sphynx

SPL

macro-assembleur

SPL mode d'emploi

PREFACE

En premier lieu nous vous souhaitons bonne chance avec votre achat du SPL et nous espérons que vous pourrez l'utiliser d'une manière plaisante et productive. Pour le cas où vous n'auriez encore aucune ou peu d'expérience avec le SPL voici une courte description.

SPL est un assembleur pour l'ordinateur individuel DAI qui permet aux utilisateurs d'appliquer les définitions macro et l'assemblage conditionnel. Les définitions macro ("Macro's") rendent possible dans un programme en assembleur, de désigner un groupe d'instructions par un mot et l'assemblage conditionnel permet d'adapter rapidement un programme pour plusieurs buts, par exemple dans l'emploi ou non du processeur arithmétique (AMD-9511) par la modification d'une seule instruction.
SPL (le nom est dérivé de System Programming Language) a été développé par nos soins durant l'été 1982. Initialement, le SPL était purement destiné à notre usage particulier. Nous reçûmes, alors que le SPL s'approchait de l'achèvement, des réactions tellement enthousiastes des personnes qui l'avaient vu en fonctionnement, que nous décidâmes de l'exporter. Pour des circonstances personnelles, il ne nous est pas possible d'effectuer nous-mêmes les activités commerciales, nous avons donc cédé les droits de diffusion à DAInamic. Avant que le SPL ne soit émis par DAlnamic, nous l'avons d'abord testé nous-même en pratique détaillée et nous l'avons adapté sur la base des expériences acquises.

Le plus grand soin et la plus grande attention ont été déployés pour le SPL. Nous n'acceptons aucune responsabilité pour d'éventuels dommages produits par l'usage. Les questions et remarques concernant le SPL sont à envoyer aux programmeurs. L'adresse de contact : Werkgroep Sphynx, p/a Gebr. Rens, Grote sloot lei, 1754 JC Burgerbrug

Many thanks to J.P. Malliën for the translations and final redaction of this manual.

MISE EN MARCHE

Vous démarrez le SPL de la manière suivante :

1) Allumez le DAI, le moniteur (ou équivalent).

2) Que vous utilisiez les cassettes classiques ou bien les DCR, les opérations sont identiques. Il suffit de passer en Utility (UT ret) et de lire le programme (R ret). Si après quelques instants le curseur ne s'arrête pas de clignoter, alors vous feriez mieux d'exécuter un CHECK pour voir ce qui se passe.

3) Quand le curseur recommence à clignoter et que le caractère guide le l'Utility (">") apparaît à l'écran sans être suivis par un point d'interrogation (autrement cela signifierait qu'il y a un problème du coté chargement), tapez Z3 (ret) G8500 (ret). Sur l'écran apparaît maintenant le titre du SPL et une adresse de départ est demandée. Cette adresse indique le début de la portion mémoire que le SPL va utiliser pour stocker le programme en assembleur et ne peut être plus grande que 83FF(H). La commande E(dit) utilise environ 525 bytes supplémentaires, pour pouvoir utiliser cette commande supplémentaire, l'adresse de départ doit donc être au maximum de 81F0(H). L'adresse est tapée sous forme hexadécimale et est suivie d'un (ret). Si au lieu d'entrer une adresse, vous tapez directement sur (ret) alors l'adresse par défaut sera utilisée. Elle est de 400(H) dans la version standard SPL V1.1.

4) Vous pouvez maintenant entrer les commandes SPL. Si vous quittez le SPL (les commandes U et u) et que vous redémarrez par UT (ret) Z3 (ret) G8500 (ret), alors apparaît à la place de la demande d'adresse un menu de l'assembleur (system : "S"), du code source (program : "P") et de la table des symboles (labels : "L") intact ou non (respectivement "OK" ou "BAD").

Si vous ne comprenez pas les termes utilisés, ouvrez la liste explicative des mots du SPL.

N.B. : les adresses ci-dessus sont valables exclusivement pour la version standard du SPL (V1.1). Si vous avez reçu une autre version, alors il se peut que d'autres adresses soient valables. C'est alors clairement mentionné.

INTRODUCTION

L'entrée des commandes utilise la routine d'entrée à l'écran du BASIC. De cette manière il est possible d'enlever des caractères avec par exemple (DEL CHAR) ou d'annuler des commandes envoyées par un (BREAK) si celui-ci est tapé avant le (RET). Le caractère guide du SPL (le 'prompt') consiste en 4 caractères : "SPL>". Pendant l'envoi des commandes, le curseur consiste en un "&" clignotant. Si vous appuyez sur (RET) alors le curseur devient un "-" ou un bloc noir pour les routines de lecture et d'écriture.

Le premier caractère de la ligne de commande envoyé indique la commande proprement dite. Une espace à cet endroit indique que c'est une commande DCR qui suit. Après la lettre de commande un ou plusieurs opérandes peuvent suivre. Ceux-ci peuvent être mutuellement (ou de la lettre de commande) séparés par autant d'espaces qu'il est souhaité. Avec les commandes R- (ou Y-) l'offset éventuel vous sert cependant à faire suivre directement le R (ou le Y) donc sans espace, parce que autrement l'offset pourrait être considéré comme une partie du nom. Du reste les noms des fichiers sources de la mémoire de masse ne commencent jamais avec des espaces, parce que les espaces d'introductions des noms sont générés par la routine d'entrée. Le nom commence donc toujours après le premier caractère suivant l'(es) espace(s), ou après le R, le W ou le Y

A l'introduction d'une commande qui attend un numéro de ligne, vous pouvez aussi taper un nom à la place de ce numéro de ligne. Le SPL cherche dans les lignes d'assembleur où se trouve ce nom pour la première fois dans la colonne LABEL et agit comme si le numéro de cette ligne était introduit.

Pour les commandes qui se rapportent à des noms dans la colonne opérande (les commandes G- et X-) vous pouvez aussi entrer le label dollar ($) à la place de ce nom.

Bien que l'éditeur attende que chaque opérande commence par un chiffre et se termine par un B-,D-,H- ou O-, la routine d'entrée pose moins d'exigences sévères. A cela près qu'une opérande numérique ne doit pas être suivie par un B-,D-,H- ou O- parce qu'avec une commande déterminée seules des opérandes décimales ou hexadécimales peuvent être utilisées et il est seulement nécessaire de faire commencer une opérande hexadécimale par un chiffre (par ex. 0) si à cette place, un nom pourrait être aussi utilisé (les commandes I-,.l- et X-

La plupart des listings sont plus rapides que l'œil ne peut suivre. Si vous appuyez sur une touche quelconque durant le listing, alors celui-ci est interrompu et le curseur devient un "~" clignotant. Ceci dure jusqu'à ce que l'on appuie sur (BREAK) ou sur (SPACE). Pour le stopper il est préférable de ne pas utiliser le (BREAK) ou le (SPACE). Le (BREAK) retourne vers l'entrée de commandes. le (SPACE) cherche à poursuivre le listing.

Ensuite en ce qui concerne la lecture de fichiers-sources dans la mémoire de masse, ce qui avec des cassettes classiques peut absorber quelque temps, un signal sonore est donné. Ce signal peut être arrêté par 'BREAK'. La composition de ce signal sonore se trouve dans la table IMPLEMENT.

LES COMMANDES DU SPL

La plupart des exemples se rapporte au fichier source RESTORE XXXX (voir cassette) dans lequel on suppose que le buffer source commence en 5000(H). (Par la commande J par exemple)
Les commandes C,A,K,R et Y vous demandent confirmation. Dans ce cas vous voyez apparaître " +? " sur l'écran. Pour confirmer, vous devez presser " + ", La pression d'une autre touche provoque l'annulation de la commande et le message - BREAK - apparaît à l'écran.

C Provoque un "cold start", presque comme si le SPL était à nouveau chargé et relancé. Si vous ne faites pas suivre cette commande d'un " + " elle sera ignorée. Le SPL vous demande une nouvelle adresse de départ pour le buffer source et les pointeurs sont remis à zéro (l'ancien contenu du buffer source n'est dès lors plus accessible). Si vous pressez directement sur - RETURN - à la place de la nouvelle adresse de départ, SPL utilisera par défaut l'adresse standard 400(H).

J adres0 Permet d'ajuster l'adresse du début du buffer source. Le code source est déplacé de telle façon que son début coïncide avec adres0. C'est pourquoi, l'espace mémoire utilisé par le SPL peut être limité ou juste augmenté afin de permettre l'édition de plus de lignes en même temps.

Z Cette commande imprimera successivement :

a. L'adresse de début du buffer source ('B')
b. L'adresse de la fin du buffer source ('E')
c. L'adresse du début de la table des symboles ('S')
Si cette table est vide, cette adresse est plus haute que l'adresse de fin le cette même table.
d. L'adresse de fin ou 'topadres' de la table des symboles ('T').
e. L'adresse de début des codes source et table des symboles combinés en vus d'une sauvegarde sur disquettes, dcr,… ('W').
f. L'adresse la plus basse de sauvegarde avec laquelle un fichier source à joindre peut être écrit ('M').
g. Le nombre de bytes encore disponibles dans le buffer source ('A').

Par exemple : La commande Z imprime pour RESTORE XXXX ceci :

B=5000
E=50F4
S=8380
T=83FF
W=828C
M=5174
A=328C

E[ligne1][ligne2] : donne l'ordre d'envoyer une partie déterminée du programme vers l'éditeur. Si les lignes 1 et 2 sont égales, un buffer d'édition vide apparaîtra. Si la ligne 2 est plus grande que la ligne 1, alors les lignes allant de la ligne 1 à la ligne 2 sont envoyées vers le buffer d'édition. La ligne 2 est exclue parce que cette disposition est plus claire lors de l'utilisation de noms de label ou d'identificateurs au lieu d'un numéro de ligne. Un nom indiquera en effet le plus souvent le début d'une nouvelle partie du programme ou un sous-programme, donc il ne sera pas nécessaire de le prendre dans la plupart des cas. Ceci notamment avec les commandes M et m. Si un numéro de ligne manque, s'il est égal à zéro ou bien plus grand que le numéro de la dernière ligne du programme, alors il est remplacé par le dernier numéro de ligne du programme + 1.
ex: E4455
Pour sortir de l'éditeur il suffit de taper - BREAK - puis - SPACE -.

L ligne1 [ligne2]: liste les lignes à partir de la ligne 1 à la ligne 2 dans une forme identique à celle montrée dans l'éditeur, mais précédées de numéros de ligne. Les règles de syntaxe pour cette commande sont les mêmes que celles de la commande E.

T Provoque un listing objet. Ce listing est imprimé comme dans l'éditeur mais avec deux colonnes supplémentaires :

1. La colonne adresse où est imprimée l'adresse où se situera le pointeur après exécution de la commande précédente

2. La colonne byte. Suivant le mnémonique 1 à 3 bytes sont imprimés. Il en est de même pour les directives DW et DB à l'exception près que la directive DB imprime au plus les trois premiers bytes dans la colonne byte. Pour les directives SET et EQU, une valeur est inscrite sur la 2^{e} et 3^{e} position, valeur qui est attribuée à un label par la directive dans l'ordre highbyte puis lowbyte alors que sur la première position on a deux caractères standards (@=). Les autres directives n'influencent pas cette colonne.

Le listing objet est orienté objet ce qui signifie que les lignes sont listées dans l'ordre suivant lequel le code objet serait organisé en mémoire suite à la commande A. Dans la version standard, les directives suivantes ne sont pas imprimées durant un listing objet :
IF, ELSE, ENDIF, call-macro, MACRO, MEND, contrôle et ### .
Après une directive UNL, le listing est suspendu tandis que la commande T continue à traiter le code source. La directive LST rétablit l'affichage du listing. Le listing objet s'arrête à la directive END.
Quand la construction ENDIF ELSE est utilisée seule la partie vérifiée est listée.

B Convertit toutes les adresses de 16 bits en binaire. Les adresses seront listées sous forme binaire jusqu'à ce que la conversion des adresses soit provoquée par une commande D, H ou O ou selon la directive PUT.

b Convertit tous les bytes en binaire. Les bytes seront listés sous forme binaire, jusqu'à interruption par les commandes d, h ou o, ou selon la directive PUT.

O Travaille comme B, mais l'échange des adresses se fait en octal. Le SPL utilise la notation octale continue contrairement à d'autres assembleurs. Ainsi FFF8 sera imprimé 1777770 et non 3773770.

o Travaille comme b, mais la conversion est octale.

D Travaille comme B, mais la conversion est décimale

d Travaille comme b, mais la conversion est décimale

H Travaille comme B, mais la conversion est hexadécimale

h Travaille comme b, mais la conversion est hexadécimale.

V Inverse le flag de hardcopy, appelle la routine d'initialisation de l'imprimante. Dans la version standard, le message 'READY?' apparaît sur l'écran. Cela vous permet d'allumer l'imprimante et quand tout est prêt, vous devez presser 'SPACE'. La pression d'une autre touche provoque l'annulation de l'ordre. Si le flac de hardcopy lors du listing convient, les caractères envoyés à l'écran sont également envoyés à l'imprimante.
Attention la commande V n'influence ni l'indicateur de ligne ni le compteur de page.

v Met, si le flag hardcopy convient, l'indicateur de ligne et de page à zéro. Cette commande vous permet de commencer un listing à partir de la page 1. Vous devez laisser tourner le papier lui-même ou le faire tourner par la commande P.

P byte0 : envoie byte0 à l'imprimante, par exemple pour pouvoir commencer le début du hardcopy avec un caractère défini. L'envoi des bytes 0C(H) et 0D(H) influence l'indicateur de ligne et de page.

P A envoie un caractère ASCII vers l'imprimante qui correspond à un linefeed.

p NB0: Donne le nombre de lignes par pages pour le hardcopy, NB0 est un nombre décimal, plus petit ou égal à 127, plus grand ou égal à 0. Un 0 provoque un listing ininterrompu, donc sans titre ni mise en page.

Q: Trie la table des symboles suivant l'ASCII. Elle est normalement construite dans l'ordre où l'éditeur croise les nouveaux noms lors de la traduction des textes du buffer d'édition dans le code source. La commande Q provoque la remise en ordre et enlève les noms qui n'ont pas été évoqués. La commande Q utilise la mémoire écran, ce qui explique les effets graphiques lors de la commande et que le nombre des noms de la table des symboles est limité à 1642(D). Cela semble être un espace suffisant. Le SPL contrôle lui-même si la table des symboles n'est pas trop grande.

S: Liste les noms dans la table des symboles suivant l'ordre où ils se trouvent avec les dernières valeurs qui ont été attribuées à ces noms. La commande S ne comporte pas de phase de placement (voir commande A).
S liste ceci pour RESTORE XXXX (le flag de hardcopy éteint) :

```
ADMTHL: DE39 CMPLNP: 031F COUNTR: 0000 DATAC : 0123 DATAD : 0124
DEC   : FFFF EIND  : 0327 EPRORM: D9F5 FINDLN: 02FC FPSTLN: 0306
NEX   : 0000 RESTOR: 02EE TXTBGN: 029F TYPE  : FFFF getbvt: 50EA
loop  : 5074
```

Attention : Lors d'un hardcopy, 6 noms au lieu de 5 sont listés. De plus, il n'y pas un mais deux espaces entre les noms.

F: Liste les labels suivant l'ordre dans lequel les valeurs ont été attribuées de la même manière que pour la commande S. Les identificateurs ne sont pas listés et les labels autant de fois que des valeurs leur ont été attribuées (ou pas du tout s'ils se trouvent dans la partie non utilisée d'un IF…).
Ex : le fichier source RESTORE XXXX donne :

```
DEC   :FFFF HEX   :0000 TYPE  :FFFF DATAC :0123 DATAD :0124
TXTBGN:029F ADMTHL:DE39 ERROPM:D9F5 RESTOR:02EE COUNTR:0002
COUNTR:0001 COUNTR:0000 FINDLN:02FC FRSTLN:0306 CMPLNP:031F
EIND  :0327
```

La commande F donne un listing objet plus simplifié et peut servir pour faire une description de la mémoire (memorymap) à l'avantage de l'instruction L de l'Utility.

l nom0: Liste toutes les lignes où le nom0 se trouve dans la colonne label précédée par l'adresse mémoire dans laquelle le byte d'état de la ligne est écrit. Vous pouvez utiliser cette commande de différentes manières quand le programme objet a endommagé le code source, pour trouver l'adresse où la ligne abîmée commence et permettre ainsi d'essayer de la rétablir en Utility après, par exemple, une commande S.
ex: ICOUNTR liste pour RESTORE XXXX :

```
506E 21 COUNTR SET 2H
506A 24 COUNTR SET COUNTR-1(H)
```

Voir aussi le chapitre "Debugging pour programmeur avancé"

l adres0 Liste la première ligne où le pointeur objet (pour le listing objet, la première colonne) est plus grand ou égal à adres0. Vous pouvez par exemple utiliser cette commande pour, après avoir testé en Utility le programme assemblé avec l'aide de l'instruction L, voir quelles lignes sources conviennent à une adresse définie.
ex : liste pour RESTORE XXXX : I 300

```
0300      33   ORA
```

i Imprime les noms de la table des symboles, précédé du numéro de ligne où ils se situent dans la colonne label, dans l'ordre dans lequel ils se trouvent dans cette table. Vous pouvez utiliser un tel listing comme référence pour un L-listing.
ex : i imprime pour RESTORE XXXX (le flag hardcopy étant éteint):

```
 8 ADMTHL 49 CMPLNR 21 COUNTR 24 COUNTR   5 DATAC
 6 DATAD   2 DEC    55 EIND    9 ERROPM  31 FIMFLN
36 FRSTLN  3 HEX    18 RESTOR  7 TEXTBGN  4 TYPE
56 getbyt 22 loop
```

G nom0[NB1][NB2): Imprime toutes les lignes où se trouve le nom0 dans la colonne opérande ou comme identificateur dans la colonne instruction. Les NB1 et NB2 indiquent respectivement le nombre de lignes précédant et suivant la ligne contenant le nom0, qui doivent être listées. Elle vous permet de voir dans quel contexte on utilise le nom0. De cette manière, il est possible de voir si certaines lignes sont répétées plusieurs fois. Si les NB1 et NB2 sont égaux à 0, les lignes seront listées de la même manière que lors de la commande I nom0, donc précédées par l'adresse mémoire où le byte d'état se situe à la place de nom0, on peut introduire $.
ex : G getbyt £ imprime pour RESTORE XXXX :

```
46              POP P
47              POP PSW
48              RET
49   CMPLNP     getbyt
50              COMP B
51              RNZ
52              getbyt
```

g Imprime les noms de la table des symboles précédés par le numéro de ligne où elle se situe dans la colonne opérande (ou pour les identificateurs dans la colonne instruction), de la même manière que la commande i. Vous pouvez utiliser un tel listing comme cross-référence, ce qui sera surtout utile pour la recherche des fautes.
ex : g imprime pour RESTORE XXXX (le hardcopy flag étant éteint) :

```
31 ADNTHL 37 CMPLNR 24 COUNTR 25 COUNTR 44 DATAC
42 DATA   10 DEC    35 ERRORM 41 ERRORM 39 FINDLN
30 FRSTLN  9 TXTBGN 10 TYPE   49 getbyt 52 getbyt
25 loop
```

l instruction0: Imprime toutes les lignes dans le programme source où une instruction définie est utilisée.
I END imprime pour RESTORE XXXX :

```
55   EIND       END
```

l byte0: Imprime toutes les lignes où L'instruction de code byte est utilisée. Parce que les 'remarques', les 'call-macro' et les 'directives de contrôle' n'ont pas été introduites avec l'aide d'une combinaison déterminée de caractères, nous ne pouvons trouver ces instructions que par leur code instruction.
Ces codes sont :

- 30 le contrôle
- 5B le call-macro
- FD la remarque (doit être introduit par : 0FD)

ex : l 30 imprime pour RESTORE XXXX :

```
22   loop       COUNTR>0H
```

x nom1 nom2: Permet de changer les noms de label et d'identificateur à la seule condition qu'un label reste un label et de même pour les identificateurs. Tous les renvois dans le code source au nom1 sont remplacés par nom2 (qui est d'abord éventuellement pris dans la table des symboles).

X label0 adres0 Change les appels au label0 dans la colonne opérande par la constante adres0.

X adres0 label0 Change les constantes adres0 dans la colonne opérande par l'appel au label0 après éventuellement avoir placé label0 dans la table des symboles.

X adres0 adres1 Change dans la colonne opérande la constante adres1 par la constante adres2. La commande X peut être surtout utile pour le changement de labels standard en labels plus pratiques lors d'un programme produit par le désassembleur SPL. A la place de label1 ou de label2 vous pouvez aussi utiliser $.

N Liste les figures où figure un nom qui n'est utilisé nulle part dans la colonne opérande ni comme identificateur dans la colonne instruction. De tels noms ne posent pas de problèmes en eux-mêmes, mais il se pourrait qu'ils déclarent une sous-routine qui n'est plus utilisée et qui dès lors, peut être effacée du programme ou bien une sous-routine qui peut être utilisée mais qui ne l'est plus guère suite à une erreur.
ex : N imprime pour RESTORE XXXX

```
 3   HEX       EQU       0(H)
18   RESTOR    PUSH PSW
55   EIND      END
```

M ligne1 ligne2 ligne3 : Copie les lignes 1 à 2 juste devant la ligne 3. Les lignes 1 et 2 restent à leurs anciennes places. Vous pouvez utiliser cette commande pour copier une suite semblable d'instruction mais avec par exemple, une utilisation différente des registres pour limiter l'éditeur.
Attention la ligne 3 ne peut se situer entre la ligne 1 et la ligne 2.
M CMPLNR EIND FINDLN imprime :

avant:

```
30              JMP       FRSTLN
31   FINDLN     CALL      ADMTHL
32   …          …
48              RET
49   CMPLNP     getbyt
50              CMP B
51              RNZ
52              getbyt
53              CMP C
54              RET
55   EIND       END
```

après:

```
30              JMP       FRSTLN
31   CFMLNR     getbyt
32              CMP P
33              RNZ
34              getbyt
35              CMP C
36              RET
37   FINDLN     CALL      ADMTHL
38   …          ...
54              RET
55   CMPLNP     getbyt
56              CMP B
57              RNZ
58              getbyt
59              CMP C
60              RET
61   EIND       END
```

^: A le même effet que mode0 en basic. L'écran glisse vers le haut jusqu'à ce que le curseur se trouve sur la quatrième ligne.

m ligne1 ligne2 ligne3: fait plus ou moins la même chose que la commande M mais, en plus, elle efface les lignes déplacées de leur ancienne place. Cela se produit après le déplacement, donc la mémoire demandée est la même que pour la commande M. Vous pouvez l'utiliser pour réorganiser le code source après, par exemple, la réunion de deux programmes (commande Y), et remettre ainsi toutes les définitions Macro's ensembles.

un exemple:
m CMPLNR EIND FINDLN

Listing avant la commande m

```
30         JMP     FRSTLN
31 FINDLN  CALL    ADMTHL
32 ...     ...
48         PET
49 CMPLNR  getbyt
50         CMP B
51         RNZ
52         getbyt
53         CMP C
54         RET
55 EIND    END
```

Listing après la commande m

```
30         JMP     FRSTLN
31 CMPLNR  getbyt
32         CMP B
33         RNZ
34         getbyt
35         CMP C
36         RET
37 FINDLN  CALL     AUITHL
...        ...
54         RET
55 EIND    END
```

K ligne1 [ligne2]: Efface les lignes de la ligne 1 à la ligne 2. Parce que ici le code source est perdu la commande demande d'abord confirmation. Les règles de syntaxe sont les même que pour la commande E. Nous vous conseillons d'utiliser la commande K pour supprimer de grandes zones de programme, mais pour effacer un petit nombre de lignes, il est préférable d'utiliser l'éditeur parce que pour ce dernier, vous disposez d'un contrôle visuel pour l'effacement. De plus, les lignes anciennes sont plus faciles à rappeler (BREAK) (←).

```
ex avant K 15 16          après K 15 16
14    ELSE                14     ELSE
15    PUT       "H"       15     ORG    2EE
16    ORG       2EE
```

W: Sauve le fichier source sur DCR, disquette, après avoir formé un bloc compact du code source et de la table des symboles. Afin de pouvoir à nouveau les séparer après la lecture, un pointeur est placé en fin de la table des symboles, pointeur qui indique le début du code source. Après la sauvegarde le programme résident reste inchangé. Parce que, seules les tables de symboles triées peuvent être jointes (voir commande Y), seules les fichiers sources avec la table des symboles triée sont sauvés (voir commande Q).
Ceci est contrôlé par le SPL.

R adres0 [chaîne0]: Lit le fichier source sauvé sur DCR, disquette. L'adres0 donne un offset grâce auquel même les fichiers source qui ont été sauvegardés par un SPL avec une autre adresse de sommet de table des symboles (voir la commande Z) peuvent être lus. Pour cela, l'offset doit prendre soin que la fin du fichier source aboutisse à l'adresse du sommet de table des symboles.
Si aucune chaîne0 n'accompagne l'instruction, le fichier suivant sur la cassette ou la disquette sera lu. Si au contraire, on tape une chaîne0 seule le fichier source dont le nom correspond sera lu.
Comme nous l'avons décrit précédemment pour la commande W, un fichier source est écrit suivant un bloc compact ou la table des symboles et le code source sont accolés, avec deux adresses de début des deux zones. Si cette dernière adresse (table des symboles) est plus petite que l'adresse de début du buffer, une mention d'erreur s'affichera. Si ce n'est pas le cas, le code source est chargé en bas, jusqu'à ce que le début de celui-ci coïncide avec l'adresse de début du buffer, la commande R supprime l'éventuel programme (code source et table des symboles) déjà présent dans le buffer. Si l'on veut empêcher cela, il faut utiliser la commande Y.

Y adres0 [chaîne0]: Sert pour la réunion du code source présent dans le buffer et d'un fichier source qui est lu dans la mémoire de masse. Les tables de symboles de ces deux programmes sont automatiquement réunies les noms communs sont réduits à un seul. Pour cela il est nécessaire que les deux tables soient triées (voir commande Q). En ce qui concerne adres0 et chaîne0 voir commande R.

$ Cette commande est utilisée pour aligner les pointeurs des opérandes $ après avoir chargé un fichier source d'une version SPL avec une autre adresse de départ. La commande vous demande cette adresse avec 'START'. Si vous devez lire un fichier source avec un offset en rapport avec une autre adresse de sommet de table de symbole, vous devez taper cet offset après l'adresse de départ. En rapport avec ceci, si vous utilisez une version SPL avec une autre adresse de départ que 8500(H), vous devez l'annoncer expressément lors de l'échange des fichiers sources. (De même pour une adresse de table de symbole différente de 83FF(H)). Quand vous utilisez la commande $, vous pouvez faire cela après le chargement. Vous ne pouvez pas utiliser la commande $ pour un fichier source qui a été lu par la commande Y. Vous devez dans ce cas, d'abord charger le fichier source, ensuite le rectifier par la commande $ et à nouveau le sauver.

dcr: Si l'instruction commence par une espace, ce qui suit est interprété comme une commande DCR.

?: Cette commande imprime les fautes de structure dans le code source du début jusqu'à la fin, ou jusqu'à ce qu'un message d'erreur apparaisse. Contrairement aux autres routines, celle-ci imprime seulement le code de l'erreur, la commande ? indique également après le code, la ligne où une erreur a été constatée ou par #MP# la ligne appelée. Nous vous conseillons de toujours précéder une commande A par une commande ? et une commande N.
Si vous placez dans RESTORE XXXX, à la ligne 56, après Macro, l'opérande X, alors la commande ? imprimera ce qui suit:

```
MP    49 CMPLNR        getbyt
```

Voir aussi le chapitre séparé sur la description des fautes.

A [adres0]: Cette commande donne l'ordre au SPL d'assembler le code source (les mnémoniques) en code objet, au terme duquel, les bytes du programme objet seront placés dans la mémoire. L'assemblage est divisé en trois phases :
- La phase de recherche, dans laquelle se passent de nombreuses fautes de contrôle dans le but d'une plus grande vitesse de travail pour les deux autres phases. Dans cette phase prend place en même temps le contrôle de la construction du programme (voir le chapitre concerné).
- La phase de composition durant laquelle le SPL établit les champs de valeurs dans la table des symboles
- La phase de mise en place durant laquelle les codes objets sont générés (et par cette commande mis en place dans la mémoire).

Vu que chaque 'call-macro' effectue une phase de composition et une phase de mise en place pour sa propre 'Définition Macro' aussi bien que pour la phase de composition et de mise en place du bloc appelé (le programme principal d'une autre Macro), le temps d'assemblage peut être considérablement allongé par l'emploi de beaucoup de macro's.
Un exemple pour éclaircir:

```
          ORG  400(H)
          DB   1H
          alpha
          END
alpha     MACRO
          DB   2H
          beta
          MEND
beta      MACRO
          DB   3H
          MEND
```

Lors d'une phase de composition de alpha une phase de composition et de placement sera exécutée pour beta. La même chose se passe pendant la phase de mise en place de alpha. Vu que alpha parcourt deux fois aussi bien la phase de composition que celle de mise en place, en liaison avec la phase de composition et de mise en place du programme principal,

beta parcourt donc quatre fois sa phase de composition et de mise en place.
Si aucune autre adres0 (offset) n'est donnée à la commande A, alors le code objet se retrouve aux adresses telles que durant le listing en d'autres mots, le programme se trouve alors à sa place et devrait, s'il est écrit sans faute, pouvoir être démarré en Utility avec par exemple la commande GXXXX. Si un offset est ajouté, alors celui-ci est compté pendant la phase de mise en place avec les opérandes des directives ORG. Cela a pour effet qu'un byte du code objet qui d'abord aurait du être placé à l'adresse X se trouve maintenant à l'adresse X + offset sans que les bytes changent eux-mêmes. Le programme objet peut être replacé par la commande M de l'Utility à une distance comparable à un offset de 0(H) et se trouve donc près pour être testé.
Un exemple chiffré : si le programme objet (et le listing objet) part suivant les directives ORG de 2EE(H) à 326(H) et est assemblé avec un offset de 1000(H), alors après l'assemblage, il se trouve dans la mémoire aux adresses 12EE(H) à 1326(H) et peut être remis à sa place en Utility par M 12EE(H) 1326(H) 2EE(H). Vous pouvez aussi sauver le code objet en mémoire de masse et le relire avec un offset négatif. Dans l'exemple : UT ret W12EE 1326 ret suivi par RF000, ce qui est un peu plus compliqué.
L'assemblage avec un offset est principalement désigné pour placer des codes objets, qui normalement devraient être mis à un endroit où l'assembleur, le code source ou la gestion de l'ordinateur se trouve l'endommageraient, ailleurs dans la mémoire où il ne peut y avoir de problèmes, jusqu'à ce que l'assemblage soit sauvé. Pendant l'assemblage le code objet peut éventuellement être écrit sur le code source, pourvu que aucun code objet ne se place à des adresses où se trouve le code source qui doit encore être assemblé. Habituellement, il y a plus de bytes du code source transformé que de bytes de code objet produit, mais vous devez faire spécialement attention avec les ORG, DS, les 'call-macro' et les contrôles.Vous devez aussi tenir compte du fait que cette surimpression détruit la source.

U : S'occupe d'un saut vers l'utility. Des checksums de l'assembleur, du code source et de la table des symboles sont rédigés et contrôlés lors du redémarrage du SPL.

u : S'occupe d'un saut vers le Basic. Ici aussi des checksums sont rédigés.

LISTE ALPHABETIQUE DES COMMANDES DU SPL

_ : Commande DCR
$: Adaptation des pointeurs d'un fichier d'une autre version SPL
? : Listing des erreurs
^ : Mode 0
A : Assemblage
B : Listing des adresses en binaire
b : Listing des bytes en binaire
C : Cold start
D : Listing des adresses en décimal
d : Listing des bytes en décimal
E : Edit
F : Structure de la mémoire (Labels placés dans l'ordre suivant l'attribution des valeurs)
G : Impression des lignes où se trouve un nom déterminé
g : Imprime les noms de la table des symboles avec leur numéro de ligne
H : Listing des adresses en hexadécimal
h : Listing des bytes en hexadécimal
I : Cherche un nom dans la colonne label
i : Liste les noms avec le numéro de ligne où ils se trouvent
J : Ajuste l'adresse de début du buffer source
K : Supprime des lignes du buffer
L : Listing normal
l : Liste les lignes où un nom spécifique est utilisé
M : Copie des lignes
m : Déplace des lignes
N : Liste les noms non utilisés
O : Listing des adresses en octal
o : Listing des bytes en octal
P : Envoie un byte (vers l'imprimante)
p : Donne le nombre de lignes par page pour un hardcopy
Q : Trie la table des symboles
R : Lit un fichier dans la mémoire de masse
S : Liste les noms avec leur dernière valeur
T : Listing objet
U : Utility
u : Basic
V : Pointeur de hardcopy
v : Initialise le titre et le hardcopy
W : Sauvegarde un fichier en mémoire de masse
X : Change des noms et/ou des constantes
Y : Relie des fichiers ensembles
Z : Montre les pointeurs

L'EDITEUR

Suite à la commande E, les noms éventuels de la ligne de commande sont traduits en numéro de ligne. Ensuite de l'espace est réservé dans le buffer source au profit du buffer d'édition. Puis la partie à éditer est placée sous forme ASCII dans le buffer d'édition. Si la partie à éditer ne s'adapte pas bien au buffer d'édition, alors l'ancienne situation est rétablie et est renvoyée avec le message 'BREAK' vers les commandes d'entrée. (Cela se passe aussi si une ligne sous la forme ASCII compte plus de 224(D) caractères comme cela peut arriver lors du listing binaire d'un 'call-macro' avec beaucoup de constantes dans la colonne opérande).
Si la partie à éditer s'adapte bien au buffer d'édition, alors vous pouvez façonner le texte comme dans l'EDIT du Basic. Examinez pour cela le manuel d'utilisation du DAI. Après le travail du texte vous pouvez quitter l'éditeur en appuyant sur la touche 'BREAK'. SPL efface l'écran et attend que la touche suivante soit frappée :

'RETURN' ni le code source, ni le buffer d'édition ne sont modifiés et on retourne vers l'éditeur. Vous pouvez l'utiliser si par exemple vous avez appuyé sur 'BREAK' par accident.

'←' La situation précédant la commande E est rétablie. Le contenu du buffer d'édition est donc ignoré. Ceci est à comparé avec le 'BREAK' 'BREAK' de l'EDIT du Basic. La pression sur ces touches n'a aucun effet si une faute de syntaxe est constatée, donc quand le curseur clignote avec une mention de faute.

'SPACE' Le texte de l'édit buffer est placé en code source, éventuellement avec des compléments à la table des symboles. A le constatation d'une faute de syntaxe, la ligne avec la faute apparaît dans le buffer d'édition en haut de l'écran, et comme symbole de curseur, clignote un caractère qui donne le type de faute. Ceci est répété jusqu'à ce que la faute soit corrigée.

SPL utilise la routine EDIT de la rom du DAI, à ceci près qu'il y est fait utilisation d'une routine de tabulation de telle sorte que la pression du 'TAB' fait sauter le curseur à une position de caractère déterminée. Quelle position ? cela dépend de la place précédente du curseur et de la table de tabulation utilisée. Le 'TAB' abandonne à l'ancienne place du curseur une petite flèche vers la droite. Tout comme vous ne pouvez placer aucun caractère à la droite d'un 'RETURN', vous ne pouvez pas non plus placer de caractère entre le 'TAB' et la colonne vers laquelle il renvoie.
Le curseur pendant la routine d'édition n'est pas un caractère habituel (comme '-'), mais un changement entre la couleur de fond et de premier plan. En standard les couleurs 8(D) et 12(D) sont utilisées.
L'écran est comme réellement divisé en quatre colonnes. Celles-ci sont de gauche à droite : la colonne LABEL, la colonne INSTRUCTION, la colonne OPERANDE et la colonne COMMENTAIRE. Chaque colonne si elle est utilisée est fermée par un 'TAB' ou un 'RETURN'

La colonne LABEL

Dans la colonne label, une des quatre combinaisons suivantes peut être placée :

1. Un point virgule, éventuellement suivi par d'autres caractères. Ceci indique une remarque et n'est pas intéressant pour le SPL. En fait, ici les trois premières colonnes tombent, de sorte que la colonne commentaire reste seule (comme si elle était listée sur toute la largeur de la ligne).
2. Un identificateur : il est utilisé pour indiquer une place déterminée dans le programme assembleur, comme la place d'une définition macro ou une adresse d'entrée pour une structure de boucle.

par ex : loop ###

3. Un label : indique une valeur par exemple, une adresse dans le programme objet, une valeur de constante déterminée ou un pointeur pour les directives conditionnelles.

par ex: COUNTR SET COUNTR-1

4. rien. Ceci est un point d'arrêt où il n'est pas fait utilisation des trois autres possibilités.

par ex: END

La colonne instruction

Dans la colonne instruction une des quatre combinaisons suivantes peut être placée :

1. Un mnémonique standard de l'INTEL 8080 (à l'exception de IN, OUT, MOV, R,R, ...) Attention dans la colonne instruction ne peut se trouver aucune opérande. Si vous voulez par exemple charger l'accumulateur avec 0D(H), ceci est le plus souvent indiqué dans les autres assembleurs par MVI 4,0D(H). Auprès du SPL cependant, 'MVI A' vient dans la colonne instruction suivi d'un 'TAB' et de '0D(H) dans la colonne opérande. Comme règle de base, vous pouvez retenir que seule la représentation du premier byte d'une instruction en langage machine peut être placée dans la colonne instruction.

par ex: MVI A 0D

2. Une directive SPL : voyez pour cela le chapitre sur les directives de l'assembleur. Ici , non plus les opérandes ne se trouvent pas dans la colonne instruction.

par ex: PUT "D"

3. Un identificateur : l'utilisation se divise en deux groupes :
   a. Les appels macros : dans l'éventuelle colonne opérande se trouvent les valeurs qui devront être prises lors de l'appel et attribuées au label après la directive macro. Ces opérandes s'évaluent donc comme une ou plusieurs valeurs 16 bits numériques.

par ex : getbyt

   b. Les instructions de contrôle : dans la colonne opérande se trouve une comparaison qui livre une condition 'vraie' ou 'fausse' (Flag).

par ex : loop COUNTR>0(H)

La colonne opérande

Ce qui peut se trouver ici dépend de ce qui se trouve dans la colonne label et dans la colonne instruction.

1. Après les mnémoniques qui indiquent une instruction en langage machine de 1 byte aucune opérande n'est permise

par ex : STC

2. Après les mnémoniques qui indiquent une instruction en langage machine de 2 bytes peuvent se trouver dans la colonne opérande :
   a. Une combinaison de labels nombres et opérateurs qui livre après évaluation une valeur inférieure ou égale à 0FF(H) et supérieure ou égale 0(H)

par ex : MVI A .0(H)-COUNTR&0FF(H)

   b. 'X' par lequel X représente un caractère quelconque
      Ceci livre, dans le programme objet, un byte avec comme valeur, la valeur ASCII du caractère.

par ex : CPI 'X'

   c. "X" par lequel X représente un caractère quelconque.
      Ceci livre, dans le programme objet, un byte avec comme valeur 80(H) augmentée de la valeur ASCII du caractère

par ex : SUI "$"

3. Après des mnémoniques qui indiquent une instruction en langage machine de 3 bytes peut se trouver une combinaison de labels, nombres et opérateurs.

par ex : COUNTR SET COUNTR-1(H)

4. Ce qui peut se trouver après une directive SPL dépend de cette directive. Voir pour cela le chapitre concerné.

par ex : ORG 750(d)

Ce qui se trouve dans la colonne opérande peut éventuellement continuer dans la colonne commentaire.

La colonne commentaire.

La colonne commentaire doit toujours commencer par un point virgule. Ce qui se trouve derrière ce point virgule sera considéré comme commentaire et ignoré par le phase d'assemblage. Comme limitation le commentaire ne peut tenir seulement que ce que les 'tabs' peuvent contenir s'ils forment des sous-parties d'une remarque (un point virgule au début de la colonne label).
Si l'éditeur refuse de placer plus de caractères à l'écran, et que cela n'est pas causé par la position du curseur, alors il est probable que le buffer d'édition est plein. La solution est de quitter l'éditeur ou de supprimer un certain nombre de caractères.

MNEMONIQUES

Le SPL utilise les mnémoniques standard de l'INTEL 8080 à l'exception de MOV R,R (ex MOV A,A) et IN et OUT qui ne sont pas définis. Si vous désirez cependant les utiliser, alors vous pouvez les introduire avec l'aide de DB.

LES DIRECTIVES DE L'ASSEMBLEUR

Toutes les directives peuvent être précédées d'un label sauf : les remarques. Pour EQU et SET, il doit figurer un label et pour ### et macro, un identificateur. Pour la clarté les labels inutilisés sont effacés par les règles de syntaxe. Le pointeur objet est augmenté de 2 par DW, par DB du nombre de bytes que comporte le code objet, par DS de la valeur de l'opérande. Le pointeur objet est placé par ORG et n'est affecté par aucune autre directive.

```
                    ORG     valeur0
ex:                 ORG     300(H)
```
ORG place le pointeur objet à la valeur 300
Si un label est placé devant ORG, celui-ci a également cette valeur0

```
                    END
ex:           EIND  END
```
END arrête le programme

```
                    DB      valeur0
ex:                 DB      ESC
```
Sous cette forme, DB donne un byte qui doit être placé dans le programme objet sur l'adresse donnée par le pointeur objet. Pour cela, la valeur de valeur0 doit être comprise entre 0(H) et 0FF(H) (mis sur un byte).

```
                    DB      byte1 (,byten)
ex                  DB      0A0(H),10(H),7(H)
```
Sous cette forme, DB donne une suite de bytes qui dans le programme objet viennent successivement dans la mémoire à partir de l'adresse donnée par le pointeur objet.

```
                    DB      'chaîne0'
ex:       STR001    DB      'ASCII'
```
Sous cette forme, DB travaille plus ou moins de la mème manière que ci-dessus, seulement une suite de caractères définis par chaîne0 est placée en mémoire sous forme de bytes avec le MSB = 0.

```
                    DB      "chaîne0"
ex :                DB      "TEXTE"
```
Sous cette forme, DB travaille de la même façon que ci-dessus, mais le MSB = 1.

```
                    DW      val0
ex:       VAL.XY    DW      BUFFER+OFFSET
```
DW définit une valeur 16 bits qui vient dans le programme objet, en mémoire de la manière suivante : le byte bas est placé à l'adresse indiquée par le pointeur objet et le byte haut à cette adresse +1 (analogue à SHLD)

```
                    DS          val0
ex :                DS          ALPHA       FF(H)
```

DS augmente le pointeur objet de la valeur indiquée par val0. Il est pour cela nécessaire que cette valeur ait déjà été définie lors de la phase d'assemblage (phase qui consiste à placer la valeur dans la table des symboles)

```
                    label0      EQU         val0
 ex :               OUTPUT      EQU         0D695(H)
```

EQU met la valeur val0 dans label0. Ici aussi il est important que cette valeur ait été définie lors de la phase d'assemblage.

```
                    Label0      SET         val0
ex :                TELLER      SET         $+1
```

SET travaille de la même manière que EQU, seulement label0 peut se trouver plusieurs fois dans la colonne label sans entraîner une 'erreur de double définition'. SET est en fait la seule instruction qui peut être utilisée quand un label apparaît deux fois dans la colonne label. Vous pouvez utiliser SET entre autre comme préparation pour IF et pour les call-macro afin de pouvoir donner des opérandes supplémentaires.

```
                    Ident0      MACRO [label1] ([.labeln])
ex :                alpha       MACRO A,B
```

MACRO désigne le début d'une définition MACRO. Chaque fois que SPL, lors de l'assemblage rencontre ident0 dans la colonne instruction, les instructions entre MACRO et MEND de ident0 MEND sont assemblées. Toutes les définitions MACRO doivent être situées après le dernier END dans le programme assembleur.

```
                    Ident0      [val0] ([.valn)]
ex :                            0ca70(H),BUFFER+OFFSET
```

C'est un appel macro (call-macro). Les valeurs de val0 à valn sont attribuées par une sorte de SET à des labels qui forment des opérandes macro. Le nombre de valeurs doit être égal au nombre de labels.

```
                    MEND
ex :                RETURN      MEND
```

MEND indique la fin d'une définition MACRO. Quand le SPL rencontre, lors de l'assemblage d'une macro, une directive MEND, alors il retourne aux instructions qui suivent l'appel macro.

```
                            IF      flag0
ex:                 LH1200  IF      $=TELLER
```

Si le flag0 est 'vrai', les instructions entre IF et son ENDIF correspondant (ou ELSE) sont également exécutées lors de l'assemblage, les éventuelles instructions entre ELSE et ENDIF ne le seront pas. Si le flag0 est 'faux', le contraire se produit.

```
                            ELSE
ex :                        ELSE
```

ELSE donne la séparation entre ce qui doit être assemblé si le flag0 après le IF correspondant est 'vrai' (les instructions entre IF et ELSE) ou ce qui doit être assemblé dans le cas contraire (les instructions entre le ELSE et ENDIF)
ELSE n'est pas absolument nécessaire.

```
                            ENDIF
ex :                        ENDIF
```

ENDIF donne la fin de la zone après le IF correspondant, zone qui suivant le falg0 doit être assemblée ou non.

```
                    ident0 ###
ex :                contrl ###
```

("entrée") indique un point dans le programme assembleur où le programme doit reprendre après l'exécution d'un contrôle.

```
                    Ident0  flag0
ex :        START   contrl  X"Z
```

Cette instruction ('contrôle') poursuit, si le flag0 est 'vrai', l'assemblage des instructions qui suivent le ### devant lequel se situe le ident0. Si le flag est 'faux', les instructions qui suivent le contrôle sont assemblées. Le contrôle se distingue du call-macro par le fait que l'opérande évalue un flag (flag0) et pas une ou plusieurs valeurs numériques. Tandis que le call-macro présente une certaine ressemblance avec l'appel de sous-routine, le contrôle est comparable à un saut conditionnel. Le contrôle peut seulement faire un saut en arrière, donc le ### correspondant doit se trouver plus tôt dans le code source. Si le contrôle fait partie d'une définition macro, alors le saut en arrière ne peut se faire plus loin que le MACRO de cette définition. Attention : il est possible, comme dans les programmes en langage machine, par une mauvaise utilisation du contrôle, de créer une boucle sans fin. Le SPL ne peut contrôler cela, donc vous devrez le faire vous-même.

```
                    TITL   'chaîne0'
ex :        PART2   TITL   'SPL par SPHYNX'
```

Dans un 'hardcopy', SPL imprime, en début de chaque page à partir de la colonne label, la chaîne0 qui se trouvait dans la colonne opérande du dernier TITL (après une commande v, une chaîne vide est utilisée, jusqu'à ce qu'un nouveau TITL ne soit donné).

```
                    UNL
ex :                UNL
```

UNL supprime, lors d'un listing objet, l'envoi de caractères vers l'écran et/ou vers l'imprimante, et ce jusqu'au moment où SPL rencontre un LST.

```
                    LST
ex :                LST
```

LST rétablit, lors d'un listing objet, l'envoi de caractères vers l'écran et/ou l'imprimante. En association avec UNL, il peut être utilisé pour n'imprimer qu'une partie définie du programme.

```
                    PUT    "char0"
ex :                PUT    "0"
```

"Char" consiste soit en : "B", "b", "0", "o", "D", "d", "H", "h". PUT rétablit ici les lettres "d'échange" pour les adresses et les bytes, exactement comme si l'une de ces commandes avait été donnée. Le but de cette directive est surtout de faciliter la compréhension des valeurs numériques lors d'un listing.

```
                    PUT    byte0
ex :                PUT    60(D)
```

Sous cette forme, il établit le nombre de lignes par page lors d'un "hardcopy". Ce nombre de lignes est donné par byte0, dans lequel 0(D) <= byte0 <= 127(D). Si cette valeur est 0(D), on fait un listing continu (donc sans form feed, titres, etc...).
Le nombre de lignes par pages n'entre en ligne de compte qu'à partir de la page suivante, donc la page présente continue avec l'ancienne valeur. Si vous voulez commencer avec une nouvelle page, vous pouvez le faire avec un PRT.

```
                PRT   byte1 (,byten)
ex :            PRT   0C(H),0D(H>

ou              PRT   'chaîne0'
ex :            PRT   'Nouvelle page'

ou              PRT   "chaîne0"
ex :            PRT   "CERTAINES IMPRIMANTES TRAVAILLENT AVEC MSB=1"
```

Lors de la routine de 'hardcopy', quand le SPL rencontre un PRT, il envoie à l'imprimante les bytes de l'opérande. Grâce à cela on peut donner des ordres spéciaux à l'imprimante, par exemple un formfeed supplémentaire (donné dans certains assembleurs par 'page') ou une lettre d'un autre type. Pour cela, examinez d'abord le manuel d'instruction de votre imprimante.

Attention : si le PRT envoie un 0C(H) ou un 0D(H) il influence le compteur de ligne et éventuellement le compteur de pages. Un byte tel 0A(H) n'a ici aucune influence.

```
                ; (chaîne0)
ex :            ; Main Program
```

Si un point virgule se trouve dans la première position de la colonne label, alors le reste de la ligne est considéré comme une remarque et ignoré par la plupart des routines. Dans une remarque, peuvent figurer des 'TAB' contrairement aux commentaires après les autres instructions

LES OPERATEURS

Les opérateurs exécutent des calculs sur les résultats des labels, nombres, et opérateurs qui sont à leur gauche (à partir du dernier alternateur, comparateur, virgule ou tab), indiqué par X, et le label ou nombre qui se trouve à leur droite, indiqué par Y. Tous les opérateurs donnent un résultat sur une valeur 16 bits (sans que l'on tienne compte du signe positif ou négatif). Un dépassement éventuel est donc ignoré

```
X # Y : OU logique entre X et Y
X % Y : OU exclusif de X et Y
X & Y : ET logique de X et Y
X ' Y : X modulo Y
X ( Y : X déplacement vers la gauche de Y (rempli à droite de 0, Y<=16 (D)
X ) Y : X déplacement vers la droite de Y (rempli à gauche de 0, Y<= 16 (D)
X * Y : X fois Y
X + Y : X plus Y
X - Y : X moins Y
X / Y : X divisé par Y
```

Les opérateurs suivants ne sont pas repris par le SPL :

NOT X	à remplacer par X % 0FFFF(H)
MINUS X	à remplacer par 0(H)- X
HIGH X	à remplacer par X) 8(H)
LOW X	à remplacer par X & 0FF(H)

LES COMPARATEURS

Les comparateurs sont utilisés pour les combinaisons de labels, nombres et opérateurs qui se trouvent à leur gauche (à partir du dernier alternateur ou tab), indiqué par X, que l'on compare avec la combinaison de labels, nombres et opérateurs , qui se trouvent à droite (jusqu'à l'alternateur suivant, tab ou return), indiqué par Y. Tous les comparateurs produisent un flag 'vrai' ou 'faux', qui ne peut être utilisé que par un alternateur ou une directive conditionnelle (IF ou contrôle). Tous les comparateurs comparent des valeurs de 16 bits, pour lesquelles il n'est pas tenu compte du positif ni du négatif. 0FFFF(H) est donc considéré comme plus grand que 0(H).

```
X < Y :  'vrai' si X est plus petit que Y, sinon 'faux'
X > Y :  'vrai' si X est plus grand que Y, sinon 'faux'
X = Y :  'vrai' si X est égal à Y, sinon 'faux'
X " Y :  'vrai' si X est différent de Y, sinon 'faux'
```

LES ALTERNATEURS

Les alternateurs sont utilisés pour combiner le résultat de combinaisons de labels, nombres, opérateurs, comparateurs et alternateurs qui se trouvent à gauche (à partir du dernier tab), indiqué par X et de combinaisons de labels, nombres, opérateurs et comparateurs qui se trouvent à droite (jusqu'à l'alternateur suivant), indiqué par Y. Tous les alternateurs comparent des flags 'vrai' ou 'faux', comme ceux produits par des comparateurs ou d'autres alternateurs, et fournissent eux-mêmes un nouveau flag 'vrai' ou 'faux'.

X ! Y : 'vrai' si aussi bien X que Y sont 'vrais', sinon 'faux' (AND)
X ? Y : 'vrai' si X ou bien Y est 'vrai', sinon 'faux' (OR)

On parlera d'une faute de syntaxe, si immédiatement à gauche ou à droite d'un opérateur, comparateur, ou alternateur il ne se trouve aucun label ou nombre.

LES SEPARATEURS

Les séparateurs sont employés pour indiquer la fin d'une opérande. Un séparateur consiste en un opérateur, comparateur, alternateur, une virgule, un tab, un return ou un espace. L'espace est utilisé comme séparateur quand par exemple, vous voulez indiquer la fin d'une opérande lors d'une commande X.
Lorsqu'il s'agit d'opérateurs de même type, l'opérande est toujours évaluée de la gauche vers la droite.

CARACTERES SPECIAUX DANS LA COLONNE OPERANDE

Lorsque après une instruction qui attend une valeur de 1 byte, la colonne opérande commence par un point ("."), le chiffre qui suit est considéré comme ayant une valeur de 2 bytes. L'avantage d'un tel nombre est qu'il peut être remplacé par un label ou qu'il peut former le début d'une opérande composée.
Si dans la colonne opérande, où normalement un label ou un nombre peut se trouver, on voit le signe dollar ("$") (pas permis dans la même ligne qu'une directive MACRO), alors, lors de l'évaluation, la valeur actuelle du pointeur objet lui sera conférée.

Vous pouvez utiliser dans un nom tous les caractères qui ne sont pas des séparateurs, et qui se trouvent entre les valeurs ASCII de 20(H) à 7F(H), à condition qu'ils ne commencent pas le nom.
En pratique ce sont les caractères suivants :

0123456789
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
$.:; ^

CONSTRUCTION D'UN PROGRAMME ASSEMBLEUR

Bien structuré un programme peut avoir des désavantages déterminés, mais il offre aussi de nombreux avantages surtout en ce qui concerne la prévention des fautes. SPL vous laisse complètement libre sur la façon de construire votre programme objet, mais pose des exigences bien précises en rapport avec l'emploi de certaines directives et leurs relations réciproques. C'est la raison pour laquelle un programme en assembleur doit être construit d'une manière particulière. La façon dont on doit scinder la construction est très bien rendue dans le diagramme de syntaxe suivant:

```
Programme               := (code)
                           END
                           (Liste des directives)
                           (macro-définitions)
Définitions macro       := MACRO
                           (code)...
                           MEND
                           (liste des directives)
Code                    := mnémonique
                        := spécial
                        := bloc
Bloc                    := IF bloc
                        := bloc de contrôle
                        := call macro
IF bloc                 := IF
                           (code)...
                           [ELSE]
                           (code)
                           ENDIF
Bloc de contrôle        := ###
                           (code)...
                           contrôle
Spécial                 := ORG
                        := EQU
                        := DW
                        := DS
                        := DB
                        := SET
                        := Liste des directives
Liste des directives    := PRT
                        := LST
                        := PUT
                        := TITL
                        := UNL
                        := Remarques
```

A gauche des ':=' se trouvent les mots dont la formation est décrite. A droite se trouve cette formation, en d'autres mots ou consignes de base, de haut en bas jusqu'au ':=' suivant. Plusieurs ':=' donnent les différentes alternatives de construction

Les consignes de base sont les mots en majuscules, call macro, ### , contrôle et remarques. Ceux-ci appartiennent aux directives assembleurs et sont décrits dans le chapitre correspondant. Ceci est valable aussi pour les mnémoniques, qui sont comptées aussi dans les consignes de base. Ce qui se trouve entre [ ] peut être inutilisé. Ce qui se trouve entre ( ) peut être employé plusieurs fois, ou bien complètement inutilisé.

Comme exemple, vous pouvez vous imaginer que dans les listings, les combinaisons suivantes sont reliées par des courbes qui ne peuvent se croiser.
Voici les principales liaisons permises :
Début du code source --- END --- fin de code source
MACRO --- MEND
IF --- ELSE --- ENDIF ou IF --- ENDIF
--- contrôle (avec le même identificateur)

La construction suivante est interdite :

```
/====           IF      X"Y
:               NOP
:    /-.contrl  ###
:    I          NOP
)====I=         ELSE
:    I          NOP
:    I-         contrl  A=0(H)
:               NOP
:====           ENDIF
```

Le 'ELSE' va créer des problèmes

Un bon exemple est le suivant :

```
/====  .callm  MACRO   A,X
:              NOP
:  /=          IF      A=X
:  :           NOP
:  :=          ENDIF
:              NOP
:====          MEND
```

LES FAUTES DE SYNTAXE

Ce sont les fautes qui lorsque l'on sort de l'éditeur (au moyen de 'BREAK' 'SPACE') sont reconnues par la routine qui traduit le contenu du buffer d'édition en code source SPL. Si une faute de syntaxe est reconnue la traduction d'ASCII en code source est interrompue, le début du buffer d'édition est placé au premier caractère de la ligne où se trouve la faute de syntaxe et on revient dans l'éditeur. L'adresse du symbole du curseur (75H) est chargée avec le code de faute rencontrée

! Erreur d'alternateur :

par ex : LXI H A?B

! ou ? apparaît dans la ligne à l'endroit de l'erreur.

& Erreur de séparateur :

par ex : DW $;

A l'endroit où l'on attend un séparateur, se trouve un autre caractère

' Erreur de guillemet :

par ex : TITL 'SPL

Il manque un ' ou un " dans la ligne

par ex : DB ""

Il ne figure rien entre les "" ou les ' ' (chaîne vide).

, Erreur de virgule :

par ex : LXI B 0H,1H

On utilise une virgule à un endroit où elle n'a aucune raison d'être.

par ex : DB 1H*1H

A la place d'une virgule se trouve un caractère quelconque.

; Erreur de remarque :

par ex : RST 5 Call screen

Lors d'un commentaire il manque un point virgule dans la ligne.

= Erreur de comparateur :

par ex : IF A=B=C

Il manque un <, =, > ou ", ou bien il y en a un de trop dans la ligne.

a Erreur d'assemblage :

par ex : XCHG

Ce n'est pas vraiment une faute de syntaxe, mais désigne une faute dans l'assembleur à cause de laquelle la ligne fautive ne peut être traduite en code source. Pour pouvoir quand même quitter l'éditeur, il faut éliminer cette ligne. Pour pouvoir la corriger, il faut recharger le SPL

d Erreur de définition :

par ex : LABEL MVI A 0H

Il n'y a pas suffisamment d'espace dans la table des symboles pour y introduire un nouveau nom. Ce peut être une question temporaire (si la SPL ne peut libérer la place nécessaire à ce moment dans la table des symboles) ou permanente (si 1624D noms figurent déjà dans la table des symboles, voir la commande Q). Pour pouvoir quitter l'éditeur, vous devez supprimer la ligne avec le nom.

i Erreur d'E/S :

par ex : PUT "Q"

Si on a une opérande interdite après une directive PUT

par ex : PUT 150D

Si la valeur de l'opérande de la directive PUT est trop grande.

l Erreur de label :

ex *ABCD DB 100,350

Un caractère interdit se trouve dans la colonne label.

ex : PRT A

Un label (ou $) se trouve comme opérande d'une directive PRT

ex : SET $

Il n'y a aucun label pour une directive EQU ou SET

ex : alpha NOP

Dans la colonne label, un identificateur n'est pas suivi par ou une directive macro dans la colonne instruction

ex : OMEGA ###

Il n'y a pas d'identificateur pour ### ou la directive MACRO

m Erreur de mnémonique :

ex : ASC 'STRING'

Dans la colonne instruction se trouve un mot qui n'est considéré ni comme mnémonique ni comme identificateur.

n Erreur numérique :

ex : MACRO 1H

Un valeur numérique ne devrait pas y exister

ex : MVI A 300D

On a une valeur numérique trop grande comme opérande

ex : PRT .10H

L'opérande d'une directive PRT commence par un point et est donc définie comme une opérande sur 2 bytes.

ex : DB

Il y a une faute d'orthographe dans la valeur numérique utilisée comme opérande.

o Erreur de restart :

ex : RST 9

Le chiffre qui suit RST n'est pas compris entre 0 et 7

p Erreur de passing

ex : label1 MACRO A+B

Dans l'opérande de la directive MACRO, se trouve un séparateur autre qu'une virgule, un tab ou un return

r Erreur de registre :

ex : STAX H

Un registre différend de B ou D est utilisé pour LDAX ou STAX

ex : MOV C,C

Deux registres identiques sont nommés après un MOV.

s Erreur d'espace :

ex : CALL X Y

Un espace suit l'opérande, ce qui est interdit

t Erreur de terminaison

ex : DB'ASCII'

Il manque un tab ou un return à un endroit où ils sont attendus

ex : NOM DB 'c'est une ligne extraordinairement longue et qui l'est même de trop

La ligne à coder compte plus de 640 bytes.

II figure un tab dans un commentaire

ex : LDA

Il manque une opérande dans la ligne

LES FAUTES DE STRUCTURE

Ces fautes sont reconnues par les routines objets lors d'une des phases d'assemblage. Une faute constatée est définie comme suit :
X numéro de la ligne fautive
'X' représente ici le code de la faute

Dans la description, il est indiqué dans quelle phase d'assemblage la faute a été reconnue. Les fautes qui sont normalement reconnues dans la phase de mise en place peuvent être également reconnues dans la phase de composition quand une instruction doit déjà connaître alors une valeur déterminée. La règle de base à appliquer est que la phase de composition ne calcule pas les valeurs qui sont placées directement en mémoire comme code objet, mais bien les valeurs qui influencent le pointeur objet.

D Erreur de double définition : (phase de recherche)
Le nom de label se trouvant dans la colonne label est déjà apparu dans la colonne label. Cela ne doit pas poser de problèmes si par exemple le label est utilisé dans les deux branchements d'une construction IF- ELSE- ENDIF.
N.B> : 'SET' ne provoque pas d'erreur de double définition.

O Dépassement de byte : (phase de mise en place)
On utilise une valeur supérieure à 0FF(H) dans une opérande de 1 byte.

S Erreur de shift : (phase de mise en place)
La valeur sur laquelle un shift '<' ou '>' doit se trouver est plus grande que 15D.

U Erreur indéfinie : (phase de mise en place)
Il n'y a pas (encore) de valeur attribuée à un label (c-à-d qu'il n'est pas encore repris dans la colonne label)

LES MESSAGES D'ERREUR

Ceux-ci consistent en l'impression d'une indication d'erreur entre deux . suivie d'un branchement vers l'introduction des commande. La différence essentielle avec les fautes de structure est la suivante : la faute qui provoque un message d'erreur peut avoir des conséquences catastrophiques. Ainsi, une erreur de shift peut avoir des conséquences relativement peu importantes et est donc considérée comme uns erreur de structure, tandis qu'une division par zéro est une erreur qui peut entraîner le SPL dans une boucle sans fin et provoque donc l'affichage d'un message d'erreur qui arrête la routine. Pour la routine ?. en plus du message d'erreur, la ligne où l'erreur est située est listée (excepté avec les #MP# où c'est la ligne appelée qui est listée). Egalement, avec la description des messages d'erreur, est indiqué quand elle a été reconnue. Voir également le chapitre concernant la construction des programmes en assembleur.

#AE# : After End error (phase de recherche`

Après un END figure (en dehors d'une définition MACRO) quelque chose d'autre qu'une remarque, TITL, PUT, PRT, LST, UNL ou bien il y a plus d'un END dans le code source.

#CT# : ConTrol error (phase de recherche)

Il y a un contrôle sans qu'il y aie une entrée correspondante au même niveau.

#DD# : Double Defined identifier error (phase de recherche)

Le nom d'un identificateur se trouve une seconde fois dans le colonne label

#DZ# : Division by Zero (phase de mise en place)

A droite de l'opérateur '/' se trouve un nombre ou un label de valeur 0(D).

#FO# : File Overflow error (commande R ou Y)

Lors de la lecture ou de la réunion de fichiers source, le fichier lu est plus grand que l'espace disponible en mémoire. Vous devez d'abord agrandir le buffer source et ensuite vous pouvez relire à nouveau le fichier source
Le code source en mémoire peut être considéré comme anéanti (donc avec la commande Y aussi bien l'ancien fichier que le nouveau).

#IF# : IF error (phase de recherche)

Il y a un ELSE ou un ENDIF sans qu'il n'existe de IF correspondant.

#L0# : LOcal error (phase de recherche)

Un label est défini par SET ou un MACRO alors qu'il est déjà défini dans un MACRO (où le programme principal) précédant

#MA# : MAcro error (phase de recherche)

Il y a un MACRO non précédé d'un END ou d'un MEND

#MP# : More Passed (ou missing pass) error (phase de recherche)

Le nombre de valeurs dans la colonne opérande d'un call-macro ne correspond pas au nombre de labels dans la colonne opérande du MACRO appelé.

#NO# : Nesting Overflow (phase de recherche et de construction)

Pendant une routine objet, il y a plus de 80 IF et /ou MACRO ou bien, il y a plus de 80 IF et /ou dans le code source.

#RC# : ReCurssing error (phase de recherche)

Un call-macro appelle une définition macro qui précède le call-macro dans le code source.

#UI# : Undefined Identifier error (phase de recherche (contrôle) ou phase de construction (call-macro))

Un contrôle ou un call-macro utilise un identificateur non (encore) défini

LES FAUTES D'INTRODUCTION

Ces messages de fautes apparaissent quand vous avez commis une faute en introduisant une commande, ou bien quand la commande ne peut être exécutée. Les messages d'erreurs comportent deux minuscules comprises entre deux # , ceci pour les distinguer des messages d'erreurs du chapitre précédent.
Avec la description des messages d'erreur, la commande avec laquelle l'erreur put survenir est indiquée. S'il n'y a rien d'indiqué, ces erreurs peuvent alors en principe, survenir avec l'introduction de n'importe quelle commande. Les différents messages sont :

#ce# : Compatibility Error (X)
Conflit entre un identificateur et un label ou valeur numérique

#cf# : Commando Fout
Le premier caractère n'est pas reconnu comme commande

#cs# : CheckSum error (R,Y)
Une faute est constatée à la lecture d'un fichier source.

#dc# : DCr error (R,W,Y)
Le DCR a constaté une erreur

#hc# : HardCopy error (P,v)
Le flag hardcopy n'est pas allumé

#in# : INstruction error (1)
Aucune instruction valable n'a été utilisée.

#ln# : Line Number error (E,K,L,M,m)
Quelque chose ne convient pas avec la numérotation des lignes (par ex : les numéros de lignes sont dans le désordre!

#mm# : MeMory error (C,J,M,m,X)
Il n'y a plus assez d'espace mémoire pour l'exécution de la commande

#nm# : NaMe error (E,G,I,K,L,M,m,x)
Un nom erroné (identificateur ou label) est utilisé

#se# : Source (ou Symbols) Empty error (F,g,i,L,S)
Le code source fait défaut ou la table des symboles est vide.

#sn# : Sorting Needed error (W,Y)
Vous avez essayé de sauver ou de réunir une table de symboles non triée.
Vous devez d'abord utiliser la commande Q.

#su# : Symboltable Underflow error (w)
Pour la sauvegarde d'un fichier source, au moins deux noms sont nécessaires dans la table des symboles, parce que leurs champs de valeur sont utilisés pour y indiquer les adresses de départ et du code source déplacé. Si lors d'une commande W un message d'erreur (#su#) apparaît, alors vous devez, éventuellement au moyen de labels factices, élargir la table des symboles jusqu'à au moins deux noms.

#sy# : SYstem error
Il y a quelque chose d'anormal avec le SPL. Vous devriez le recharger.

#vo# : Value Overflow error ($,A,C,E,G,I,J,K,L,M,m,P,p,R,X,Y)
Une valeur numérique trop grande a été utilisée.

DEBUGGING POUR PROGRAMMEUR AVANCE

Ce chapitre est réservé à ceux qui sont déjà avancé dans le SPL et qui veulent savoir comment restaurer un fichier source qui a été détérioré par un programme en langage machine. Il ne faut cependant pas oublier que vous ne pouvez réussir que dans un nombre de cas assez restreints. S'il y a par exemple, trop de bytes affectés dans le fichier source, vous avez plutôt intérêt à le relire et si nécessaire à relire le SPL avant. Ensuite, vous ne devez pas oublier de rechercher l'erreur qui a provoqué l'accident et de la corriger. Comme connaissance de base il est nécessaire d'avoir une idée sur la construction du code source du SPL, c'est pourquoi nous vous donnons ici une description de ce dernier.
Un fichier source est composé en fait de deux parties : la table des symboles et le code source. Nous allons d'abord nous occuper du code source.
L'unité de base dans un code source est la ligne source. C'est une forme codée de la ligne en assembleur, donc une ligne de texte dans l'éditeur ou du listing. Dans le code source, chaque ligne commence par un soi-disant byte d'état. Les différents bits de ce byte ont une signification particulière. Les six bits les plus bas, c-à-d du bit0 jusqu'au bit5 inclus, indiquent sous forme binaire, le nombre de byte qui séparent ce byte d'état du suivant, en d'autres termes la longueur de la ligne source, non-compris le byte d'état, en bytes. Le bit7, bit le plus élevé, également appelé bit de signe, est set (mis à 1) lorsque la ligne assembleur commence car un label. Le bit6 du byte d'état a différentes significations suivant l'instruction utilisée.

- Pour les mnémoniques de 1 byte, comme ORA M, le bit6 du byte d'état n'est pas utilisé et est donc égal à 0. Ce type de mnémonique est uniquement suivi d'un commentaire éventuel après le code de l'instruction. La présence de commentaire transparaît alors toujours dans la grande valeur des bits0 à 5 du byte d'état, valeur qui est supérieure au total de bytes utilisé par l'éventuel label ou identificateur, le code de l'instruction et les opérandes (1B pour un mnémonique de 1 byte sans label et 11B avec label). Ce commentaire est codé en ASCII (bit de signe = 0). Le point-virgule précédent le commentaire n'est pas repris dans la ligne source.

- Pour les mnémoniques de 2 bytes comme CPI 2H, le bit6 du byte d'état n'est pas allumé (donc égal à 0) si l'opérande doit être représentée sur un byte simple. La valeur de ce byte est directement indiquée par le byte qui suit le code de l'instruction. Un commentaire éventuel peut suivre. Si le bit6 du byte d'état pour les mnémoniques à 2 bytes, est à 1, il se présente alors plusieurs possibilités :

Si le code de l'instruction est suivi d'un byte 00H ou 80H, cela signifie que l'opérande, qui est codée sur 16 bits après évaluation, devra avoir son plus haut byte à 0. Dans les listings et dans l'éditeur cela se représente par une valeur numérique précédée d'un point. Pour la méthode de stockage, voir la description pour les mnémoniques à 3 bytes. Le bit7 du byte à 00H ou 80H joue ici le même rôle que le bit6 du byte d'état dans les mnémoniques à 3 bytes (0 : opérande commençant par un nombre, 1 : opérande commençant par un label).

Si le code de l'instruction commence par un byte de valeur différente de 00H ou 80H, cela représente la valeur ASCII d'un caractère. Les caractères ASCII avec le bit7 à 0 seront listés entre ' ' et les caractères avec le bit7 à 1 seront listés comme le caractère correspondant mais entre "".

- Pour les mnémoniques de 3 bytes comme CALL OUTPUT, l'opérande est construite comme suit :
Après le code d'instruction suivent 2 bytes. Si le bit6 du byte d'état est à 0, ces deux bytes représentent une valeur numérique de 16 bits, dans l'ordre lowbyte et highbyte (analogue aux instructions du 8080). Si le bit6 du byte d'état est à 1, alors ces deux bytes forment un pointeur qui indique une adresse dans la table des symboles (voir plus loin dans ce chapitre). Après ces deux bytes, il se présente trois possibilités :

1. Le nombre de bytes de la ligne, donné par le byte d'état est épuisé, le byte suivant est dès lors, le byte d'état d'une nouvelle ligne source.
2. Le bit7 du byte suivant est à 0. Cela indique le début d'un commentaire.
3. Le bit7 est à 1 et la ligne non encore épuisée, cela indique un opérateur dont la valeur ASCII est donnée par les bits 0 à 5 de ce byte, mais alors avec les bits 6 et 7 à 0. Le bit6 du byte opérateur indique si les deux bytes suivants doivent être considérés comme une valeur numérique ou comme un pointeur vers un label, de manière analogue au bit6 du byte d'état en relation avec les deux premiers bytes de l'opérande. Après le byte opérateur avec ses deux bytes de valeur, nous obtenons à nouveau trois possibilités, aussi longtemps qu'il n'y a pas de commentaires ou que la fin de la ligne source n'est pas atteinte.

Après les mnémoniques nous allons traiter maintenant des différentes directives :

END, ###, ELSE, ENDIF, MEND, LST, UNL et remarques sont codées dans les lignes sources comme un mnémonique d'un byte (seulement il ne peut figurer de nom devant une remarque et il doit toujours figurer un identificateur devant ###).

ORG, EQU, SET, DW, DS et IF sont codées comme des mnémoniques à 3 bytes, seulement il doit figurer un label devant EQU et SET alors que IF utilise également à coté des opérateurs, des comparateurs et des alternateurs.

Un identificateur doit toujours figurer devant MACRO, mais il ne doit pas être nécessairement suivi par des opérandes. Dans ce cas, le bit6 du byte d'état est à 0. Si la MACRO est suivie par des opérandes, le bit6 du byte d'état est à 1. L'opérande MACRO ne peut consister qu'en un ou plusieurs labels séparés par des virgules, qui sont codés de la même manière que des opérateurs, comparateurs et alternateurs. Ceci pour rester compatible avec les mnémoniques à 3 bytes en vue du listing.

Call-macro est codé comme suit :

- byte d'état
- (éventuellement) un pointeur vers un label
- un byte d'instruction du call-macro
- un pointeur vers un identificateur
- éventuellement suivi d'une virgule séparatrice et une ou plusieurs opérandes, chacune codée comme un mnémonique de 3 bytes mais séparées entre elles par des séparateurs.
- éventuellement un commentaire.

Contrôle est codé comme suit :

- byte d'état
- éventuellement un pointeur vers un label
- un byte d'instruction du contrôle
- un pointeur vers un identificateur
- une virgule séparatrice
- une opérande construite comme pour IF'
- éventuellement suivi d'un commentaire

Pour DB, le code d'instruction est suivi d'un byte de longueur qui indique le nombre de bytes que DB doit placer en mémoire pendant l'assemblage, suivi de ces bytes. L'état du bit6 du byte d'état décide si ces bytes vont être considérés comme des valeurs numériques ou comme chaîne ASCII, de la mène manière que pour les mnémoniques de 2 bytes. Avec une chaîne ASCII le bit7 détermine si le premier caractère de la chaîne doit se trouver entre ' ' ou " ". Dans une chaîne ASCII, les bits de signe de tous les caractères sont égaux. Des labels ou opérandes composées sont aussi possibles avec DB, de la même manière que pour les mnémoniques à 2 bytes. Cependant, seulement un seul byte peut être défini.

TITL est codé comme DB, cependant l'opérande peut seulement consister en une chaîne ASCII et doit donc se trouver entre deux ' ' (bit7 = 0)

PRT est codé comme DB, mais les labels et opérandes composées ne sont pas autorisées.

PUT est codé comme un mnémonique à 2 bytes, seules les caractères B, D, H, O, b, d, h, o sont autorisés, tous ces caractères sont entre "" (le bit de signe = 1). La valeur pour un byte opérande

est seulement une valeur plus petite ou égale à 7F(H) et plus grande ou égale à 00(H). La fin d'un code source est donnée par byte 00(H) à la place où devrait se trouver un byte d'état.

LES CODES D'INSTRUCTIONS

Les mnémoniques sont représentés par des codes standards de l'INTEL 8080. Il n'est cependant pas possible d'utiliser les mnémoniques MOV R,R ; IN et OUT parce que leurs codes sont utilisés pour les directives.
Ainsi les différents codes des directives deviennent :

```
08(H) : ORG          40(H) : IF           0CB(H) : END
10(H) : EQU          49(H) : ELSE         0D3(H) : PRT
18(H) : DW           52(H) : ENDIF        0D9(H) : LST
20(H) : ###          5B(H) : call-macro   0DB(H) : PUT
28(H) : DS           64(H) : MACRO        0DD(H) : TITL
30(H) : contrôle     6D(H) : MEND         0ED(H) : UNL
38(H) : DB           7F(H) : SET          0FD(0) : remarque (;)
```

LA TABLE DES SYMBOLES

La table des symboles est construite par l'assembleur, à partir d'une adresse fixe, vers le bas (cette adresse est toujours 0XX00(H)).
Par nom introduit, il y a toujours 8 bytes disponibles. Les 6 premiers bytes de ce champ contiennent le nom du label introduit (ou identificateur). Pour cela, seuls les 7 bits inférieurs de chaque byte sont utilisés, le bit de signe peut être utilisé comme drapeau, mais doit être égal à 0 dans la plupart des routines. Le premier caractère du nom se trouve à l'adresse la plus basse des six bytes, le deuxième sur la suivante et ainsi de suite. Si le nom est plus court que six caractères, les bytes inutilisés sont remplis de 0.
Les deux autres bytes du champ servent comme lieu de stockage de la valeur qui est attribuée à un label (ou identificateur). Les six et deux bytes sont respectivement nommés champ du nom et champ de valeur. Si dans le code source on désigne un nom dans la table des symboles cela se produit au moyen d'un pointeur de 2 bytes (dans l'ordre lowbyte - highbyte). Ce pointeur désigne l'adresse la plus basse du champ de valeur.
Le label $, qui donne la valeur actuelle du pointeur objet, n'est pas repris dans la table des symboles, mais a une place fixe dans l'assembleur lui-même. C'est aussi la raison pour laquelle après la lecture d'un fichier source d'une autre version SPL, vous devez adapter les appels vers le labels $ par la commande $.

TRUCS POUR LA REPARATION DES FICHIERS SOURCES ENDOMMAGES

En quittant le SPL par la commande U ou u, certains checksums sont réalisés et automatiquement contrôlés lors du retour dans le SPL. S'il y a une erreur dans l'assembleur, il est préférable de relire le SPL ainsi que le dernier fichier source. Si c'est la table des symboles qui est endommagée, cela signifie en général que l'un des noms est tronqué. Un listing des symboles vous montrera cela rapidement surtout quand la table des symboles a été triée depuis peu de temps. Une telle erreur est réparée en Utility par l'instruction D pour rechercher le nom et ensuite, par l'instruction S, corriger la faute. Si vous ne voyez rien de particulier dans le listing des symboles, l'erreur se trouve dans un des champs de valeur. Cette faute n'est pas grave et se corrige d'elle-même.
S'il semble que le code source soit endommagé, cela peut être facilement réparé par un listing. Parfois l'erreur peut être détectée facilement si par ex :

- Le byte d'état a été endommagé (le listing des lignes suivantes est également tronqué).
- Le pointeur de la table de symboles est modifié (un autre nom peut être imprimé) ou plus vraisemblablement, une suite de signes sans aucune signification.
- Un code instruction est modifié en un autre code ayant des exigences différentes pour les opérandes par ex, un mnémonique d'un byte transformé en mnémonique 3 bytes.

Dans de tels cas vous feriez mieux de rechercher le dernier label ou identificateur dans la colonne label ou opérande et respectivement au moyen des commandes I et G, demander l'adresse où le byte d'état de cette ligne se trouve. Par Utility, au moyen de l'instruction D, on peut montrer la zone avec l'erreur et, lorsqu'elle est de l'instruction D, on peut montrer la zone avec l'erreur et, lorsqu'elle est localisée la corriger avec l'instruction S. Si cela pose trop de problèmes, vous pouvez remplacer le byte d'état de la ligne endommagée par un byte dont le bit6 et le bit7 sont tous deux égaux à 0 et les autres bits indiquent le byte d'état suivant, suivi par un byte à 00(H) et autant de codes 20(H) (espace commentaire) qu'il faut pour atteindre le prochain byte d'état. Il est aussi possible au lieu d'utiliser le byte 00(H), un byte 38(H) (DB). Il doit être alors suivi par un byte indiquant le nombre de bytes supplémentaires après DB jusqu'au prochain byte d'état.

Après que la ligne source ait été provisoirement réparée de cette manière, vous pouvez retourner au SPL, où avec l'éditeur vous pouvez corriger la source. Si au début, il ne ressort pas du listing où figure l'erreur, celle-ci peut éventuellement détectée par la commande ?. En général, il n'est pas nécessaire de passer par Utility, souvent l'éditeur suffit. N'utilisez l'éditeur que quand vous êtes certain que les lignes à éditer peuvent être listées normalement. Soyez également prudents avec les routines objets, en tout cas, quand les pointeurs de labels sont endommagés. Si la commande ? ne donne rien, vous feriez mieux de recommencer à lire le fichier source. Après la réparation du fichier source, l'erreur qui a provoqué l'incident dans le programme doit encore être résolue nous donnons ici quelques règles de base :

1. Le niveau de pile

Cela provoque souvent un branchement vers une adresse en dehors du programme et des choses bizarres avec le curseur se produisent. (sauts ou arrêts).

Cela se produit souvent lors d'un emploi déséquilibré de PUSH et POP, ou par un JMP là où un CALL aurait dû figurer. Cela arrive souvent quand l'on saute en dehors ou dans une sous-routine. Surtout pour les programmeurs débutants, il est recommandé d'utiliser des programmes moins compliqués et d'utiliser souvent un peu plus de bytes et de temps machine. Si le programme tourne convenablement, alors on peut l'optimiser.

2. Les sauts conditionnels

Se produit-il un saut à la bonne condition ? Surtout si vous utilisez des sous-routines de test, qui influencent les flags (ex Z ou NC), vous pouvez alors vous tromper. Cette forme d'erreur se caractérise par aucune action ou une action de trop. Des effets graphiques ont souvent cela comme cause, par ex une boucle avec le mnémonique STAX dedans. Si une telle chose apparaît à l'écran, alors souvent le SPL sera endommagé avant que la ram vidéo ne soit atteinte.

Lors de la préparation des sauts conditionnels, vous pouvez utiliser un petit truc. un CPI (ou CMP) ont la même influence sur les flags qu'un SUI (ou SUB) mais ils n'influencent pas le registre A. Faites attention en général à la façon avec laquelle les mnémoniques agissent sur les flags. Ainsi, INX et DCX n'ont aucune influence sur les flags tandis que DCR et INR influencent le flag carry

3. Les erreurs de registre

Souvent se sont de simples fautes telles que MOV A,B à un endroit où l'on aurait du écrire MOV B,A. Parfois il s'agit d'une surcharge, quand par exemple, une sous routine C est utilisé comme compteur alors que BC est utilisé comme pointeur dans une boucle principale.

Une combinaison de PUSH et de POP peut être souvent la solution pour ce genre de problèmes. Il se peut aussi qu'une sous-routine attende une valeur dans E alors que le programme l'a placée dans L. Au moyen d'un détour qui veille à l'adaptation, on peut trouver une solution à moins qu'il ne soit plus facile d'adapter la routine principale ou la sous routine.

4. Fautes diverses

Par exemple la non incrémentation d'un compteur ou l'appel d'une sous routine erronée.

LEXIQUE DU SPL

Dans ce manuel nous avons utilisé des termes techniques. C'est pour cette raison que nous en décrivons ici quelques-uns des plus utilisés.

DESCRIPTION SYNTAXIQUE DES COMMANDES, DIRECTIVES,…

Ce qui se trouve entre crochets est optionnel.
Ce qui se trouve entre parenthèses peut être répété.
Les mots soulignés sont des mots standards.
Celui-ci peut être remplacé par chaque série de caractères qui répond aux exigences syntaxiques du mot standard. Dans les descriptions syntaxiques, les mots standards sont respectivement suivis par 0, 1, 2 ou N

Mots standards :

adres	: nombre hexadécimal de maximum 16 bits.
byte	: nombre hexadécimal de maximum 8 bits.
car	: caractère dont la valeur ASCII se trouve entre 20(H) et 7F(H).
dcr	: commande DCR, par exemple LOOK.
NB	: valeur numérique.
ident	: nom identificateur : minuscule suivie au maximum de 5 caractères qui ne sont pas des séparateurs.
label	: nom label : majuscule suivie de 5 caractères au maximum qui ne sont pas des séparateurs.
ligne	: indication de ligne, composée soit d'un nombre décimal, d'un identificateur ou d'un label.
chaîne	: suite de caractères dont la valeur ASCII est comprise entre 20(H) et 7F(H).
flag	: combinaison de nombres, labels, opérateurs, comparateurs et/ou alternateurs qui après évaluation donnent une condition logique vraie ou fausse.
val	: combinaison de labels, opérateurs et/ou nombre résultant en une valeur ds l6 bits.

Autres

Adresse de début	: adresse ou commence le buffer source
Adresse de départ	: adresse avec laquelle et appelé le SPL Par G (Utility) ou CALLM (Basic). Dans le SPL V1. standard elle est de 8500H)
Adresse de sauvegarde	: adresse de départ du code source combiné avec la table des symboles pour la sauvegarde sur DCR.… etc....
Adresse sommet	: la plus haute adresse de la table de symboles
Assemblage	: conversion d'un fichier source en code objet.
Buffer d'édition	: espace dans le buffer source où l'éditeur conserve le texte pendant l'édition
Buffer source	: partie de la mémoire où le fichier source et le buffer l'édition sont stockés.
Champ de valeur	: deux bytes après un nom dans la table des symboles où la valeur d'un label, ou l'adresse d'un identificateur est stockée.
Code objet	: ce sont les bytes du programme en langage machine tels que l'assembleur le construit à partir du code source.
Code source	: La forme codée du programme assembleur telle que le SPL le stocke en mémoire.
Colonne adresse	: Colonne dans les listings avec la valeur actuelle du pointeur objet.
Colonne byte	: colonne dans le listing qui comprend au maximum trois bytes qui sont produits par une ligne source durant l'assemblage.
Colonne commentaire	: voir la description de l'éditeur.
Colonne instruction	: " " "
Colonne label	: " " "

Colonne opérande : " " "
Commande : instruction du SPL
Compteur de lignes : compte les lignes pendant le listing en hardcopy pour la générations de formfeed
Compteur de pages : compte le nombre de pages pendant le listing sur imprimante, ceci afin de déterminer le numéro de page dans la première ligne.
Confirmation : les commandes C, A; K, R et Y demandent que la touche + soit enfoncée par sécurité.
Directive : dans une ligne assembleur, c'est l'instruction donnée à l'assembleur pour exécuter quelque chose de spécial.
Echange d'adresse : on défini si les valeurs de 16 bits sont listées comme opérandes binaires, octales, hexadécimales, décimales.
Echange de byte : on fait de même pour les valeurs de 8 bits.
Fichier source : combinaison du code source et de la table des symboles.
Plage de hardcopy : on détermine ici si les caractères sont à envoyer vers l'imprimante par exemple pendant un listing.
Identificateur : nom symbolique, utilisé pour donner une adresse déterminée dans le fichier source (pour et MACR0).
Implément : à adapter par l'utilisateur.
Label : nom symbolique, pour déterminer une valeur dans le programme assembleur.
Ligne assembleur : partie du programme assembleur telle que l'éditeur liste sur une seule ligne.
Ligne d'en-tête : Ligne telle qu'elle est listée pendant un hardcopy au dessus des pages. Par exemple, V1.1 est composé d'une ligne 'SPL V1.1 PAGE', un numéro de page et éventuellement un titre.
Ligne source : forme codée d'une ligne assembleur.
Listing objet : listing détaillé par lequel on montre comment l'assemblage met le code objet en mémoire.
Macro : une suite d'instruction, dans le programme assembleur commençant par MACRO et se terminant par MEND et indiquée par un identificateur. Peut figurer à n'importe quelle place d'un programme objet au moyen de call-macro.
Mnémonique : représentation symbolique d'un code d'instruction compréhensible par le microprocesseur.
Numéro de ligne : nombre décimal par lequel on désigne une ligne en assembleur.
Offset : distance sur laquelle est déplacé quelque chose comparativement avec sa place normale.
Phase : étape déterminée pendant une routine objet, voir la commande A.
Pointeur objet : un compteur qui retient pendant les routines objets, à partir de quelle adresse devrait être placé le code objet produit par une ligne source pendant l'assemblage dans la mémoire.
Routine : partie du programme SPL qui est appelée par une commande SPL.
Routine objet : routine par laquelle les phases d'assemblages sont exécutées. Pour le SPL se sont A, F, I (adresse) S, T et ?.
Tab : caractère de tabulation.

DISPLAY

Display est un programme d'aide auxiliaire du SPL. Le nom DISPLAY vient de DISassembleur et SPL, AY n'existant que pour la prononciation. Pour la compréhension du fonctionnement de DISPLAY, il est fortement recommandé d'étudier d'abord d'une manière approfondie les chapitres sur l'assembleur, car sans cette connaissance, vous vous trouverez devant de nombreux problèmes. Les concepteurs de DISPLAY ne sont pas responsables des suites de l'utilisation de ce programme, les suites auxquelles ils pensent principalement étant l'éventuel copyright des programmes désassemblés. Ils réalisent certes, que d'éventuels pirates, avec ce programme en combinaison avec le SPL possèdent un outil d'effraction sûr. Ils considèrent que l'emploi abusif éventuel ne tombe pas sur leur responsabilité. Ils font un appel pressant à ceux qui utilisent ce programme de ne l'employer que dans un but légal.
DISPLAY est spécialement conçu pour le décodage (désassemblage) des programmes en langage machine. Par l'assemblage de DISPLAY, le set de commandes est étendu de quelques instructions. En autre on peut mettre le SPL dans un soi-disant 'départ froid' (par TOFILE TRUE SET), de telle façon qu'à côté de la version en langage machine de DISPLAY, vous pouvez sauver une version assortie au SPL. A côté des commandes réelles de désassemblage, plusieurs commandes auxiliaires du DISPLAY vous aident à avoir une vue d'ensemble des programmes. Ce sont:

% [adres1][adres2]:

ici les adresses 1 et 2 sont divisibles par 16(D), ou sont sectionnées. Cette instruction montre le contenu des mémoires de l'adresse1 à l'adresse2. Cela se fait par bloc de 16 bytes par ligne, précédé par une adresse qui est toujours divisible par 16.
Les blocs de bytes sont montrés successivement sur la ligne comme nombres de deux hexadécimaux, près desquels chaque fois après un byte sur une adresse impaire, suit un espace et les signes ASCII (correspondants). Les caractères avec une valeur inférieure à 20(H) sont montrés par un point ('.'). Sur demande, vous pouvez choisir un autre caractère (au moyen de la commande CONTRL).
Vous pouvez interrompre temporairement ou définitivement cette vue à volonté, tout comme les autres listings du SPL.

\+ [adres1]

Cette routine recherche tous les endroits, où une adresse déterminée est citée dans la mémoire, et les montre. Ceci peut être utilisé pour voir à partir de quel endroit une routine déterminée est appelée, où une table déterminée est utilisée, etc.

* [adres]

Cette instruction fait la même chose que la précédente, mais a comme condition supplémentaire, que l'adresse en question doit être précédée, en mémoire, d'un code machine qui utilise cette adresse comme opérande. Ce code est montré en même temps que l'adresse. De plus cette instruction est montrée en mnémoniques SPL.
La partie la plus importante de DISPLAY est une collection d'instructions de désassemblage. Celles-ci s'écartent d'une manière importante du désassemblage précédent de DAInamic. Elles ne génèrent notamment pas de sorties visibles, mais en lieu et place elles font un fichier source SPL qui, à l'aide des commandes du SPL, peut être adapté. Cela a deux avantages qui sautent aux yeux :

1. Parce que DISPLAY ne doit pas attendre le lent œil humain ou l'imprimante, il peut atteindre une grande rapidité. Vous devez cependant tenir compte que le désassemblage est néanmoins pourtant clairement plus lent que l'assemblage du même programme. Pour des grands programmes cela peut même durer des minutes (cela durera probablement rarement plus de 5 minutes).
2. Puisque vous disposez maintenant de la source d'un programme, vous ne devez plus la taper séparément. Quand vous avez l'expérience du recopiage ds programmes hors des revues, vous saurez très bien combien juste cette copie peut prendre de temps.

Les curieux d'entre nous, peuvent au moyen dune combinaison DISPLAY-SPL suivre très simplement le travail d'un programme déterminé, compléter le listing source de commentaires,

donner des dénominations adéquates aux labels et toutes sortes d'autres choses en plus, avec lesquelles vous pourrez mieux percer le fonctionnement du programme. Toutes ces choses peuvent aussi hâter notre propre habileté de programmation en langage machine.
DISPLAY est un désassembleur orienté vers le listing. Cela signifie que, premièrement une liste de commandes précise a été établie, liste dans laquelle il est indiqué si une région précise de la mémoire doit être traduite en codes machine, en ASCII, ou quelque chose d'autre. Sphynx a choisi cette construction, dans l'intérêt d'une des plus grandes propriétés du désassembleur, notamment la génération automatique de labels. Maintenant, pour les tables (et autres) des labels sont placés à la bonne place (du moins si vous avez établi une bonne liste de commandes). En outre, pour la rapidité de fonctionnement on a opté pour une construction de table de symbole triée. Ainsi les temps de recherche de, en particulier, les labels dans la colonne label a été remarquablement raccourci. Cela a cependant eu pour suite que chaque désassemblage doit commencer avec une telle table des symboles. A cet effet, pour chaque désassemblage, un 'départ froid' est généré. C'était aussi une des raisons de la construction tabulaire. La liste des commandes reste valable de sorte que vous pouvez la réutiliser à souhait si par exemple, il apparaît que vous avez désassemblé une partie de la mémoire comme du code machine alors que c'était en fait une table ASCII. Par la construction triée, la commande Q n'est plus nécessaire, vous pouvez sauvegarder à souhait la source ainsi créée (s'il .y a au moins 2 labels dans la table des symboles).
La construction de la liste des commandes se déroule comme suit : vous introduisez une commande de travail (" # ' : = ~ , ; < > ou une commande que l'on définit soi-même), suivie par une adresse (en hexadécimal). Cela signifie, désassembler, pendant l'exécution de la liste, une région à partir de l'adresse précédente dans la liste (ou 0(H) si c'est la première commande), jusqu'à l'adresse donnée, de la manière indiquée par la commande. Dans la liste, au début se trouve déjà la commande FFFF. Les adresses sont automatiquement rangée par ordre croissant des adresses. A l'introduction d'une adresse existante, la vieille commande est remplacée par le nouvelle (impossible pour ~FFFF). Dans la version standard cette liste est placée dans le buffer BASIC ENVELOPE. Cela signifie que cette liste peut être longue de 80(H) bytes (donc de 42(D) commandes). Lors de l'établissement de cette liste, DISPLAY veille aussi au non dépassement de cette limite. Vous devez, en outre, tenir compte que le SPL utilise ce buffer lors de l'évaluation des MACRO's. Si entre-temps, vous avez assemblé un programme avec des MACRO's, alors votre liste est écrasée. Si vous voulez utiliser une plus grande liste, vous devez adapter les limites de la liste (DMPBUF et BUFMAX). Pour l'adaptation de cette liste vous disposez de la suite des commandes suivantes :

!	Montre la liste de désassemblage Cette représentation se divise sous la forme : code (espace) adresse (hexa). Vous pouvez interrompre cette liste d'une manière temporaire ou définitive.
)	Vide la liste. Cette commande demande une confirmation.
/ [adres]	Supprime cette adresse de la liste.
& (adres)	Désassembler à l'aide de la liste, dans laquelle l'adresse donnée indique un offset, d'une manière comparable à celui donné par une commande A pour le SPL.

Vous pouvez utiliser un tel offset par exemple, quand le programme à désassembler se trouve à un endroit qui est occupé normalement par le SPL.
Un 'départ froid' est automatiquement généré, de sorte que la source SPL éventuellement présente soit détruite. Avec ce départ à froid, l'adresse de départ de la source SPL à former est demandée avec 'START?'. Vous devez choisir cette adresse de manière à ce que le programme en langage machine à désassembler ne soit pas écrasé.
Si, durant le désassemblage, la région entre E000(H) et EFFF(H) doit être désassemblée autrement que par '~', alors le numéro de banque est demandé.

Le désassemblage de plusieurs banques E en même temps est impossible, cela en raison des labels. Si la première commande ne consiste pas en '~', une ORG 0(H) est placée au début. A la fin de la source SPL, un END est placé automatiquement. A l'adresse de chaque commande '~' un ORG est placé, exepté pour ~FFFF(H). Tout cela se passe sous réserve d'un espace mémoire suffisant. Dans le cas éventuel d'une insuffisance, la production de source est interrompue.
Pendant le désassemblage, toutes les opérandes 16 bits utilisées par 'DW' ou ';' sont stockées dans la table des symboles dans l'ordre croissant des labels. Ces labels sont composés des lettres LH, suivis de la représentation hexadécimale de la valeur, mais vous pouvez ici, choisir une autre combinaison. Pendant la deuxième phase du désassemblage, ces labels sont placés dans la colonne label de la ligne source, qui commence à l'adresse concernée ou, éventuellement définie par un EQU en cas d'interruption.

Le désassembleur connaît les ordres de travail suivant :

#	: désassembler comme des instructions en langage machine.
,	: " " " nombres de 8 bits
'	: " " " caractères ASCII de 8 bits (MSB = 0)
"	: " " " " " " " (MSB = 1)
=	: " " " " " " " (MSB = 0 ou 1)
:	: " " " nombres de 16 bits (low - high)
;	: " " " nombres de 16 bits (high - low)
~	: Définir seulement des labels dans cette zone.

Avec ' " et =, les nombres qui ne satisfont pas à la condition MSB concernée, et/ou dont les 7 bits les plus bas indiquent une valeur inférieure à 20(H), sont désassemblés comme des valeurs numériques de 8 bits.
Avec la commande ____ les valeurs qui représentent un code spécial SPL sont traitées comme un nombre de 8 bits.
Dans les deux cas spéciaux ci-dessus, il y a un nombre de 8 bits par ligne.
Lors du traitement de ',' il en vient d'habitude 8 sur une ligne, à moins que l'adresse de sommet ne soit atteinte plutôt, ou que l'on rencontre un label. La longueur standard pour les fonctions ASCII est de 16 caractères.
Pour ';' vu que le SPL ne connait pas de directives spéciales pour les nombres de 16 bits dans l'ordre high - low, une méthode spéciale est appliquée. Admettons que l'adresse 1234 se trouvent en code machine dans l'ordre high - low, alors ' ; ' donne le résultat suivant :

```
[label0]        DB    LH11234>8H
[label1)        DB    LH1234&0FFH.
```

Dans DISPLAY, il y a deux commandes standard pour le décodage des tables composées.
Ce sont :

<	: décode le byte et adresse dans l'ordre suivant low - high
>	: décode le byte de longueur, la chaîne ASCII, l'adresse en low - high.

Vous pouvez définir facilement des routines composées par vous-même. Comme code de commande vous pouvez utiliser '[' ou ']'. Dans ce cas, vous ne devez pas oublier de reprendre la commande dans DECTBL, ZETTBL et PLCTBL. Dans ces tables vous aller trouver des renvois vers, comme exemple, des routines à utiliser (< et >). Si vous n'avez pas besoin de l'une ou l'autre instruction, alors vous pouvez à souhait supprimer la routine de DISPLAY, pour ainsi créer plus d'espace pour des programmes en langage machine et des sources SPL.
Parce que dans le DAI, les RST 1, 4 et 5 utilisent toujours un byte de data, ce byte est décodé comme tel en standard. Quand vous ne voulez pas renoncer à cette possibilité, vous devez mettre le label RSTDAT sur FALSE SET.

TRANSLATOR

TRANSLATOR est un programme d'aide auxiliaire pour l'assembleur du SPL, dont le but est de traduire des programmes qui, par exemple, ont été écrit avec l'assembleur du DNA, en code éditeur du SPL, qui seront alors traduit par l'éditeur du SPL en code source SPL. Cette traduction vers le code éditeur comprend notamment :

- La mise à 0 des MSB des codes ASCII.
- La disparition du linefeed après le carriage return.
- La mise en place correcte des tab's.
- L'élimination des espaces superflus.
- Le remplacement des mnémoniques spéciaux du DNA en mnémoniques standards.
- L'adaptation de la notation des constantes.
- Le placement à l'endroit ad hoc des points virgules.

Le fonctionnement du TRANSLATOR se déroule comme suit :

Vous commencer par assembler avec le TRANSLATOR à l'endroit voulu, et si désiré enregistrer sur bande la version en langage machine. Vous pouvez également si elle est présente, lire la version en langage machine. Dans chacun des cas vous avez besoin du SPL lui-même. Si vous avez la version de DNA qui transcris les fichiers sources sont sous un '#', lisez d'abord cet assembleur et avec son aide le fichier source souhaité. Quand votre version sauve sous un '1' vous pouvez lire ce fichier source en Utility. Ces fichiers sources ne doivent naturellement pas écraser le SPL et/ou le TRANSLATOR. Le début du fichier source doit être égal à la valeur DNABAS du TRANSLATOR (standard 3000(H)). Ensuite vous démarrez TRANSLATOR à l'adresse où se trouve son origine. Après la mise en route. TRANSLATOR demande par 'START?' l'adresse de départ du buffer SPL à former. Cette adresse doit au moins être inférieure de 200(H) à DNABAS. Ensuite extérieurement tout est calme, pendant que le TRANSLATOR traduit la source. Quand tout est terminé vous vous trouvez dans l'éditeur du SPL. A ce moment, vous feriez mieux de taper 'BREAK' 'SPACE' car dans la plupart des cas, l'éditeur va traduire le contenu du buffer d'édition en source SPL sans aucune indication d'erreur. Si cela devait se produire (par exemple à cause d'une faute dans la source DNA originale), alors cela apparaîtra de lui-même à l'écran. Durant cette phase de traduction, vous ne pouvez pas quitter l'éditeur par 'BREAK' '← '. Ceci pour éviter de devoir tout recommencer si vous avez choisi ce moyen par accident.

IMPLEMENTATION

L'exécution des modifications au SPL se déroule comme suit : vous commencez par lire le SPL, ensuite vous le recopiez à un endroit sûr par exemple au moyen de l'instruction Utility M. Ensuite, vous chargez le fichier source IMPLEMENT et vous y choisissez les modifications souhaitées. Ensuite, vous assemblez IMPLEMENT avec un offset qui correspond à la version déplacée du SPL. Vous glissez celui-ci (avec les modifications) à nouveau à sa place exacte et vous le sauvegardez vers un périphérique (DCR, disquettes...)
Vous faites la sélection des modifications en plaçant le label flag correspondant à une valeur déterminée. Dans la situation 'par défaut' l'assemblage de IMPLEMENT ne changerait rien, seuls les labels auxquels vous donnez une valeur auront un effet. Ci-dessous une liste de labels flags avec leurs positions possibles, et la signification de celles-ci. Le mot 'extern' indique qu'a l'endroit où le flag est utilisé, vous devez encore changer quelque chose.

TOFILE	TPUE/FALSE	C'est la mise en route d'un 'départ froid' du SPL.
SRCTOP	SAME/CHANGD	Modification du sommet du buffer, par ex pour pouvoir transcrire comme un seul fichier le SPL et DISPLAY. Changez aussi TOPADR.
HIDING	SAME/CHANGD	Le listing ou non de certaines instructions pendant le listing objet. Seules 8 instructions peuvent être ainsi écartées (pas de nop). Extern.
DFWORD	SAME/CHANGD	Le format listing 'par défaut' des constantes 16 bits. Extern
DFBYT	SAME/CHANGD	Le format listing 'par défaut' des constantes 8 bits. Extern
FILECH	SAME/CHANGD	Le caractère lecture/écriture. Extern.
RGDEFL	SAME/CHANGD	Le nombre de lignes par défaut sur une page (sans en-tête). Extern
BASGEB	KILLED/SAVED	Conservation ou destruction des programmes Basic.
BASPRG	SAME/CHANGD	Début du Basic. Extern.
SOUND	SAME/CHANGD	Le signal. Code analogue à TALK. Extern.
COLORS	SAME/CHANGD	Les couleurs du curseur. Extern.
WIDDTH	nombre	Largeur des caractères sur l'imprimante.
TABELS	SAME/CHANGD	Les tables de tabulation. Extern.
DCR	PRSENT/NPRSNT	Le DCR est présent ou non.
PRINTR	SERIAL/PARALL/NOPRIN	Type d'imprimante (noprin : pas d'imprimante). Extern.
BDRATE	nombre	Byte pour le registre de baud du 5501..

RESTORE

RESTORE XXXX est non seulement un bel exemple pour illustrer ou pour mieux faire comprendre le fonctionnement des différentes commandes du SPL mais aussi, possède un autre objectif. Si vous assemblez ce fichier source, cela forme une bonne acquisition pour vos programmes en basic.
Un des (rares) désavantages du DAI-BASIC est qu'il n'est pas possible au moyen de RESTORE de placer le pointeur DATA à une autre ligne que la première ligne contenant des DATA. Pour d'autres Basic's cela est bien possible au moyen de l'instruction RESTORE XXXX, où XXXX représente le numéro de ligne où vous allez lire les DATA suivants au moyen de l'instruction READ. Pour ce problème RESTORE XXXX trouve la solution. Vous devez pour cela faire la chose suivante :

- Assemblez RESTORE XXXX ou lisez un fichier objet de RESTORE XXXX.
- Placez le 'HEAP POINTER' (adresses 29B(H) et 29C(H)) sur 327(H), donc plus loin que RESTORE XXXX.
- Introduisez l'instruction NEW ou un CLEAR XXXX en BASIC, de sorte que les autre pointeurs soient également adaptés.
- Introduisez votre programme BASIC.
- Placez dans vos programmes BASIC, partout où vous désirez RESTORE XXXX, la combinaison P%=XXXX:CALLM 750,P% ou des instructions semblables. Ici P% représente une variable entière et arbitraire et XXXX le numéro de ligne vers lequel le pointeur de DATA doit indiquer,
- RUN de votre programme.

RESTORE XXXX place le pointeur de DATA sur le numéro de ligne indiqué par la variable entière. Si ce numéro de ligne ne figure pas dans le programme Basic il apparaît une indication de faute : UNDEFINED LINE NUMBER, pendant un run suivi d'un 'IN LINE NUMBER NNNN' où NNNN représente le numéro de la ligne avec CALLM. La ligne vers laquelle le pointeur DATA renseigne, ne doit pas nécessairement contenir de DATA. Si les DATA manque, la première instruction READ recherchera la première ligne après la ligne indiquée par le pointeur de DATA. Elle recherche donc la première ligne qui contient des DATA. Si elle ne trouve pas une telle ligne, suit l'indication d'erreur habituelle : OUT OF DATA, pendant un RUN suivi de 'IN LINE NUMBER NNNN' où NNNN représente le numéro de ligne qui contient l'instruction READ.
Le fichier source RESTORE XXXX a été conçu très large, ceci pour indiquer clairement les différentes directives SPL il peut être simplifié mais nous laissons cela volontiers à votre bon gré.

TABLEAU DES MNEMONIQUES

INSTRUCTION	FONCTION																	HEX							
MOVE GROUP																		reg	A	B	C	D	E	H	L	M
MOV	A,	reg	(	A	)	←	(	reg	)										7F	78	79	7A	7B	7C	7D	7E
MOV	B,	reg	(	B	)	←	(	reg	)										47	40	41	42	43	44	45	46
MOV	C,	reg	(	C	)	←	(	reg	)										4F	48	49	4A	4B	4C	4D	4E
MOV	D,	reg	(	D	)	←	(	reg	)										57	50	51	52	53	54	55	56
MOV	E,	reg	(	E	)	←	(	reg	)										5F	58	59	5A	5B	5C	5D	5E
MOV	H,	reg	(	H	)	←	(	reg	)										67	60	61	62	63	64	65	66
MOV	L,	reg	(	L	)	←	(	reg	)										6F	68	69	6A	6B	6C	6D	6E
MOV	M,	reg	(	M	)	←	(	reg	)										77	70	71	72	73	74	75	—
ACCUMULATOR GROUP																										
ADD	reg		(	A	)	←	(	A	)	+	(	reg	)					*	87	80	81	82	83	84	85	86
ADC	reg		(	A	)	←	(	A	)	+	(	reg	)	+	(	CY	)	*	8F	88	89	8A	8B	8C	8D	8E
SUB	reg		(	A	)	←	(	A	)	-	(	reg	)					*	97	90	91	92	93	94	95	96
SBB	reg		(	A	)	←	(	A	)	-	(	reg	)	-	(	CY	)	*	9F	98	99	9A	9B	9C	9D	9E
ANA	reg		(	A	)	←	(	A	)	∧	(	reg	)					*	A7	A0	A1	A2	A3	A4	A5	A6
XRA	reg		(	A	)	←	(	A	)	⊻	(	reg	)					*	AF	A8	A9	AA	AB	AC	AD	AE
ORA	reg		(	A	)	←	(	A	)	∨	(	reg	)					*	B7	B0	B1	B2	B3	B4	B5	B6
CMP	reg						(	A	)	-	(	reg	)					*	BF	B8	B9	BA	BB	BC	BD	BE
INCREMENT / DECREMENT REGISTER																										
INR	reg		(	reg	)	←	(	reg	)	+	1							**	3C	C4	0C	14	1C	24	2C	34
DCR	reg		(	reg	)	←	(	reg	)	-	1							**	3D	C5	0D	15	1D	24	2D	35
REGISTER PAIR GROUP																			rp	B	D	H	SP	PSW		
INX	rp		(	rp	)	←	(	rp	)	+	1									03	13	23	33	—		
DCX	rp		(	rp	)	←	(	rp	)	-	1									0B	1B	2B	3B	—		
LDAX	rp		(	A	)	←	(	(rp)	)											0A	1A	—	—	—		
STAX	rp		(	(rp)	)	←	(	A	)											02	12	—	—	—		
DAD	rp		(	H,L	)	←	(	H,L	)	+	(	rp	)	***						09	19	29	39	—		
PUSH	rp		(	(SP)-1	)	←	(	rh	)	,	(	(SP)-2	)	←	(	rl	)			C5	D5	E5	—	F5		
			(	SP	)	←	(	SP	)	-	2															
POP	rp		(	rl	)	←	(	(SP)	)	,	(	rh	)	←	(	(SP)	+1)			C1	D1	E1	—	F1		
			(	SP	)	←	(	SP	)	+	2															
DIRECT ADRESS GROUP																										
LDA	addr		(	A	)	←	(	addr	)											3A	al	ah				
STA	addr		(	addr	)	←	(	A	)											32	al	ah				
LHLD	addr		(	L	)	←	(	addr	)	,	(	H	)	←	(	addr	- 1)			2A	al	ah				
SHLD	addr		(	addr	)	←	(	L	)	,	(	addr+1	)	←	(	H	)			22	al	ah				
IMMEDIATE GROUP																										
MVI	A,	data	(	A	)	←	data													3E	dd					
MVI	B,	data	(	B	)	←	data													06	dd					
MVI	C,	data	(	C	)	←	data													0E	dd					
MVI	D,	data	(	D	)	←	data													16	dd					
MVI	E,	data	(	E	)	←	data													1E	dd					
MVI	H,	data	(	H	)	←	data													26	dd					
MVI	L,	data	(	L	)	←	data													2E	dd					
MVI	M,	data	(	M	)	←	data													36	dd					
ADI	data		(	A	)	←	(	A	)	+	data							*		C6	dd					
ACI	data		(	A	)	←	(	A	)	+	data	+	(	CY	)			*		CE	dd					
SUI	data		(	A	)	←	(	A	)	-	data							*		D6	dd					
SBI	data		(	A	)	←	(	A	)	-	data	-	(	CY	)			*		DE	dd					
ANI	data		(	A	)	←	(	A	)	∧	data							*		E6	dd					
XRI	data		(	A	)	←	(	A	)	⊻	data							*		EE	dd					
ORI	data		(	A	)	←	(	A	)	∨	data							*		F6	dd					
CPI	data						(	A	)	-	data							*		FE	dd					
LXI	B,	addr	(	B	)	←	ah,			(	C	)	←		al					01	al	ah					
LXI	D,	addr	(	D	)	←	ah,			(	E	)	←		al					11	al	ah					
LXI	H,	addr	(	H	)	←	ah,			(	L	)	←		al					21	al	ah					
LXI	SP,	addr	(	SP h	)	←	ah,			(	SP l	)	←		al					31	al	ah					

INSTRUCTION		FONCTION			HEX		
JUMP GROUP							
JMP	addr		(PC) ← addr		C3	al	ah
JNZ	addr	If Z=0 ,	(PC) ← addr		C2	al	ah
JZ	addr	If Z=1 ,	(PC) ← addr		CA	al	ah
JNC	addr	If CY=0 ,	(PC) ← addr		D2	al	ah
JC	addr	If CY=1 ,	(PC) ← addr		DA	al	ah
JPO	addr	If P=0 ,	(PC) ← addr		E2	al	ah
JPE	addr	If P=1 ,	(PC) ← addr		EA	al	ah
JP	addr	If S=0 ,	(PC) ← addr		F2	al	ah
JM	addr	If S=1 ,	(PC) ← addr		FA	al	ah
PCHL		(PC h) ← (H) ,	(PC l) ← (L)		E9		
CALL GROUP							
CALL	addr		(TOS) ← (PC) ,	(PC) ← addr	CD	al	ah
CNZ	addr	If Z=0 ,	(TOS) ← (PC) ,	(PC) ← addr	C4	al	ah
CZ	addr	If Z=1 ,	(TOS) ← (PC) ,	(PC) ← addr	CC	al	ah
CNC	addr	If CY=0 ,	(TOS) ← (PC) ,	(PC) ← addr	D4	al	ah
CC	addr	If CY=1 ,	(TOS) ← (PC) ,	(PC) ← addr	DC	al	ah
CPO	addr	If P=0 ,	(TOS) ← (PC) ,	(PC) ← addr	E4	al	ah
CPE	addr	If P=1 ,	(TOS) ← (PC) ,	(PC) ← addr	EC	al	ah
CPE	addr	If S=0 ,	(TOS) ← (PC) ,	(PC) ← addr	F4	al	ah
CM	addr	If S=1 ,	(TOS) ← (PC) ,	(PC) ← addr	FC	al	ah
	N.B.	(TOS) ← (PC) designates the following :-					
		((SP)=1)← (PC h) ,	((SP)-2) ← (PC l) ,	(SP) ← (SP)-2			
RETURN GROUP							
RET			(PC) ← (TOS)		C9		
RNZ		If Z=0 ,	(PC) ← (TOS)		C0		
RZ		If Z=1 ,	(PC) ← (TOS)		C8		
RNC		If CY=0 ,	(PC) ← (TOS)		D0		
RC		If CY=1 ,	(PC) ← (TOS)		D8		
RPO		If P=0 ,	(PC) ← (TOS)		E0		
RPE		If P=1 ,	(PC) ← (TOS)		E8		
RPE		If S=0 ,	(PC) ← (TOS)		F0		
RM		If S=1 ,	(PC) ← (TOS)		F8		
	N.B.	(PC) ← (TOS) designates the following :-					
		(PC l) ← ((SP)) ,	(PC h) ← ((SP)+1) ,	(SP) ← (SP)+2			
RESTART GROUP							
RST 0		(TOS) ← (PC) ,	(PC) ← 0016		C7		
RST 1		(TOS) ← (PC) ,	(PC) ← 0816		CF		
RST 2		(TOS) ← (PC) ,	(PC) ← 1016		D7		
RST 3		(TOS) ← (PC) ,	(PC) ← 1816		DF		
RST 4		(TOS) ← (PC) ,	(PC) ← 2016		E7		
RST 5		(TOS) ← (PC) ,	(PC) ← 2816		EF		
RST 6		(TOS) ← (PC) ,	(PC) ← 3016		F7		
RST 7		(TOS) ← (PC) ,	(PC) ← 3816		FF		
ROTATE/CONTROL/SPECIAL GROUP							
RLC		(A n+1) ← (A n) ,	(A0) ← (A7) ,	(CY) ← (A7) ***	07		
RRC		(A n) ← (A n+1) ,	(A7) ← (A0) ,	(CY) ← (A0) ***	0F		
RAL		(A n+1) ← (A n) ,	(A0) ← (CY) ,	(CY) ← (A7) ***	17		
RAR		(A n) ← (A n+1) ,	(A7) ← (CY) ,	(CY) ← (A0) ***	1F		
NOP		No operation			00		
HLT		Processor stopped until interrupt or reset			76		
DI		Interrupts disabled			F3		
EI		Interrupts enabled after next instruction			F8		
XTHL		(L) ← ((SP))			E3		
SPHL		(SP h) ← (H)			F9		
XCHG		(H) ← (D)			E8		
DAA		Decimal adjust accumulator		*	27		
CMA		(A) ← ($\overline{A}$)			2F		
STC		(CY) ← 1		***	37		
CMC		(CY) ← ($\overline{CY}$)		***	3F		
OUT	port)	not used in DCE Systems			D3	port	
IN	port)				Db	port	