CBS
ELECTRONICS

Boite Postale 50016
Z.I. Paris Nord II
95945 Roissy Charles de Gaulle Cedex

ADAM™

MACRO ASSEMBLEUR "MACADAM"©

CONCEPTION: RÉGICIEL
RÉALISATION: J.-L. PRONIER
P. SANTONI

CBS
ELECTRONICS

SOMMAIRE

I MISE EN OEUVRE

Il suffit de mettre la cassette dans le lecteur puis de faire "RESET".Le système se charge de lancer le programme.
Attendez que le premier menu s'affiche. En tapantsur les touches "flèche haute" et "flèche basse", vous modifiez la couleur du fond et du texte.

II-0 LE MENU DE SELECTION

Le prenier menu est simplement un menu de sélection d'autres menus.

1- le mode édition
2- le menu de contrôle des entrées/sorties
3- le menu d'assemblage
4- le menu d'initialisation
5- le menu des paramètres d'environnement

Pour acceder à l'un de ces menus, il vous suffit de taper sur 1,2,3,4 ou 5

II-1 ENVIRONNEMENT

Ce menu vous permet de choisir la configuration dans laquelle vous allez travailler.
Ce choix permet de sélectionner pour la mémoire de masse, les unités de lecture et d'écriture, ainsi que pour l'imprimante, le choix du papier, du nombre de lignes imprimées et du nombre de lignes dans la page.
Il vous permet aussi d'introduire un titre à afficher au début de vos listes. Pour modifier une option, positionnez-vous à l'aide des flèches "haut" et "bas" sur l'option. Tapez ensuite la valeur de votre option. Tapez sur "RETOUR T/T" pour revenir au menu principal.
Quand vous introduisez un titre, vous pouvez utiliser les caractères du clavier, ainsi que que les touches "droite" et "gauche".

II-2 LE MENU D'INITIALISATION

C'est le plus simple. Il demande simplement la confirmation de la destruction du buffer de travail. Son but est d'eviter une destruction du programme suite à une fausse manoeuvre

II-3 LE MODE D'EDITION

II-3-1 LE FONCTIONNEMENT

Le principe est de travailler plein écran.
L'écran est une fenêtre de votre texte qui vous permet d'inclure des phrases ou de modifier ce texte. Chaque phrase du texte est représentée par deux lignes d'écran. Ce qui permet d'écrire des phrases jusqu'à 48 caractères. Chaque phrase est séparée de la suivante par le caractère "RETURN". Vous avez à votre disposition l'ensemble du texte dans lequel vous pouvez vous déplacer. L'appui d'une touche provoque l'insertion du caractère à l'endroit où se trouve le curseur.

II-3-2 LES CARACTERES SPECIAUX

II-3-2-1 DEPLACEMENT DU CURSEUR

Le déplacement du curseur se fait à l'aide du pavé des flèches à droite sur le clavier.
La touche "<---" déplace vers la gauche.
La touche "--->" déplace vers la droite.
La touche flèche "basse" déplace vers le début de la ligne suivante.
La touche flèche "haute" déplace vers le début de la ligne précédente.
La touche "R" positionne le curseur au début de l'écran.
La touche "contrôle <---" positionne au début de votre texte.
La touche "contrôle --->" positionne en fin de votre texte.
La touche "contrôle flèche haute" déplace l'écran de 4 phrases vers le haut.
La touche "contrôle flèche basse" déplace l'écran de 4 phrases vers le bas.
Le déplacement du curseur se fait d'un caractère du buffer au suivant. Il n'est pas possible d'accéder aux zones après "TAB" ou "RETURN" ainsi qu'après le 48 ième caractère. L'éditeur gère des tabulatures correspondant aux zones labels et opérandes. La touche "tab" permet de formater les phrases sous forme de tabulation. Ces tabulations positionnent respectivement au code opération et à l'opérande.

II-3-2-2 AUTRES CARACTERES SPECIAUX

La touche "RETOUR f/f" retour au menu principal.
les touches "I" et "II" sont reliées et permettent des recherches dans le texte.
La touche "I" permet d'introduire le texte à rechercher. Quand vous introduisez le texte vous pouvez utiliser les caractères du clavier ainsi que la touche "RETURN", pour effacer le dernier caractère introduit. "RETURN" marque la fin de l'introduction. "RETURN f/f" provoque l'abandon de la saisie du texte.
La touche "II" positionne à la première occurance de la chaine de caractères introduite à l'aide de "I" . Si n'y a pas eu de choix cela provoque un positionnement en fin de fichier. La recherche débute après l'endroit où est positionné le curseur. Si la chaine est trouvée, l'écran est positionné sur la ligne correspondante. Si elle n'est pas trouvée, l'écran est positionné en fin de fichier.
La touche "III" est reliée à la touche "DEPLACE".
La touche "III" permet de marquer le caractère à l'endoit du curseur. Un second appui efface cette marque. La touche "DEPLACE" déplace les zones ainsi marquées. Le texte, compris entre le premier caractère marqué et le caractère précédent le second caractère marqué, est transféré après le troisième caractère marqué. Si il n'y a pas 3 marques, rien ne se passe. Si il y a plus de 3 marques, le deplacement ne s'opère que sur les 3 premières occurences rencontrées.
La touche "impression" permet de sortir sur l'imprimante le contenu de l'écran (hard-copy).
La touche "RETOUR" permet de détruire le caractère sur lequel est positionné le curseur.
La touche "SUPRES" détruit toute la fin de la ligne y compris le caractère "RETURN".
La touche "EFFACER" détruit de l'endroit où est le curseur jusqu'à la fin du fichier, à moins d'avoir limité la destruction en marquant le caractère (voir explication sur la touche III".

II-3-3 LE RETOUR DE L'EDITEUR

Avant d'afficher le menu principal, les lignes sont analysées et reformatées. Les lignes blanches sont détruites. Les blancs après les labels, le code opération et l'opérande sont détruits. La partie label est tronquée à 6 caractères. La partie code opération est tronquée à 4 caractères. La partie opérande est laissée telle que mais avec suppression des blancs qui l'entourent. Ceci n'est toutefois valable que pour les lignes qui ne sont pas des commentaires.

II-4 LE MENU DE CONTROLE DES ENTREES/SORTIES

Il permet de communiquer avec le module mémoire de masse que vous disposez. La lecture et l'ecriture se font sur l'unité que vous avez désigné. Ce menu permet de charger à la suite du buffer programme un autre programme. Si l'on ne veut pas mélanger les programmes, il est nécessaire de détruire le programme précédent. Ce menu permet aussi la sauvegarde du programme source et du programme généré. Ce dernier peut s'opérer de deux manières.

- Sauvegarde normale: c'est la création d'un fichier répertorié contenant le fichier généré. Le fichier généré est la zone mémoire comprise entre l'adresse minimum et l'adresse maximum implicitement définie par votre programme.
- Sauvegarde sur un bloc particulier: dans ce cas la sauvegarde va s'opérer directement sur le bloc que vous allez désigner. Systématiquement sera sauvé à partir de votre adresse de début le nombre d'octets correspondant à la longueur du bloc. Le programme ne vérifie pas la logique du numéro de bloc que vous introduisez et tente la sauvegarde. Ce numéro doit être une valeur allant de 0 à FFFFH. Il ne vérifie pas non plus si votre généré est trop long ou trop court. (voir les particularités d'ADAM).

Le programme vous demandera le nom du fichier à charger ou à sauver. Vous pouvez utiliser les caractères du clavier ainsi que la touche "RETOUR" pour effacer le dernier caractère. "RETURN" marque la fin de votre introduction. "RETOUR T/T" provoque l'abandon de la sauvegarde ou du chargement.

II-5 LE MENU D'ASSEMBLAGE ET DE LISTES

Ce menu permet de déclencher l'assemblage et fournit des listes. Il est à remarquer qu'à l'exception de la simple liste, les autres listes exigent que le programme soit correct.

II-5-1 ASSEMBLAGE

Avec ce choix, seuls seront imprimés les messages erreurs.

II-5-2 ASSEMBLAGE AVEC LISTE INTEGRALE

Agit comme le précédent mais fournit la liste intégrale des instructions y compris celles générées par macro-instruction. La liste n'apparaît que si les deux premières étapes n'ont pas donné lieu à erreur. Son but est de permettre la validité des macro-instructions créées.

II-5-3 ASSEMBLAGE SANS LISTE DES MACRO-INSTRUCT.

Idem mais les macro-instructions n'apparaissent pas. La liste n'apparaît que si les deux premières étapes n'ont pas donné lieu à erreur.

II-5-4 LISTE SIMPLE

Cela provoque l'impression du programme dans son intégralité.

II-5-5 DUMP HEXADECIMAL

Fournit le code hexadecimal du généré. La liste n'apparaît que si le programme n'a pas donné lieu à erreur.

II-5-6 TABLE DES SYMBOLES

Fournit la table des symboles utilisés dans le programme. La liste n'apparaît que si le programme n'a pas donné lieu à erreur.

II-5-7 IMPRIMANTE

La couleur de fond du menu change avec cette sélection. La manière dont sortent les listes dépend de vos choix dans le menu environnement. En principe si vous avez choisi des listes continues, il n'est pas possible d'interrompre le travail. Dans les autres cas, il vous faudra taper une touche pour que le travail se poursuive. A ce niveau, en appuyant sur "RETOUR T/T" vous pouvez interrompre le travail.

III LES CONVENTIONS

III-1 LE FORMAT DES INSTRUCTIONS

Les lignes ont une longueur maximum de 48 caractères. on distingue deux types de formats.
- Les commentaires: Ils commencent avec "*"en colonne 1 et peuvent être suivis de 47 caractères. Ils ne sont pas analysés et sont destinés à documenter le programme.
- Les instructions réelles: Elles sont composées de 3 champs physiques.

1) le champ étiquette: formé de 6 caractères. une étiquette est un nom permettant de référencer la zone mémoire où sera assemblée l'instruction. Le premier caractère doit une lettre ascii >=41H. Les 5 suivants sont des codes ascii >=30H. Cas particulier: les étiquettes servant à référencer les macro-instructions ne doivent avoir que 4 caractères maximun, car intervenant dans le champ code opération.
2) le champ zone code opération: formé de 4 caractères. Il est destiné à recevoir les codes opérations Z80, les pseudo-instructions et les macro-instructions.
3) Le dernier champ à deux fonctions: On doit tout d'abord trouver l'opérande relatif au mnemonique puis, séparé par au moins un blanc, une zone commentaire facultative et non prise en compte par l'analyseur.

III-2 LES DIFFERENTS OPERANDES

On distinge 4 types:
1) Les opérandes relatifs au langage Z80. Ils servent à préciser un mnemonique (ex: LD A,H).
2) les opérandes sous forme d'un octet (oct): Ils interviennent dans certaines instructions Z80 (LD A,3) et dans la pseudo instruction dO.
3) Les opérandes sous forme de mots de 2 octets (mot/adresse). Ils interviennent dans certaines instruction Z80 (LD DE,3)et dans plusieurs pseudo-instructions.
4) Les opérandes liés à l'appel d'une macro-instruction. Il s'agit de chaine de caractère ne comportant ni blanc ni virgule.

Ils sont séparés par des virgules et leur définition s'achève par un blanc ou un "RETURN". Il en est de même pour l'opérande caractère des pseudo-instructions MOVE et CHG.

III-3 LA GESTION DES OPERANDES

Les opérandes "octet": ils obeissent aux mêmes régles que les opérandes "mot" mais n'est gardé que l'octet de poids faible du résultat.
Les opérandes "mot": Ce sont des expressions arithmétiques sans parenthèses et ne contenant que des + ou des -. Peuvent être utilisés:
- Des valeurs décimales: écrites telles quo.
- Des valeurs hexa.: commencent par "£".
- Des valeurs binaires: commencent par "!".
- Le caractère F: la valeur associée est la valeur de l'adresse où s'implante l'instruction
- Des caractères: ils sont entourés de quotes.
- Des "mots": leur valeur est celle affectée au "mot" et a pu être déterminée de 2 manières.

1) Si ce sont des labels d'instructions, leur valeur est celle de l'adresse où s'implantera l'instruction.
2) Si ce sont des labels de la pseudo-instruction "EQU", leur valeur sera celle de l'opérande de cette pseudo-instruction.

- Les variables %0 et %1: elles sont remplacées par la dernière valeur donnée. remarque: leur initialisation est aléatoire.
- Les opérandes des macro-instructions: le système gère le passage des paramètres en vérifiant que leur nombre est correct. Ce n'est qu'à l'assemblage qu'il pourra constater une incohérence au niveau du généré . L'erreur sera à ce niveau et non à celui de l'appel.
- Détermination de la valeur des opérandes "mots": remarque:pour les pseudo-instructions il est nécessaire de pouvoir déterminer leur valeur dès le premier passage de l'analyseur. Les "mots" doivent être définis avant leur utilisation. Echappent à cette règle DATA et DW. Le système travaille sur les "mots" en opérant des décalages. Il ne génère jamais d'overflow. Ainsi F2345H est interprété comme 2345H, "XYZ" comme "YZ". L'absence d'opérande est interprété comme un 0 .
  LD D, ---> LD D,0 : LD A,"" ---> LD A,0

III-4 LA STRUCTURE DE LA DEMANDE

La demande doit être formée de deux parties.
1) la déclaration des macro-instructions.
2) le programme qui est séparé des macro-instructions par la pseudo instruction PGM et qui se termine par END.

IV LE LANGAGE

Le langage est composé de 3 types d'instructions
1) Les pseudo-instructions.
2) Le langage assembleur Z80.
3) L'appel des macro-instructions définis par le programmeur. Les instructions assembleur génèrent directement du langage machine tandis que les pseudo-instructions servent à controler le déroulement de l'assemblage.

IV-1 LES PSEUDO-INSTRUCTIONS

- ASC : syntaxe LABEL ASC "TEXTE"
Génére les codes ascii du texte mis entre les " "
- CHG : syntaxe CHG CARACTERE,CHAINE CARACTERE
Permet de modifier la variable &6 par traduction en remplaçant le premier caractère par la chaine. Si le résultat dépasse 6 caractères, la chaine est tronquée. EX: si &6 = ABCABC
CHG A,E ---> &6=EBCEBC : CHG A,C ---> &6=CBCCBC
CHG F,A ---> &6=ABCABC : CHG A,EF ---> &6=EFBCEF
- DATA : syntaxe LABEL DATA OCT1,OCT2,...
Génére la représentation sur un octet de chaque nombre indiqué.
- DO : syntaxe DO N
Assemble N fois le texte situé après la ligne DO et avant la pseudo-instruction ENDO. Il est possible d'imbriquer des boucles bien que l'analyseur ne provoque pas de rejet. Chaque utilisation de DO provoque l'initialisation d'une variable spécifique qui se décrémente à la rencontre d'un ENDO et provoque l'analyse après le dernier DO rencontré.

```
EX: DO 8                        INC HL
    INC HL                      DEC D
    DO 3          GENERE        DEC D
    DEC D                       DEC D
    ENDO
```

Le premier DO n'est pas prise en considération.
- DS : syntaxe DS OCT1,OCT2
Permet de générer OCT1 octets initialisés à la valeur OCT2. OCT1 doit être déterminé dés sa rencontre.
- DW : syntaxe LABEL DW MOT
Génére la représentation du nombre sur 2 octets avec les conventions Z80. Octet de poids fort après.

- EGU : syntaxe NOM EGU MOT
Permet d'affecter une valeur à un mot. MOT doit pouvoir être calculé dès sa première rencontre. Cette pseudo-instruction permet de gérer des variables propres au macro-instruction. La différence avec EQU est que le programmeur peut modifier un nom déja affecté. Au moment de l'affectation la nouvelle valeur remplacera la précédente. Cette possibilité bien gérée est puissante mais risque de provoquer des erreurs car on peut ainsi modifier les labels. Aussi est-il conseillé, par exemple, de déclarer ces variables avec un nom commençant par un Z qui leur sera réservé.
- END : syntaxe END
Marque la fin de l'analyse d'un programmeur. N'est pas forcément la dernière instruction.
- ENDI : syntaxe ENDI
Marque la fin du programme analyse après un IF.
-ENDO : syntaxe ENDO
Marque la fin d'une boucle (voir DO).
Le programme vérifie d'abord si la variable associée au DO est nulle. Si oui, l'assemblage continue. Si non, la variable est décrémentée de 1 et l'analyse reprend à la position du dernier DO rencontré si la variable associée n'est toujours pas nulle. Il en résulte que DO 0 est executé une fois. Le test ayant lieu en fin de boucle.
-EQU : syntaxe NOM EQU MOT
Permet d'affecter une valeur à un NOM. MOT doit pouvoir être calculé dès sa première rencontre. Cette pseudo-instruction permet de gérer:
1) Des constantes. EX : OPEN EQU FCC0
2) Des références externes.

```
EX:  LIGNE DS 30
     DEBUT EQU LIGNE
     MIL EQU LIGNE+15
     FIN EQU LIGNE+27
```

-IF : syntaxe IF= MOT
Teste si la valeur %1 est égale à MOT. SI oui le texte est assemblé. Si non le texte qui suit jusqu'à la rencontre de ENDIF est sauté. MOT doit pouvoir être calculé dès sa première rencontre. Cette instruction permet un assemblage conditionnel. Cela permet (par exemple) d'optimiser certaines macro-instructions

```
EX:  macro ajoutant à HL la valeur du paramètre.
EXP MACC 1
    SET1 &1          EXP 3 génére  INC HL
    IF< 9                          INC HL
    DO %1                          INC HL
    INC HL
    ENDO             EXP 12 génére PUSH DE
    OUTM                           LD DE,12
    ENDI                           DAD HL,DE
    PUSH DE                        POP DE
    LD DE,&1
    ADD HL,DE
    POP DE
    OUTM
```

-IF<> : syntaxe IF<> MOT
Idem à IF sauf que l'on teste la différence.
-IF>= : syntaxe IF>= MOT
Idem à IF sauf que l'on teste la supériorité ou l'égalité de %1. Remarque: -3 est > 4. Les MOTS sont considérés comme des termes positifs de 16 bits.
-MACC : syntaxe LABEL MACC VAL
VAL doit être <=5 et >=0
voir paragraphe consacré aux macro-instrutions.
-MOVE : syntaxe MOVE VAL1,VAL2,CHAINE CARACTERES
Permet d'initialiser la variable &6. On doit avoir : 6>= VAL2 >= 1 : 20>= VAL1 >=0
&6 prend la valeur de la chaîne de caractères, de longueur VAL2, prise à partir du VAL1 ième caractère. Le premier caractère est le caractère 0 ième. Si la chaine en opérande à une longueur inférieure à VAL1+VAL2, des blancs sont générés.

```
EX:  MOVE 2,2,ABCDEF      &6= CD
     MOVE 4,1,ABCDEF      &6= E
     MOVE 3,2,ABCDEF      &6= DE
```

-OUTM : syntaxe OUTM
Marque la fin d'une macro-instruction. Cela provoque un retour à la ligne de l'appel avec, si cet appel était dans une macro-instruction, l'initialisation des paramètres.
-PGM : syntaxe PGM
Marque le début de l'assemblage, il doit suivre la déclaration des macro-instructions. Ce qui précede est considéré comme faisant partie des déclarations de macro-instruction. ADRS marque l'adresse de début de l'assemblage. Il faut que cette variable soit indiquée sinon le système considérera que l'assemblage demarre en 0.
-SET0 : syntaxe SET0 MOT
Permet d'initialiser la variable %0. MOT doit pouvoir être calculé dès sa première rencontre.

-SET1 : syntaxe SET1 MOT
Permet d'initialiser la variable %1. MOT doit pouvoir être calculé dès sa première rencontre.

IV-2 LE LANGAGE ASSEMBLEUR Z80

Pour une bonne utilisation du langage assembleur il convient de se reporter à des ouvrages donnant des exemples de programmation. Les instructions sont très simples mais font peu de choses en elles-mêmes. C'est la structure d'ensemble du programme qui donnera des résultats. Cela nécessite un mode de résolution des problèmes d'avantage analytique, mais il existe un certain nombre de structures de programme que l'on acquiert progressivement et qui rend la programmation aussi aisée que celle du basic. De cette effort de compréhension vous aurez la satisfaction de voir à quelle vitesse iront vos programmes et de mieux connaître le fonctionnement des processeurs. Nous nous contenterons ici de rappeler les mnémoniques utilisés pour le Z80.

NOTATION

NUM désigne 00H,08H,10H,18H,20H,28H,30H,38H.
NBIT désigne 0,1,2,3,4,5,6,7.
I désigne 0,1,2.
OCT désigne un octet.
ADR désigne une adresse.
REG désigne A,B,C,D,E,H,L.
G désigne A,B,C,D,E,H,L,(HL),(IX+DEP),(IY+DEP).
SS désigne BC,DE,HL,SP.
QQ désigne BC,DE,HL,AF.
PP désigne BC,DE,IX,SP.
RR désigne BC,DE,IY,SP.
GG désigne BC,DE,HL,SP,IX,IY.
CC désigne Z,NZ,C,NC,P,M,PO,PE.
(IND) désigne le contenu de IND.
Ranger X dans la pile système (stack),signifie que X est mis dans la pile et que le pointeur de pile est décrémenté de 2. La pile fonctionne dans le sens décroissant de la mémoire.
Recuperer X dans la pile système signifie que X est sortie de la pile et que le pointeur de pile est incrémenté de 2.

L'appel d'un programme signifie que l'adresse de l'incrustation suivant l'appel est rangée dans la pile système et qu'il y a branchement à l'adresse du sous programme. Le retour du sous programme s'effectue en dépilant la pile pour récupérer l'adresse de retour.
Les drapeaux. Il se positionnent selon le résultat de certaines opérations.
Z est l'indicateur de mise à 0 après l'opération.
C est l'indicateur de dépassement de capacité, après l'opération.
P est l'indicateur de parité après l'opération ou d'overflow.
S est l'indicateur de signe après l'opération (0 si positif).
H est l'indicateur de dépassement de capacité après l'opération au niveau du demi octet.

CODE		OPERATION REALISEE
ADC	HL,SS	HL=HL+SS+CARRY
ADC	A,G	A=A+CARRY
ADC	A,OCT	A=OCT+CARRY
ADD	IY,RR	IY=IY+RR
ADD	IX,PP	IX=IX+PP
ADD	HL,SS	HL=HL+SS
ADD	A,G	A=A+G
ADD	A,OCT	A=A+OCT
AND	G	A=A "et logique" G
AND	N	A=A "et logique" OCT
BIT	NBIT,G	met dans Z le contenu du bit NBIT de G
CALL	ADR	appel du programme situé en ADR
CALL	CC,ADR	appel du programme situé en ADR si la condition CC est réalisée
CCF		complemente le carry flag
CP	G	comparaison de A et de G
CP	OCT	comparaison de A et de OCT
CPD		compare A et (HL) décrémente HL et BC
CPDR		compare A et (HL) décrémente HL et BC.Répete sauf si BC=0 ou A=(HL)
CPI		compare A et (HL) incrémente HL et décrémente BC
CPIR		compare A et (HL) incrémente HL et décrémente BC. Repéte sauf si BC=0 ou A=(HL)
CPL		complémente A

DAA		ajustement décimal de l'accumulateur
DEC	GG	GG=GG-1
DEC	G	G=G-1
DI		interdit les interruptions
DJNZ	ADR	décrément B et débranchement en ADR si B différent de 0. L'adresse est relative.
EI		permet les interruptions
EX	AF,AF'	échange de AF et de AF'
EX	DE,HL	échange de DE et de HL
EX	(SP),IY	échange de (SP) et de IY
EX	(SP),IX	échange de (SP) et de IX
EX	(SP),HL	échange de (SP) et de HL
EXX		échange de BC,DC,HL avec BC',DE',HL'
HALT		arrêt du processeur jusqu'à RESET ou une intèrruption
IM	T	positionnement des intèrruptions
IN	REG,(C)	met dans REG le contenu du port (C)
IN	A,(N)	met dans A le contenu du port (N)
INC	GG	GG=GG+1
INC	G	G=G+1
IND		charge (HL) avec le contenu du port (C). Décrémente HL et B
INDR		charge (HL) avec le contenu du port (C). Décrémente HL et B répete sauf si B=0
INI		charge (HL) avec le contenu du port (C). Incrémente HL et décrémente B
INIR		charge (HL) avec le contenu du port (C). Incrémente HL et décrémente B. Répéte sauf si B=0
JP	ADR	débranchement en ADR
JP	(IY)	débranchement en (IY)
JP	(IX)	débranchement en (IX)
JP	HL	débranchement en (HL)
JP	CC,ADR	débranchement en ADR si la condition CC est réalisée
JR	ADR	débranchement en ADR saut relatif
JR	C,N	débranchement en ADR si carry=1 saut relatif
JR	NC,N	débranchement en ADR si carry=0 saut relatif
JR	Z,N	débranchement en ADR si Z=1 saut relatif

```
JR      NZ,N        débranchement en ADR si Z=0
                    saut relatif
LD      A,(DE)      A=(DE)
LD      (DE),A      (DE)=A
LD      A,(BC)      A=(BC)
LD      (BC),A      (BC)=A
LD      A,I         A=I
LD      I,A         I=A
LD      A,R         A=R
LD      R,A         R=A
LD      A,(ADR)     A=(ADR)
LD      (ADR),A     (ADR)=A
LD      G,REG       G=REG
LD      REG,G       REG=G
LD      REG,OCT     REG=OCT
LD      GG,(ADR)    GG=(ADR)
LD      GG,ADR      GG=ADR
LD      (ADR),GG    (ADR)=GG
LD      SP,IY       SP=IY
LD      SP,IX       SP=IX
LD      SP,HL       SP=HL
LDD                 met (HL) dans (DE)
                    décrémente HL,DE,BC
LDDR                met (HL) dans (DE)
                    décrémente HL,DE,BC
                    répète jusqu'à BC=0
LDI                 met (HL)dans (DE).Incrémente HL,DE
                    décrémente BC. Répète jusqu'à BC=0
NEG                 A=-A arithmétique(complément à 2)
NOP                 non opération
OR      G           A=A "ou logique" G
OR      N           A=A "ou logique" OCT
OTDR                met (HL) sur le port (C)
                    décrémente HL,DE,BC
                    répète jusqu'à BC=0
OTIR                met (HL) dans le port (C)
                    incrémente HL, décrémente BC
                    répète jusqu'à BC=0
OUT     (C),REG     met sur le port (C) le contenu de
                    REG
OUT     (N),A       met sur le port (N) le contenu de
                    A
OUTI                met (HL) sur le port (C)
                    incrémente HL, décrémente BC
OUTD                met (HL) sur le port (C)
                    décrémente HL,DE,BC
POP     GG          récupère GG dans la pile système
```



```
PUSH    GG          met GG dans la pile système
RES     NBIT,G      met à 0 le NBIT ième bit de G
RET                 retour de sous programme
RET     CC          retour de sous programme si la
                    condition CC est réalisée
RETI                retour d'interruption masquable
RETN                retour d'interruption non masquable
RL      G           rotation gauche de G d'un bit avec
                    le carry, G0=C : C=G7
RLA                 rotation gauche de l'accumulateur
                    d'un bit avec le carry
RLC     G           rotation gauche de G d'un bit
                    G0=G7; C=G7
RLCA                rotation gauche de l'accumulateur
                    d'un bit
RLD                 rotation gauche d'un demi-octet
                    entre le demi-octet bas de
                    l'accumulateur et les deux demi-
                    octets de HL
RR      G           rotation droite de G d'un bit avec
                    le carry: G7=C : C=G0
RRA                 rotation droite de l'accumulateur
                    d'un bit avec le carry
RRC     G           rotation droite de G d'un bit
                    G7=G0 : C=G0
RRCA                rotation droite de l'accumulateur
                    d'un bit
RRD                 rotation droite d'un demi-octet
                    entre le demi-octet bas de
                    l'accumulateur et les deux demi-
                    octets de HL
RST     NUM         appel de NUM
SBC     HL,SS       HL=HL-SS-CARRY
SBC     A,G         A=A-G-CARRY
SBC     A,OCT       A=A-OCT-CARRY
SCF                 mise à 1 du carry
SET     NBIT,G      met à 1 le NBIT ième bit de G
SLA     G           rotation logique gauche de G d'un
                    bit:  G0=0 ; C=G7
SRA     G           rotation gauche de G d'un bit
                    G7=G7  ;  C=G0
SRL     G           rotation logique droite de G d'un
                    bit:  G7=0  ;  C=G0
SUB     A,G         A=A-G
SUB     A,OCT       A=A-OCT
XOR     G           A=A "ou exclusif logique" G
XOR     N           A=A "ou exclusif logique" G
```

V GESTION DES MACRO-INSTRUCTIONS

V-1 DEFINITION D'UNE MACRO-INSTRUCTION

Il s'agit d'un modèle de programme. Une macro-instruction est composée d'instructions en assembleur qui sont destinées à s'implanter chaque fois que dans son programme principal, le programmeur fait appel à elles. Ce ne sont donc pas des sous programmes. A chaque appel les instructions seront générées. Leur but est d'éviter des répétitives en fournissant au programmeur des pseudo-instructions facilitant l'écriture des programmes. La puissance de ces pseudo-instructions dépendra de vos définitions. Le programmeur peut se contenter d'utiliser le macro-assembleur comme un simple assembleur ou alors créer sont propre langage. Cette possibilité est d'autant plus grande que l'outil permet de contrôler le déroulement de l'assemblage. Sans qui y ait réellement de frontière, on peut distinguer plusieurs niveaux d'utilisations.

1) un niveau de gestion de répétitive.
2) un niveau de passage des paramètres à des sous programme.
3) un niveau d'autogénération évoluée.

Les exemples qui suivent appartiennent aux 2 premiers niveaux, les programmeurs abordant le troisième niveau n'ayant plus besoin de conseils, ou devant se documenter sur les techniques de compilations.

V-2 L'OUTIL

Le macro-assembleur que vous avez, dispose des possibilités suivantes.

V-2-1 GESTION DES PARAMETRES

A la déclaration de la macro-instruction on indique le nombre de paramètres qu'elles possedent. Ce nombre doit être compris entre 0 et 5. Dans le corps de la macro-instruction, ces paramètres seront désignés par &1,&2,&3,&4,&5, et ne pourront définir qu'un label ou un code opération ou un opérande. Un paramètre ne peut définir 2 champs simultanément. La régle est que le paramètre est remplacé dans le champ où il apparait jusqu'à ce qu'il remplisse le champ correspondant.

Chaque appel ne peut définir pour un paramètre qu'une chaine de 6 caratères. EX:
si l'appel de la macro-instruction définit &1=ABCDEF et si la macro-instruction contient
X&1Y , X&1Y , X&1Y : le généré sera :
XABCDE , XABC , XABCDEFY.
Le label a été tronqué à 6 caractères, le code opération à 4 et la zone opérande conservée en entier, la coupure ne se produisant qu'au 48ème caractère. Il n'y a aucune restriction à l'usage de ces remplacements. La seule règle est que le généré soit compatible avec le reste du programme et ait une signification. En particulier on peut générer l'appel à une macro-instruction, en revanche il n'est pas possible d'en définir une.

V-2-2 GESTION RECURSIVE DES MACRO-INSTRUCTIONS

Une macro-instruction a la possibilité d'en appeler une autre, y compris elle même. Les paramètres de la macro-instruction appelant sont gérés dynamiquement. Au retour de l'appel les paramètres reprendront leur valeur d'avant l'appel. EX:

```
MC1   MACC 1
      INC &1
      OUTM
MC2   MACC 3          L'appel MC2 H,D,SP générera
      MC1 &1                  INC H
      MC1 &2                  INC D
      MC1 &3                  INC SP
      MC1 &1                  INC H
      OUTM
```

&1 n'a pas été modifié par les appels précédents. Cette gestion dynamique n'est possible que grâce à une pile. Cette pile ayant une taille limitée, le programmeur pourra se trouver confronté à un overflow et devra donc trouver des astuces. Chaque appel utilise 5 octets + 6 octets par paramètre, la pile représente environ 8K octets.
EX: macro servant à pusher 4 registres (ou moins) quelconques.

```
PUS   MACC 4
      SET1 "&1"  (%1 prend la valeur ascii du
                  contenu du premier paramètre)
      IF<>       (test si DE 0)
      PUSH &1
      PUS &2,&3,&4
      ENDI
```

```
      OUTM
```

L'appel PUS BC,HL,HL,IX générera PUSH BC/PUSH HL/PUSH HL/PUSH IX :mais consommera (5+6*4)*4=116 octets dans la pile. Bien que moins "joli" il est préférable d'écrire :

```
PUS   MACC 4
      PUS1 &1
      PUS1 &2
      PUS1 &3
      PUS1 &4
      OUTM
PUS1  MACC 1
      SET1 "&1"
      IF<>
      PUSH &1
      ENDI
      OUTM
```

qui ne consomme 5+24=29 octets lors d'un appel. Pour compléter cet exemple, il est à remarquer que les macro-instructions suivantes sont encore plus puissantes.

```
PUSA  MACC 4
      PUS1 USH,&1
      PUS1 USH,&2
      PUS1 USH,&3
      PUS1 USH,&4
      OUTM
POPA  MACC 4
      PUS1 OP,&1
      PUS1 OP,&2
      PUS1 OP,&3          PUSA AF,HL,, génére
      PUS1 OP,&4          PUSH AF/PUSH HL.
      OUTM                POPA DE,DE,HL, génére
PUS1  MACC 2              POP DE/POP DE/POP HL
      SET1 "&2"
      IF
      P&1 &2
      ENDI
      OUTM
```

V-2-3 LE PSEUDO PARAMETRE &D

Il désigne en fait un des paramétres de l'appel. L'indice du paramètre désigné est contenu dans %0. Il convient de bien se souvenir que %0 est une variable statique. EX: si &1=A,&2=B,&3=C,&4=D

```
SETO  2
INC   &D --> INC &2 --> INC B
```



```
SETO  4
INC   &D --> INC &4 --> INC D
```

L'usage du pseudo paremètre est evidemment lié à celui de boucles.

```
EXPL  MACC 4     est equivalent à EXPL MACC 4
      SETO 1                          INC &1
      DO 4                            INC &2
      INC &D                          INC &3
      SETO %0+1                       INC &4
      ENDO                            OUTM
      OUTM
```

V-2-4 LA VARIABLE CARACTERE AUXILAIRE &6

Comme les paramètres, cette variable est une chaîne de 6 caractères maximum. Mais contrairement à ces derniers il s'agit d'une variable statique. 2 pseudo-instructions permettent d'agir sur elle: MOVE et CHG EX: &D et &6 permettent d'écrire différenent la macro-instruction PUSH, POP en la complétant.

```
OPRG  MACC 5
      MOVE 0,1,&1    OPRG I,B,B,H,H génére
      CHG I,INC      INC B/INC B/INC H/INC H
      CHG D,DEC
      CHG O,POP      OPRG O,BC,BC,AF,HL génére
      CHG U,PUSH     POP BC/POP BC/POP AF/POP HL
      SETO 2
      DO 4           OPRG D,H,H,D,H génére
      SET1 "&D"      DEC H/DEC H/DEC D/DEC H
      IF<>
      &6 &D          OPRG U,AF,HL,, génére
      SETO  &D+1     PUSH AF/PUSH HL
      ENDO
      ENDI
      OUTM
```

V-3 EXEMPLES D'UTILISATIONS

V-3-1 UTILISATION DES VARIABLES STATIQUES %0, %1

Ces variables permettent surtout d'utiliser efficacement les boucles DO et les tests. Elles peuvent être aussi utilisées comme constantes ou pour gérer des labels, mais la pseudo-instruction CGU pernet d'en faire autant tout en offrant un plus grand nombre de possibilités. Elles permettent aussi de récupérer les paramètres d'une macro-instruction pour les tester. Dans ce domaine il faut se méfier de certains piéges.

```
EXEM  MACC  1
      SETD  "&1"
      SET1  &1
```

Si le paramètre passe est A: %D prendra la valeur correspondant au code ascii de "A":41
%1 prendra la valeur correspondant à l'adresse du label A. Si le paramètre passé est BCDAA: %D prendra la valeur correspondant au code ascii "AA":4141. %1 prendra la valeur correspondant à l'adresse du label BCDAA.

V-3-2 UTILISATION POUR GERER DES REPETITIVES

La technique consiste à créer une macro-instruction dont le nom est destiné à remplacer une instruction. Cela peut aller du simple remplacement d'une instruction dont on veut rendre plus claire l'appellation à la création d'instructions spécifiques. EX:

```
JPHL MACC        AHL MACC          SI  MACC  2
     JP HL           LD A,(HL)         CP &1
     OUTM            INC HL            JP Z,&2
J=   MACC 1          OUTM              OUTM
     JP Z,&1     XC MACC 2
     OUTM            LD HL,(&1)
                     LD (&2),HL
J>=  MACC 1          OUTM
     JP P,&1
     OUTM
```

V-3-3 USAGE DE GESTION DES SOUS PROGRAMMES

La technique consiste à passer les paramètres du sous programme comme paramètres de la macro-instruction qui est chargée de faire réellement le passage. Cela évite de consulter la manière dont il faut initialiser les paramètres. Ceci est valable pour les programmes de l'EDS mais aussi pour ses propres sous programmes auxquels on peut associer une macro-instruction. EX:

```
WRAM  MACC 3   appel de l'écriture dans le VDP
      LD HL,&1
      LD BC,&2
      LD DE,&3
      CALL
       FD1A
      OUTM
CONS  MACC 1    HOME MACC       CRLF MACC
```



```
      LD A,&1         CONS 80         CONS C
      CALL            OUTM            OUTM
       FC39      CUR MAC 2
       OUTM          LD D,&1
                     LD E,&2
                     CONS 1C
                     OUTM
```

VI LES MESSAGES ERREURS

Les messages erreurs sont à relier à la manière dont le programme d'analyse travaille. Cette analyse se fait en trois temps. Chaque étape n'est enclanché que si la précédente n'a pas donné lieu à erreur.
Première étape: l'analyseur recherche les macro-instruction et le début du programme. Cela peut conduire à plusieurs messages erreurs.
NO PGM: il manque la pseudo-instruction PGM
INVALID NUMBER: vous avez déclaré une macro-instruction avec plus de 5 paramètres.
MULTIPLE MACRO: vous avez déclaré une macro-instruction avec un nom déjà attribué. Ne pas oublier que le nom est formé de 4 caractères, l'analyseur ne détecte pas d'erreurs si l'on en met 5 ou 6. Il se contente de tronquer.
Deuxième étape: l'analyseur connaît désormais les macro-instructions. Il va chercher à résoudre toutes les références de labels. Pour cela il lui faut déterminer la longueur du généré de chaque instruction. A la fin il regarde si il a suffisemment de place disponible pour implanter le généré.
INVALID NUMBER PARAMETERS: vous ne passez pas le nombre de paramètres attendus. Ne pas oublier que la virgule est un séparateur.
COMMANDE INVALID: la zone code opération ne correspond ni à un nom de macro-instruction, ni à un code opération.
INDEFINED REFERENCE: une référence obligatoire à ce niveau est inconnue.
INVALID OPERAND: la structure de la zone opérande n'est pas bonne. Cela peut être un paramètre de plus de 6 caractères ou une mauvaise utilisation de MOVE ou CHG.
OUT OF MEMORY: si cela se produit avant l'instruction END c'est qu'il n'y a pas assez de place pour mettre les labels dans la table. Avec l'instruction END cela correspond au fait qu'il n'y aura pas assez de place pour le généré.

INVALID NUMBER: vous utilisez &O alors que %O a une valeur supérieure à 6.
STRING INVALID: une chaîne de caractères est utilisée sans être délimitée.
MULTIPLE LABEL: label déjà utilisé.
STACK FULL: vos appels récursifs conduisent à un débordement de la pile.
Troisième étape: le macro-assembleur est apte à résoudre toutes les références et peut implanter le programme. Il va reprendre l'étape 2 en précisant exactement le généré. Cela peut conduire à la détection de nouvelles erreurs.
INVALID REFERENCE: il s'agit d'une référence d'une instruction Z80 non obligatoire au niveau précédent.
INVALIDE OPERAND: la zone opérande n'est pas celle attendue.

VII LES PARTICULARITES DU SYSTEME ADAM

Sans que l'on puisse ici aborder toutes les possibilités d'ADAM, il convient de connaître quelques points pour pouvoir utiliser plus facilement le langage machine.
1) le partage mémoire: ADAM possède un système d'exploitation (EOS) lui permettant de gérer les différents périphériques. Ce système occupe la mémoire depuis l'adresse D390H. On y accède par une table de saut située de FC30 à FD50. Jusqu'à l'adresse 100 se situent les interruptions. Les adresses sont à la disposition du programmeur. Elles sont initialisées pour le retour. A noter que 66H (interruption non masquable) est déclenché par le VDP tous le 1/50ème de secondes.
2) quelques points d'entrées:
Le clavier: FD20. Le système renvoie dans la valeur ascii de la touche appuyée. Il y a attente de cette appui. Normalement le drapeau Z est mis à 1.la séquence d'appel est donc/

```
LAB CALL FD20
    JR Z,LAB
```

3) Le VDP (video display processor).
Il s'agit du TMS 9928A de TEXAS INSTRUMENTS. Il est souhaitable de connaître son fonctionnement avant de l'utiliser. Sa particularité est de pouvoir décrire la mémoire d'écran (16K indépendant dans ADAM: VRAM), de différentes

manières et selon plusieurs modes graphiques.
Il existe différentes tables dans la VRAM et dont les registres donnent la position. L'EOS permet d'y accéder par:
FD20: écriture d'un registre, B=numéro registre, C=valeur à écrire.
FD1A: écriture dans la VRAM; HL adresse du buffer contenant les octets; DE adresse en VRAM (16K disponibles); BC nombre d'octets à transférer.
FD26: écriture dans la VRAM par remplissage; A caractère à envoyer; HL adresse en VRAM; BC nombre d'octets à transférer.
L EOS fournit deux autres adresses facilitant l'écriture de l'initialisation.
FD26: positionnement d'une table dans le VDP; A numéro de la table; 0 table des attributs de sprites; 1 table des sprites; 2 table d'écran des formes; 3 table du dessin des formes; 4 tables des couleurs des formes; HL position de table dans la VRAM.
FD17: permet d'envoyer le dessin des caractères d'ADAM dans le VDP; HL numéro du premier caractère à prendre; BC nombre de caractères à envoyer; DE adresse de la VRAM. Chaque caractère occupe 8 octets. De plus l'EOS fournit un système automatique de gestion de fenêtre d'écran.
FC36: permet d'initialiser ce système; B nombre de caractères de la fenêtre (<32); C nombre de lignes de la fenêtre (<23); D numéro du caractère 1 de la fenêtre; E numéro de la première ligne; HL adresse en VRAM de l'écran.
FC39: envoie d'un caractère dans la fenêtre précédemment définie (contenu dans A). Caractères spéciaux comme le basic avec de plus 1C qui positionne le curseur en (D,E).
Séquence d'initialisation:

```
LD B,0          registres d'état
LD C,0          graphique mode 1
CALL FD20
LD B,1
LD C,E0
CALL FD20
LD B,7          couleur de fond
LD C,0
CALL FD20
LD A,0          table des attributs de sprites
LD HL,1F00
```

```
CALL FD29
LD A,1          table des sprites
LD HL,3800
CALL FD29
LD A,2          table d'écran
LD HL,1800
CALL FD29
LD A,3          table des générateurs
LD HL,0
CALL FD29
LD A,4          table des couleurs des générateurs
LD HL,2000
CALL FD29
LD HL,0         chargement total dessin caractères
LD BC,80
LD DE,0
CALL FD27
LD HL,0
LD BC,80        de 80 à FF
LD DE,400       même dessin que les précédents
CALL FD27
LD HL,2000      couleurs des caractères de 0 à 1F
LD A,F0
LD DE,10
CALL FD26
LD HL,2010      couleurs des caractères de 80 à FF
LD A,0F         inversion des précédents
LD DE,10
CALL FD26
LD BC,1F17      définition d'une fenêtre d'écran
LD DE,0         (ici écran total)
LD HL,1800
CALL FC36
EXEMPLE: écrivons "OK".
LD A,"O"
CALL FD39
LD A,"K"
CALL FD39
```

Accès à l'imprimante. Il se fait par FC66 qui envoie A. Il suffit d'envoyer le code ascii. Au retour de Z=1 tout est "OK" sinon il vaut mieux recommencer. Attention A a été détruit.
Séquence classique:

```
TEMP DS 1,0
PRT  LD (TEMP),A
BCL  LD A,(TEMP)
     CALL FC66
     JNZ BCL
```


PARTICULARITES DES FICHIERS:

Le basic permet avec la fonction BLOAD de charger des programmes codes. Mais il faut tenir compte du fait que le basic a besoin de certains renseignements pour pouvoir charger ces codes. Il en résulte que le fichier doit contenir avant le programme proprement dit:
2 octets spécifient l'adresse d'implantation du programme. Pour cela il suffit de mettre en début du source:

```
PGM £ADR-5    (ex:£3000-5)
DATA 1,0,2
DW £ADR
...
```

ADR est l'adresse d'implantation et d'exécution du programme.
Le bloc 0:
Quand un "RESET" se produit le système charge le contenu du bloc 0 à l'adresse C800H et déclenche l'exécution à partir de cette adresse. Vous pouvez donc profiter de cette possibilité pour créer des cassettes auto start en langage machine.

Conception : REGICIEL.
Réalisation : J.L. PRONIER / P. SANTONI.