SOMMAIRE

0.1 - PRESENTATION :

MAX est un ASSEMBLEUR destiné aux micro-ordinateurs construits à partir de micro-processeurs de la famille des TMS7000 de TEXAS INSTRUMENTS.

Il est spécialement adapté à l'Ordinateur EXL100 disposant de l'extension CMOS RAM (exelmémoire) ou du systéme d'exploitation disque (DOS).

MAX vous permettra au moyen de son EDITEUR puissant de créer ou modifier des programmes en langage assembleur TMS7000.

Il vous permettra également de traduire ces programmes en langage machine (ASSEMBLER), de créer des modules COMMANDES exécutables en CRAM,et,d'accéder au CROS et au DOS.

Il vous donnera enfin la possibilité de choisir la COULEUR des caractéres affichés à l'écran,ainsi que la COULEUR du fond.

MAX supporte les fichiers sur CRAM et sur DISQUETTES à condition que les noms de ces fichiers soient normalisés.
Le nom doit etre constitué de 1 à 8 caractéres alphanumériques, suivis d'un point et de 3 caractéres alphanumériques indiquant le type du fichier.
Ce nom doit etre précédé de la lettre précisant le support concerné (C -> CRAM, A ou B -> DISQUES, I -> IMPRIMANTE).

exemples : **A:MYPROG.ASM** --> fichier disque 0, 100, A
B:MYPROG.LST --> fichier disque 1, 101, B
C:MYPROG.COM --> fichier cram (commande)
I: --> imprimante parallele (pas de nom)

MAX supporte jusqu'à 10 fichiers ouverts en meme temps. Il controle la validité du nom de fichier ainsi que l'existence du support.
Toutefois, en fonction du type de support, le nombre maximum de fichiers peut etre inférieur à 10. (par exemple, la CRAM n'autorise qu'un seul fichier en écriture).

0.2 - MISE EN PLACE ET UTILISATION :

Introduisez le MODULE ASSEMBLEUR face avant dans le logement prévu à cet effe
Mettez votre moniteur et votre EXL100 sous tension.
Vous obtenez alors un écran menu **(MENU PRINCIPAL)** :

```
          ------------------------
          ! E X E L M A X  1 . 6 !
          ------------------------

 ---------------------------------------------------------
 !     !                                                   !
 !  1  !  Select terminal color                           !
 !  2  !  Editor                                           !
 !  3  !  Assembler                                       !
 !  4  !  Relocatable  -->  COM                           !
 !  5  !  Peripheral Setup                                !
 !  6  !  CROS access                                      !
 !  7  !  DOS  access                                     !
 !     !                                                   !
 ---------------------------------------------------------

  Your Selection ? -
```

```
+-----------------------+---+
! 1.0  Your Selection ? ! 1 !
+-----------------------+---+
```

(Select terminal color)

Cette OPTION vous permet de choisir vos couleurs de travail.
Une grille de 64 cases apparait sur l'écran, chaque case représente une combinaison de COULEURS.
Il vous suffit, au moyen des 4 touches fléchées, de déplacer le curseur dans ces cases, et, d'actionner la touche RETOUR quand votre choix est fait.
Vous revenez alors au MENU PRINCIPAL.

```
+-----------------------+---+
! 2.0  Your Selection ? ! 2 !
+-----------------------+---+
```

(Editor)

Cette OPTION vous permet de créer et (ou) modifier un programme assembleur en utilisant l'EDITEUR pleine page, mono fichier, fourni avec MAX.

Le programme sous EDITEUR sera généralement un programme SOURCE, mais, vous pouvez également modifier un programme OBJET issu de l'assemblage (voir en annexes le format d'un programme objet).

Durant l'EDITION votre programme est stocké en mémoire VDP. L'EDITEUR fonctionne en mode INSERTION, vous ne pouvez donc pas surcharger un caractére par un autre. Toutefois, la commande **CTRL O** permet de passer en mode recouvrement (OVERRIDE), qui facilite les corrections (remplacement d'un caractére par un autre). Pour revenir au mode INSERTION il suffit d'appuyer à nouveau sur les touches **CTRL O**.

La ligne est la partie du texte que se termine par un RETOUR CHARIOT. Sa longueur n'est pas limitée, mais, MAX dans sa phase d'assemblage tronquera les lignes à 80 caractéres.

2.1 Edit File ?

Ce message vous permet de préciser le support et le nom du fichier dont le contenu doit etre chargé en mémoire VDP (ex : C:ESSAIS.ASM)
Si vous appuyez sur la touche RETOUR sans donner de nom, vous passez directement en EDITEUR sans charger (ou recharger) la mémoire VDP.
Si MAX affiche **Unable to open Edit File**, c'est que le fichier n'existe pas.

2.2 Utilisation des touches fléchées :

Cet EDITEUR est plein écran, vous pouvez donc vous déplacer à volonté dans le texte en utlisant les touches fléchées (droite, gauche, haut, bas).

La touche **'fleche inclinée'** place le curseur en début d'écran (en haut à gauche).

Vous pouvez faire défiler votre texte vers le haut ou vers le bas en positionnant le curseur sur la premiere ou sur la derniere ligne de l'écran, puis, en actionnant les touches 'fleche en haut' ou 'fleche en bas'.

La touche **'fleche coudée' (RETOUR)** à une utilité différente selon l'endroit où se trouve le curseur, et, selon que vous travaillez en creation (ligne vide) ou en mise à jour (curseur sur texte existant).

2.2.1 Le curseur est placé sur une ligne vide.

En mode **INSERTION**, la touche RETOUR marque la fin de ligne.

2.2.2 Le curseur est placé sur une ligne de texte.

En mode INSERTION :
Si le curseur est sur le premier caractére d'une ligne, cette touche permet d'insérer une ligne vide au dessus (le texte desc d'une ligne).
Si le curseur est placé apres le dernier caractére d'une ligne, il y a insertion d'une ligne vide au dessous (le texte remonte d'une ligne).
Si le curseur est sur un caractére quelconque, la touche RETOU permet de couper la ligne à partir du curseur.

En mode RECOUVREMENT :
Cette touche permet de placer le curseur en début de ligne suivante.

2.3 Utilisation de la touche 'DEL' :

Vous pouvez supprimer un caratére en positionnant le curseur derriere caractére puis, en actionnant la touche DEL.
Si vous appuyez sur DEL alors que le curseur se trouve sur le premier caractére d'une ligne, cette derniere viendra se placer au bout de la ligne précédente.

2.4 Utilisation de la touche 'TAB' :

Cette touche provoque le déplacement du curseur de 8 colonnes en 8 colonnes, avec, insertion de caractéres espace (en mode INSERTION).

2.5 Utilisation de la touche 'CTL' :

La touche **CTL** associée à la touche **C**, vous permet de quitter l'EDITEUR pour revenir au MENU PRINCIPAL , soit, directement (si la mémoire VDP est vide), soit, apres avoir sauvegardé la mémoire VDP.

Save (Y/N) ? vous permet de sauvegarder votre texte sur CRAM ou DISQUETTE (dans ce cas répondre Y).

File Name ? vous permet de fournir le nom du fichier à écrire ou reécrire (ex : C:ESSAIS.ASM), toutefois, si à ce moment vous appuye sur ESC et RETOUR, vous revenez au MENU PRINCIPAL sans sauvegarder.

Replace (Y/N) ? MAX vérifie que le support ne contient pas de fich de meme nom. S'il trouve un fichier, il vous demande l'autorisation le détruire (ce qu'il fera si vous répondez Y).

2.6 Utilisation de la touche 'ESC' :

Sous EDITEUR cette touche vous met en mode **COMMANDE**, en appuyant une seconde fois sur cette touche, vous sortez du mode COMMANDE.

Command.? affiché à l'écran (au début de la ligne où se trouve le curseur vous indique que vous etes en mode **COMMANDE**, vous pouvez alors manipuler votre texte au moyen d'ordres simples :

CTL I : Insertion de ligne.

Le texte descend d'une ligne, et, le curseur se place au début d'une ligne vide.

D : Suppression. **Delete**

L suppression d'une ligne.
B supression d'un bloc (Voir CTL S et CTL E ci apres).
T suppression du texte (dans ce cas vous confirmerez en répondant **Y** au message **Sure (Y/N)?**)

G : Déplacement. **Goto**

E en fin de texte.
S en début de texte.

F : Recherche de mot. **Word?**

Frappez le mot que vous recherchez, puis RETOUR, il s'affichera à la suite de **Word?** puis le curseur se positionnera à la 1 ere occurence du mot recherché.
La recherche se fait à partir de la position courante du curseur.

R : Remplacement de mot. **Change ?**

Frappez le mot que vous voulez changer, puis RETOUR.
With ?
Frappez le mot qui remplacera le précédent, puis RETOUR.
L'échange se fait à partir de la position courante du curseur.

S : Marque de début de bloc.

Le texte s'affiche dans une couleur différente à partir de la position du curseur jusque la fin du texte.

E : Marque de fin de bloc.

Le texte reprend sa couleur originelle à partir de la position du curseur jusque la fin du texte.

Tout ce qui se situe entre la marque de début de bloc et la marque de fin de bloc, constitue un BLOC.

C : Copie d'un bloc.

Le bloc est copié à l'endroit où se trouve le curseur.
Il est dupliqué (il ne disparait pas de l'endroit de départ).

CTL M : Mouvement d'un bloc.

Le bloc est supprimé de l'endroit où il était, puis il est déplacé là où se trouve le curseur.

O : Changement de mode.

Passage de mode INSERTION en mode RECOUVREMENT, et, vice versa. Le type de curseur change en fonction du mode.

2.7 Tableau récapitulatif des commandes :

CTL	C		Fin éditeur (retour au menu)
ESC	I		Insertion de ligne
ESC	D	L	Suppression de ligne
		B	" de bloc
		T	" de texte
ESC	G	S	Déplacement au début du texte
		E	" en fin du texte
ESC	F		Recherche de mot
ESC	R		Echange de mot
ESC	S		Marque début de bloc
ESC	E		Marque fin de bloc
ESC	C		Copie d'un bloc
ESC	M		Mouvement d'un bloc
ESC	O		Switch insertion/recouvrement

```
+------------------------+---+
! 3.0  Your Selection ? ! 3 !    (Assembler)
+------------------------+---+
```

Cette OPTION vous permet d'assembler votre programme source. Le fichier issu de l'assemblage est un fichier ASCII, éventuellement éditable (voir ANNEXE A).

Vous pouvez abandonner la phase d'assemblage en appuyant sur les touches ESC et RETOUR au lieu de répondre aux questions posées.

3.1 QUESTIONS POSEES AVANT ASSEMBLAGE :

3.1.1 Source File Name (VDP) ?

Appuyez sur la touche RETOUR si le source est en mémoire VDP, sinon, donnez le nom et le support du fichier contenant le source.

3.1.2 Object File Name (NUL) ?

Si RETOUR, MAX ne produira aucun fichier objet. Sinon donnez le nom et le support du fichier destiné à contenir le programme objet.

3.1.3 Replace (Y/N) ?

MAX vérifie si le nom donné ci.dessus (3.1.2) existe sur le support concerné, et, s'il trouve ce nom, demande l'autorisation de remplacer le fichier ancien (répondre Y pour remplacer).

3.1.4 List File Name (CON) ?

Si RETOUR, le résultat de l'assemblage sera affiché à l'écran. Sinon donnez le nom du fichier destiné à recevoir le résultat (généralement I: pour l'imprimante).

3.1.5 Options ?

Appuyez sur la touche RETOUR, ou, entrez une ou plusieurs lettres :

- **W** en cas d'erreur d'assemblage, MAX attend l'appui d'une touche pour continuer l'affichage des résultats.
- **L** annule le pseudo opérateur UNL.
- **B** annule l'option BUNLST.
- **T** annule l'option TUNLST.
- **D** annule l'option DUNLST.

```
+-----------------------+---+
| 4.0  Your Selection ? ! 4 !   (Relocatable --> COM)
+-----------------------+---+
```

Cette OPTION vous permet de créer des modules commandes exécutables en CRAM.
Vous pouvez abandonner cette option en appuyant sur les touches ESC et RETOUR au lieu de répondre aux questions posées.

4.1 QUESTIONS POSEES :

4.1.1 Relocatable File Name ?

Donnez le nom et le support d'un fichier issu de l'assemblage.

4.1.2 Command File Name ?

Donnez le nom et le support du fichier destiné à contenir le module commande.
Le type de ce fichier doit etre **COM** (ex : C:XXXXX.COM).

4.1.3 Run Time Address ?

Donnez l'adresse de chargement du module commande au moment de son exécution.
Cette adresse peut etre donnée en hexadécimal (>8004) ou en décimal (32772) ou en binaire (?100000000000100).
L'adresse de chargement sera stockée dans le fichier COM.

4.2 POURQUOI RELOGER UN MODULE ?

Examinons le résulat d'assemblage suivant

```
0000 5204              MOV  %4,B
0002 00        COMPAR  NOP
0003 AD000C            CMPA @ZON(B) <- Adresse par rapport au début
0006 E301              JHS  CONT            du programme (000C)
0008 0A                RETS
0009 CAF7      CONT    DJNZ B,COMPAR
000B 0A                RETS
000C 0A141E28  ZON     BYTE 10,20,30,40
0010                   END
```

Imaginons le meme programme relogé à l'adresse >8000

```
0000 5204              MOV  %4,B
0002 00        COMPAR  NOP
0003 AD800C            CMPA @ZON(B) <- Adresse par rapport au point
0006 E301              JHS  CONT            de chargement du programme
0008 0A                RETS                 (8000 + 000C = 800C)
0009 CAF7      CONT    DJNZ B,COMPAR
000B 0A                RETS
000C 0A141E28  ZON     BYTE 10,20,30,40
0010                   END
```

```
+-------------------------+---+
! 5.0  Your Selection ? ! 5 !    (Peripheral Setup)
+-------------------------+---+
```

Cette OPTION vous permet de remettre en fonction un périphérique qui etait éteint lors du lancement de MAX (en particulier l'imprimante).

```
+-------------------------+---+
! 6.0  Your Selection ? ! 6 !    (CROS access)
+-------------------------+---+
```

Cette OPTION vous permet d'acceder au CROS (Gestion de l'extension CMOS RAM).

```
+-------------------------+---+
! 7.0  Your Selection ? ! 7 !    (DOS access)
+-------------------------+---+
```

Cette OPTION vous permet d'acceder au DOS (Systeme d'exploitation disquette).

8.0 LE PROGRAMME SOURCE

Le programme source peut se trouver en mémoire VDP, sur CRAM, ou sur DISQUETTE. Il peut se composer de plusieurs parties, chacune constitua fichier différent, ces différents fichiers etant appelés par l'ordre moment de l'assemblage.
L'EDITEUR travaillant en mémoire VDP, cette possibilité de fragmentat permet de régler les problémes de place qui pourraient se poser.

Une ligne source se compose de :

- un **champ label** (facultatif)
- un **champ opérateur**
- un **champ opérandes** (facultatif)
- un **champ commentaires** (facultatif)

Chaque champ sera séparé du champ suivant par au moins un espace.
Le champ **label** est constitué de 6 caractéres alphanumériques plus #, ce champ fait plus de 6, les caractéres suivants seront ignorés.
Le champ **instruction** est constitué d'INSTRUCTIONS TMS7000, de NOMS de MACROS, ou, de PSEUDO-INSTRUCTIONS.
Le champ **opérandes** est fonction de l'instruction codée, il peut ne pa etre présent. Tout opérande superflu est purement et simplement ignor
Le champ **commentaires** doit etre précédé de **;**, son contenu n'est pas vérifié.

8.1 CODAGE DES EXPRESSIONS

Les opérandes d'une expression peuvent se coder de maniére différente

- 12 represente la valeur décimale 12, codée en décimal.
- ?1100 represente la valeur décimale 12, codée en binaire.
- >0C represente la valeur décimale 12, codée en hexadécimal.
- l'**apostrophe** précédera et suivra les chaines de caractéres.
- **% @ *** précéderont certaines expressions (voir adressage ci.apres).
- **(B)** terminera certaines expressions (voir adressage ci.apres).

Certains opérateurs n'affectent que la donnée qui les suit (**opérateur unaires**) :

- **+** n'affecte pas la valeur.
- **-** inverse arithmétiquement la valeur qui suit.
- **NOT** complément à 1.
- **LOW** partie basse (8 bits) d'une valeur 16 bits.
- **HIGH** partie haute (8 bits) d'une valeur 16 bits.

Certains opérateurs permettent une comparaison de valeurs :

- **EQ =** teste l'égalité.
- **NE <>** teste l'inégalité.
- **LT <** teste plus petit que.
- **GT >** teste plus grand que.
- **GE >=** teste plus grand ou égal.
- **LE <=** teste plus petit ou égal.

Si FAUX, la valeur retournée est 0, si VRAI, la valeur est >FFFF.

L'opérateur **NUL** permet dans une MACRO, de tester la présence d'un parametre.

8.1.1 Opérateurs arithmétiques

+ (addition) - (soustraction) * (multiplication) / (division)

8.1.2 Opérateurs logiques

```
MOD   reste de division entiere
SHL   décalage logique gauche
SHR   décalage logique droite
OR    ou inclusif
XOR   ou exclusif
AND   et
```

8.1.3 Priorité des opérateurs

La liste ci.dessous donne la priorité des opérateurs entre eux, les premiers cités sont de plus basse priorité.

```
OR    XOR
AND
=  <>  >  >=  <  <=  EQ  NE  GT  GE  LT  LE
+  -                                          BINAIRE
*  /  MOD  SHL  SHR
+ -                                           UNAIRE
NOT  HIGH  LOW  NULL
%  @  *  (B)
```

8.1.4 Exemple de codage d'expressions

```
0000 000C      = A1  EQU   >0C
0000 0003      = A2  EQU   3
0000 0004      = A3  EQU   4
0000           +---- IF      A1/A2=A3 OR (A3>A2 AND A1>A2)
0000 220F      !     MOV     %A1 OR %A2,A
0002 2200      !     MOV     %A1 XOR %A1,A
0004 2206      !     MOV     %A1 SHR 1,A
0006 2218      !     MOV     %A1 SHL 1,A
0008 2202      !     MOV     %11 MOD %3,A
000A           +---- ELSE
000A           !     MOV     %20,A
000A           +---- ENDIF
000A 17        ! Z1  BYTE -+ (2/1)+2+3+NOT 0+LOW 2567+HIGH 2567
000B FF        ! Z2  BYTE -! NOT 0
000C 07        ! Z1  BYTE -! LOW 2567
000D 0A        ! Z4  BYTE -! HIGH 2567
000E 4C274558  ! Z5  BYTE -! 'L''EXL100'
     4C313030  !           !
0016           !     END   !
               +-----------+------> Assemblage conditionnel
                           !
                           +--> Réservations (voir ci.apres)
```

8.2 LES MODES D'ADRESSAGE DU TMS 7000

8.2.1 Les registres du TMS 7000

Le TMS 7000 dispose de nombreux registres internes, tels que :

128 registres 8 bits à usage général (R0 à R127).
256 registres 8 bits d'entrées/sorties (P0 à P255). (PORT)
1 pointeur de pile SP de 8 bits.
1 registre d'état ST de 8 bits dont 4 utilisés.
1 compteur ordinal PC de 16 bits.

Le registre R0 s'appelle A, et, c'est l'accumulateur.
Le registre R1 s'appelle B, et, c'est le registre index.
Les registres P0 à P255 correspondent à des registres de périphériqu
qui peuvent exister ou pas sur votre EXL100.
Les registres P2 et P3 sont les 2 registres du TIMER.
Le registre d'état comporte 4 bits utiles :

```
+-------------------------------+
! C ! N ! Z ! I !   !   !   !   !
+-------------------------------+
```

C : retenue (opérations arithmétiques)
N : bit signe de l'opérande destination d'une instructio
Z : à 1 lorsque le résultat est nul
I : à 1 pour que les interruptions soient reconnu

8.2.2 Adressage simple registre

L'opérande de l'instruction est contenue dans un des 128 registres, l'instruction occupera 1 octet (DEC A) ou 2 octets (DEC R120).

8.2.3 Adressage double registre

Similaire à ci.dessus, sauf utilisation de 2 registres (registre source et registre destination), l'instruction occupera 1 octet (MOV A,B), 2 octets (MOV Rx,A), ou, 3 octets (MOV Rx,Ry).

8.2.4 Adressage registres périphériques

Adressage double registre, mais, un des registres est un registre périphérique (Pn), l'instruction occupera 2 octets (MOVP Pn,A), ou, octets (MOVP %x,Pn).

8.2.5 Adressage immediat

La donnée utilisée par l'instruction suit immédiatement celle.ci, la notation pour ce type d'adressage, utilise le symbole %
(MOV %12,A ou MOV %>0C,A).

8.2.6 Adressage relatif

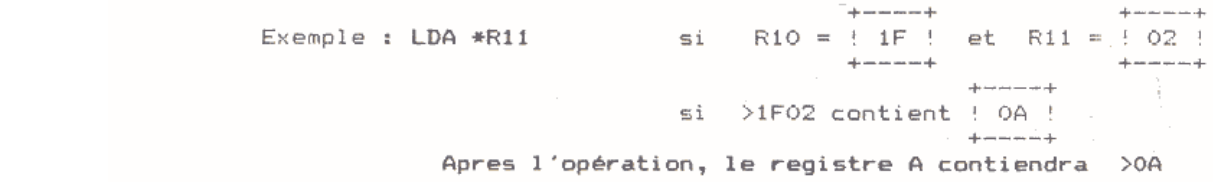
Ce mode est utilisé lors des opérations de branchements conditionnel ou inconditionnels, l'instruction de branchement est suivie par un octet contenant le déplacement (nombre d'octets entre l'instruction qui suit le branchement et la destination de celui.ci).
Vous n'avez pas à vous préoccuper de cela, codez simplement la réfé-
-rence de l'instruction destination, MAX fera le reste (JMP BOUCLE).
Attention aux déplacements supérieurs à +127 ou -128 (un seul octet)

8.2.7 Adressage étendu direct

Ce mode d'adressage manipule des adresses à 16 bits, vous pouvez donc accéder à toute la mémoire adressable.
L'instruction est suivie de l'adresse codée sur 2 octets, le symbole @ précise le mode d'adressage (LDA @>F2C1).

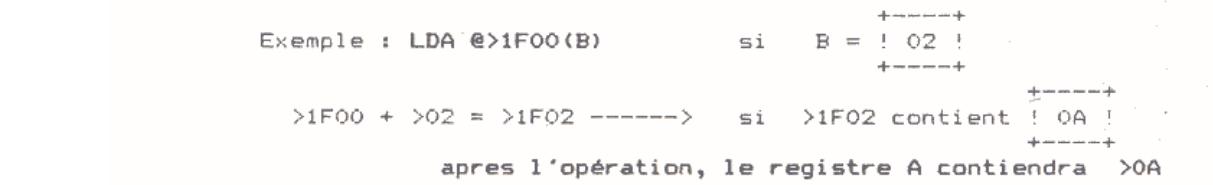
8.2.8 Adressage indirect

L'instruction est suivie d'un nom de registre (R1 à R127). Soit Rx ce registre, le TMS 7000 va chercher dans Rx-1 (poids forts) et dans Rx (poids faibles) une adresse de 16 bits.
Le symbole * indique ce type d'adressage.



8.2.9 Adressage étendu indexé

Le contenu d'un registre index (registre B) est ajouté à la valeur (codée sur 16 bits) qui suit l'instruction, le résultat donnera l'adresse où se trouve la donnée.



8.3 LES PSEUDO-OPERATEURS

Les pseudos-opérateurs sont des opérateurs qui ne génèrent aucun code particulier, mais, facilitent le travail du programmeur assembleur.

8.3.1 Réservation d'emplacements mémoire

BYTE permet la réservation et l'initialisation d'octets, l'adresse associée au label sera celle du 1er octet.

```
exemple : ZONE  BYTE 13,>AA,?011011,'BONJOUR',0
```

DATA permet la réservation et l'initialisation de mots (16 bits), l'adresse associée au label sera celle du 1er octet du 1er mot.

```
exemple : ZONE  DATA >FFFF,?011100 SHL 3,0
```

TEXT permet la réservation de mémoire, et, son initialisation par du code ASCII. L'adresse du 1er octet du texte sera associé au label.

```
exemple : ZONE  TEXT '1''EXL100 EST UN ORDINATEUR'
```

BES réserve de la place en mémoire, mais, n'initialise pas les octets réservés. L'adresse associée au label, sera celle du 1er octet qui suit la zone réservée.

```
exemple : 0068                    EQU     $
          00B8          BUFFER    BES     80
          00B8                    EQU     $
          00B8 00B8 = REFER       EQU     BUFFER
```

BSS réserve de la place en mémoire, mais, n'initialise pas les octets réservés. L'adresse associée au label, sera celle du 1er octet de la zone réservée.

```
exemple : 0068                    EQU     $
          00B8          BUFFER    BSS     80
          00B8                    EQU     $
          00B8 0068 = REFER       EQU     BUFFER
```

8.3.2 Pseudo-opérateurs divers

AORG n'est présent que pour assurer une certaine compatibilité avec l'assembleur Texas Instruments, mais, il 'est ignoré par MAX.
L'adresse d'implantation réelle n'est pas donnée par AORG, mais par l'OPTION 4 de MAX (Relocatable -> Com).

COPY permet l'inclusion de fichiers externes dans le source, et, de ce fait permet de travailler sur des sources dont la taille est supérieur à celle de l'EDITEUR.
le niveau maximal d'imbrication de COPY est de 4, le fichier appelé doit etre sur CRAM ou sur DISQUETTE.

```
exemple : COPY    A:PARTUN.ASM
          COPY    C:PARTDE.ASM
```

END indique la fin du source à assembler, tout ce qui suit ne sera pas assemblé.

IDT permet de donner un nom au module, ce nom (8 caractéres maxi) sera présent dans le module objet à l'issue de l'assemblage.

8.3.3 Assemblages conditionnels

IF débute un bloc IF, il permet un assemblage conditionnel d'une partie du programme.

ENDIF termine un bloc IF.

ELSE codé dans un bloc IF (entre IF et ENDIF), permet de donner une alternative à une condition préalablement testée par IF.

```
exemple : VENDU   EQU   0           --+
                  IF    VENDU <> 1    ! Si au moment de l'assemblage
                  RETS                ! VENDU=1, le programme sera
                  ELSE                ! complet, sinon, seule l'ins-
                  LDA   .....         ! -truction RETS sera assemblée
                  ...                 !
                  ENDIF             --+
```

Voir autre exemple d'assemblage conditionnel en 8.1.4.

8.3.4 Equivalences

EQU permet de donner des équivalences aux opérandes d'une instruction cette possibilité offre l'avantage de rendre le programme moins ésotérique.
Ainsi, **OR %PRESEN,FLAG** est préférable à **OR %?000100,FLAG**.

```
exemple :     PRESEN  EQU ?000100
              IMPR    EQU P55
                      ...
                      MOVP A,IMPR
                      OR   %PRESEN,FLAG
                      ...
```

SET Identique à EQU, mais, en plus, il est possible de redéfinir la valeur de SET.

Voir exemple d'utilisation de SET au chapitre des MACRO-INSTRUCTIONS

8.3.5 Parametrage de l'édition

LIST Annule l'effet de UNL, permet donc le listage du programme apres assemblage.

OPTION Comporte 3 opérandes, qui permettent de n'éditer le généré de BYTE (BUNLST), de DATA (DUNLST), ou, de TEXT (TUNLST), que sur une seule ligne.

```
exemple : 0000                    OPTION TUNLST,BUNLST
          0000 00010002  Z1       DATA   01,02,03,04,05,06
               00030004
               00050006
          000C 01020304  Z2       BYTE   1,2,3,4,5,6
          0012 31323334  z3       TEXT   '123456'
          0018                    END
```

PAGE permet d'indiquer à MAX le nombre maxi de lignes que vous voulez écrire sur chaque page ainsi que la taille de la feuille de papier (e nombre de lignes).

```
exemple :       PAGE  50,66
                      !  !
  nombre de lignes  <--- +  +---> taille de la page
```

Ce pseudo-opérateur seul (sans opérande) permet de provoquer un saut de page à la demande.

TITLE permet à MAX d'imprimer un titre au moment du listage.

exemple : **TITLE 'XXXXXX - VERSION 01.00 DU 20/04/86'**

UNL permet de stopper le listage d'un programme, le listage ne reprendra qu'à la rencontre du pseudo-opérateur LIST.

8.3.6 Tableau récapitulatif des pseudo-opérateurs

AORG	Implantation réelle (compatibilité TEXAS INSTR.)
BES	Réservation de mémoire (Beginning end symbol)
BSS	Réservation de mémoire (Beginning start symbol)
BYTE	Réservation et initialisation d'octets
COPY	Inclusion de fichiers 'source' externes
DATA	Réservation et initialisation de mots
ELSE	Alternative à IF dans un bloc IF - ENDIF
END	Fin de programme (Fin d'assemblage)
ENDIF	Marque la fin d'un bloc IF
EQU	Permet les équivalences
IDT	Donne un nom au module
IF	Assemblage conditionnel (IF - ELSE - ENDIF)
LIST	Demande le listage du résultat d'assemblage
OPTION	Directives données pour le listage
PAGE	Saut de page et caractéristiques d'une page
SET	Identique à EQU, mais, peut etre redéfini
TEXT	Réservation et initialisation de texte (ASCII)
TITLE	Impression d'un titre au listage
UNL	Demande l'arret du listage

8.4 LES INSTRUCTIONS DU TMS 7000

Le TMS 7000 possède un jeu d'instructions permettant d'effectuer des opérations arithmétiques, des branchements, des comparaisons, d'intervenir directement sur le fonctionnement du micro-precesseur, de manipuler des registres et des zones mémoire, d'effectuer des opérations logiques et de décalages registres, et, de réaliser des opérations d'entrées-sorties.

ADC - Add with carry - Addition avec retenue

Syntaxe : ADC source,destination
Opération : source + destination + bit C -> destination
Adressage : double registre
Registre etat : C à 1 si retenue
Z positionné selon résultat
N positionné selon résultat

Exemple : ADC R66,R117

Observations : ADC peut etre utilisé pour une addition en précision multiple, d'entiers signés ou non, par exemple, l'entie 16 bits des registres R2-R3 peut etre additionné à l'entier 16 bits des registres A-B

```
ADD  R3,B  -> addition des poids faibles
ADC  R2,A  -> addition des poids forts
```

MNEMONIQUE	CODE
ADC Rn,A	19
ADC %n,A	29
ADC Rn,B	39
ADC Rn,Rn	49
ADC %n,B	59
ADC B,A	69
ADC %n,Rn	79

ADD - Add - Addition

Syntaxe : ADD source,destination
Opération : source + destination -> destination
Adressage : double registre
Registre etat : C à 1 si retenue
Z positionné selon résultat
N positionné selon résultat

Exemple : ADD A,B

Observations : ADD est utilisé pour une additionner 2 octets.

MNEMONIQUE	CODE
ADD Rn,A	18
ADD %n,A	28
ADD Rn,B	38
ADD Rn,Rn	48
ADD %n,B	58
ADD B A	68
ADD %n,Rn	78

AND - And - ET logique

Syntaxe : AND source,destination
Opération : ET logique de source et destination -> destination
Adressage : double registre.
Registre etat : C à 0
Z positionné selon le résultat du ET logique
N positionné selon le résultat du ET logique

Exemple : AND %>1,R12 (met à 0 tous les bits de R12 sauf b0).

Observations : AND effectue un ET logique sur chacun des 8 bits des 2 opérandes.

Table de décision du ET logique :

source	destination	destination (résultat)
0	0	0
0	1	0
1	0	0
1	1	1

MNEMONIQUE	CODE
AND Rn,A	13
AND %n,A	23
AND Rn,B	33
AND Rn,Rn	43
AND %n,B	53
AND B,A	63
AND %n,Rn	73

ANDP - And peripheral register - ET avec registres périphériques

Syntaxe : ANDP source,destination
Opération : ET logique de source et destination -> destination
Adressage : registres périphériques.
Registre etat : C à 0
Z positionné selon le résultat du ET logique
N positionné selon le résultat du ET logique

Exemple : ANDP %>DF,P6 (met à 0 le bit 5 du port 6)

Observations : ANDP permet de remettre à zéro un ou plusieurs bits d
périphérique (port).

MNEMONIQUE	CODE
ANDP A,Pn	83
ANDP B,Pn	93
ANDP %n,Pn	A3

BR - Branch - Branchement inconditionnel

Syntaxe : BR destination
Opération : effectue un branchement inconditionnel à l'adresse 16 bits indiquée
Adressage : tous types étendus
Registre etat : non affecté

Exemple : BR @Adresse ou BR @Adresse(B) ou BR *Registre

Observations : BR permet d'aller à un endroit précis d'un programme
Le branchement indexé est un moyen d'exécuter une actio
ou une autre en fonction de parametres divers.

Exemple d'utilitation du branchement indexé :

```
BRIX      EQU  $
          MOV  R3,B          R3 contient 1 ou 2 ou 3 ... ou 128
          RL   B             X par 2 (déplacement dans la table)
          BR   @TABLE(B)     aller à l'instruction JMP adéquate
*
DISPATCH  EQU  $             table des branchements
          JMP  ACTION0            --- poste 1 de la table
          JMP  ACTION1            --- poste 2 de la table
          ...
          JMP  ACTIONn
ACTION0   EQU  $             si R3 = 0
          ...
ACTION1   EQU  $             si R3 = 1
          ...
```

MNEMONIQUE	CODE
BR @n	8C
BR *Rn	9C
BR @n(B)	AC

BTJO - Bit test and jump if one - Test d'un bit et saut si un

Syntaxe : BTJO source,destination,déplacement
Opération : ET logique de source et destination et saut si le résultat est différent de 0. Source et destination ne sont pas modifiés.
Adressage : double registre et relatif (pour le saut).
Registre etat : C à 0
Z positionné selon le résultat du ET logique
N positionné selon le résultat du ET logique

Exemple : BTJO %>14,R4,ETIQ saute à l'étiquette ETIQ si le bit b2 ou le bit b4 de R4 est à 1

Observations : Utilisez BTJO pour savoir si au moins un bit à son bit correspondant à 1 dans les 2 opérandes.
Source peut etre utilisé comme un masque destiné à tester à 1 les bits destination. Soit par exemple :

BTJO %>34,R2

	b7	b6	b5	b4	b3	b2	b1	b0	
si R2 =	0	1	1	0	1	1	1	1	>4F
masque =	0	0	1	1	0	1	0	0	>34

saut parce que -------> *

MNEMONIQUE	CODE
BTJO Rn,A	16
BTJO %n,A	26
BTJO Rn,B	36
BTJO Rn,Rn	46
BTJO %n,B	56
BTJO B,A	66
BTJO %n,Rn	76

BTJOP - Bit test and jump if one peripheral - Test d'un bit périphérique et saut si un

Syntaxe : BTJOP source,périphérique,déplacement
Opération : ET logique de source et périphérique et saut si le résultat est différent de 0. Source et périphérique ne sont pas modifiés.
Adressage : registres périphériques et relatif (pour le saut).
Registre etat : C à 0
Z positionné selon le résultat du ET logique
N positionné selon le résultat du ET logique

Exemple : BTJOP %>81,P4,ETIQ saute à l'étiquette ETIQ si le bit b0 ou le bit b7 de P4 est à 1

Observations : Utilisez BTJOP pour savoir si au moins un bit à son bit correspondant à 1 dans les 2 opérandes.

MNEMONIQUE	CODE
BTJOP A,Rn	86
BTJOR B,Rn	96
BTJOP %n,Rn	A6

BTJZ - Bit test and jump if zero - Test d'un bit et saut si zéro

Syntaxe : BTJZ source,destination,déplacement
Opération : ET logique de source et du complément de destination et saut si le résultat est différent de 0. Source et destination ne sont pas modifiés.
Adressage : double registre et relatif (pour le saut).
Registre etat : C à 0
Z positionné selon le résultat du ET logique
N positionné selon le résultat du ET logique

Exemple : BTJZ A,R23,ETIQ saute à l'étiquette ETIQ si au moins un bit à 1 de A correspond à un bit à 0 dans R23

Observations : Utilisez BTJZ pour tester à 0 le ou les bits de la destination.

MNEMONIQUE	CODE
BTJZ Rn,A	17
BTJZ %n,A	27
BTJZ Rn,B	37
BTJZ Rn,Rn	47
BTJZ %n,B	57
BTJZ B,A	67
BTJZ %n,Rn	77

BTJZP - Bit test and jump if zero peripheral - Test d'un bit périphé-
-rique et saut si zéro

Syntaxe : BTJZP source,périphérique,déplacement
Opération : ET logique de source et du complément de périphérique et saut si le résultat est différent de 0. Source et péri-
-phérique ne sont pas modifiés.
Adressage : registres périphériques et relatif (pour le saut).
Registre etat : C à 0
Z positionné selon le résultat du ET logique
N positionné se on le résultat du ET logique

Exemple : BTJZP %>21,P4,ETIQ saute à l'étiquette ETIQ si le bit b0 ou le bit b5 de P4 est à 0

Observations : Utilisez BTJZP pour savoir si au moins un bit du second opérande (port) est à 0.
Les bits testés sont ceux positionnés à 1 dans source.

MNEMONIQUE	CODE
BTJZP A,Pn	87
BTJZP B,Pn	97
BTJZP %n,Pn	A7

CALL - Call - Appel de sous programme

Syntaxe : CALL destination
Opération : sauvegarde le PC sur la pile puis saute à l'adresse
Adressage : tous types étendus
Registre etat : non affecté

Exemple : CALL @Adresse CALL @Adresse(B) CALL *Registre

Observations : CALL permet d'appeler une sous routine placée n'importe où en mémoire. Il appartient à la routine appelée de sauvegarder (PUSH/POP) les registres uttilisés.

MNEMONIQUE	CODE
CALL @n	8E
CALL *Rn	9E
CALL @n(B)	AE

CLR - Clear - Mise à zéro

Syntaxe : CLR destination
Opération : mise à zéro de destination
Adressage : simple registre.
Registre etat : C à 0
Z à 1
N à 0

Exemple : CLR B (met B à zéro)

Observations : CLR permet de remettre à zéro tout registre.

MNEMONIQUE	CODE
CLR A	B5
CLR B	C5
CLR Rn	D5

CLRC - Clear the carry bit - Mise à zéro du bit C

Syntaxe : CLRC
Opération : mise à zéro du bit C du registre d'etat.
Adressage : implicite.
Registre etat : C à 0
Z positionné selon le contenu du registre A
N positionné selon le contenu du registre A

Exemple : CLRC

Observations : CLRC est utilisée pour remettre à zéro le bit carry avant une instruction arithmétique ou de décalage registre.
Cette instruction est identique a TSTA.

MNEMONIQUE	CODE
CLRC	B0

CMP - Compare - Comparaison

Syntaxe : CMP source,destinaton
Opération : soustrait source de destination et positionne les bits du registre d'etat en fonction du résultat. Source et destination ne sont pas modifiés.
Adressage : double registre.
Registre etat : C à 1 si destination est logiquement plus grande
Z à 1 si le résultat de la soustraction est négatif
N à 1 si source et destination sont égales

Exemple : CMP R13,R89

Observations : Les nombres négatifs sont considérés comme arithméti-
-quement plus petits, mais, comme logiquement plus grands que des nombres positifs. Apres CMP vous pouvez utiliser les branchements conditionnels suivants :

JC/JHS -> saut si dest. logiquement >= source
JN -> saut si dest. arithmetiquement < source
JNC/JL -> saut si dest. logiquement < source
JNZ/JNE -> saut si dest. non égal source
JP -> saut si dest. arithmétiquement > source
JZ/JEQ -> saut si dest. égal source
JPZ -> saut si dest. arithmétiquement >= source

MNEMONIQUE	CODE
CMP Rn,A	1D
CMP %n,A	2D
CMP Rn,B	3D
CMP Rn,Rn	4D
CMP %n,B	5D
CMP B,A	6D
CMP %n,Rn	7D

CMPA - Compare accumulator extended - Comparaison étendue avec A

Syntaxe : CMPA source
Opération : soustrait source de A et positionne les bits du registre d'etat en fonction du résultat. A et source non modifiés.
Adressage : tous modes étendus.
Registre etat : C à 1 si destination est logiquement plus grande
Z à 1 si le résultat de la soustraction est négatif
N à 1 si source et destination sont égales

Exemple : CMPA @Adresse CMPA @Adresse(B) CMPA *Registre

Observations : CMPA peut etre utilisée pour comparer A et un opérande à adressage long (via mode direct, indexé, indirect).
CMPA est tres utile pour les recherches en table.

MNEMONIQUE	CODE
CMPA @n	8D
CMPA *Rn	9D
CMPA @n(B)	AD

DAC - Decimal add with carry - Addition décimale avec retenue

Syntaxe : DAC source,destination
Opération : BCD de source + destination + C -> destination
Adressage : double registre
Registre etat : C à 1 si résultat > ou = 100
Z positionné selon résultat
N positionné selon résultat

Exemple : DAC %>32,A : SI A = >34, apres opération, A = >66

Observations : DAC est utilisé pour additionner 2 octets BCD.
Si opérande à 0, DAC équivaut à incrément conditionnel de l'opérande destination
Le bit carry est addiionné, ce qui permet d'additionner une suite d'octets DCB. Il faut donc penser à remettre le bit carry à 0 avant la premiere addition.

La routine ci-dessous permet d'additionner entre eux, une suite d'octets (255 C maxi) stockés aux adresses STR1 et STR2, et de placer le résultat en STR2. Le registre B contient le nombre d'octets à additionner

```
        CLRC            carry à 0
        PUSH ST         sauveg. status sur pile
LOOP    LDA  @STR1-1(B) chargement octet en cours de STR1
        MOV  A,R2       sauveg. cet octet dans R2
        LDA  @STR2-1(B) chargement octet suivant de STR2
        POP  ST         récup. carry précédente addition
        DAC  R2,A       addition des octets décimaux
        PUSH ST         sauveg. carry addition ci-dessus
        STA  @SR2(B)    stockage résultat addition
        DJNZ B,LOOP     boucle nombre d'octets à additionner
        POP  ST         récupération status
        RETS            retour au module appelant
```

MNEMONIQUE	CODE
DAC Rn,A	1E
DAC %n,A	2E
DAC Rn,B	3E
DAC Rn,Rn	4E
DAC %n,B	5E
DAC B,A	6E
DAC %n,Rn	7E

DEC - Decrement - Décrémentation

Syntaxe : DEC destination
Opération : destination - 1 -> destination
Adressage : simple registre
Registre etat : C à 0 si passage de 00 à FF
Z positionné selon résultat
N positionné selon résultat

Exemple : DEC R117 diminue de 1 le contenu du registre R117

Observations : DEC est utile pour le comptage ou l'adressage de tableaux d'octets.

MNEMONIQUE	CODE
DEC A	B2
DEC B	B2
DEC Rn	B2

DECD - Decrement double - Décrémentation double

Syntaxe : DECD destination
Opération : diminue de 1 le mot de 16 bits contenu dans la paire de registres destination, et, place le résultat dans cette paire de registres
Adressage : simple registre
Registre etat : C à 0 si passage de 00 à FF des 8 bits de poids fort
Z positionné selon résultat sur 8 bits de poids fort
N positionné selon résultat sur 8 bits de poids fort

Exemple : DECD R51 diminue de 1 le mot de 16 bits contenu dans R50 (poids fort) et R51 (poids faible)

Observations : Cette instruction peut etre utilisée pour décrémenter une adresse indirecte placée dans un registre.

La routine ci-dessous permet de rechercher un octet dans une table de 500 octets, et, retourne dans les registres R2 et R3 l'adresse de cet octet.
Au moment de l'appel la routine les registres R2 et R3 doivent etre initialisés à l'adresse de fin de table, les registres R4 et R5 contiennent la taille de la table, et, le registre A contient l'octet à rechercher.

```
LOOP    CMPA  *R3      test de l'octet en cours
        JZ    FOUND    si egal, il est trouvé (carry à 0)
        DECD  R3       si non, décrémenter l'adresse
        DECD  R5       faire - 1 sur le nombre de postes
        JNZ   LOOP     si poids fort <> 0, on continue
        CMP   %0,R5    si poids faible <> 0, on continue
        JNZ   LOOP
        SETC           si fin de table, carry à 1
FOUND   RETS           retour au module appelant
```

MNEMONIQUE	CODE
DECD A	BB
DECD B	CB
DECD Rn	DB

DINT - disable interrupts - interdiction des interruptions

Syntaxe : DINT
Opération : met à 0 le bit I du registre d'etat afin d'interdire les interruptions.
Adressage : implicite.
Registre etat : C à 0
Z à 0
N à 0
I à 0

Exemple : DINT

Observations : Le bit d'autorisation des interruptions etant stocké dans le registre d'état, les instructions POP ST et RETI peuvent rendre les interruptions possibles malgré DINT. Durant le traitement d'une interruption, le bit I est automatiquement mis à 0 aprés sauvegarde du reg. d'état.

MNEMONIQUE	CODE
DINT	06

DJNZ - Decrement and jump if not zero - Décrémente et saute si non zéro

Syntaxe : DJNZ source,destination
Opération : diminue de 1 le contenu de source, et, place le résultat dans source, puis, saute à destination si le contenu de source est différent de 0.

Adressage : simple registre et relatif (pour le saut).
Registre etat : non affecté.

Exemple : DNJZ R22,ETIQ

Observations : Cette instruction permet de controler une boucle.

MNEMONIQUE	CODE
DJNZ A	BA
DJNZ B	CA
DJNZ Rn	DA

EINT - enable interrupts - autorisation des interruptions

Syntaxe : EINT
Opération : met à 1 le bit I du registre d'etat afin d'autoriser les interruptions.
Adressage : implicite.
Registre etat : C à 1
Z à 1
N à 1
I à 1

Exemple : EINT

Observations : Le bit d'autorisation des interruptions etant stocké dans le registre d'état, les instructions POP ST et RETI peuvent interdire les interruptions malgré EINT.
Dans une routine de service d'interruption, il faut effectuer un EINT pour permettre les interruptions imbri-quées.

MNEMONIQUE	CODE
EINT	05

IDLE - Idle until interrupt - Arret en attente d'interruption.

Syntaxe : IDLE
Opération : arret du programme en attendant une interruption.
Adressage : implicite.
Registre etat : non affecté

Exemple : IDLE

Observations : Le TMS 7000 est arreté en attente d'une interruption o
d'un RESET. Vous devez positionner le bit I du registr
d'état et, vous assurer que les autorisations d'inter-
-ruptions sont o.k dans le registre de controle, avant
d'exécuter IDLE.

MNEMONIQUE	CODE
IDLE	01

INC - Increment - Incrémentation

Syntaxe : INC destination
Opération : destination + 1 -> destination
Adressage : simple registre
Registre etat : C à 1 si passage de FF à 00
Z positionné selon résultat
N positionné selon résultat

Exemple : INC A

Observations : Cette instruction ajoute 1 dans un registre, puis,
place le résultat dans ce registre.

MNEMONIQUE	CODE
INC A	B3
INC B	C3
INC Rn	D3

INV - Invert - Inverse

Syntaxe : INV destination
Opération : inversion bit à bit de la destination
Adressage : simple registre
Registre etat : C à 0
Z positionné selon résultat
N positionné selon résultat

Exemple : INV A

Observations : Cette instruction effectue un complémeent à 1 de l'opérande. Pour effectuer un complément à 2, il suffit de faire suivre l'instruction INV par l'instruction INC

MNEMONIQUE	CODE
INV A	B4
INV B	C4
INV Rn	D4

J(cond) - Jump on condition - Saut conditionnel

Syntaxe : J(cond) déplacement
Opération : saut si la condition est vraie

Adressage : relatif
Registre etat : non affecté

Exemple : JNC ETIQ

Observations : J(cond) est généralement utilisée après CMP.
Apres un MOV, MOVP, LDA, STA, l'instruction JZ ou JNZ permet de savoir si la valeur mouvementée est égale à 0.
JN et JPZ permettent de tester le bit signe.

INSTRUCTION	MNEMONIQUE	CODE	ETAT DES BITS POUR SAUT C	N	Z
saut si retenue	JC	E3	1	x	x
saut si égal à zéro	JEQ	E2	x	x	1
saut si supérieur ou égal	JHS	E3	1	x	x
saut si inférieur	JL	E7	0	x	x
saut si négatif	JN	E1	x	1	x
saut si pas de retenue	JNC	E7	0	x	x
saut si non égal	JNE	E6	x	x	0
saut si différent de zéro	JNZ	E6	x	x	0
saut si positif	JP	E4	x	0	0
saut si positif ou nul	JPZ	E5	x	0	x
saut si nul	JZ	E2	x	x	1
saut si plus petit	JLT	E1	x	1	x
saut si plus grand	JGT	E4	x	0	0
saut si plus grand ou égal	JGE	E5	x	0	x

LDSP - Load stack pointer - Chargement du pointeur de pile

Syntaxe : LDSP
Opération : place le contenu de B dans le pointeur de pile.
Adressage : implicite.
Registre etat : non affecté.

Exemple : LDSP

Observations : LDSP est utilisée pour initialiser le pointeur de pile.

MNEMONIQUE	CODE
LDSP	0D

LVDP - Load VDP - Chargement VDP

Syntaxe : LVDP
Opération : place un octet de la mémoire VDP dans A.
Adressage : implicite.
Registre etat : C à 0
Z positionné en fonction de la valeur lue.
N positionné en fonction de la valeur lue.

Exemple : LVDP

Observations : LVDP permet de lire la mémoire VDP. L'adresse de lecture devra etre positionnée par un TRAP 9 ou un TRAP 10. Votre documentation VDP vous donnera les précisions utiles.

MNEMONIQUE	CODE
LVDP	D7

MOV - Move - déplace

Syntaxe : MOV source,destination
Opération : place le contenu de source dans destination
Adressage : double registre.
Registre etat : C à 0
Z positionné selon la valeur déplacée
N positionné selon la valeur déplacée

Exemple : MOV A,B (met le contenu de A dans B)
MOV R35,R105 (met le contenu R35 dans R105)
MOV %12,R2 (met 12 (décimal) dans R2)

Observations : MOV permet de transférer des valeurs entre registres, ou une valeur immédiate dans un registre.

MNEMONIQUE	CODE
MOV Rn,A	12
MOV %n,A	22
MOV Rn,B	32
MOV Rn,Rn	42
MOV %n,B	52
MOV B,A	62
MOV %n,Rn	72

MOVD - Move double - Double déplacement

Syntaxe : MOVD source,destination
Opération : déplace une valeur 16 bits de la paire de registres source vers la paire de registres destination
Adressage : spécial :

MOVD %n,Rn	CODE	poids fort	poids faible		d
MOVD Rn,Rn	CODE		s		d
MOVD %n(B),Rn	CODE	poids fort	poids faible		d

Registre etat : C à 0
Z positionné selon l'octet poids fort du mot déplacé
N positionné selon l'octet poids faible du mot déplacé

Exemple : MOVD %>1234,R3 (met 1234 (hexa) dans R2 et R3)
MOVD R5,R3 (déplace R4 et R5 dans R2 et R3)
MOVD %TAB(B),R3 (place l'adresse indexée calculée dans R2 et R3)

Observations : MOVD %n,Rn est utilisée pour initialiser une paire de registres utilisables pour l'adressage indirect.
MOVD Rn,Rn permet de transférer le contenu de 2 registres en une seule fois.
MOVD %n(B),Rn permet de stocker une adresse indexée dans une paire de registres, pour utiliser plus tard un adressage indirect. Au moment de l'opération, le contenu du registre B sera ajouté à la valeur 16 bits de l'opérande %n, et, le résultat sera placé dans la paire de registres Rn-1,Rn (poids fort dans Rn-1, poids faible dans Rn).

MNEMONIQUE	CODE
MOVD %n,Rn	88
MOVD Rn,Rn	98
MOVD %n(B),Rn	A8

MOVP - Mov to/from peripheral - Déplacement de/vers un périphérique

Syntaxe : MOVP source,destination
Opération : déplace le contenu de source dans destination
Adressage : registres périphériques.
Registre etat : C à 0
Z positionné selon la valeur déplacée
N positionné selon la valeur déplacée

Exemple : MOVP A,P2 (initialise la valeur du timer (A dans P2))
MOVP P4,B (lit une donnée périphérique dans B)

Observations : MOVP est utilisée pour transférer des données depuis ou vers un périphérique. Au cours de ce type d'instruction, le périphérique est lu, si pour des raisons tenant au matériel (hardware), cette lecture est indésirable, il vous faudra utiliser STA avec une adresse mémoire désignant le périphérique. Cette adresse sera obtenue en ajoutant >100 (256) au numéro de périphérique (port).

MNEMONIQUE	CODE
MOVP A,Pn	82
MOVP B,Pn	92
MOVP %n,Pn	A2
MOVP Pn,A	80
MOVP Pn,B	91

MPY - Multiply - Multiplication

Syntaxe : MPY source,destination
Opération : source x destination -> A,B. Poids forts dans A
Adressage : double registre
Registre etat : C à 0
Z positionné selon octet poids fort du résultat
N positionné selon octet poids fort du résultat

Exemple : MPY R3,R4 (multiplie R3 par R4 et place le résultat dans A et B)

Observations : MPY effectue une multiplication 8 bits et place le résultat dans 16 bits (A et B). Une multiplication par une puissance de 2, peut etre effectuée par des instructions de décalage (RLC).

La routine ci-dessous permet de multiplier un nombre de 16 bits (dans R2 et R3) par un nombre de 8 bits (dans A)
Au retour le résultat est placé dans A,R2,R3.
Si dépass. capacité, bit 0 = 0 sinon bit 0 = 1.

```
MPY16X8  EQU   $
         PUSH  B       sauvegarde des registres de travail
         PUSH  R4
         MOV   A,R4    copie A
         MPY   R3,A    A=A*R3 (poids fort) B=A*R3 (poids faible)
         PUSH  A       sauvegarde de A*R3
         MOV   B,R3    copie résultat A*R3 (poids faible)
         MOV   R4,A    restauration de A
         MPY   R2,A    A=A*R2 (poids fort) B=A*R2 (poids faible)
         POP   R2      récupérer dans R2 résultat de A*R3
         ADD   B,R2    R2 = (A*R3) + (A*R2)
         ADC   %0,A    transférer carry bit dans overflow byte
         POP   R4
         POP   B       restaurer les registres de travail
         RETS
```

MNEMONIQUE	CODE
MPY Rn,A	1C
MPY %n,A	2C
MPY Rn,B	3C
MPY Rn,Rn	4C
MPY %n,B	5C
MPY B,A	6C
MPY %n,Rn	7C

NOP - No operation - Pas d'opération

Syntaxe : NOP
Opération : le PC passe à l'instruction suivante.
Adressage : implicite.
Registre etat : non affecté

Exemple : NOP

Observations : NOP est utile comme instruction de remplissage, il perme
alors de compléter ou modifier le programme directement
en langage machine (PATCH,S).

MNEMONIQUE	CODE
NOP	00

OR - Or - OU logique

Syntaxe : OR source,destination
Opération : OU logique de source et destination -> destination
Adressage : double registre.
Registre etat : C à 0
Z positionné selon le résultat du OU logique
N positionné selon le résultat du OU logique

Exemple : OR %>0F,A (met à 1 les 4 bits de poids faible de A).

Observations : OR effectue un OU logique sur chacun des 8 bits des 2 opérandes.

Table de décision du OU logique :

source	destination	destination (résultat)
0	0	0
0	1	1
1	0	1
1	1	1

MNEMONIQUE	CODE
OR Rn,A	14
OR %n,A	24
OR Rn,B	34
OR Rn,Rn	44
OR %n,B	54
OR B,A	64
OR %n,Rn	74

ORP - Or peripheral register - OU logique avec registres périphériques

Syntaxe : ORP source,destination
Opération : OU logique de source et destination -> destination
Adressage : registres périphériques.
Registre etat : C à 0
Z positionné selon le résultat du OU logique
N positionné selon le résultat du OU logique

Exemple : ORP %>01,P6 (met à 1 le bit 0 de P6 (port B))

Observations : ORP permet de mettre à 1 chacun des bits d'un registre périphérique (port).
Le périphérique etant lu avant que le OU ne soit réalisé il se peut que l'instruction ne fonctionne pas pour des périphériques ayant des fonctions differentes en lecture et en écriture.

MNEMONIQUE	CODE
ORP A,Pn	84
ORP B,Pn	94
ORP %n,Pn	A4

POP - POP from stack - Extraction d'une valeur de la pile

Syntaxe : POP destination
Opération : place la valeur située au sommet de la pile dans la destination, puis, diminue de 1 le contenu du pointeur de pile.
Adressage : simple registre.
Registre etat : C à 0
Z positionné selon la valeur récupérée
N positionné selon la valeur récupérée

Exemple : POP R32 ou POP ST

Observations : La pile peut etre utilisée pour sauvegarder ou passer des données, par exemple, dans des sous-programmes ou dans des routines de service d'interruptions.
POP récupére la donnée au sommet de la pile.
POP ST charge le registre ETAT à partir de la pile.

MNEMONIQUE	CODE
POP A	B9
POP B	C9
POP Rn	D9
POP ST	08

PUSH - PUSH on stack - Mise sur la pile

Syntaxe : PUSH source
Opération : Augmente le pointeur de pile d'une unité, et, place le contenu de source sur la pile.
Adressage : simple registre.
Registre etat : C à 0
Z positionné selon la valeur placée sur la pile
N positionné selon la valeur placée sur la pile

Exemple : PUSH A ou PUSH ST

Observations : La pile est utilisée pour sauvegarder des données, par exemple, dans des sous-programmes ou dans des routines de service d'interruptions.
L'instruction PUSH place une valeur dans la pile.

MNEMONIQUE	CODE
PUSH A	B8
PUSH B	C8
PUSH Rn	D8
PUSH ST	OE

RETI - Return from interrupt - Retour d'interruption

Syntaxe : RETI
Opération : charge le PC avec la valeur contenue au sommet de la pile diminue de 2 (PC sur 16 bits) le pointeur de pile, puis, charge le registre d'état avec la valeur contenue sur la pile, et, diminue de 1 le pointeur de pile.
Adressage : implicite.
Registre etat : chargé à partir de la valeur retirée de la pile.

Exemple : RETI

Observations : C'est la derniere instruction d'une routine de service d'une interruption. Elle restitue le registre etat à sa valeur d'avant interruption, puis, retourne au programme interrompu, là où il en etait au moment de l'inter-
-ruption.

MNEMONIQUE	CODE
RETI	OB

RETS - Return from subroutine - Retour de sous-programme

Syntaxe : RETS
Opération : charge le PC avec la valeur contenue au sommet de la pil
et diminue de 2 (PC sur 16 bits) le pointeur de pile.
Adressage : implicite.
Registre etat : non affecté.

Exemple : RETS

Observations : C'est la derniere instruction d'un sous-programme, son
exécution permet de revenir à l'instruction qui suit le
CALL.

MNEMONIQUE	CODE
RETS	0A

RL - Rotate left - rotation gauche

Syntaxe : RL destination
Opération : effectue une rotation à gauche d'un bit de destination,
tous les bits se décalent, le bit b7 passe dans carry et
dans b0 de destination.
Adressage : simple registre.
Registre etat : C égal au bit 7 de destination avant rotation
Z positionné selon le résultat de la rotation
N positionné selon le résultat de la rotation

Exemple : RL R102

Observations : Soit B qui contient >93, si RL B, apres l'opération, le
registre B contient >27

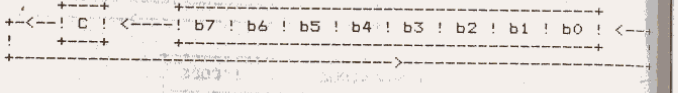
MNEMONIQUE	CODE
RL A	BE
RL B	CE
RL Rn	DE

RLC - Rotate left through carry- rotation gauche par la retenue

Syntaxe : RLC destination
Opération : effectue une rotation à gauche d'un bit de destination, tous les bits se décalent, carry passe dans le bit b0, le bit b7 passe dans carry.
Adressage : simple registre.
Registre etat : C égal au bit 7 de destination avant rotation
Z positionné selon le résultat de la rotation
N positionné selon le résultat de la rotation

Exemple : RLC R102

Observations : Soit B qui contient >93, et le bit C à 0, si RLC B, apres l'opération, le registre B contient >26 et C contient 1. La rotation gauche multiplie par 2, donc plusieurs rotations permettent de multiplier par des puissances de 2. Attention toutefois au contenu du bit Carry.



MNEMONIQUE	CODE
RLC A	BF
RLC B	CF
RLC Rn	DF

RR - Rotate right - rotation droite

Syntaxe : RR destination
Opération : effectue une rotation à droite d'un bit de destination, tous les bits se décalent, le bit b0 passe dans carry et dans b7 de destination.
Adressage : simple registre.
Registre etat : C égal au bit 0 de destination avant rotation
Z positionné selon le résultat de la rotation
N positionné selon le résultat de la rotation

Exemple : RR R102

Observations : Soit B qui contient >93, si RR B, apres l'opération, le registre B contient >C9 et le bit Carry est positionné

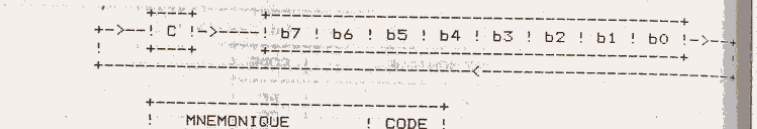
MNEMONIQUE	CODE
RR A	BC
RR B	CC
RR Rn	DC

RRC - Rotate right through carry- rotation droite par la retenue

Syntaxe : RRC destination
Opération : effectue une rotation à droite d'un bit de destination, tous les bits se décalent, carry passe dans le bit b7, le bit b0 passe dans carry.
Adressage : simple registre.
Registre etat : C égal au bit 0 de destination avant rotation
Z positionné selon le résultat de la rotation
N positionné selon le résultat de la rotation

Exemple : RRC R102

Observations : Soit B qui contient >93, et le bit C à 0, si RRC B, après l'opération, le registre B contient >49 et C contient 1. Si bit C à 0, RRC divise par 2 le contenu de destination vous pouvez donc diviser par des puissances de 2. Attention toutefois au bit Carry.



MNEMONIQUE	CODE
RRC A	BD
RRC B	CD
RRC Rn	DD

SBB - Subtract with borrow - Soustraction avec retenue

Syntaxe : SBB source,destination
Opération : destination - source - 1 + C -> destination
Adressage : double registre
Registre etat : C à 1 s'il n'y a pas de retenue
Z positionné selon le résultat
N positionné selon le résultat

Exemple : SBB %23,B soustrait 23 (décimal) de B, résultat dans B

Observations : SBB est utilisée pour les soustractions en complément à 2. Si valeur immédiate à 0, la soustraction équivaut à une diminution conditionnelle de la destination. Quand source = 0, si carry = 0, la destination est diminuée, sinon, pas de changement. Il y a retenue quand le résul--tat est négatif (dans ce cas carry = 1).

MNEMONIQUE	CODE
SBB Rn,A	1B
SBB %n,A	2B
SBB Rn,B	3B
SBB Rn,Rn	4B
SBB %n,B	5B
SBB B,A	6B
SBB %n,Rn	7B

SETC -Set carry bit - Mise à un du bit C

Syntaxe : SETC
Opération : mise à un du bit C du registre d'etat.
Adressage : implicite.
Registre etat : C à 1
Z à 1
N à 0

Exemple : SETC

Observations : SETC est utilisée pour mettre à un le bit carry avant une instruction arithmétique ou de décalage registre.

MNEMONIQUE	CODE
SETC	07

STA - Store A register - Stockage du registre A

Syntaxe : STA destination
Opération : place le contenu de A dans destination
Adressage : tous modes étendus.
Registre etat : C à 0
Z positionné selon la valeur déplacée
N positionné selon la valeur déplacée

Exemple : STA @Adresse STA @Adresse(B) STA *Registre

Observations : STA est utilisée pour stocker une valeur n'importe où en mémoire adressable. L'adressage direct permet d'accéder directement à une variable en mémoire. L'adressage indi-rect permet de gérer des tables, quant à l'adressage indexé, il permet de gérer des tables importantes. DNJZ associée à ces 2 derniers modes d'adressage, permet de créer des boucles de programme de gestion de table.

MNEMONIQUE	CODE
STA @n	8B
STA *Rn	9B
STA @n(B)	AB

STSP - Store stack pointer - Stockage du pointeur de pile

Syntaxe : STSP
Opération : place le contenu du pointeur de pile dans B.
Adressage : implicite.
Registre etat : non affecté

Exemple : STSP

Observations : STSP est utilisée pour obtenir une copie du pointeur de pile. Cette instruction peut etre utilisée pour connaitre la taille de la pile. L'adressage indexé permettra de lire des valeurs dans la pile, par exemple :

```
STSP
LDA  @>0000(B)  <- lecture sommet de la pile
LDA  @>FFFE(B)  <- lecture poste précédent
```

MNEMONIQUE	CODE
STSP	09

SUB - Subtract - Soustraction

Syntaxe : SUB source,destination
Opération : destination - source -> destination
Adressage : double registre
Registre etat : C à 1 si le résultat est positif ou nul
Z positionné selon le résultat
N positionné selon le résultat

Exemple : SUB R19,B soustrait contenu de B du contenu de R19 et place le résultat dans B

Observations : SUB est utilisée pour les soustractions en complément à 2.

MNEMONIQUE	CODE
SUB Rn,A	1A
SUB %n,A	2A
SUB Rn,B	3A
SUB Rn,Rn	4A
SUB %n,B	5A
SUB B,A	6A
SUB %n,Rn	7A

TRAP - Trap to subroutine - Appel vectorisé de sous-programme

Syntaxe : TRAP numéro (numéro compris entre 0 et 23)
Opération : sauvegarde le PC sur la pile, puis, augmente de 2 unités le pointeur de pile, et, saute au sous-programme dont l'adresse est spécifiée par le numéro qui suit TRAP.
Adressage : implicite.
Registre etat : non affecté.

Exemple : TRAP 1

Observations : l'opérande est un muméro de vecteur. La documentation du moniteur vous donne la liste des 'TRAP' utiles ainsi que leur action.
Description de la table des vecteurs :

```
>FFD0 (65488) ->  Adresse poids fort   TRAP 23
>FFD1 (65489) ->  Adresse poids faible TRAP 23
                          ---
                          ---
>FFFE (65534) ->  Adresse poids fort   TRAP  0
>FFFF (65535) ->  Adresse poids faible TRAP  0
```

MNEMONIQUE	CODE
TRAP 23	E8
TRAP 22	E9
TRAP 21	EA
TRAP 20	EB
TRAP 19	EC
TRAP 18	ED
TRAP 17	EE
TRAP 16	EF
TRAP 15	F0
TRAP 14	FA
TRAP 13	F2
TRAP 12	F3
TRAP 11	F4
TRAP 10	F5
TRAP 9	F6
TRAP 8	F7
TRAP 7	F8
TRAP 6	F9
TRAP 5	FA
TRAP 4	FB
TRAP 3	FC
TRAP 2	FD
TRAP 1	FE
TRAP 0	FF

TSTA - Test A register - Test du registre A

Syntaxe : TSTA
Opération : positionne N et Z en fonction du contenu de A.
Adressage : implicite.
Registre etat : C à 0
Z positionné selon le contenu de A
N positionné selon le contenu de A

Exemple : TSTA

Observations : TSTA positionne les bits du registre etat en fonction du contenu de A. Similaire à l'instruction CLRC.

MNEMONIQUE	CODE
TSTA	B0

TSTB - Test B register - Test du registre B

Syntaxe : TSTB
Opération : positionne N et Z en fonction du contenu de B.
Adressage : implicite.
Registre etat : C à 0
Z positionné selon le contenu de B
N positionné selon le contenu de B

Exemple : TSTB

Observations : TSTB positionne les bits du registre etat en fonction du contenu de B. Similaire à l'instruction XCHB B.

MNEMONIQUE	CODE
TSTB	C1

WVDP - Write to VDP - Ecriture en mémoire VDP

Syntaxe : WVDP source
Opération : Mouvement d'un octet de source en mémoire VDP.
Adressage : périphérique.
Registre etat : C à 0
Z positionné selon la valeur déplacée
N positionné selon la valeur déplacée

Exemple : WVDP A WVDP B WVDP %12

Observations : WVDP permet d'écrire en mémoire VDP. L'adresse où sera écrite la valeur dépend du TRAP B, TRAP 9, TRAP 10 qui précéde l'instruction VDP.
WVDP A est équivalent à MOVP A,P46
WVDP B est équivalent à MOVP B,P46
WVDP %12 est équivalent à MOVP %12,P46

MNEMONIQUE	CODE
WVDP A	82
WVDP B	92
WVDP %n	A2

XCHB - Exchange with B register - Echange avec le registre B

Syntaxe : XCHB destination
Opération : échange les contenus respectifs de B et de destination
Adressage : simple registre.
Registre etat : C à 0
Z positionné selon le contenu initial de B
N positionné selon le contenu initial de B

Exemple : XCHB A XCHB R3

Observations : XCHB est utilisée pour échanger le contenu d'un registre quelconque avec B, et, ce, sans zone intermédiaire.
XCHB B est identique à TSTB.

MNEMONIQUE	CODE
XCHB A	B6
XCHB B	C6
XCHB Rn	D6

XOR - Exclusive or - OU exclusif

Syntaxe : XOR source,destination
Opération : OU exclusif de source et destination -> destination
Adressage : double registre.
Registre etat : C à 0
Z positionné selon le résultat du OU exclusif
N positionné selon le résultat du OU exclusif

Exemple : XOR %>1,R2 (change d'état le bit b0 de R2).

Observations : XOR effectue un OU exclusif sur chacun des 8 bits des 2 opérandes.
Si un bit destination à besoin d'etre inversé, il suffit d'un XOR avec le bit correspondant du source à 1.

Table de décision du OU exclusif :

source	destination	destination (résultat)
0	0	0
0	1	1
1	0	1
1	1	0

MNEMONIQUE	CODE
XOR Rn,A	15
XOR %n,A	25
XOR Rn,B	35
XOR Rn,Rn	45
XOR %n,B	55
XOR B,A	65
XOR %n,Rn	75

XORP - Exclusif OR peripheral - OU exclusif périphérique

```
Syntaxe       : XORP source,destination
Opération     : OU exclusif de source et destination -> destination
Adressage     : registres périphériques.
Registre etat : C à 0
                Z positionné selon le résultat du OU exclusif
                N positionné selon le résultat du OU exclusif

Exemple       : XORP %>01,P6    (change d'état le bit 0 de P6 (port B))

Observations  : XORP inverse l'état des bits du registre périphérique en
                destination.
                Les bits à inverser seront mis à 1 dans source.
```

MNEMONIQUE	CODE
XORP A,Pn	85
XORP B,Pn	95
XORP %n,Pn	A5

8.5 LES MACRO-INSTRUCTIONS

8.5.1 Généralités

Le langage MACRO, permet de simplifier la programmation. Les définitions de macros permettent de stocker un groupe d'instructions, qui, sera restitué au moment de l'assemblage grace au nom de la macro placé dans le champ instruction.

Les définitions de macros peuvent etre stockées dans une bibliothéque des macros, puis, appelées au moment de l'assemblage par l'ordre COPY.

8.5.2 Les Macros 'in-line'

Se sont les plus simples, elles forcent l'assembleur à relire, sous le controle d'un compteur ou d'une liste de texte, le corps de la macro, et, à effectuer l'assemblage sur ce corps.
Ces macros se composent des groupes **REPT-ENDM**, **IRPC-ENDM** et **IRP-ENDM**.

Le groupe REPT-ENDM est écrit comme une séquence d'instructions assembleur, débutant par le pseudo opérateur **REPT**, et, se terminant par le pseudo opérateur **ENDM**. Sa forme est la suivante :

```
label   REPT    valeur
        instruction
        instruction
        .....
        ENDM
```

Le label est facultatif. La valeur non signée qui suit REPT représente le nombre d'expansions de la macro.

```
Exemple :                          BIDON  SET   0
                                          REPT  3
                                   BIDON  SET   BIDON+1
                                          BYTE  BIDON
                                          ENDM

Donnera à l'assemblage : 0000 0000    * BIDON  SET   0
                         0000         * BIDON  SET   BIDON+1
                         0000 01              BYTE  BIDON
                         0001         * BIDON  SET   BIDON+1
                         0001 02              BYTE  BIDON
                         0002         * BIDON  SET   BIDON+1
                         0002 03              BYTE  BIDON
```

Le groupe IRPC-ENDM provoque l'assemblage répétitif du corps de la macro sous le controle d'une liste de caractéres. Sa forme est la suivante :

```
label   IRPC    Identificateur,Liste de caractéres
        instruction
        instruction
        .....
        ENDM
```

Le label est facultatif. A chaque passage, à l'apparition de l'identi-
-ficateur dans le corps de la macro, MAX substitue le caractére associé.

Exemple :

```
                        IRPC  X,'AZERT '
LAB\X\                  BYTE  '\X\'
                        ENDM
                        IRPC  Y,FLASH
                        BYTE  '\Y\'+>80
                        ENDM
```

Donnera à l'assemblage :

```
0000 41        LABA   BYTE  'A'
0001 5A        LABZ   BYTE  'Z'
0002 45        LABE   BYTE  'E'
0003 52        LABR   BYTE  'R'
0004 54        LABT   BYTE  'T'
0005 40        LAB    BYTE  ' '
0006 C6               BYTE  'F'+>80
0007 CC               BYTE  'L'+>80
0008 C1               BYTE  'A'+>80
0009 D3               BYTE  'S'+>80
000A C8               BYTE  'H'+>80
```

Le groupe IRP-ENDM provoque l'assemblage répétitif du corps de la macr sous le controle d'une liste d'éléments. Sa forme est la suivante :

```
label   IRP     Identificateur,<el1,el2,el3,...eln>
        instruction
        instruction
        .....
        ENDM
```

Le label est facultatif. A chaque passage, à l'apparition de l'identi-
-ficateur dans le corps de la macro, MAX substitue la valeur élément jusqu'à la rencontre du caractére >.

Exemple :

```
                IRP   E,<INIT,GET,PUT,FIN>
                BR    @\E\
                ENDM
INIT            EQU   $
GET             EQU   $
PUT             EQU   $
FIN             EQU   $
```

Donnera à l'assemblage :

```
0000 8C000C           BR    @INIT
0003 8C000C           BR    @GET
0006 8C000C           BR    @PUT
0009 8C000C           BR    @FIN
000C 000C   = INIT    EQU   $
000C 000C   = GET     EQU   $
000C 000C   = PUT     EQU   $
000C 000C   = FIN     EQU   $
```


8.5.3 Les Macros mémorisées

Les macros de ce type permettent au programmeur de créer une séquence prototype, pour l'inclure à différentes places du processus d'assem-
-blage. Le traitement de macro consiste simplement à manipuler du texte au moment de l'assemblage.
Le prototype d'une macro stockée, est donné par le corps de la macro, précédé par le pseudo opérateur **MACRO**, et, terminé par **ENDM**.

Sa forme est la suivante :

```
nom de macro    MACRO p1,p2,p3...pn
                instruction
                instruction
                ....
                ENDM
```

Le nom de la macro est obligatoire car, il permet de retrouver la macro lors de l'appel.
Le corps de la macro est stocké dans la mémoire, son expansion n'étant effectuée que lors de l'appel, et ce, à la rencontre d'une instruction du type suivant :

nom de la macro p1,P2,p3...Pn

A chaque apparition des parametres, MAX les remplace par la valeur des parametres d'appel. Les parametres sont positionnels, la valeur d'un parametre omis n'est pas 0, mais, une chaine de longueur nulle.

L'opérateur **NUL**, employé comme dernier opérateur d'une expression, permet de tester un parametre vide. C'est un opérateur unaire produisant une valeur VRAI si son argument à une longueur 0.

MAX permet **l'imbriquation** des définitions de macros, seule, la taille de la mémoire peut limiter cette imbriquation.

Exemple simple d'écriture de macros :

```
AFF     EQU     >C800
DISPL   EQU     4
*
SAVE    MACRO
        PUSH    B
        PUSH    A
        ENDM
*
RESTOR  MACRO
        POP     A
        POP     B
        ENDM
*
WCHAR   MACRO   A
        MOV     %'\A\',A
        MOV     %DISPL,B
        CALL    @AFF
        ENDM
*
        SAVE
        WCHAR   H
        WCHAR   I
        RESTOR
        RETS
```

provoquera le généré suivant :

```
        PUSH    B
        PUSH    A
        MOV     %'H',A
        MOV     %DISPL,B
        CALL    @AFF
        MOV     %'I',A
        MOV     %DISPL,B
        CALL    @AFF
        POP     A
        POP     B
        RETS
```


Exemple d'utilisation de l'opérateur NUL :

```
          PARAM  MACRO   A
         *---> \A\ PARAMETRE(S)
                ENDM
         TSTNUL MACRO   P1,P2,P3
         CTR    SET     0
                IF      NOT NUL \P1\
                MOV     %\P1\,R121
         CTR    SET     CTR+1
                ENDIF
                IF      NOT NUL \P2\
                MOV     %\P2\,R122
         CTR    SET     CTR+1
                ENDIF
                IF      NOT NUL \P3\
                MOV     %\P3\,R123
         CTR    SET     CTR+1
                ENDIF
                NBPARAM %CTR
                ENDM
                TSTNUL  11,,13
                TSTNUL  12
                TSTNUL
                END
```

provoquera le généré suivant :

```
0000 0000    * CTR     SET     0
0000                   IF      NOT NUL 11
0000 720B79            MOV     %11,R121
0003 0001    * CTR     SET     CTR+1
0003                   ENDIF
0003                   IF      NOT NUL
0003                   MOV     %,R122
0003           CTR     SET     CTR+1
0003                   ENDIF
0003                   IF      NOT NUL 13
0003 720D7B            MOV     %13,R123
0006 0002    * CTR     SET     CTR+1
0006                   ENDIF
             * ---> 2 PARAMETRE(S)
0006 0000    * CTR     SET     0
0006                   IF      NOT NUL 12
0006 720C79            MOV     %12,R121
0009 0001    * CTR     SET     CTR+1
0009                   ENDIF
0009                   IF      NOT NUL
0009                   MOV     %,R122
0009           CTR     SET     CTR+1
0009                   ENDIF
0009                   IF      NOT NUL
0009                   MOV     %,R123
0009           CTR     SET     CTR+1
0009                   ENDIF
           * ---> 1 PARAMETRE(S)
```

```
0009 0000     * CTR     SET     0
0009                    IF      NOT NUL
0009                    MOV     %,R121
0009            CTR     SET     CTR+1
0009                    ENDIF
0009                    IF      NOT NUL
0009                    MOV     %,R122
0009            CTR     SET     CTR+1
0009                    ENDIF
0009                    IF      NOT NUL
0009                    MOV     %,R123
0009            CTR     SET     CTR+1
0009                    ENDIF
              * ---> 0 PARAMETRE(S)
0009                    END
```

Exemple d'appel d'une macro stockée dans une bibliothèque :

Soit la macro suivante (stockée sur CRAM)

```
DEBUT   MACRO   PGM,VERS,AUT,DATE
* ************************************
*       PROGRAMME : \PGM\
*       VERSION   : \VERS\
*       AUTEUR    : \AUT\
*       DATE      : \DATE\
* ************************************
        ENDM
```

Soit le programme suivant

```
TEST    EQU     $
        COPY    C:DEBUT.MAC
        DEBUT   'DEMOPROG','01.00','EXEL MAX','21/07/86'
        BYTE    '1'
        END
```

provoquera le généré suivant :

```
0000 0000 = TEST    EQU     $
0000                COPY    C:DEBUT.MAC
                  * ************************************
                  *       PROGRAMME : DEMOPROG
                  *       VERSION   : 01.00
                  *       AUTEUR    : EXEL MAX
                  *       DATE      : 21/07/86
                  * ************************************
0000 31             BYTE    '1'
0001                END
```


Exemple de définition imbriquée de macro :

```
        LOCAL   MACRO   A
        CLAB    SET     0
        LOCAL   MACRO   B
        \B\     SET     CLAB
        CLAB    SET     CLAB+1
                ENDM
                LOCAL   \A\
                ENDM
        *
        LABEL   MACRO   A
        L\A\
                ENDM
        *
        INSTR   MACRO   PART1,PART2
                \PART1\L\PART2\
                ENDM
        *
        TIME    MACRO   SECOND
                LOCAL   BIDON
                INSTR   'BR     @',%BIDON
        TIMER
                .....
                RETS
                LABEL   %BIDON
        TIME    MACRO   A
                MOVD    %\A\ SHL 4,TEMPO
                CALL    @TIMER
                ENDM
                TIME    \SECOND\
                ENDM

                TIMER   4
                TIMER   2
```

provoquera le généré suivant :

```
                BR      @L0
        TIMER
                .....
                RETS
        L0
                MOVD    %4 SHL 4,TEMPO
                CALL    @TIMER
                MOVD    %2 SHL 4,TEMPO
                CALL    @TIMER
```

La macro **TIME** permet d'attendre un certain laps de temps. Afin de limiter le généré, le 1 er appel à cette macro créera une routine ainsi que le branchement à cette routine, tandis que les appels ultérieurs ne généreront que le branchement.
Les macros **LOCAL**, **LABEL**, et, **INSTR** permettent de définir des labels locaux.

8.5.4 Conventions d'évaluation des parametres des Macros

Les espaces précédant le parametre sont défaits.

Le premier caractére du parametre est évalué.

Si le premier caractére est un indicateur de chaine ', le parametre prend la valeur de la chaine, quotes exclues (sauf double ').

Si le premier caractére est <, le parametre prend la valeur de la chaine jusqu'à la rencontre de >.

Si le premier caractére est % l'identificateur qui suit est recherché dans la table des symboles, et, sa valeur, traduite sous forme de chaine de caractéres est prise par l'argument.

LE MODULE OBJET ISSU DE L'ASSEMBLAGE

Le fichier issu de l'assemblage est un fichier ASCII éditable par l'EDITEUR de MAX, il se compose de TAGs et de données provenant de votre programme.

Si vous avez à effectuer une correction mineure dans votre programme, vous pouvez modifier directement le résultat de l'assemblage, ce qui vous permet de ne pas réassembler, et donc de gagner du temps.

Attention toutefois à ne pas oublier de corriger le source en prévision d'un futur assemblage.

Soit le source assemblé suivant :

```
0000                        IDT   'DEMOPROG'
0000 1F42      = ADRESS     EQU   >1F42
0000 0000      = NUL        EQU   0
0000 5200                   MOV   %NUL,B
0002 8A1F42                 LDA   @ADRESS
0005 AD0010                 CMPA  ZON(B)
0008 E201                   JEQ   LABL
000A 000A      = FIN        EQU   $
000A 0A                     RETS
000B 000B      = LABL       EQU   $
000B AB0010                 STA   ZON(B)
000E E0FA                   JMP   FIN
0010 010203      ZON        BYTE  1,2,3
0013                        END
```

Voici le contenu du fichier résultant de l'assemblage :

```
K0013DEMOPROGA0000B5200*8AB1F42*ADC0010B
E201*0A*ABC0010BFFFFF          MAX1.6
BE0FA*01B02038FFFFF
                               MAX1.6
:   MAX Version 1.6  @1986 EXELvision
```

Liste des tags :

K xxxx	Longueur du programme
A xxxx	Adresse relative du programme (par rapport au début)
* xx	Octet à charger
B xxxx	Mot à charger
C xxxx	Mot à reloger puis à charger
7 xxxx	Checksum à vérifier
8 xxxx	Checksum à ne pas vérifier
F	Fin de ligne
:	Fin de module

A noter apres les 4 octets indiquant la longueur du programme, le nom du module (8 octets) tel qu'il a été donné par **IDT.** Si le pseudo-opérateur IDT est absent, le module aura comme nom : **NO$IDT.**

TABLE DE CONVERSION DECIMAL-HEXADECIMAL

Utilisez la table ci-aprés pour convertir des nombres décimaux en nombres hexadécimaux et inversement.

Conversion décimal -> hexadécimal :

```
14000 = >36B0    car   14000
                     - 12288 ->  >3
                       -----
                     =  1712
                     -  1536 ->  >6
                       -----
                     =   176
                     -   176 ->  >B
                       -----
                     =     0 ->  >0
```

Conversion hexadécimal -> décimal :

```
>36B0 = 14000    car   12288 ->  >3
                     +  1536 ->  >6
                     +   176 ->  >B
                     +     0 ->  >0
                       -----
                     = 14000
```


3		2		1		0	
H	DEC	H	DEC	H	DEC	H	DEC
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	960	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3076	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

TABLEAU RECAPITULATIF DES INSTRUCTIONS DU TMS 7000

CODE	SYNTAXE	TYPE	PAGE
ADC	ADC source,destination	arithmétique	22
ADD	ADD source,destination	arithmétique	23
AND	AND source,destination	logique	24
ANDP	ANDP source,destination	périphérique	25
BR	BR destination	branchement	26
BTJO	BTJO source,destination,déplacement	branchement	27
BTJOP	BTJOP source,destination,déplacement	périphérique	28
BTJZ	BTJZ source,destination,déplacement	branchement	29
BTJZP	BTJZP source,périphérique,déplacement	périphérique	30
CALL	CALL destination	branchement	31
CLR	CLR destination	logique	32
CLRC	CLRC	controle	33
CMP	CMP source,destination	comparaison	34
CMPA	CMPA source	comparaison	35
DAC	DAC source,destination	arithmétique	36
DEC	DEC destination	arithmétique	37
DECD	DECD destination	arithmétique	38
DINT	DINT	controle	39
DJNZ	DJNZ source,déplacement	branchement	40
EINT	EINT	controle	41
IDLE	IDLE	controle	42
INC	INC déplacement	arithmétique	43
INV	INV déplacement	arithmétique	44
JC	JC déplacement	branchement	45
JEQ	JEQ déplacement	branchement	45
JGE	JGE déplacement	branchement	45
JGT	JGT déplacement	branchement	45
JHS	JHS déplacement	branchement	45
JL	JL déplacement	branchement	45
JLT	JLT déplacement	branchement	45
JMP	JMP déplacement	branchement	45
JN	JN déplacement	branchement	46
JNC	JNC déplacement	branchement	45
JNE	JNE déplacement	branchement	45
JNZ	JNZ déplacement	branchement	45
JP	JP déplacement	branchement	45
JPZ	JPZ déplacement	branchement	45
JZ	JZ déplacement	branchement	45
LDA	LDA source	branchement	45
LDSP	LDSP	mouvement	47
LDVP	LDVP	mouvement	48
MOV	MOV source,destination	périphérique	49
		mouvement	50

(1/2)

CODE	SYNTAXE	TYPE	PAGE
MOVD	MOVD source,destination	mouvement	51
MOVP	MOVP source,destination	périphérique	52
MPY	MPY source,destination	arithmétique	53
NOP	NOP	controle	54
OR	OR source,destination	logique	55
ORP	ORP source,destination	périphérique	56
POP	POP destination	mouvement	57
PUSH	PUSH source	destination	58
RETI	RETI	branchement	59
RETS	RETS	branchement	60
RL	RL destination	décalage	61
RLC	RLC destination	décalage	62
RR	RR destination	décalage	63
RRC	RRC destination	décalage	64
SBB	SBB source,destination	arithmétique	65
SETC	SETC	controle	66
STA	STA destination	mouvement	67
STSP	STSP	mouvement	68
SUB	SUB source,destination	arithmétique	69
TRAP	TRAP numéro	branchement	70
TSTA	TSTA	comparaison	71
TSTB	TSTB	comparaison	72
WVDP	WVDP source	périphérique	73
XCHB	XCHB destination	mouvement	74
XOR	XOR source,destination	logique	75
XORP	XORP source,destination	périphérique	76

(2/2)

EXEMPLES DE PROGRAMMES 'ASSEMBLEUR'

Tri bulle d'une table de 150 octets

```
FLAG    EQU     R2              Si égal à 1 => échange de caractéres
*
SORT    CLR     FLAG            Remise à zéro de l'indicateur d'échange
        MOV     %149,B          Registre B sur dernier caractére de table
LOOP1   LDA     @TABLE(B)       Chargement d'un caractére dans A
        CMPA    @TABLE-1(B)     Comparaison avec le précédent
        JL      LOOP2           Si plus petit => saut
        INC     FLAG            1 dans l'indicateur
        PUSH    A               sauvegarde de A sur la pile
        LDA     @TABLE-1(B)     Echange des 2 caractéres (1ere partie)
        STA     @TABLE(B)       -----
        POP     A               Récupération de la valeur sauvegardée
        STA     @TABLE-1(B)     Echange des 2 caractéres (2eme partie)
LOOP2   EQU'    $
        DNJZ    B,LOOP1         Boucle jusqu'à ce que la table soit triée
        BTJO    %>FF,FLAG,SORT  Si échange => re-balayer la table
```


Multiplication de 2 nombres de 16 bits chacun

```
*                           XH      XL          X multiplicande
*                    X      YH      YL          Y multiplicateur
*                    ------------------
*                           XLYLm   XLYL1       l = lsb (poids faible)
*                    XHYLm  XHYL1               m = msb (poids fort)
*                    XLYHm  XLYH1
*             + XHYHm XHYH1
*           ---------------------------------
*              RSLT3  RSLT2  RSLT1  RSLT0
*
XH      EQU     R2          Opérande supérieur de X
XL      EQU     R3          Opérande inférieur de X
YH      EQU     R4          Opérande supérieur de Y
YL      EQU     R5          Opérande inférieur de Y
RSLT3   EQU     R6          Poids fort du résultat final
RSLT2   EQU     R7
RSLT1   EQU     R8
RSLT0   EQU     R9          Poids faible du résultat final
*
MPY32   CLR     ACC2        Effacement valeur actuelle
        CLR     ACC3
        MPY     XL,YL       Multiplication des poids faibles
        MOV     B,RSLT0     Stockage poids faible dans R0
        MOV     A,RSLT1     Stockage poids fort   dans R1
        MPY     XH,YL       Obtention de XHYL
        ADD     R1,RSLT1    Ajout au résultat actuel XLYL
        ADC     R0,RSLT2    Ajout de carry si présent
        MPY     XL,YH       Multiplication pour obtenir XLYH
        ADD     R1,RSLT1    Ajout au résultat actuel XLYL+XHYL
        ADC     R0,RSLT2    Ajout aux résultats existants et carry
        ADC     %0,RSLT3    Ajout si carry présent
        MPY     XH,YH       Multiplication des poids forts
        ADD     R1,RSLT2    Ajout une fois encore au registre résultat
        ADC     R0,RSLT3    Effectuer la derniere addition au reg. résultat
```

EXEMPLES DE PROGRAMMES 'ASSEMBLEUR'

Tri bulle d'une table de 150 octets

```
FLAG    EQU     R2              Si égal à 1 => échange de caractéres
*
SORT    CLR     FLAG            Remise à zéro de l'indicateur d'échange
        MOV     %149,B          Registre B sur dernier caractére de table
LOOP1   LDA     @TABLE(B)       Chargement d'un caractére dans A
        CMPA    @TABLE-1(B)     Comparaison avec le précédent
        JL      LOOP2           Si plus petit => saut
        INC     FLAG            1 dans l'indicateur
        PUSH    A               sauvegarde de A sur la pile
        LDA     @TABLE-1(B)     Echange des 2 caractéres (1ere partie)
        STA     @TABLE(B)       -----
        POP     A               Récupération de la valeur sauvegardée
        STA     @TABLE-1(B)     Echange des 2 caractéres (2eme partie)
LOOP2   EQU'    $
        DNJZ    B,LOOP1         Boucle jusqu'à ce que la table soit triée
        BTJO    %>FF,FLAG,SORT  Si échange => re-balayer la table
```


Multiplication de 2 nombres de 16 bits chacun

```
*                           XH      XL          X multiplicande
*                    X      YH      YL          Y multiplicateur
*                    ------------------
*                           XLYLm   XLYL1       l = lsb (poids faible)
*                    XHYLm  XHYL1               m = msb (poids fort)
*                    XLYHm  XLYH1
*             + XHYHm XHYH1
*           ---------------------------------
*              RSLT3  RSLT2  RSLT1  RSLT0
*
XH      EQU     R2          Opérande supérieur de X
XL      EQU     R3          Opérande inférieur de X
YH      EQU     R4          Opérande supérieur de Y
YL      EQU     R5          Opérande inférieur de Y
RSLT3   EQU     R6          Poids fort du résultat final
RSLT2   EQU     R7
RSLT1   EQU     R8
RSLT0   EQU     R9          Poids faible du résultat final
*
MPY32   CLR     ACC2        Effacement valeur actuelle
        CLR     ACC3
        MPY     XL,YL       Multiplication des poids faibles
        MOV     B,RSLT0     Stockage poids faible dans R0
        MOV     A,RSLT1     Stockage poids fort   dans R1
        MPY     XH,YL       Obtention de XHYL
        ADD     R1,RSLT1    Ajout au résultat actuel XLYL
        ADC     R0,RSLT2    Ajout de carry si présent
        MPY     XL,YH       Multiplication pour obtenir XLYH
        ADD     R1,RSLT1    Ajout au résultat actuel XLYL+XHYL
        ADC     R0,RSLT2    Ajout aux résultats existants et carry
        ADC     %0,RSLT3    Ajout si carry présent
        MPY     XH,YH       Multiplication des poids forts
        ADD     R1,RSLT2    Ajout une fois encore au registre résultat
        ADC     R0,RSLT3    Effectuer la derniere addition au reg. résultat
```

EXEMPLES DE PROGRAMMES 'ASSEMBLEUR'

Tri bulle d'une table de 150 octets

```
FLAG    EQU     R2              Si égal à 1 => échange de caractéres
*
SORT    CLR     FLAG            Remise à zéro de l'indicateur d'échange
        MOV     %149,B          Registre B sur dernier caractére de table
LOOP1   LDA     @TABLE(B)       Chargement d'un caractére dans A
        CMPA    @TABLE-1(B)     Comparaison avec le précédent
        JL      LOOP2           Si plus petit => saut
        INC     FLAG            1 dans l'indicateur
        PUSH    A               sauvegarde de A sur la pile
        LDA     @TABLE-1(B)     Echange des 2 caractéres (1ere partie)
        STA     @TABLE(B)       -----
        POP     A               Récupération de la valeur sauvegardée
        STA     @TABLE-1(B)     Echange des 2 caractéres (2eme partie)
LOOP2   EQU'    $
        DNJZ    B,LOOP1         Boucle jusqu'à ce que la table soit triée
        BTJO    %>FF,FLAG,SORT  Si échange => re-balayer la table
```


Multiplication de 2 nombres de 16 bits chacun

```
*                           XH      XL          X multiplicande
*                    X      YH      YL          Y multiplicateur
*                    ------------------
*                           XLYLm   XLYL1       l = lsb (poids faible)
*                    XHYLm  XHYL1               m = msb (poids fort)
*                    XLYHm  XLYH1
*             + XHYHm XHYH1
*           ---------------------------------
*              RSLT3  RSLT2  RSLT1  RSLT0
*
XH      EQU     R2          Opérande supérieur de X
XL      EQU     R3          Opérande inférieur de X
YH      EQU     R4          Opérande supérieur de Y
YL      EQU     R5          Opérande inférieur de Y
RSLT3   EQU     R6          Poids fort du résultat final
RSLT2   EQU     R7
RSLT1   EQU     R8
RSLT0   EQU     R9          Poids faible du résultat final
*
MPY32   CLR     ACC2        Effacement valeur actuelle
        CLR     ACC3
        MPY     XL,YL       Multiplication des poids faibles
        MOV     B,RSLT0     Stockage poids faible dans R0
        MOV     A,RSLT1     Stockage poids fort   dans R1
        MPY     XH,YL       Obtention de XHYL
        ADD     R1,RSLT1    Ajout au résultat actuel XLYL
        ADC     R0,RSLT2    Ajout de carry si présent
        MPY     XL,YH       Multiplication pour obtenir XLYH
        ADD     R1,RSLT1    Ajout au résultat actuel XLYL+XHYL
        ADC     R0,RSLT2    Ajout aux résultats existants et carry
        ADC     %0,RSLT3    Ajout si carry présent
        MPY     XH,YH       Multiplication des poids forts
        ADD     R1,RSLT2    Ajout une fois encore au registre résultat
        ADC     R0,RSLT3    Effectuer la derniere addition au reg. résultat
```

TABLEAU DES MESSAGES D'ERREURS A L'ASSEMBLAGE

MESSAGE	SIGNIFICATION
String too long	Chaine trop longue, tronquée
Unmatched < >	Il n'y a pas autant de < que de >
Unmatched ( )	Il n'y a pas autant de (que de)
Expression too complex	Expression trop complexe
Invalid expression	Expression non conforme
Invalid operator	Opérateur non conforme
Operator missing	Opérateur absent
Syntax error	Erreur de syntaxe
Undefined symbol	Symbole non défini
Symbol table overflow	La table des symboles est trop petite
Symbol too long	Symbole trop grand, tronqué
Too many operand	Trop d'opérandes
Operand missing	Manque opérande(s)
Invalid operand	Opérande non conforme
Displacemnent too big	Déplacement trop grand
Invalid index	Le registre index est obligatoirement (B)
Duplicate definition	Définition multiple
Pass 1/Pass 2 conflict	Erreur due à des erreurs précédentes
Too many IF	Trop de IF
Too many ELSE	Trop de ELSE
Too many ENDIF	Trop de ENDIF
Too many ENDM	Trop de ENDM
Unknown instruction	Instruction inconnue (ou MACRO inconnue)
ENDM missing	Manque ENDM
ENDIF missing	Manque ENDIF
Memory full	La mémoire VDP est pleine