7

SMAKY6

SMILE, EPRO, ETEX, AS:

EDITEURS ET ASSEMBLEURS

Septembre 1981

EPSITEC-system sa

EDITEURS ET ASSEMBLEURS POUR SMAKY6

TABLE DES MATIERES

1. DESCRIPTION GENERALE

1.1 Introduction

La collection de programmes décrits dans cette notice a pour but d'aider au développement de programmes, au dépannage de ceux-ci et à la rédaction de documents. Ce but ne peut être atteint au moyen d'un outil unique et implique un développement pas à pas.
Le premier pas est la rédaction de documents préalables, tels que cahier des charges, documentation préliminaire, notice. Cette édition se fait au moyen du programme éditeur de textes (ETEX) qui permet la correction aisée en cours de route.
Le second pas est l'édition et le test des différents modules du programme. Un éditeur-assembleur-debugger (SMILE) facilite grandement la mise au point des différentes routines composant le programme. Le morcellement d'un programme de grande taille en plusieurs modules testés séparément augmente beaucoup la vitesse de développement.
L'assemblage de tous les modules en un programme final et la mise au point de celui-ci utilise trois programmes spécialisés, l'éditeur, l'assembleur et le debugger.
La rédaction définitive des modes d'emploi et des descriptions se fait à nouveau avec l'éditeur de textes.
Les programmes utilitaires développés pour le SMAKY 6 permettent de travailler de façon efficace et rapide.

1.2 Environnement hardware

Le système SMAKY 6 supporte tous les programmes mentionnés dans le précédent paragraphe. Ce système comprend un processeur (Z80), 64k bytes de mémoire RAM, 2k bytes de mémoire ROM, un écran et un clavier. La mémoire ROM contient l'*operating system* et une librairie de routines. L'écran alphanumérique-graphique de 20 lignes de 64 caractères et 256x120 points travaille en mode DMA directement dans la mémoire RAM du système. Le tout est agrémenté d'une multitude d'interfaces avec le monde extérieur (clavier, interfaces série/parallèle, contrôleur de mémoire de masse..). Trois types de mémoire de masse sont actuellement disponibles:

- Micro-floppy
- Micro-disque
- Cobus

La mémoire de masse Cobus donne la possibilité de dialoguer à plusieurs avec une mémoire de masse importante au travers d'un réseau. Cobus est employé à l'Ecole Polytechnique Fédérale de Lausanne.

1.2 Environnement software

Les différents programmes présentés dans les précédentes sections ne peuvent être opérationnels que s'ils sont entourés d'un système opératoire et d'une collection de routines système; on se référera aux autres notices SMAKY pour étudier ces questions.

2. LES FDITEURS

Les différents éditeurs à disposition sont:

ETEX	éditeur de textes
EPRO	éditeur de programmes
SMILE	éditeur de programmes et assembleur

Les descriptions qui suivent réunissent les caractéristiques des trois éditeurs. Les chapitres consacrés à chacun d'eux traiteront des exceptions et des différences.

2.1 Organisation générale de l'éditeur

2.2 Le clavier, la syntaxe

Un éditeur doit être aussi simple à utiliser qu'une machine à écrire. Les éditeurs du SMAKY 6 ont une philosophie d'actions naturelles instantanées. L'action sur une touche correspond naturellement à l'inscription sur la touche pressée, et se traduit immédiatement par son résultat définitif. Par exemple l'édition de textes se fait directement dans le texte déjà écrit et non dans une ligne de commande.

Pour éviter une floraison de touches correspondant chacune à une fonction d'édition, un codage simple est obtenu à l'aide de 7 touches fonctions et de quelques touches lettres repérées par des flèches (collées sur les tranches). Chaque combinaison de touche(s) fonction génère en fait un nouveau clavier (un peu comme la touche SHIFT pour les minuscules et les majuscules).

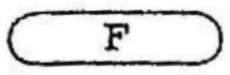
Sur le dessin du clavier standard ci-dessous, les touches fonction encadrent la barre d'espacement.

Chaque touche fonction doit être maintenue pressée pendant le temps nécessaire à la dépression et au relâchement de la touche associée

1) SHOW 3 2) SHOW 3 3) SHOW 3 4) SHOW 3 5) SHOW 3

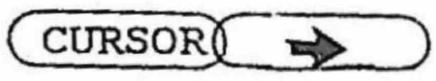
Exemple: SHOW du buffer #3

La représentation graphique suivante a été définie pour mettre en évidence les actions sur les touches qui sont parfois simultanées et parfois consécutives.

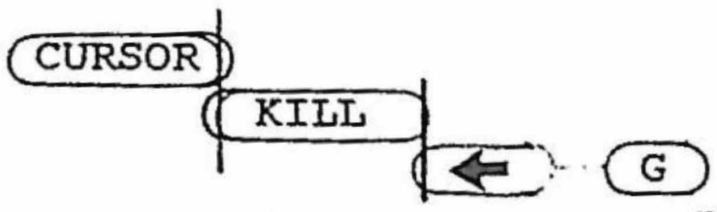


La touche F est pressée isolément

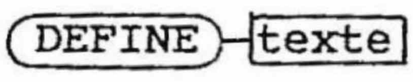
La touche H puis la touche A sont actionnées.



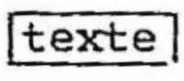
La touche CURSOR est maintenue pressée durant le temps de 

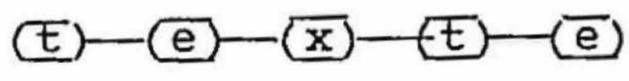


CURSOR et KILL sont maintenus pressés pendant 
puis (une fois que tout a été relâché) G est pressé.



La touche DEFINE est pressée et relâchée, puis un texte quelconque peut être écrit.

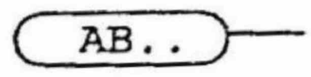


est identique à 

2.3 Insertion de caractères

Si l'on ignore les touches fonctions, le clavier est standard et l'insertion de caractères dans un texte est une opération ne nécessitant pas de commande spéciale. Le programme éditeur réagit comme une machine à écrire avec en plus la facilité de corriger le texte écrit sans laisser de trace. Quelques touches ont des fonction spéciales: il s'agit des touches RETURN et TAB . La première a pour effet de terminer la ligne en cours et de passer à la ligne suivante. La seconde sert à se positionner à des endroits bien précis sur l'écran, généralement toutes les huit colonnes.

Le graphisme adopté pour l'insertion de caractères est le suivant:



insertion

La correction en cours de frappe se fait avec la touche BS qui efface le dernier caractère tapé. D'autres modes de correction plus évolués seront décrits plus loin.

2.4 Déplacements du pointeur

Contrairement à la machine à écrire, l'insertion d'un nouveau caractère au milieu d'un texte est possible. Le pointeur ou curseur clignotant sert à repérer l'emplacement de la prochaine insertion. Le pointeur se déplace au moyen de la touche fonction CURSOR employée en conjonction avec certaines touches du clavier. Voici en détail la représentation graphique de tous les déplacements de pointeur possibles.
Dans ce qui suit nous appelons *champ* une suite de caractères terminés par une virgule, un tabulateur, un point-virgule ou un retour de chariot.

CURSOR ↑	Déplace le pointeur au début du texte
CURSOR ↓	Déplace le pointeur à la fin du texte
CURSOR ←	Déplace le pointeur d'une position vers la gauche
CURSOR →	Déplace le pointeur d'une position vers la droite
CURSOR ←	Déplace le pointeur d'un champ vers la gauche ou à la fin du champ courant
CURSOR →	Déplace le pointeur d'un champ vers la gauche ou au début du champ courant
CURSOR ↑	Déplace le pointeur au début de la ligne précédente ou de la ligne courante
CURSOR ↓	Déplace le pointeur à la fin de la ligne suivante ou de la ligne courante
CURSOR ↑	Déplace le pointeur de 4 lignes en arrière ou au début du paragraphe précédent
CURSOR ↓	Déplace le pointeur de 4 lignes en avant ou au début du paragraphe précédent

Pour résumer les 10 commandes précédentes on écrira:

CURSOR ↑ ↓ ← →	Déplacements du curseur

D'autres commandes permettent de bouger le pointeur seulement dans une direction (horizontalement ou verticalement).
Par exemple, déplacer le curseur dans la ligne inférieure sans le déplacer horizontalement.

(CURSOR( )CTRL( ) ↑) Déplace le pointeur vers le haut, si possible, sans le déplacer horizontalement

(CURSOR( )CTRL( ) ↓) Déplace le pointeur vers le bas, si possible sans le déplacer horizontalement

(CURSOR( )CTRL( ) ⇑) Déplace le pointeur de 4 lignes vers le haut si possible sans le déplacer horizontalement

(CURSOR( )CTRL( ) ⇓) Déplace le pointeur de 4 lignes vers le bas, si possible sams le déplacer horizontalement

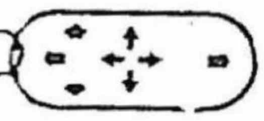
(CURSOR( )CTRL( ) ←) Déplace le pointeur d'une position vers la gauche

(CURSOR( )CRTL( ) →) Déplace le pointeur d'une position vers la droite

(CURSOR( )CTRL( ) ⇐) Déplace le pointeur au début de la ligne visible sur l'écran

(CURSOR( )CTRL( ) ⇒) Déplace le pointeur à la fin de la ligne visible sur l'écran

Pour résumer

(CURSOR( )CTRL( ) 

2.5 Effacement de textes

La conjonction des touches fonction CURSOR et KILL a pour effet de détruire (enlever, effacer) des parties de texte. La dimension du texte détruit est donnée en appuyant sur une touche de déplacement. Il est possible de détruire un caractère, un champ, une ligne, 4 lignes ou un paragraphe, ou tout jusqu'à la fin du texte et ceci en avant ou en arrière.

(CURSOR ( ) KILL) (✦ ✦ ✦ ✦) effacement

Afin de faciliter les corrections au moment de l'édition, deux touches naturelles permettent d'effacer un caractère en avant ou en arrière.

(BACK SPACE)
(DELETE)

(CURSOR ( ) KILL (←)
(CURSOR ( ) KILL (→)

En d'autres termes, la touche BACK SPACE efface le caractère situé en arrière du pointeur, alors que la touche DELETE efface le caractère sous le pointeur. Pour effacer tout le texte en amont et en aval du pointeur, on tape respectivement:

(CURSOR ( ) KILL (↓)
(CURSOR ( ) KILL (↑)

Le système demande *êtes-vous sûr ?*. Presser sur la lettre O pour exécuter la commande.

Une pression sur la touche (ESC) permet d'annuler la ou les dernières destructions, c'est-à-dire de faire réapparaître ce qui vient d'être effacé.

2.6 Travail dans les dix buffers

L'éditeur permet l'édition de dix textes ou parties de textes simultanément. Il y a en effet dix zones de travail (appelées buffers) à disposition de l'utilisateur. La touche SHOW permet le passage d'une zone à l'autre.

Soit le buffer 0 qui contient un texte; nous passons dans le buffer 1, qui lui ne contient rien en tapant:

(SHOW ( ) 1)

Suite à cet ordre, l'écran se divise en deux parties, l'ancien buffer (0) en bas, le nouveau buffer (1) en haut. Ceci permet la comparaison des deux textes. Dès maintenant, il vous est possible d'éditer dans le buffer 1. Pour afficher ce buffer sur tout l'écran, il faut répéter la commande

(SHOW ( ) 1).

En résumé:

(SHOW ( ) 0..9)	affiche le buffer sélectionné en écran double puis simple.

Pour tuer le contenu d'un buffer qu'on ne voit pas, il faut taper:

(CURSOR ( ) KILL ( ) 0..9)

Le système demande *êtes-vous sûr ?*. Presser sur la lettre O pour tuer le buffer.

2.7 Copies et transferts

Afin de faciliter les copies et transferts d'informations d'un buffer à l'autre, il existe des commandes réalisant ces fonctions.
Dans une première opération, il s'agit d'initialiser le buffer de destination de la copie; le buffer de source est toujours le buffer dans lequel on se trouve. Ceci est réalisé au moyen de la touche COPY suivie du numéro du buffer de destination.
Le transfert d'information se fait en pressant simultanément la touche COPY et une touche de déplacement en avant.

Copions par exemple une partie du buffer 0 dans le buffer 1; nous tapons

(COPY ( ) 1)

L'écran se sépare en deux et montre en haut le buffer source (buffer 0) et en bas le buffer destination (buffer 1).
Tapons maintenant

(COPY ( ) ➡)

La commande COPY a pour effet de copier 4 lignes du buffer 0 dans le buffer 1.

Pour des raisons évidentes, il n'est pas possible de copier en arrière.
La commande

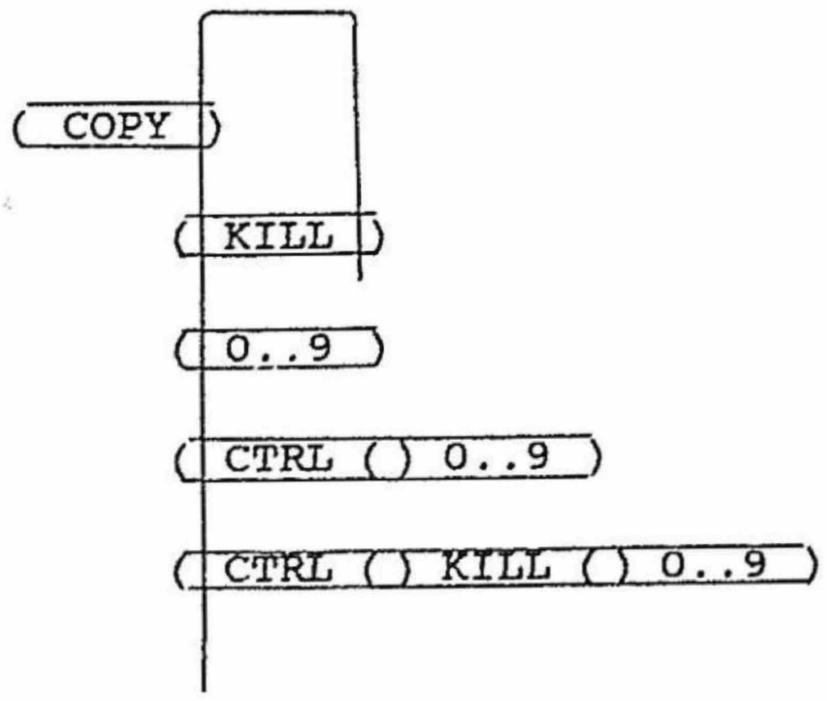
(COPY ( ) ⬅)

correspondant à un déplacement arrière a une signification spéciale: elle copie tout le buffer source quelle que soit la position du pointeur dans le texte.

De la même façon, il est possible de transférer (copier puis détruire) tout ou partie de buffer source dans le buffer destination. Cette opération est réalisée comme dans le cas des effacements en associant à la touche COPY et la touche KILL .
Par exemple le transfert d'une ligne du buffer source 1 dans le buffer destination 2 se fera de la façon suivante:

(COPY ( ) 2)
(KILL ( ) COPY ( ) ⬇)

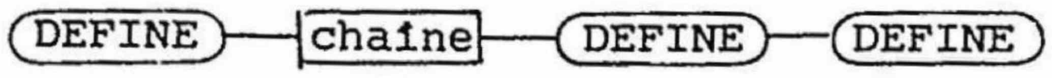
Notons qu'il est également possible d'insérer sous le pointeur le contenu d'un buffer autre que le buffer principal. Ceci est employé pour insérer des en-têtes, des titres et autres motifs dans le buffer principal. La commande est la même que l'initialisation d'une copie, excepté que le numéro du buffer est donné en maintenant la touche CTRL pressée. Si la touche fonction KILL est simultanément maintenue pressée, l'original de la copie est détruit.



2.8 Recherches et comptages

Pour se déplacer dans de grands textes, plutôt que de se servir de la touche CURSOR, il est préférable de définir une chaîne de caractères et de rechercher cette chaîne dans le texte par un ordre approprié. Un autre ordre permet de dénombrer les occurences de cette chaîne (par exemple pour compter des mots-clés ou le nombre de répétitions d'un mot trop souvent utilisé).

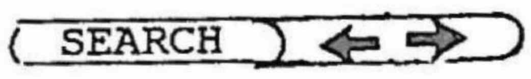
Avant de commencer les deux opérations précédemment décrites, il est nécessaire d'éditer la chaîne de caractères à chercher ou compter. Cette chaîne de caractères doit être éditée dans un buffer réservé à cet effet, inaccessible par l'utilisateur au moyen des commandes usuelles.
Pour définir cette chaîne, la séquence suivante doit être introduite:



La première pression sur la touche DEFINE a deux conséquences:
- Elle détruit le précédent contenu du buffer de définition.
- Elle affiche sur l'écran le buffer de définition dans la partie supérieure et le buffer d'édition dans la partie inférieure.

Pour éditer la chaîne de caractères, toutes les facilités d'édition sont à disposition (déplacements, effacements). La seconde pression sur la touche DEFINE a pour seul effet d'insérer dans le texte édité un double crochet pointu. La troisième pression sur la touche DEFINE affiche de nouveau le buffer principal sur tout l'écran, la chaîne définie étant mémorisée de façon invisible.

La recherche de la chaîne de caractères définie préalablement se fait au moyen de la touche fonction SEARCH combinée aux déplacements avant ou arrière.

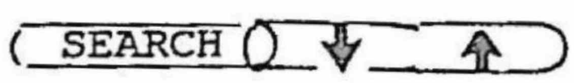


recherches

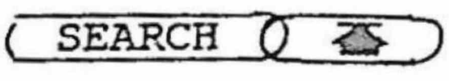
Le curseur s'arrête sur la première occurence en avant ou en arrière, trouvée dans le texte. Il est placé au début de la chaîne qu'il fallait trouver.

La recherche s'effectue indifféremment avec les majuscules et les minuscules. Par exemple, la chaîne de recherche "Fenetre" permet de trouver "fenêtre", "FENETRE", "FeneTRe", etc.
La recherche peut durer plusieurs secondes. Une pression sur la touche END permet d'interrompre la recherche.

Les comptages se font sans déplacement du curseur. Le comptage du nombre d'occurences se fait en aval, en amont ou à travers tout le texte. Le résultat du comptage est affiché au haut de l'écran

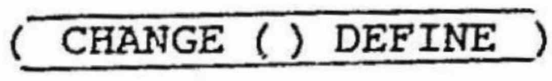


comptages aval/amont



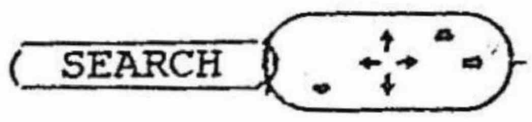
comptages partout

Pour réafficher le buffer de définition sans détruire son contenu, il suffit de presser sur



On peut éditer la chaîne avec les ordres usuels.
Il faut presser sur DEFINE pour sortir de ce mode.

Résumé des ordres de recherches et comptages:



2.9 Changements, échanges.

Il est non seulement possible de rechercher des chaînes de caractères, mais aussi de les changer contre une chaîne de remplacement. Celle-ci peut d'ailleurs être de longueur nulle, ce qui revient à détruire une chaîne donnée.
Avant de commencer une opération d'échange, il est nécessaire de définir la chaîne à rechercher et la chaîne de remplacement. Celles-ci sont éditées dans le buffer réservé à cet effet, inaccessible à l'utilisateur au moyen des commandes usuelles. Seule la séquence d'ordres ci-dessous permet d'éditer les deux chaînes.

(DEFINE)—[chaîne1]—(DEFINE)—[chaîne2]—(DEFINE)

La première pression sur la touche DEFINE a deux conséquences:
- Elle détruit le précédent contenu du buffer de définitions.
- Elle affiche sur l'écran le buffer de travail dans la partie inférieure.

Il faut ensuite taper la chaîne de caractères à changer. La seconde pression sur la touche DEFINE a pour seul effet d'insérer dans le texte édité un double crochet à droite qui est à interpréter comme *devient* ou *subsitué par*.
Il faut encore taper la chaîne de remplacement.
La troisième pression sur la touche DEFINE affiche à nouveau le buffer principal sur l'écran.

Le changement de l'occurence précédente ou suivante s'obtient en combinant la touche fonction CHANGE à un déplacement d'un caractère en arrière ou en avant.

(CHANGE ( ) ← →)

Si l'on de veut pas forcément changer l'occurence suivante, on peut alterner

(SEARCH ( ) →) et (CHANGE ( ) →).

Si les changements à effectuer sont nombreux, il est possible d'échanger toutes les occurences au-dessus ou au-dessous du pointeur, voire même toutes les occurences du buffer.

(CHANGE ( ) ↑ ↓) changements multiples

L'élimination d'une chaîne de caractères donnée est en fait le remplacement de cette chaîne par une autre chaîne de longueur nulle. La définition se fait de la façon suivante:

(DEFINE)—[texte]—(DEFINE)—(DEFINE)

La syntaxe des changements est la même que ci-dessus.

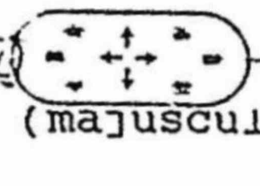
Résumé des commandes de changements:

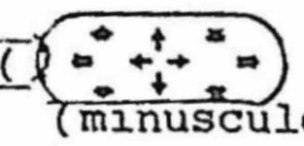
(CHANGE ( ) ↑ ← → ↓) changements

2.10 Conversion majuscules minuscules

Il est possible de changer une lettre, un mot, une phrase, etc. de majuscules et minuscules, ou inversement. Les flèches de déplacement sont utilisées pour définir l'amplitude du changement.

Si la lettre correspondant au déplacement est majuscule, la conversion minuscule --> majuscule est effectuée et si elle est minuscule, la conversion minuscule --> majuscule est effectuée.

(PROGRA( )CHANGE( (majuscules) — conversion minuscules --> majuscules

(PROGRA( )CHANGE( (minuscules) conversion majuscules --> minuscules

Les lettres minuscules accentuées sont transformées en majuscules, et les lettres majuscules sont transformées en minuscules sans accents (forcément).

2.11 Macros

Les macros offrent la possibilité de mémoriser une séquence de touches (insertion de caractères, déplacements, destruction,...), puis de répéter cette séquence.

(PROGRA( )MACRO) — démarre la mémorisation d'une séquence de touches. Le nom de l'éditeur (sur la première ligne de l'écran) apparaît en inversé pour indiquer ce mode MACRO. Toutes les commandes réagissent maintenant comme avant, l'éditeur mémorise simplement tout ce que l'utilisateuzr fait. Une pression de (PROGRA( )END) termine la mémorisation.

(MACRO) — Exécute la séquence de touches mémorisée

La séquence de touches est mémorisée dans le buffer numéro 12. Il est possible de travailler dans ce buffer avec la commande (SHOW( )MACRO). Chaque touche mémorisée occupe une ligne du buffer 12, de la façon suivante:

par exemple 000 - 101 correspond à A

code de la touche fonction en octal

code de la touche principale en octal

Le buffer 12 peut être utilisé comme n'importe quel autre buffer. Par exemple, il est possible de le modifier (d'ajouter de nouvelles touches), de la sauver sur disque, etc .

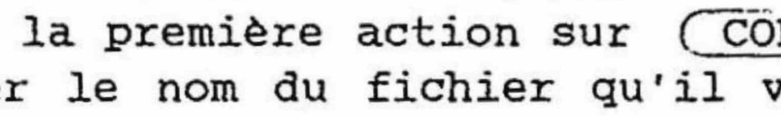
(CHANGE( )MACRO) — permet de compléter la mémorisation d'une séquence de touches. Les nouvelles touches tapées sur le clavier sont insérées à l'endroit où est situé le curseur dans le buffer 11. Après l'exécution d'une MACRO (MACRO), le curseur est à la fin du buffer 12.

(CURSOR( )MACRO) — Exécute répétivement la séquence de touches mémorisées. La touche END permet de stopper à la fin de l'exécution de la MACRO (SHIFT( )END) permet de stopper n'importe quand

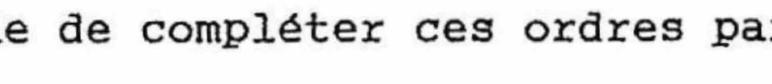
Si une touche fonction quelconque (CURSOR, COPY, KILL, ...) est pressée pendant l'exécution, l'écran montre tout ce qui se passe. L'exécution est donc nettement plus lente.

2.12 Entrées et sorties

Un éditeur de textes/programmes sans communication avec le monde extérieur est inutilisable. Il est nécessaire de pouvoir sauver le texte édité sur une mémoire de masse ou un périphérique, puis de récupérer ce texte lors d'une séance de travail ultérieure.
Pour permettre le sauvetage du texte édité, la commande COPY DEFINE suivie d'un nom de fichier aura l'effet désiré. Après l'éxécution de cette commande, la totalité du buffer d'édition aura été recopiée sur le fichier, voire le périphérique, mentionné.



L'opération inverse, la lecture d'un fichier, se fait au moyen de la commande SHOW DEFINE suivie du nom du fichier à lire. Celui-ci est inséré à partir de la position du pointeur jusqu'à concurence de la place disponible. En général, on insère dans un buffer vide.



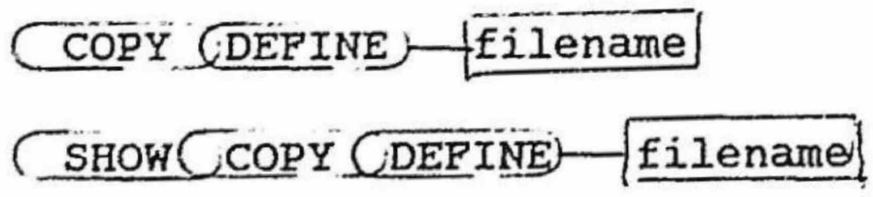
Pour éditer un fichier plus long que la demi-place maximale disponible sur la disquette, il faut éditer son fichier par tranches de dimensions inférieures au maximum, le reste du texte résidant sur disque.

La commande COPY SHOW DEFINE est une combinaison des deux commandes précédentes. Chaque action de celle-ci écrit sur le disque le contenu du buffer d'édition et, après avoir rendu la place ainsi libérée, amène la tranche suivante du fichier en mémoire jusqu'à concurence des deux-tiers de la place disponible. A la première action sur COPY SHOW DEFINE, l'opérateur est invité à taper le nom du fichier qu'il veut éditer. Deux cas se présentent alors:
- le fichier n'existe pas, et l'éditeur le crée, chaque commande COPY SHOW DEFINE écrivant et ajoutant le contenu du buffer sur le disque.
- le fichier existe, et l'éditeur amène la première tranche en mémoire.

Il est possible de compléter ces ordres par une réservation entre crochets.

Si le fichier est gros, COPY SHOW DEFINE FILE charge le buffer courant avec une partie du fichier, en laissant environ 10'000 caractères pour rajouter du texte ou utiliser les autres buffers.
L'ordre COPY SHOW DEFINE permet de passer à la partie suivante du fichier, en sauvant le début.

Résumé des ordres d'entrée-sortie:



REMARQUE: filename peut être un nom de fichier disque ou un nom de périphérique (par exemple: $LP, $PR, $PP, ...).

2.13 Appel de l'éditeur et retour au système

Pour appeler l'éditeur, on tape son nom suivi du nom du fichier à créer ou à modifier. Par exemple EPRO somme↵

Si le programme "somme.SR" existe, il est chargé par EPRO.
Dans le cas contraire, la question *Crée SOMME.SR ?* est posée, et l'utilisateur doit confirmer en tapant O.

Pour quitter l'éditeur la commande suivante rappelle le système d'exploitation. Dans le cas ou l'éditeur est en train de traiter un fichier par tranches, cette commande finira de recopier le fichier d'entrée sur le fichier de sortie.

PROGRA() END — sortie de l'éditeur

Après cette commande l'utilisateur doit répondre à une question; la première ligne de l'écran apparaît comme suit:

M(ise à jour, E(diteur, C(atastrophe

Si l'utilisateur répond 'M', l'éditeur met à jour la copie du fichier sur disque. Dans notre exemple, il sauve le programme édité dans le buffer 0 sous le nom "SOMME.SR", en détruisant l'ancienne version "SOMME.SR".

Si l'utilisateur répond 'C' on quitte l'éditeur sans mettre à jour la copie sur disque donc sans recopier les modifications éventuelles effectuées. La touche 'E' permet de retourner à l'éditeur si la commande PROGRA-END a été donnée par erreur.

2.14 Messages d'erreurs

En cas d'erreurs ou d'opérations spéciales, des messages apparaissent dans la première ligne de l'écran. En voici leur signification:

Mémoire pleine signale que la mémoire texte est pleine; il faut sauver ou détruire une partie des textes se trouvant dans les buffers.

Commande impossible est la réponse à une commande impossible et qui n'a pas été exécutée.

Chaîne pas trouvée signifie que la chaîne à chercher n'a plus pu être trouvée.

Chaîne inexistante apparaît lors d'une recherche ou d'un changement lorsque la chaîne de recherche n'a pas été définie

Macro incorrecte apparaît pendant l'exécution d'une macro lorsque le contenu du buffer 11 n'est pas correct

Détruit l'ancien fichier XXX signale que l'on essaie de copier sur un fichier qui existe déjà et demande s'il faut détruire l'ancienne version pour la remplacer par la nouvelle. Une frappe sur les touches Y ou O le permet; toute autre touche fait avorter l'ordre.

Map.dr error et Sys.dr error signalent une panne hardware du système ou une destruction d'une partie du programme. Si aucune erreur de manipulation ne s'est produite, faire subir un test de fiabilité au système.

3. LES ASSEMBLEURS

Les assembleurs AS et SMILE ont un certain nombre de caractéristiques communes décrites dans ce chapitre. Les commandes et exceptions spécifiques à ces programmes sont données dans les sections particulières de ces programmes.

3.1 Le langage

CALM (Common Assembly Language for Microprocessors) est un ensemble de notations pour assembleurs de microprocesseurs, développé à l'EPFL dès 1974. Il est caractérisé par un nombre restreint de mnémoniques clairs pour exprimer la fonction exécutée par l'instruction, et par des opérandes dont la liste suit le mnémonique, le mode d'adressage de chaque opérande étant décrit par une expression indiquant avec précision le calcul effectué pour obtenir l'adresse. Quelques signes spéciaux sont nécessaires pour préciser ces modes d'adressage. Le mode normal, pour lequel l'adresse est une adresse directe en mémoire, ne nécessite aucun signe.
Par exemple, pour lire la valeur de la position 42500, appelée SAVPC, on écrit

```
    LOAD  A,SAVPC
```

Pour lire la position suivante, on peut écrire

```
    LOAD  A,SAVPC+1  (c'est l'assembleur qui fait l'addition).
```

Pour préciser que ce n'est pas le contenu de la position mémoire, mais la valeur de l'adresse que l'on veut transférer, il faut écrire

```
    LOAD  HL,#SAVPC
```

HL contient alors une adresse mémoire valable, que l'on peut utiliser pour pointer et lire le contenu de cette position, en écrivant

```
    LOAD  A,(HL)
```

La parenthèse indique que ce n'est pas la valeur dans HL qui est transférée dans A (il y aurait d'ailleurs absurdité car A est un registre 8 bits et ne peut pas recevoir le mot de 16 bits contenu dans HL).
Des processeurs évolués (16 bits) admettent des expressions compliquées comme

```
    LOAD    A,(HL)+DEPL ;la valeur  DEPL  est ajoutée
                        ;à la valeur dans HL pour
                        ;obtenir l'adresse

    LOAD    A,(HL)+(B)  ;la valeur 8 bits dans B est
                        ;ajoutée à la valeur dans HL
                        ;pour obtenir l'adresse
```

Lorsque la position mémoire d'adresse donnée contient l'adresse de la valeur, et non pas la valeur, on parle d'adressage indirect et on utilise le signe @ (at).
Par exemple

```
    LOAD    A,@ADADVAL
```

Peu de microprocesseurs actuels ont l'adressage indirect.
Avec les microprocesseurs 8 bits, une lettre simple est utilisée pour un registre 8 bits: B, C. Une lettre double pour un registre 16 bits: BC, IX.
Les microprocesseurs 16 bits ont une structure plus riche qui oblige à préciser la longueur du registre dans le code de l'instruction:

```
    LOAD.B  R0,ADVAL    charge les 8 bits à l'adresse ADVAL
    LOAD.W  R0,ADVAL    charge les 16 bits aux adresses ADVAL
                        et ADVAL + 1
    LOAD.L  R0,ADVAL    charge 32 bits (long word).
```

3.2 Définitions

Séparateur:

Un séparateur est à choix une suite non vide d'espaces (noté SPACE, BLANK ou), de virgules ou de tabulateurs (TAB). Le tabulateur définit l'emplacement du prochain caractère comme étant au moins une position plus loin et à une distance multiple de 8 du premier caractère de la ligne. Dans la pratique, le choix entre la virgule ou le tabulateur est fixé par le contexte. L'espace est rarement utilisé comme séparateur.

Expression:

Une expression est formée de nombres, symboles, parenthèses et opérations arithmétiques. Lorsque la valeur des symboles utilisés est connue, la valeur de l'expression peut-être calculée, avec un résultat entier. Suite à une division, la partie fractionnaire est tronquée, sans arrondi. La représentation en mémoire est un nombre arithmétique dont la longueur dépend de l'instruction. L'assembleur signale les dépassements de capacité lorsqu'il calcule une expression. La définition est récursive étant donné que le facteur d'une expression peut être une expression, avec les règles de parenthèses habituelles.

Exemples: 612+(20-1)*40
(15+(200/(2*3)))*((2+3)-1)

Les opérations de groupage (parenthèses) et de puissance (^) ont la priorité sur les opérations multiplicatives, fois (*), divise (/) et ET logique (&) qui elles ont la priorité sur les opérations additives plus (+), moins (-) et OU logique (!). Le complément à deux d'un terme s'obtient au moyen du moins unaire (-). Le "non booléen" s'obtient au moyen du signe "~" (par exemple: ~0 vaut 377 ~1 vaut 0 ~2 vaut 0 .... ~377 vaut 0).

Exemples: (2*3)+(4*5) est équivalent à 2*3+4*5,
mais (2+3)*(4+5) n'est pas égal à 2+3*4+5

Symboles et nombres:

Avec les notations de symboles et de nombres, on atteint le dernier niveau de définitions. Les diagrammes de syntaxe ne contiennent plus de rectangles impliquant de nouvelles définitions. Un symbole commence toujours par une lettre, un point d'interrogation (?) ou un souligné (_) , pour le distinguer d'un nombre. Le point isolé est un symbole particulier qui a valeur du compteur d'adresse pour la ligne considérée.

Exemples de symboles: TOTO
?DICAR
HELLO?
UN_SYMBOLE

Exemples de nombres: 123
177777
19.

Un nombre peut être une suite de chiffres dans la base courante ou une suite de chiffres terminée par un point auquel cas le nombre sera analysé comme décimal. Une dernière catégorie de nombres est le caractère ASCII précédé par une apostrophe (par exemple. 'B vaut 102 octal). Dans ce dernier cas ce sera le code du caractère qui sera considéré.

Exemples d'expressions complexes:

```
TRUC!2^MACHIN!(5-BIDULE)!1
(100000-FINMEM)/400
(ADRESSE&(400-1))+OFFSET
```

3.3 Le fichier source

Un fichier à assembler est une suite de lignes contenant toute l'information permettant de créer automatiquement un fichier binaire exécutable.
Le fichier est formé de lignes séparées par des retours de ligne notés CR (↵).
Une première analyse montre que certaines lignes contiennent les instructions proprement dites (lignes d'instructions), d'autres sont des lignes d'affectation qui servent à assigner une valeur à un symbole. D'autres lignes sont des pseudo-instructions servant de guide pour l'assembleur et définissant des symboles et positions mémoire. Une ligne de fin termine le fichier. Elle peut éventuellement être suivie par d'autres lignes, mais celles-ci seront ignorées par l'assembleur.

3.3.1 La ligne d'instruction

Une ligne d'instruction est formée d'une étiquette, qui caractérise l'emplacement mémoire, de l'instruction proprement dite et d'un commentaire. Une ligne d'instruction peut ne pas avoir d'étiquette ou avoir plusieurs étiquettes, sur la même ligne ou non.

Exemples:

```
                instruction
        LOOP:   instruction
ETIQ1: ETIQ2:   instr.    ou    ETIQ1:          ou    ETIQ1:
                                ETIQ2:  instr.        ETIQ2:
                                                            instr.
```

Etiquette:

Chaque étiquette est contituée d'un symbole, c'est-à-dire d'un mot représentant un nombre (en l'occurence une adresse) suivi du signe ":". Des séparateurs peuvent être utiles de part et d'autre du symbole, mais ne sont pas nécessaires. Certaines étiquettes sont classées spécialement et portent le nom d'étiquettes locales. Elles sont caractérisées par le fait qu'elles comportent un ou deux chiffres suivi du signe '$' et que leur domaine de validité n'existe qu'entre deux étiquettes normales.

Exemples:
```
LOOP:
10$:    (étiquette locale)
```

Instruction:

Une instruction comporte au moins un code mnémotechnique ou mnémonique caractérisant l'opération effectuée par le processeur. Un séparateur précède généralement ce code et doit suivre si une condition de test et/ou des opérandes précisent l'opération.

Commentaires:

Un commentaire est caractérisé par le signe ";". Un texte quelconque peut-être écrit jusqu'à la fin de la ligne, celle-ci étant caractérisée par un retour à la ligne. Les commentaires sont ignorés par l'assembleur.

3.3.2 La ligne d'affectation

Une affectation est caractérisée par le signe "=". Elle permet d'assigner une valeur à un symbole.

Exemples:
```
BIDULE = 1
CHOSE = (BIDULE+5)*3
```

3.3.3 Les pseudo-instructions

On distingue les pseudo-instructions de commande, qui sont des instructions générales pour l'assembleur (mise en pages, action sur compteur d'adresses, choix de la base, etc.) et les pseudo-instructions de génération qui réservent ou assignent des positions mémoire.

Pseudo-instructions de commande

.TITLE
donne un titre au programme. Ce titre est répété au début de chaque page du listing. Il est recommandé de n'employer cette pseudo que comme première ligne du programme afin de faciliter la gestion de la table des matières.

```
Exemple:        .TITLE  DEMONSTRATION   ;commence une nouvelle page
```

.SBTTL
permet de nommer des parties de programme en leur donnant un sous-titre qui sera imprimé au début de chaque page, à coté du titre du programme. De plus, une définition de sous-titre éjecte une page du listing afin de débuter le nouveau chapitre au début d'une page.

.LOC
initialise le compteur d'adresses courant à la valeur suivant la pseudo. Si cette pseudo n'est pas définie au début du listing, SMILE prendra 100000 comme valeur par défaut, tandis que AS prendra 0 comme valeur part défaut.

Il est conseillé de ne pas utiliser la pseudo-instruction .LOC avec SMILE lorsque l'on veut exécuter le programme assemblé avec (PROGRA( ) V). Cette valeur 104000 risque d'être changée dans les révisions ultérieures.

.ALIGN
permet d'aligner le PC de l'assembleur à une valeur à choix (donc par exemple de faire commencer des parties de programmes à des adresses rondes)

```
Exemple:        .ALIGN  400
                TABLE:  ...
```

.PC
permet l'utilisation de plusieurs compteurs d'adresses. La valeur qui suit la pseudo est le numéro du compteur et doit être comprise entre 0 et 7 inclus.

```
Exemple:        ROM =   0
                RAM =   1
                .PC     ROM
                .LOC    0
                .PC     RAM
                .LOC    100000
                ;variables
                .PC     ROM
                ;programme
                ...
                .PC     RAM
                ;une variable supplémentaire
                .PC     ROM
                ;suite du programme
```

.END
signale à l'assembleur la fin du fichier source. Les lignes suivantes sont ignorées. La valeur qui suit la pseudo est l'adresse à laquelle le programme doit commencer son éxécution lors de son chargement. Si aucune valeur n'est spécifiée, ce programme ne pourra pas être chargé, mais sera quand même assemblé.

.PAGE
éjecte une page de listing à un endroit quelconque du programme afin d'en améliorer la lisibilité.

.LINES
permet de choisir le nombre de lignes par page. .LINES 0 (zéro) donnera un listing continu.

.LIST
permet de n'imprimer que certaines parties d'un programme suivant la valeur de l'expression booléenne qui suit.

```
Exemple:        .LIST   0       supprime le listing
                .LIST   1       autorise le listing
```

.PROC
permet de générer du code pour un processeur au choix, pour autant que le fichier de paramètres soit disponible sur la mémoire de masse.

```
Exemple:        .PROC   Z80
```

.REF
déclare que le fichier dont le nom suit la pseudo doit être pris comme fichier de définition (évite de surcharger le fichier source de définitions et autres affectations). L'extension réservée à ces fichiers de définition est .ST

```
Exemple:        .REF    SM6     charge le fichier de définition SM6.ST
```

.RADIX (ou .RDX)
définit la base du système de numération utilisée par l'assembleur. La valeur qui suit la pseudo est analysée dans la base courante, donc dans *l'ancienne* base. Pour éviter toute confusion, il est conseillé de donner la valeur de la nouvelle base au moyen d'un nombre décimal (terminé par un point). La base d'entrée par défaut est la base 8 (octal).

```
Exemple:        .RADIX  16.
                passe au système hexadécimal
```

.OCT, .HEX
définissent la base de sortie. C'est dans cette base que sera imprimé le listing. La base de sortie par défaut est l'octal.

.IF
assemblage conditionnel si la valeur de l'expression booléenne qui suit la pseudo est différente de zéro

.ENDIF
signale la fin d'une partie assemblée conditionnellement.

.ELSE
située entre un .IF et un .ENDIF, cette pseudo a pour effet d'inverser la condition calculée au .IF précédent.

```
Exemple:        LOAD    A,TRUC
                .IF     SMAKY
                SUB     A,#2
                .ELSE
                ADD     A,#5
                .ENDIF
                LOAD    C,A
                etc.
```

.INS
permet d'insérer un source dans un programme. Ce source sera assemblé de la même manière que le programme maître, sauf les erreurs éventuelles qui sont simplement affichées sur l'écran, sans être copiées dans le nouveau source.

```
Exemple:        .INS SYS.RF
                a même effet final que .REF SM6
                mais est plus lent
```

Pseudos de génération

.BLKB
réserve en mémoire le nombre de bytes indiqué.
Cette valeur doit être calculable lors de la première passe de l'assembleur.
Exemple:
```
.BLKB   64.
réserve 64 bytes
```

.BLKW
réserve en mémoire le nombre de mots indiqué

.BYTE ou .B
génère du code pour toutes les valeurs qui suivent sous forme de bytes consécutifs
Exemples:
```
.RADIX  16
.BYTE   123
.BYTE   99.
.BYTE   'A
.BYTE   0A3     ;comme les nombres doivent
                ;commencer par un chiffre,
                ;on a ajouté un 0 (zéro) devant
                ;le nombre hexa A3
```
Dans ce cas, on peut écrire plus simplement:
```
.B      123,99.,'A,0AB
```
(virgule, espace ou tab pour séparer)

.WORD ou .W
génère du code pour toutes les valeurs qui suivent sous forme de mots consécutifs.
Exemple:
```
.WORD   34567
```

Il est possible de générer des bytes et des mots mélangés en déclarant une pseudo du type .BWBBW qui, dans ce cas générera un byte suivi d'un mot, de deux bytes et d'un mot et ceci cycliquement.

.ASCII
mémorise dans des paires de positions mémoire successives les codes ASCII de tous les caractères entre guillemets
Exemple:
```
.ASCII  "BONJOUR"
.ASCII  "texte"
```

.ASCIZ
comme ASCII, mais ajoute un byte nul après les codes ASCII des caractères (les codes spéciaux à insérer dans la chaîne sont mis entre parenthèses pointues.
Exemple:
```
.ASCIZ  "texte de <15><12> lignes"
.ASCIZ  "signe<'"> et signe<'<>"
```

3.4 Le travail de l'assembleur

L'assembleur transforme le fichier source en code binaire conformément aux pseudo-instructions de commande et à l'aide de la table de description du processeur. L'assembleur travaille en deux passes:

Lors de la premiere passe, il se contente de collectionner tous les symboles en leur attribuant une valeur si celle-ci est définie. Il contrôle également la syntaxe des différentes instructions et enlève les anciens messages d'erreurs qui subsistent.

Lors de la seconde passe, l'assembleur génère le code binaire dans le fichier spécifié. Il fournit également un listing de l'assemblage si celui-ci a été demandé. Le travail le plus caché est l'insertion des messages signalant les erreurs dans le fichier source directement. A la fin de la seconde passe, l'assembleur adjoint une cross-référence map au fichier listing permettant de retrouver plus facilement les différents symboles dans le programme. Le travail terminé, il ne reste plus qu'à signaler le nombre d'erreurs à l'utilisateur (auquel cas le fichier binaire est automatiquement détruit) ou que l'assemblage a été réalisé sans découverte d'erreur. Ceci ne veut pas dire que le programme assemblé fonctionne de façon satisfaisante !

3.5 Le fichier objet

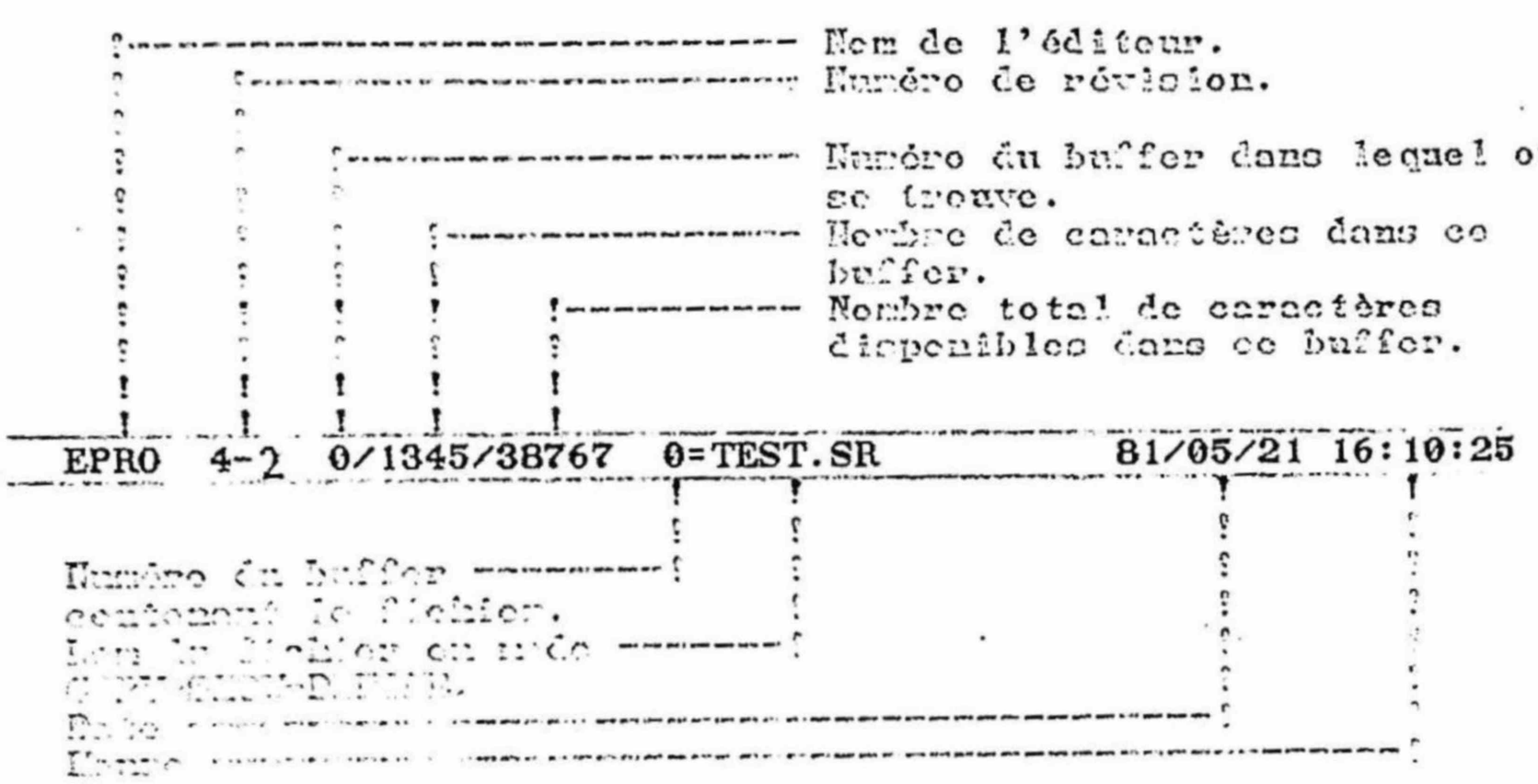
Le fichier objet généré par l'assembleur est directement une image mémoire du fichier tel qu'il sera en mémoire lors de son exécution.

L'avantage de cette forme de représentation réside dans un chargement très rapide, aucune conversion de code n'ayant lieu.

Le désavantage est la place occupée sur disque par des programmes occupant les deux extrémités de la mémoire. Ces derniers n'étant pas les plus courants, l'utilisateur s'accomodera de cette petite faiblesse.

4. EPRO: l'éditeur de programmes

Pour éditer le programme source, on utilise l'éditeur EPRO .
Cet éditeur permet de travailler avec des fichiers ayant l'extension .SR par défaut. Lorsque l'éditeur est chargé, la ligne supérieure de l'écran donne les informations suivantes:



Les caractéristiques de cet éditeur ont été décrites au début de la notice. On se contentera ici de donner des indications sur les diverses manières de charger un programme source, de le corriger et de le sauver.

EDITION D'UN PROGRAMME

Il y a deux façons d'éditer un programme, suivant sa grandeur:

1) Edition d'un programme court

On entend par programme court un programme qui peut être contenu entièrement dans un buffer d'EPRO.

Depuis le CLI, taper

EPRO↵ qui charge l'éditeur

Il est alors possible d'éditer un nouveau programme.

Pour lire un ancien programme depuis la disquette, taper

(SHOW ( ) DEFINE)─┤nom↵

Pendant l'édition du programme, il est prudent de faire régulièrement des copies de sécurité, en tapant par exemple:

(COPY ( ) DEFINE)─┤T↵

Lorsqu'un ancien fichier T.SR existe déjà sur la disquette, il faut répondre (O) à la question "Détruit l'ancien fichier T ?"

Lorsque l'édition est terminée, on donne l'ordre

(COPY ( ) DEFINE)─┤nom↵

pour sauver le programme sur la disquette. Si le fichier "nom.SR" existe déjà et qu'on désire le remplacer par la nouvelle version, on tape (O) en réponse à la question "Détruit l'ancien fichier nom.SR ?".

L'ordre (PROGRA ( ) END) permet de quitter EPRO.
Si (PROGRA( )END) est tapé par erreur, il suffit de presser sur E pour retourner à l'éditeur.

Pour retourner au CLI, il faut encore presser sur (S).

2) Edition d'un long programme

Un long programme est un programme qui ne peut pas être contenu en entier dans le buffer d'EPRO; dans ce cas on parlera de pages, une page étant le contenu d'un buffer d'EPRO.

(EPRO) charge l'éditeur

(COPY ( ) SHOW ( ) DEFINE) — nom↵ lit la première page du programme "nom.SR" dans un buffer d'EPRO

Il est alors possible d'éditer la première page du programme.

(COPY ( ) SHOW ( ) DEFINE)

écrit la page courante sur la disquette, puis lit la page suivante.

(PROGRA ( ) END) permet de quitter d'éditeur.

(E) retourne à l'éditeur
(M) recopie la totalité du programme sur le disque (même les pages que l'on n'aurait pas lues)
(C) retourne au CLI sans mettre à jour le programme

Remarque:

La commande (EPRO) — nom↵ est équivalente à la séquence

EPRO↵
(COPY ( ) SHOW ( ) DEFINE) — nom↵

De plus il est possible d'éditer des programmes courts comme des longs programmes (le contraire n'étant évidemment pas vrai !)

La commande EPRO nom nouveau/N permet de créer le fichier nouveau.SR à partir du fichier nom.SR. Après avoir fait quelques modifications puis (PROGRA ( ) END) , le fichier nouveau.SR contiendra donc les modifications, tandis que le fichier nom.SR ne sera pas changé.
Cette commande peut être utile pour éditer de très long fichier en mettant le fichier à éditer sur DX0 et en créant le nouveau fichier sur DX1 avec la commande EPRO nom DX1:nouveau/N

```
                                EEEEE  PPPP    RRRR     000
                                E      P   P   R   R   0   0
                                EEE    PPPP    RRRR    0   0
                                E      P       R   R   0   0
                                EEEEE  P       R   R    000
```

Majuscules minuscules. --(flèches ( ) CHANGE)

Minuscules majuscules. --(FLECHES ( ) CHANGE)

Aide à l'utilisateur. ---------- (H)

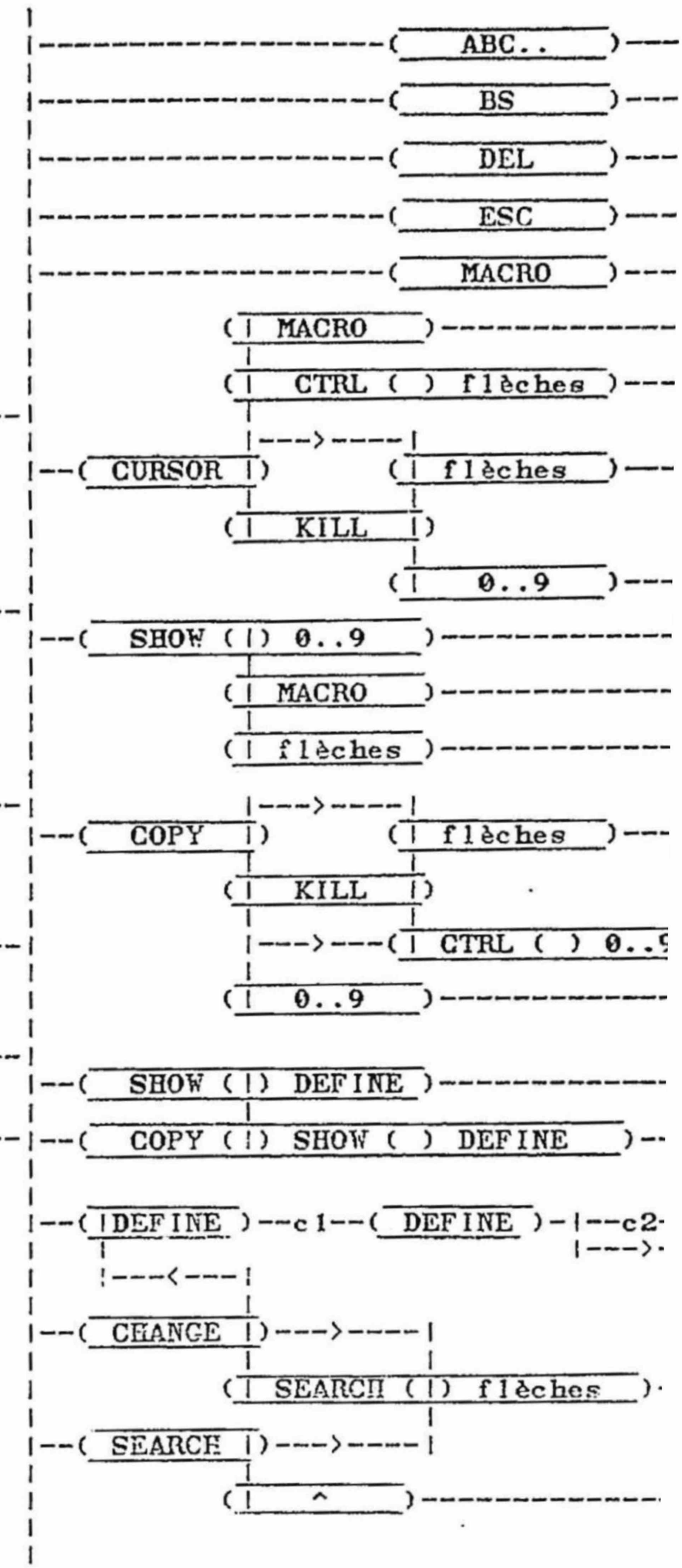
Affiche les CRs, TABs. ---------- (CR, TAB)

Supprime les CRs, TABs. ---------- (CR, TAB)

Détruit un fichier. --- file ------ (DEFINE)

Démarre ou stoppe l'enregistrement d'une macro. ---------- (MACRO)

Quitte l'éditeur. ---------- (END ( ) PROGRA)



5. ETEX: l'éditeur de textes

Ce programme est prévu pour l'édition de textes au kilomètre, texte qui sera mis en page par d'autres programmes ou par des imprimantes intelligentes.

Les principales différences entre ETEX et EPRO sont:

1) Lorsque l'on a tapé une ligne de 64 caractères:

Avec EPRO: si l'on continue à taper (sans presser RETURN), les caractères suivants sont mémorisés, mais restent invisibles pour l'utilisateur. Il y a toutefois une possibilité de les visualiser en déplaçant latéralement l'écran au moyen de l'ordre (SHOW) combiné avec une déplacement horizontal (S, D, F ou G).
Après le RETURN, un "!" signale que la ligne déborde.

Avec ETEX: le changement de ligne se fait automatiquement. Les mots de fin de ligne qui sont trop longs sont reportés à la ligne suivante (sans être coupés).

2) Déplacements:

Les déplacements avec la touche CURSOR n'ont pas le même effet.

Par exemple:
Avec EPRO, (CURSOR ( ) C) descend d'une ligne,
Avec ETEX, (CURSOR ( ) C) se déplace d'une phrase, c'est à dire se positionne sur le prochain point.

3) (PROGRA( ) P) permet de voir le texte comme il serait avec EPRO, c'est-à-dire que chaque ligne de l'écran correspond à un paragraphe. Dans ce mode, les déplacements sont identiques à ceux d'EPRO, y compris les déplacements horizontaux (SHOW( ) flèches).
(PROGRA(T) permet de revenir à l'affichage en mode ETEX.

4) Extensions prises par défaut:

Avec EPRO: .SR
Avec ETEX: .TX

```
EEEEE   TTTTT   EEEEE   X   X
E         T     E       X X
EEEE      T     EEEE      X
E         T     E       X X
EEEEE     T     EEEEE   X   X
```

Majuscules minuscules. --(flèches ( ) CHANGE)

Minuscules majuscules. --(FLECHES ( ) CHANGE)

Affiche en coupant les mots. ---------- (T)

Affiche une ligne par paragraphe. ---------- (P)

Aide à l'utilisateur. ---------- (H)

Affiche les CRs, TABs. ---------- (CR, TAB)

Supprime les CRs, TABs. ---------- (CR, TAB)

Détruit un fichier. --- file ------ (DEFINE)

Démarre ou stoppe l'enregistrement d'une macro. ---------- (MACRO)

Quitte l'éditeur. ---------- (END ( ) PROGRA)

(PROGRA) (PROGRA) (PROGRA) (KILL) (CHANGE) (PROGRA)

(ABC..)
(BS)
(DEL)
(ESC)
(MACRO)

(MACRO) (CTRL ( ) flèches)

(CURSOR) (flèches) (KILL) (0..9)

(SHOW ( ) 0..9) (MACRO) (flèches)

(COPY) (flèches) (KILL) (CTRL ( ) 0..9) (0..9)

(SHOW ( ) DEFINE)

(COPY ( ) SHOW ( ) DEFINE)

(DEFINE) --c1-- (DEFINE) --c2-->

(CHANGE) (SEARCH ( ) flèches) (SEARCH)

Pour forcer un saut de page: ^L
(SHIFT ( ) RETURN)

L'éditeur de texte est complété par un programme de justification: JUSTIF.

U T I L I S A T I O N D E J U S T I F . S M

Ce programme permet, à partir d'un texte tapé au km avec ETEX, de le mettre en page en choisissant la largeur de la colonne et le type de justification.

On appelle le programme en tapant:

JUSTIF ↵

Puis on répond aux questions qui apparaissent sur l'écran.

On indique ainsi successivement:

. Le nom du texte à traiter avec son extension
. Le nom du fichier de sortie (un nom de fichier disque
  ou $LP pour obtenir directement une copie papier)
. La largeur de la colonne (en nombre de caractères)
. Le type de justification.

L (left)	correspond à une justification à gauche seulement
R (right)	correspond à une justification à droite seulement
C (center)	centre chaque ligne (sans justification)
A (aligned)	justifie des deux côtés

(Attention: ces commandes doivent être en majuscules)

Dans la version actuelle de JUSTIF les tabulateurs sont considérés comme des espaces simples.

6. AS: l'assembleur paramétrisable

Prendre une disquette contenant les fichiers:

EPRO.SM	(éditeur de programmes)
SM6.ST ou FLO.ST	(symbols prédéfinis)
AS.SM	(assembleur)
Z80.SM	(module pour le processeur Z80)
XREF.SM	(générateur de cross-références)

Pour éditer le programme source, on utilise l'éditeur EPRO .

On indique au début du programme les pseudos utilisées, soit en général:

```
.TITLE ESSAI

.PROC   Z80
.REF    SM6

.LOC    100000
```

Pour assembler le programme ESSAI.SR, les ordres suivants peuvent être donnés:

AS ESSAI assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM . S'il y a eu une ou plusieurs erreurs d'assemblage, ESSAI.SR contient les messages d'erreurs, et le fichier ESSAI.SM n'existe pas.

AS ESSAI/L assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM et un listing ESSAI.LS .

AS ESSAI TOTO/L assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM et un listing TOTO.LS .

AS ESSAI/L/X assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM, un listing ESSAI.LS et une cross-référence ESSAI.XR . Les lignes dans le fichier listing seront numérotées pour permettre une recherche à l'aide de la cross-référence.

AS ESSAI TOTO/B assemble le fichier ESSAI.SR et génère un binaire TOTO.SM .

AS ESSAI/S assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM et une table des symbols ESSAI.ST . Cette table des symbols pourra être utilisée à l'aide de la pseudo-instruction .REF ESSAI .

AS ESSAI/E assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM et un fichier contenant les messages d'erreurs ESSAI.ER . ESSAI.ER contient uniquement les erreurs rencontrés dans les .INS . Les autres erreurs sont dans le fichier ESSAI.SR .

AS ESSAI/M assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM et un fichier ESSAI.MS contanant les messages qui apparaissent normalement sur l'écran.

AS ESSAI TOTO/N assemble le fichier ESSAI.SR et génère un binaire ESSAI.SM et un nouveau fichier source TOTO.SR . S'il y a une ou plusieurs erreurs, elles seront donc insérées dans le fichier TOTO.SR .

Exemple d'ordre complexe:
AS ESSAI/L TOTO/B/X TITI/E TATA/S
Assemble le fichier ESSAI.SR et génère les fichiers:
- ESSAI.SR nouveau source avec les erreurs éventuelles
- ESSAI.LS listing
- TOTO.SM binaire (s'il n'y a pas eu d'erreurs)
- TOTO.XR cross-référence
- TITI.ER erreurs dans les .INS
- TATA.ST table des symbols

RESUME:

switch	extension	contenu du fichier
/B	.SM	binaire
/L	.LS	listing (*)
/X	.XR	cross-référence (*)
/S	.ST	table des symbols (pour un .REF)
/E	.ER	erreurs dans les .INS (*)
/M	.MS	message de l'écran (*)
/N	.SR	nouveau source (*)

Les (*) indiquent des fichiers qui peuvent être imprimés ou édités.

EXEMPLE:

Soit le source incorrect ESSAIKO.SR suivant:

```
                .TITLE ESSAI

          .PROC   Z80
          .REF    SM6

          .LOC    100000

ESSAI:    LOAD    C,#LINES
          .W      ?IDIS
LOOP:     .W      ?GET
          .W      ?DITEX
          .W      ?RETURN
          JUMP    .LOOP
          .END    ESSAI
```

Si on donne l'ordre AS ESSAIKO
on créera sur la disquette, au lieu du fichier exécutable ESSAIKO.SM, un nouveau source ESSAIKO.SR où les erreurs sont mises en évidence.

Nous visualisons ce source avec EPRO en donnant l'ordre EPRO ESSAIKO et nous voyons maintenant sur l'écran:

```
                .TITLE ESSAI

        .PROC   Z80
        .REF    SM6

        .LOC    100000

ESSAI:  LOAD    C,#LINES
        .W      ?IDIS
LOOP:   .W      ?GET
                ^ Undefined symbol
        .W      ?DITEX
        .W      ?RETURN
        JUMP    LOOP
        .END ESSAI
```

Nous corrigeons la faute en remplaçant ?GET par ?GETLINE (le programme ôtera lui-même le message "^ Undefined symbol"), nous sauvons le fichier corrigé sous le nom ESSAI.SR et redonnons l'ordre AS ESSAI .

Cette fois l'assemblage se fait correctement et le fichier exécutable ESSAI.SM est créé sur la disquette.

Pour créer également un fichier listing, on donne l'ordre AS ESSAI/L qui crée, en plus du binaire ESSAI.SM, le fichier listing ESSAI.LS, que nous pouvons visualiser en donnant l'ordre EPRO ESSAI.LS .

```
30/01/81        12:17:43                TABLE           OF          CONTENTS
01-01


        01      ESSAI

                30/01/81                12:17:43                            ESSAI
01-01


                                                .TITLE ESSAI

                                        .PROC   Z80
                                        .REF    SM6

        100000                          .LOC    100000

        100000  016 024         ESSAI:  LOAD    C,#LINES
        100002  347 126                 .W      ?IDIS
        100004  347 005         LOOP:   .W      ?GETLINE
        100006  347 006                 .W      ?DITEX
        100010  347 043                 .W      ?RETURN
        100012  030 370                 JUMP    LOOP
                        100000          .END ESSAI


000009 references
Source file 000011 usefull lines long
Binary file 000014 bytes long
Assembly time: 0001 seconds     0660 lines/min
```

Remarques:

Les noms qui suivent l'ordre AS peuvent être aussi des noms de périphériques ou de fichiers sur une autre unité de disquette.
AS ESSAI $LP/L créera un listing du programme

Pour assembler les très longs programmes, par exemple le BASIC, on réserve une disquette pour le source, et l'autre pour le programme assembleur. On donne alors l'ordre: AS DX1:BASIC[573]

Pour pouvoir insérer les erreurs dans le source, l'assembleur utilise un fichier temporaire ayant le même nom que le fichier source, mais avec l'extension .SC . Les étapes successives sont:

1) destruction du fichier .SC
2) lecture ligne par ligne du fichier source .SR
3) écriture ligne par ligne dans le fichier .SC avec en plus l'écriture des erreurs éventuelles
4) lorsque l'assemblage est terminé, destruction du fichier .SR
5) changement de nom du fichier .SC en fichier .SR

Pour pouvoir générer la cross-référence, l'assembleur génère un fichier intermédiaire avec l'extension .XR ainsi qu'une table des symboles avec l'extension .ST . Lorsque l'assembleur a terminé ses deux passes, il démarre le programme XFER.SM qui lit les fichiers .XR et .ST et qui crée une cross-référence imprimable avec l'extension .XR .
Si l'assembleur n'a pas suffisemment de place sur la disquette pour générer le fichier intermédiaire .XR, le message "Xref map suppressed" apparait sur l'écran, puis l'assembleur continue normalement son travail.

Attention: l'assembleur utilise beaucoup de fichiers. La disquette doit pouvoir les contenir tous !

Pour chercher les erreurs, on charge le source avec EPRO, et on donne l'ordre (SEARCH ( ) ^).
On corrige les instructions erronées seulement; l'assembleur ôte lui-même les messages d'erreur lors de la prochaine passe.

7. SMILE: l'éditeur-assembleur

SMILE permet de créer et de tester rapidement de petits programmes écrits en langage d'assemblage CALM pour processeur Z80.

Création d'un programme

1) Appeler l'éditeur-assembleur en tapant SMILE↵
2) Taper le programme
3) Sauver temporairement le programme toutes les 10 minutes en donnant l'ordre (COPY ( ) DEFINE)—T↵
4) Sauver en donnant l'ordre (COPY ( ) DEFINE)—nom↵
5) Quitter l'éditeur par (PROGRA ( ) END) suivi de (O)

Modification d'un programme

1) Il y a deux façons de charger un programme depuis la disquette
   depuis le CLI: taper SMILE nom↵
   depuis SMILE: taper (SHOW ( ) DEFINE)—nom↵
2) Corriger le source
3) Le sauver en donnant dans tous les cas l'ordre
   (COPY ( ) DEFINE)—nom↵
4) Quitter l'éditeur par (PROGRA ( ) END) suivi de (O)

Assemblage d'un programme

Donner l'ordre (PROGRA ( ) Z)
S'il y a des erreurs, le programme insérera des messages d'erreur dans le source

Recherche des erreurs

Donner l'ordre (SEARCH ( ) ^)

Exécution d'un programme

Avant d'exécuter le programme s'assurer que le source est sauvé.
Donner l'ordre (PROGRA ( ) V)
Si nécessaire, presser sur BREAK pour interrompre et sur SPACE pour afficher les registres, puis encore une fois sur SPACE pour revenir dans SMILE.

Sauvetage du binaire

Donner l'ordre (PROGRA ( ) B)
puis le nom du programme lorsque l'écran montre FILE:

Fichier listing

Donner l'ordre (PROGRA ( ) L)
puis le nom du programme lorsque l'écran montre FILE:

Transmission du binaire format PDP11 par SIMSER

Assembler, puis donner l'ordre (PROGRA ( ) T)

Passage au moniteur

Donner l'ordre (PROGRA ( ) Q)
Retour à SMILE en pressant S.

Destruction du binaire

Donner l'ordre (KILL( )CURSOR( ) B)
pour détruire le binaire en mémoire (créé par (PROGRA( ) Z)). Cela permet de gagner de la place pour éditer.

```
 SSSS   M   M   III   L       EEEEE
S       MM MM    I    L       E
 SSS    M M M    I    L       EEEE
    S   M   M    I    L       E
SSSS    M   M   III   LLLLL   EEEEE
```

Assemble. ---------- (Z)

Crée un fichier binaire. ---------- (B)

Crée un fichier listing. ---------- (L)

Transmet le binaire en format PDP. ---------- (T)

Moniteur. ---------- (Q)

(PROGRA) ---

Exécute. ---------- (V)

Continue après un TRAP. ---------- (C)

Majuscules minuscules. -- (flèches ( ) CHANGE)

Minuscules majuscules. -- (FLECHES ( ) CHANGE)

Aide à l'utilisateur. ---------- (H)

(SHOW) ---

Affiche les registres. ---------- (R)

Affiche les CRs, TABs. ---------- (CR, TAB)

Supprime les CRs, TABs. ---------- (CR, TAB)

(KILL) ---

Détruit un fichier. --- file ------ (DEFINE)

Quitte l'éditeur. ---------- (END ( ) PROGRA) ---

```
|---------------------( ABC.. )--
|
|---------------------( BS )--
|
|---------------------( DEL )--
|
|---------------------( ESC )--
|
|
|
|                 ( CTRL ( ) flèches )--
|                      |
|                      |--->----|
|--( CURSOR )          ( flèches )--
|                      |
|                 ( KILL )
|                      |
|                 ( 0..9 )--
|
|--( SHOW ( ) 0..9 )-----------
|
|                 ( flèches )-----------
|
|                      |--->----|
|--( COPY )            ( flèches )--
|                      |
|                 ( KILL )
|                      |
|                 |--->---( CTRL ( ) 0..
|                      |
|                 ( 0..9 )-----------
|
|--( SHOW ( ) DEFINE )-----------
|                      |
|--( COPY )            |
|                      |
|--( DEFINE )--c1--( DEFINE )-|--c2
|                             |--->
|---<---|
|       |
|--( CHANGE )--->----|
|                    |
|              ( SEARCH ( ) flèches )
|                    |
|--( SEARCH )--->----|
|       |
|       ( ^ )-----------
|
```