler les adresses relatives de la macro instruction d'une valeur égale à la longueur de la partie du programme déjà assemblé.
- * Remplacer les paramètres formels de la macro instruction par les paramètres réels définis dans le programme appelant.

1. Traitement des Macro Instructions non Standards :

- ⇒ Les macro instructions non standards sont définies par l'utilisateur,
- ⇒ La définition de la macro instruction se fait avant son appel,
- ⇒ La macro instruction n'est pas traduite lorsqu'elle est définie, mais seulement au moment de son utilisation (appel).

• Traitement d'une définition de macro instruction :

Lorsque l'Assembleur rencontre une définition de macro instruction :

- * Il range le nom de la macro instruction dans la TI (ou dans une autre table bien spécifique), et place une marque pour le distinguer des autres identificateur;

- * Il place l'adresse du début de la macro instruction dans la zone mémoire;
- * Le corps de la macro instruction est conservé dans une zone mémoire (zone de macros).

- Traitement de l'appel de la macro instruction :

L'Assembleur Rencontre dans le champs Cmde, le nom d'une macro instruction, il fait une recherche initiale dans la TCO, et une fois qu'il ne le trouve pas, dans ce cas il va fait la recherche dans la TI (table spécifique pour macro) pour voir s'il s'agit bien d'une macro instruction

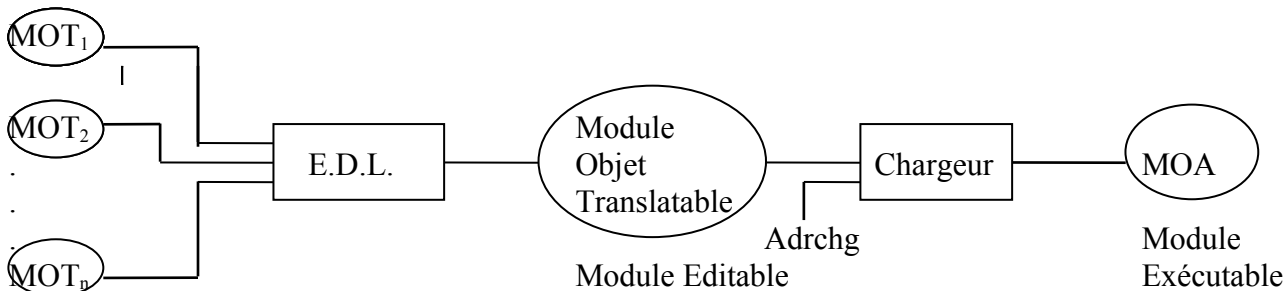
L'assembleur va donc traduire et insérer la macro instruction à la place de l'appel de la manière suivante :

- * Mise en correspondance dans la TI (à partir d'une marque) les paramètres formels et des paramètres effectifs
- * L'assemblage du corps de la macro instruction (prise de la zone de sauvegarde des macros se fait comme un seul bloc de la façon suivante :
 - ◊ la traduction se fait ligne par ligne,
 - ◊ tout paramètre formel est remplacé par le paramètre réel correspondant (adresse relative du paramètre formels est celle du paramètre réels),
 - ◊ insertion dans la TI les identificateurs locaux à la macro instruction
 - ◊ la rencontre de la fin de la macro instruction provoque l'effacement dans la TI de tout les paramètres formels afin d'économiser l'espace mémoire.

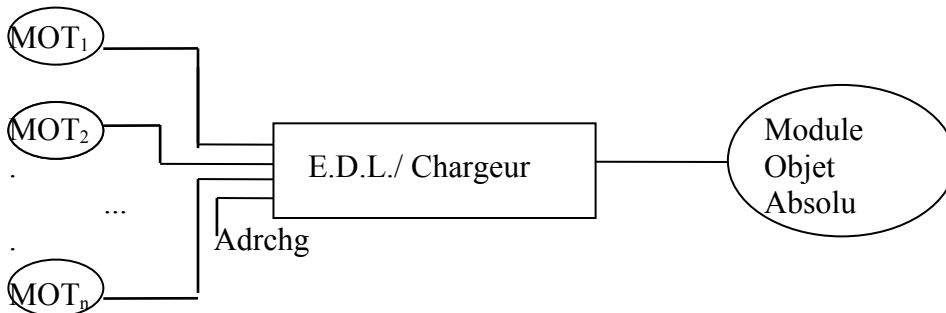
CHARGEUR EDATEUR DE LIENS

Le chargeur éditeur de liens reçoit en entrée plusieurs Modules Objet Translatables (MOTs) qu'il doit relier entre eux et charger en mémoire à partir d'une adresse donnée,

On a deux schémas possibles :



ou bien



Il doit réaliser les fonctions suivantes :

- a)- déterminer l'adresse de chargement de chacun des modules.
- b)- réaliser , pour chacun des modules la réimplantation des informations translatables,
- c)- réaliser la liaison des références externes.

Chaque module objet doit contenir , outre les informations de réimplantation, les informations permettant de faire la liaison des références externes .

C'est-à-dire :

- a)- la liste des noms des références externes faites dans le module.
- b)- la table des noms des définitions externes.

structure d'un module translatable.

en-tête [< nom-module, taille, adresse début d'exécution ou Nil >

table des ref. externes [nb, <identificateurs >

Corps [< a,n,r,texte >

table des def. Externes [nb, < identificateurs, adresse relative >

La signification d'un enregistrement du corps du module est maintenant la suivante :

a: l'emplacement du texte (adresse relative du code)

n: longueur en octets du texte

r :information de la translatabilité du texte

r = 0 : information non translatable

r = 1 : information translatable (référence interne)

r < 0 : information externe. dans ce cas, on doit ajouter à l'information "texte"

l'adresse absolue de l'emplacement désigné par l'identificateur qui porte le
numéro -r dans la table des références externes du module.

Quelque soit l'ordre de lecture des modules par l'EDL , des références externe peuvent apparaître avant la définition correspondante(2 modules peuvent chacun faire référence à l'autre)

De même qu'un assembleur, un EDL travaille donc, en 2 passes ou en une seule avec chaînage des références.

Editeur de liens à 2 passes

Fonctions de la première passe :

- allocation de la mémoire aux modules,
- constitution de la table globale des symboles externes.

Fonctions de seconde passe :

- liaison des références externes et transformation correspondantes des adresses,
- détermination de l'adresse début d'exécution.

Une table globale des identificateurs externes, analogue à la table des symboles d'un assembleur, est construite au moyen de deux procédures :

entrer (id, val)

insère dans la table le couple:

id : identificateur

val : adresse absolue correspondante.

recherche (id, val)

recherche l'identificateur id dans la table:

- Si échec Alors l'identificateur ne figure pas dans la table et la valeur de **val** est sans signification.

Sinon le couple (id, val) figure dans la table.

Editeur de liens

- paramètres :
- adresse de chargement en mémoire,
 - identificateurs des modules à relier (idf) ,
 - bibliothèque pour la recherche des références non satisfaites.

Algorithme éditeur de liens;

- récupérer les paramètres;
- adrdebut = adrcharg;
- première-passe;
- seconde-passe

fin;

Procédure première-passe ;

tantque liste des modules non épuisée faire

lire (nom-module, taille)

traiter-en-tête;

tantque table-ref-ext non épuisée faire

lire (idf-ref);

traiter-ref-externe

fin;

lire (corps-du-module); % pas de
traitement %

tantque table-def-ext non épuisée faire

lire (idf-def , adrdef);

traiter-def-externe

fin;

fin;

fin-passe1

fin;

Procédure traitement-en-tête;

Recherche(nom module, val);

si echec ou val = 0 alors

entrer (nom module , adrdebut)

sinon

erreur (" double définition ")

fin;

Base := adrdebut ;

adrdebut := adrdebut + taille;

% vérifier si taille permet le chargement %
fin;

Procédure traiter-def-externes

recherche (idf-def , val);

si echec ou val = 0 alors

adr := base + adrdef; % adresse absolue de
idf-def %

entrer (id-def , adr)

sinon

erreur (" double definition ")

fin

fin;

Procédure traiter-ref-externes;

recherche (idf-ref , val);

si echec alors

enter (idf-ref , 0)

fin

fin;

Procédure seconde-passe;

tantque liste des modules non épuisée faire

lire (nom-module, taille);

recherche (nom-module , val)

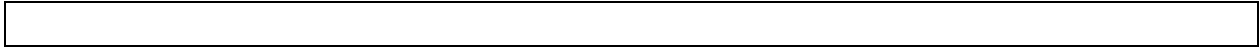
base := val;

lire (table-des ref-externes);

tantque corps-du-module non épuisé faire

lire (a,n,r,texte);
traiter-enreg-texte

fin
fin;
fin-passe2
fin;



Procedure *fin-passe2*;

```
    recherche ( debut , val );  
  
    si echec ou val = 0 alors  
        erreur (" adresse inconnue ")  
  
    sinon  
  
        adrexec := val  
    fin;  
    imprimer le contenu de la table des externes;  
    branchement à l'adresse adrexec  
fin;
```

Procedure *traiter-enreg-texte (a,n,r,texte)*;

```
    adr = base + a  
  
    si r = 1 alors  % information translatable %  
  
        texte = texte + base  
  
    sinon  
  
        si r < 0 alors  % reference externe %  
  
            i = - r;  
  
            idf-ref := entrée N° i dans table ref du module;  
            recherche ( idf-ref , val );  
            texte := texte + val  
  
        fin  
    fin;  
fin;
```