



- MAXAM 464/664/6128 -

Note : Cette 'traduction' ne suit pas toujours exactement le texte original mais en contient cependant l'essentiel . Seuls les programmes exemples les plus représentatifs sont reproduits (Parfois légèrement adaptés) dans ces pages .

Section 1 : Qu'est-ce que MAXAM ? (V.O. Page 4) .

C'est un logiciel qui permet de générer un code binaire à partir d'un fichier source édité en ASCII . Pour parvenir à ses fins il dispose de :

1 Editeur permettant la création des fichiers source joint à un assembleur générant le code binaire à partir du fichier ASCII précédemment édité .

1 Moniteur / désassembleur destiné à fouiller dans la RAM ou les ROMS basic ou externes .

1 Option particulières à MAXAM permet aussi de créer du code source à partir de lignes BASIC .

De nombreuses options secondaires que vous découvrirez lors de la lecture de cette notice .

Section 2 : Pour commencer ! (V.O. Page 5).

MAXAM existe en 3 versions : Voici la procédure à suivre pour charger et lancer chaque version .

1 / MAXAM ROM : Référez vous à la notice de votre ROM BOARD pour l'insertion de la puce MAXAM dans celui-ci . Le numéro de ROM affecté à MAXAM doit être compris entre 1 et 6 . Lorsque MAXAM ROM est connecté , une zone de 255 octets située juste sous le HIMEM est réservée aux RSX de MAXAM , si vous avez besoin de récupérer cet emplacement mémoire , tapez !MAXOFF . Cette commande déconnecte MAXAM , effectue un RESET et remet le HIMEM au maximum .

Notes : a / Après !MAXOFF seul un reset par CTRL+SHIFT+ESC permet de retrouver MAXAM .

b / A l'initialisation , un bip sonore suivi du message 'Maxam checksum error' signifie sans doute que la puce est mal emboîtée ou que vos connecteurs apprécieraient une friction à l'alcool à brûler !

2 / MAXAM DISC : TRES IMPORTANT !! Toujours commencer par un reset par CTRL+SHIFT+ESC !! Ensuite , placer le disque MAXAM et faire RUN"MAXAM" ou RUN"DISC" . Le 1er menu vous propose 3 options :

1 : Charger le programme complet .

2 : Charger l'éditeur seul .

3 : Charger TOUT SAUF l'éditeur .

Les options 2 & 3 de chargement partiel servant évidemment à économiser la RAM disponible dans le CPC .

Le programme sera automatiquement relogé à l'adresse la plus haute possible .

Une fois chargé , le programme demande combien d'octets vous souhaitez réserver au code . Vous pouvez vous contenter de taper sur ENTER et laisser MAXAM se débrouiller ou de lui assigner un nombre d'octets précis qui abaissera la HIMEM d'autant . (Voir détails en section 5) .

3 / MAXAM CASSETTE : TRES IMPORTANT !! Toujours commencer par un reset par CTRL+SHIFT+ESC !! Ensuite , placer la cassette MAXAM et faire RUN"MAXAM" ou RUN" . Le programme vous demandera : 'Do you want to load the editor ?' . répondez 'N' si vous n'avez pas besoin de l'éditeur et 'O' dans tout autre cas . La suite des opérations se déroule comme pour la version disquette .

Section 3 : Usage général (V/O. Page 6) .

L'activation de MAXAM implante en RAM une série de commandes RSX , les voici :

Toutes versions .

!MAXAM,n : Cette commande permet d'accéder au menu principal (n prend la valeur 1 ou 2 selon le mode d'écran que l'on souhaite . Ce paramètre est facultatif)

!M : Comme !MAXAM .

!ASSEMBLE : Provoque la génération du code binaire à partir d'un listing implanté en BASIC et l'affichage du listing .

!ASSEM : Comme !ASSEMBLE mais sans l'affichage du listing .

!CAT : Catalogue du drive actif .

!MODE,n : Comme MODE en basic avec n=1 ou 2 .

!SPEED : Vitesse de chargement pour la cassette .

!CLEAR ou MCLEAR : Détruit le fichier source entré avec l'éditeur .

!FIND ou MFIND : Recherche une chaîne donnée dans un listing source rédigé sous BASIC .

Commandes disponibles SEULEMENT SUR VERSION ROM .

!HELP ou MHELP : Liste toutes les ROMS connectées et donne leur numéro de version .

!HELP,n ou MHELP,n : Liste les commandes RSX liées à la ROM dont le numéro est donné dans n . Ex : HELP,7 liste les RSX du drive .

!MAXOFF : Adieu MAXAM . Reset pour le retrouver .

!MSL : Place la RAM écran à l'adresse basse &4000 .

!MSH : Place la RAM écran à l'adresse haute &C000 .

!ROMOFF : Désactive la ROM sélectionnée .

Note : Certaines commande ont 2 syntaxes possibles : précédées ou non de 'M' ceci ne sert qu'à les différencier de commandes similaires que vous pourriez trouver dans d'autres ROMS externes .

Généralités :

La touche ESC fonctionne comme en BASIC . 1 Appui interrompt l'opération , un second appui sur ESC la stoppe définitivement alors que tout autre touche permet de reprendre le déroulement normal .

DEL : Fonctionne comme en BASIC .

Lorsque le curseur est visible à l'écran , c'est que MAXAM attend une commande ou une réponse . Une opération d'impression avec l'imprimante hors service provoque la réapparition du curseur et une attente . Dans ce cas , activer l'imprimante ou maintenir ESC enfoncé pour annuler l'opération .

Si vous utilisez MAXAM en mode 1 sur un écran monochrome : tapez INK 3,18 pour un affichage plus lisible .

Section 4 : Un exemple plus concret (V.O. Pages 7/8) .

Note au sujets des exemples : Seuls les listings essentiels sont traduits ici . Ces derniers sont parfois un peu modifiés par rapport à la V.O. Pour tout listing non reproduit ici figure une mention : Voir aussi V.O. Page xx .

Ecrit en BASIC ce programme donnerait :

```
10 FOR i=32 to 127: ?CHR$(i);:NEXT
```

Et affiche les caractères de 32 à 127 .

```
10 MEMORY HIMEM-11
20 start=HIMEM+1
30 GOSUB 80
40 PRINT:PRINT CHR$(24)" EXECUTION DU PROGRAMME ASSEMBLE "CHR$(24):PRINT
50 CALL start
60 END
70 '
80 MODE 2:PRINT:PRINT CHR$(24)" OPERATION D'ASSEMBLAGE "CHR$(24):PRINT
90 ;ASSEMBLE ;Assembler ce qui suit .
100 'LD A,32 ;Place le code ASCII 32 dans l'accumulateur .
110 '.LOOP ;Définit le label : LOOP .
120 'CALL &BB5A ;Routine système qui affiche le caractère donné dans A
.
130 'INC A ;Caractère suivant .
140 'CP 128 ;Plus grand que 128 ?
150 'JR C,LOOP ;Non continuer .
160 'RET ;Oui , fini .
170 'END ;Fin de l'assemblage .
180 RETURN ;Fin du GOSUB .
```

Explications : En ligne 10 on réserve 11 octets pour le code binaire , en ligne 20 la variable 'start' est initialisée à l'adresse de départ du binaire (En l'absence de directive ORG celle-ci est HIMEM+1) , puis la ligne 30 va assembler le code pour enfin l'exécuter en ligne 40. Voir aussi V.O. Page 8 .

Section 5 : L'assembleur en détail . (V.O. Pages 9/25) .
Usage à partir du basic . (V.O. Page 9) .

Le programme à assembler peut indifféremment être rédigé à partir du BASIC (Section 4) ou depuis l'éditeur (section 8) . Le programme à assembler se compose d'une suite d'instructions dont chacune a le format suivant : LABEL INSTRUCTION ;Commentaire.

Le LABEL est facultatif et n'est utilisé que pour identifier une section précise du programme . Il faut noter que majuscules et minuscules sont indifférenciées et que du point de vue de l'assembleur , DEBUT et DebuT sont exactement la même chose . Lors de la rédaction d'un code source sous BASIC , ce label doit être précédé d'un point afin que l'assembleur ne le confonde pas avec une instruction ! Ex :

```
10' .DEBUT ;C'est bon .
```

```
10' DEBUT ;C'est une erreur !
```

L'instruction peut être tapée en majuscules comme en minuscules . LD A,(HL) ou ld a,(hl) sont tout aussi valables . En mode BASIC vous pouvez utiliser plusieurs instructions sur une même ligne pourvu qu'elles soient séparées par un double point . Ex : 1000 'LD A,(HL):INC HL:CALL &BB5A:RET

Le commentaire est facultatif et dépend de vos besoins . Il doit toujours être précédé d'un point virgule et suivre l'instruction . Si l'on veut utiliser le point dans un commentaire , il faut commencer la ligne par 2 points virgule faute de quoi , ce qui suivrait le point serait pris pour un label ! Ex : 230 'INC A ;Incrémente l'accumulateur
240 'LD (HL),A ;;Stocke l'accu. dans la case pointée par HL

Il est préférable de ne pas utiliser le double point dans un commentaire car il pourrait être pris pour un séparateur entre 2 instructions !

Le mode direct peut aussi être employé pour assembler un bref programme , essayez : !ASSEMBLE: 'LD A,"A":CALL &BB5A:LD A,"B":JP &BB5A puis ENTER ... L'apostrophe (REM) qui précède chaque séquence assembleur est indispensable . Sans celui-ci , le BASIC voudra interpréter la ligne à sa manière et manifestera sa perplexité par un 'Syntax error in ...'

Correction des erreurs (V.O.Page 10) .

Les corrections se font exactement comme en basic . Toute faute dans la partie assembleur du listing provoque un 'Syntax error in ...' et passe en mode édition .

Déboguage et points d'arrêt .

Une instruction spéciale : 'BRK' permet de stopper le programme à chaque fois qu'il la rencontre . Le contenu de tous les registres est alors affiché à l'écran . A ce stade , si vous appuyez sur ESC le programme stoppe définitivement ou continue si vous appuyez sur tout autre touche . Par exemple , pour le programme donné en section 4 , ajouter : 111 'BRK et vous verrez l'effet de cette commande . Vous verrez aussi que pris dans une boucle cette instruction provoque un scrolling souvent peu désirable ! L'astuce qui suit vous évitera cet ennui :

```
10000 ' .BREAK ;Label BREAK
10010 'PUSH HL ;Préserver tous les registres .
10020 'PUSH DE
10030 'PUSH BC
10040 'PUSH AF
10050 'LD HL,&0101 ;Curseur en haut à gauche .
10060 'CALL &BB75 ;Routine système LOCATE .
10070 'POP AF ;Récupérer tous les registres .
10080 'POP BC
10090 'POP DE
10100 'POP HL
10110 'BRK ;Interrompre .
'RET ;Fini .
```

Il suffit de l'inclure à votre listing et de faire : 'CALL BREAK . Le contenu des registres sera toujours affiché à la position donnée dans HL .

Note : Cette instruction BRK utilise le RST &30 (Dit RST 6 dans la V.O.) qui est un point de Restart libre situé à l'adresse &30 en RAM CPC . Les spécialistes sauront , en cas de besoin , redétourner ce saut vers une de leurs propres routines pour qu'elle soit activée par BRK .

Messages d'erreur de l'assembleur (V.O. Page 11) .

On trouvera 3 types de messages :

1 / FATAL ERROR (Erreur fatale) : Ce type de message signifie que l'assembleur a perdu le scénario et ne peut suivre vos instructions . L'assemblage s'interrompt aussitôt après le bip sonore .

2 / WARNING (Attention !) : Ces messages ne bloquent pas l'assemblage et ne sont affichés que lors de la seconde passe . Ils se rapportent généralement à des labels absents , erreurs de syntaxe dans une expression , expression au delà de la capacité d'un registre (Ex : LD A,450 ou LD HL,120000) , Etc ... Il vaut mieux rectifier et réassembler avant de lancer le programme ...

3 / ERROR (Erreur) : Tout autre type d'erreur qu'il vaut mieux corriger avant de lancer le programme .

La signification des messages sera détaillée en fin de notice .

Location du code objet (V.O. Page 12) .

Le programme binaire doit bien sur être placé dans une zone mémoire où rien ne viendra le perturber . Pour le détail de la carte mémoire CPC , voyez la V.O. Page 12 .

En mode assemblage sous basic , c'est MEMORY qui permet de redéfinir la place réservée au binaire . Avec MAXAM disquette ou cassette , vous pouvez utiliser la zone mémoire comprise entre la fin du BASIC et le début de MAXAM. Ces 2 adresses varient selon la longueur du programme BASIC et la version (Complète ou partielle) de MAXAM logée en mémoire . Avec MAXAM ROM vous disposez de la zone comprise entre la fin du BASIC et l'adresse &A575 à moins qu'avec 'Symbol after' vous ayez un peu abaissé le HIMEM .

Si il faut employer 'SYMBOL AFTER' faites le avant de modifier MEMORY !

Un exemple : Si votre code binaire doit commencer en 10000 , faites : MEMORY 9999 et le basic ne pourra plus perturber le code binaire . La page 12 de la V.O. montre une autre méthode , a chacun son truc is it not ?...

La directive ORG (V.O. Page 13) .

Noter qu'une directive n'est pas une instruction ! L'instruction implante un code binaire en RAM , la directive donne un ordre précis à l'assembleur . Que l'on utilise l'assemblage sous BASIC ou l'éditeur , ces directives ont exactement le même rôle !

Revenons à ORG pour dire que c'est cette instruction qui dit à l'assembleur où commencer l'assemblage.ORG peut s'exprimer sous 2 formes:

1 / ORG <Expression> : Expression peut être une simple adresse , Ex : ORG 10000 commence l'assemblage en 10000 , le résultat d'un calcul Ex : ORG 5000*2 commence toujours l'assemblage en 10000 , voire contenir des labels définis par le programme , Ex : ORG ADRESSE+7 assemblera en 10007 .
ADRESSE EQU 10000

2 / ORG <Expression1 , Expression2> : Cette forme sera souvent utilisée avec les versions disquette ou cassette . En effet , on aura parfois besoin de fichiers binaires exécutables à une adresse occupée par MAXAM au cours de l'assemblage . Dans ce cas , <Expression1> contiendra l'adresse où DOIT S'EXECUTER LE CODE BINAIRE et <Expression2> l'adresse où l'assembleur STOCKE LE CODE BINAIRE .

Un exemple concret : La version finale de votre programme de 200 octets doit fonctionner depuis l'adresse 40000 . D'abord vous la mettez au point depuis une adresse sans problèmes (Sous MAXAM) . Ceci fait , vous ajouterez une directive ORG 40000,10000 . Le code résultant sera stocké à l'adresse 10000 MAIS TOUTES LES ADRESSES SERONT CALCULEES POUR UNE EXECUTION EN 40000 ! L'assemblage terminé , il suffira de faire :SAVE"PRG.BIN",B,10000,200 pour le stocker sur la disquette et par la suite , pour le réutiliser , LOAD"PRG.BIN",40000:CALL 40000 . N'oubliez jamais que dans ce cas votre programme est sauvegardé depuis une adresse n'ayant rien à voir avec sa véritable location et que celle-ci doit IMPERATIVEMENT être spécifiée après le LOAD .

Note : Plusieurs directives ORG peuvent être incluses dans un même programme .

La directive LIMIT (V.O. Page 13) .

Cette directive interdit à MAXAM de poursuivre l'assemblage au delà de la valeur donnée dans LIMIT . Si votre code source déborde , le message : FATAL ERROR : Code limite exceeded , sera affiché . Initialement fixée par MAXAM en &A375 , vous pouvez la réviser à votre guise pour :

- 1 : Préserver une zone mémoire où se trouve un autre code .
- 2 : Limiter la taille d'un programme .
- 3 : Assembler en RAM écran ou dans la zone CPM (#BE80) .

LIMIT n'affecte que le stockage du code en mémoire et pas son emplacement réel si il est différent (ORG 10000,40000) par exemple .

Note : Vous disposez en effet d'un peu plus de 100 octets en #BE80 pour implanter une courte routine qui survivra même à un reset du CPC ce qui est souvent fort utile. Cette zone RAM n'est utilisée par le CPC que sous CPM.

Les directives CODE et NOCODE (V.O. Page 14) .

NOCODE permet de vérifier un assemblage sans rien stocker en RAM . A partir du moment où cette directive est rencontrée dans un listing , le stockage en RAM du binaire est interrompu et ne reprend que lorsqu'une directive CODE est trouvée . Attention , si un CALL routine est situé dans une zone CODE et que la routine proprement dite est dans une section NOCODE le label de la routine sera considéré comme absent !

La directive END (V.O. Page 14) .

L'usage de END signale à l'assembleur la fin de son travail . Quoique son usage soit facultatif elle permet cependant de stopper l'assemblage avant la fin normale du programme .

Les expressions (V.O. Page 14) .

Chaque fois que c'est nécessaire on peut utiliser une expression arithmétique avec l'assembleur .

Nombres : Décimaux : (132)
Hexadécimaux : Au choix &BB5A ou #BB5A .
Binaire : %10001000

Caractères : "A" ou "'" . L'apostrophe remplace le guillemet comme délimiteur si l'on veut voir le guillemet apparaître dans un texte et réciproquement .

Opérateurs arithmétiques : +, -, *, /, MOD (Comme en basic)
Opérateurs logiques : AND , OR , XOR .

Toute expression est évaluée sous forme d'entiers non signés . En cas de dépassement de capacité , ce sont les 16 bits de poids faible que l'on retrouve dans le résultat .

Les symboles (V.O. Page 15) .

L'assembleur se réfère à une table de symboles auquel l'assemblage assigne une valeur sur 16 bits . Un symbole peut être comparé à une variable BASIC . C'est lors de la seconde passe que l'assembleur calcule les adresses de branchement . A ce moment , si un symbole est employé sans avoir été défini un WARNING sera affiché .

Note : Le symbole peut inclure des chiffres mais ne doit JAMAIS COMMENCER PAR UN CHIFFRE .

On peut définir 4 types de symboles .

1 / Le label ou étiquette : Ce symbole fait référence à une routine destinée à être appelée par le programme ou un point de branchement pour saut . Exemple :

```
CALL LABEL
JP ETIQUETTE
;
LABEL LD A,7
RET
;
ETIQUETTE JP #BB5A
```

Ce même label peut aussi bien être affecté à une zone de données (Equivalent de DATA en BASIC) . Exemple :

```
LD HL,TEXTE ;Ceci permet d'afficher la phrase complète
BOUCLE LD A,(HL) ;pointée par texte .
OR A
RET Z
CALL #BB5A
INC HL
JR BOUCLE
;
TEXTE DB "Ceci est un texte",0
```

2 / Comme une constante par la directive EQU . Dans ce cas , le symbole contient DEFINITIVEMENT la valeur affectée . On ne peut redéfinir une seconde fois le même symbole dans un même programme .

```
VALEUR1 EQU 4000
VALEUR2 EQU 10000
LD HL,VALEUR1 ;Met 4000 dans HL .
LD DE,VALEUR2 ;Met 10000 dans DE .
RET
```

Note : On peut bien sur utiliser des expressions et écrire : VALEUR3 EQU VALEUR1+VALEUR2 . Dans ce cas VALEUR3 vaudra 14000 si on a pris la précaution de définir PREALABLEMENT les 2 termes de l'expression .

3 / Comme avec EQU mais par LET . Dans ce cas on peut redéfinir plusieurs fois le symbole .

4 / Par GET et PUT . Ces 2 commandes étant spécifiques à MAXAM et à l'assemblage sous BASIC , nous les verrons dans un chapitre spécial .

Placer des données dans un programme objet . (V.O. Page 16) .

Les directives qui suivent sont l'équivalent de DATA dans un programme BASIC . Pour chaque opération plusieurs syntaxes sont possibles pour conserver une certaine compatibilité avec d'autres assembleurs .

1 : BYTE , TEXT , DEFB , DB , DEFM : Ces 5 directives (Au choix) ordonnent à l'assembleur de stocker une suite de données 8 bits exprimées en valeurs numériques où en chaînes de caractères . Une chaîne composée d'un seul caractère est considérée comme valeur numérique ce qui permet la syntaxe un peu bizarre : DEFB "TEXT","E"+#80 . Dans ce cas , le 7ème bit du dernier caractère est mis et peut servir de reconnaissance de fin de chaîne (Pour RSX par exemple) .

2 : DEFW , DW , WORD : 3 Manières possibles de stocker des valeurs 16 bits en mémoire .

3 : DEFS , DS , RMEM : Ici 3 possibilités de réserver un nombre d'octets donnés en RAM . Exemples : DS 10 Réserve 10 octets . DS 10,&FF Réserve toujours 10 octets ET leur donne la valeur &FF .

Voici un exemple montrant les possibilités de stockage :

```
10 MEMORY 10000
20 ;ASSEM
30 'ORG 10000
40 'DB 10,20,30,40,"TEXTE"
50 'DW 10000,20000,30000
60 'DS 10,&FF
70 'END
80 MODE 2 ; PRINT" AFFICHAGE DES VALEURS STOCKEES"
90 PRINT:PRINT" Par DB : ";;FOR I=10000 TO 10003:PRINT PEEK(I);:NEXT
100 FOR i=10004 TO 10008:PRINT CHR$(PEEK(i));:NEXT
110 PRINT:PRINT:PRINT" Par DW : ";
120 FOR i=10009 TO 10013 STEP 2:PRINT PEEK(i)+256*PEEK(i+1);:NEXT
130 PRINT:PRINT:PRINT" Par DS : ";;FOR i=10015 TO 10024:PRINT
PEEK(i);:NEXT
140 END
```

GET et PUT (V.O. Pages 17/19) .

Ces 2 commandes sont spécifiques à l'usage sous basic et permet un échange entre BASIC et assembleur lors de l'assemblage . Cela peut servir :

- 1 : A passer à l'assembleur l'adresse de stockage du code .
- 2 : Passer les variables qui contrôlent l'assemblage conditionnel .
- 3 : Retourner au basic les adresses des points d'entrée .

Pour faire bon usage de GET et PUT , les paramètres à utiliser doivent suivre derrière la commande !ASSEMBLE (Ou !ASSEM) . Ex : !ASSEMBLE,debut,x,@debut+1 . Chaque paramètre doit être une constante ou une variable numérique . Pour conserver la compatibilité avec les CPC 464 , l'utilisateur de 6128 aura soin de toujours passer les variables chaînes et réelles par leur pointeur ! L'utilisateur de CPC 464 n'a pas le choix ...

PUT Fonctionne un peut comme POKE mais sur 2 octets au lieu d'un . L'assembleur ne reconnaît pas directement les variables de type BASIC et il faudra jongler un peu ! On fera :

- 1 : Création d'une variable basic : entree=0
- 2 : Avertir l'assembleur qu'il doit utiliser cette variable : !ASSEM,@entree .
- 3 : Transformer la variable BASIC en variable assembleur par GET . Le nom de la variable assembleur est indifférent mais il est préférable d'en choisir un qui rappelle la variable initiale : GET entree_adr
- 4 : Au point d'entrée du code , définir un label .entree . (Facultatif mais utile pour plus de lisibilité du code) .
- 5 : Dépôt de l'adresse d'entrée dans la variable BASIC par PUT : PUT entree_adr,\$ ou PUT entree_adr,entree .

L'exemple complet qui suit sera peut-être un peu moins obscur ...

```
10 MEMORY 10000
20 GOSUB 1000:MODE 2
40 CALL hexout2,7:PRINT:CALL hexout2,&A0:PRINT:CALL hexout2,&C1
50 PRINT
60 CALL hexout4,&1E2B
70 PRINT:CALL hexout4,&A03F:PRINT:CALL hexout4,&F03A
999 END

1000 'ROUTINE ASSEMBLEUR QUI AFFICHE LE NOMBRE DONNE EN HEXA DECIMAL
1010 hexout2=0:hexout4=0
1020 !ASSEM,@hexout2,@hexout4      'Adresse des 2 routines possibles .
1030 'GET hexout2_ref,hexout4_ref ;Conversion en variables assembleur .
1040 'LET outtxt=&BB5A             ;Le symbole outtxt prend la valeur &BB5A
1050 '
1060 '.hexout4 PUT hexout4_ref,$   ;Ceci recupère au bon moment la valeur
1070 'LD A,(IX+1)                  ;16 bits transmise .
1080 'call hexout2                  ;Sortir octet fort .
1090 'LD A,(IX+0)
1100 'CALL hexout2                  ;Puis octet faible .
1110 'RET
1120 '
1130 'PUT hexout2_ref,$           ;Ceci recupère au bon moment la valeur
1140 'LD A,(IX+0)                  ;8 bits transmise .
1150 '.hexout2
1160 'PUSH AF                       ;Préserver valeur .
1170 'RRCA:RRCA:RRCA:RRCA         ;Inverser les 2 quartets .
1180 'CALL hexout1                 ;Sortir 1er chiffre .
```

```

1190 'POP AF ;Récupère valeur et sort second chiffre.
1200 '.hexout1
1210 'CALL binasc ;Adapter format binaire à format ASCII
1220 'CALL outtxt ;Sortir caractère .
1230 'RET
1240 '
1250 '.binasc
1260 'AND &F ;Isoler quartet faible .
1270 'ADD A,&30 ;Décalage ASCII de 0 à chr$("0")
1280 'CP &3A ;Si < 10 c'est OK .
1290 'RET C
1300 'ADD A,7 ;Sinon ajouter décalage ASCII entre "9"
et
1310 'RET ;"A" pour valeur supérieure à 9 .
1320 'END
1330 RETURN

```

Note : L'instruction CALL suivie de paramètres modifie les registres comme suit : A = Nombre de paramètres . IX=Adresse du dernier paramètre transmis .

Assemblage conditionnel . (V.C.Pages 20/21)

Ce type d'assemblage est utile lorsque vous avez besoin de plusieurs versions d'un même programme . (Ex : Disque ou cassette) . L'inclusion des directives appropriées permet d'assembler la section de programme en rapport avec l'usage voulu . Plus concrètement , on définit certains blocs du code source qui ne seront assemblés que si la condition d'entrée est remplie .

La condition est en réalité une valeur 16 bits signée et considérée comme vraie si ce nombre est positif (De 1 à 32767) et fausse si il est négatif . Un exemple qui permet d'assembler un code pour 464 OU pour 6128 :

```

CPC464 EQU 1
IF CPC464
Ici un bloc de code source pour 464
ELSE
Ici un bloc de code source pour 6128
ENDIF

```

Dans ce cas , c'est le bloc de code CPC 464 qui sera assemblé . Pour assembler le code CPC 6128 il faut modifier la 1ère ligne : CPC464 EQU 0 .

IF peut aussi remplacer la directive LIMIT . On déclare un label 'debut' situé au commencement du programme et un label 'fin' sur le dernier octet . Dans l'exemple qui suit , un message vous avertira si le programme devient trop encombrant . Ceci présente l'avantage de ne pas interrompre l'assemblage en cas de débordement comme c'est le cas avec LIMIT . Dans l'exemple qui suit 'longmax' contient la longueur maximale à ne pas dépasser .

```

1000 !ASSEMBLE,longmax
1010 'GET longmax
1020 '.debut
1030 'Le code source se met ici .
1040 '.fin
1050 'IF fin-debut-longmax
1060 'PRINT"Code trop long"
1070 'ENDIF
1080 'RETURN

```

Les opérateurs logiques AND , OR , XOR , peuvent aussi être utilisés avec les conditions mais ce n'est que très très rarement utile . Voyez la V.O. Page 21 .

On peut aussi imbriquer des blocs IF (Jusqu'à 10) . Si par exemple on a besoin de 3 versions d'un même programme : ROM,DISQUE,CASSETTE , on fera :

```
IF ROM
Code source pour ROM
ELSE:IF DISQUE
Code source pour disque
ELSE
Code source pour cassette
ENDIF
ENDIF
```

Pour conclure , 3 formes particulières de IF .

IFNOT . C'est le contraire de IF . La section de code sera assemblée si la condition donnée N'EST PAS VERIFIEE .

Les 2 autres formes ne sont utiles que pour afficher des messages . Ce sont IF1 et IF2 dont l'un sera vrai et l'autre faux selon la passe (1 ou 2) d'assemblage en cours . Ex :

```
IF1
PRINT "1ère passe"
ENDIF
IF2
PRINT "2ème passe"
ENDIF
```

Lecture d'un code source à partir d'un fichier . (V.O. Page 22) .

La place RAM dans votre CPC est limitée et cette directive READ'fichier' vous sera utile à partir du moment où votre code source ne tient pas dans la RAM . Elle assemble directement depuis la disquette ou la cassette et laisse toute la RAM disponible pour le code binaire . Le fichier peut être du BASIC ou de l'ASCII écrit avec l'éditeur de MAXAM ou tout autre traitement de texte aux normes ASCII . Le véritable intérêt de cette commande est de pouvoir assembler plusieurs fichiers source à la suite , donc de scinder un même programme en plusieurs codes source . Par exemple , la zone des variables , le corps du programme , les sous routines . Dans ce cas vous créez un 4ème fichier qui contiendra :

```
10 !ASSEMBLE
20 'ORG ou vous voulez
30 'READ "PRGINIT.MAX"
40 'READ "PRGMAIN.MAX"
50 'READ "PRGSUB .MAX"
60 'END
```

Et que vous sauvegardez sous le nom : 'PRGASSEM.BAS' . Le lancement de ce fichier générera le code source désiré . Pour savoir où vous en êtes , il est avisé de commencer chacune des 3 sections par :

```
1 PRINT 'J'assemble le fichier ....'
```

Le numéro de ligne 1 remet à 0 le compteur de ligne ce qui est indispensable en cas d'erreur. Faute de ceci, une erreur dans le 3ème programme tiendra compte de la totalité des lignes déjà traitées et vous aurez du mal à la retrouver.

Usage de READ avec la cassette. Un conseil : N'utilisez pas cette commande avec un lecteur de cassette, vous joueriez avec vos nerfs. Si vous y tenez vraiment, voyez la V.O. Page 22.

Ecrire d'un code binaire à partir d'un fichier source. (V.O. Page 22)

C'est l'inverse de READ. La directive WRITE "fichier" assemble le code source et écrit le fichier binaire sur la disquette. Un détail pour les utilisateurs de CPM : Donner l'extension '.COM' au nom de fichier génère un code directement utilisable par CPM (ORG &100). Ce code spécifique n'est pas utilisable par l'AMSDOS. Un WRITE doit toujours se terminer par un CLOSE qui signale que le fichier est fini.

```
Ex: ORG ou on veut
     WRITE 'PRG.BIN'
     Le code source .
     CLOSE
     END
```

Pour utiliser avec READ on écrira :

```
10 !ASSEMBLE
20 'ORG ou vous voulez
25 'WRITE "PRG.BIN"
30 'READ "PRGINIT.MAX"
40 'READ "PRGMAIN.MAX"
50 'READ "PRGSUB .MAX"
55 'CLOSE
60 'END
```

2 remarques. 1 : WRITE doit toujours être placé avant READ. 2 : Le fichier sauvegardé par WRITE ne contient pas la véritable adresse d'assemblage dans l'en-tête ! Il est indispensable d'utiliser la formule : LOAD "PRG.BIN", adresse. Adresse étant celle donnée comme ORG au programme.

Les commandes de l'assembleur (V.O. Page 23/25)

LIST et NOLIST : Ces 2 commandes activent ou désactivent l'affichage du listing à l'écran lors de l'assemblage. LIST P Envoie le listing sur l'imprimante.

Une ligne de listing se présente comme ceci :

00010	9C44	FE 0A	TEST	CP 10	;Voir si c'est 10
↑	↑	↑	↑	↑	↑
Numéro	Adresse	Octets du	Label	Instruction	Commentaire .
de ligne	du code	code (1à4)		et opérande	

PRINT "Quelque chose" : Permet d'afficher un texte repère à l'écran pendant l'assemblage.

PAUSE : Interrompt l'assemblage et attend l'appui sur une touche pour reprendre .

DUMP : Affiche ou imprime la liste des symboles utilisés par le programme.

PLEN :. Sans cette commande , le listing est imprimé en continu sans sauts de page . PLEN n (N entre 40 et 255) définit le nombre TOTAL de lignes sur une page .

PAGE <expression> : Provoque l'éjection de la page en cours . L'expression est facultative et servira de nouveau numéro de page (De 0 à 255) .

TITLE "Votre titre" : Servira si vous voulez voir un titre imprimé en haut de chaque page .

Section 6 : Programmes d'application . Voir la V.O.Pages 26/29 .

Section 7 : Les utilitaires accessibles par menus (V.O. Pages 30/33) .

Pour accéder à ces utilitaires , tapez !MAXAM,1 ou !MAXAM,2 selon le mode écran voulu . !MAXAM tout court conserve le mode en cours . Ceci fait vous verrez apparaître un menu . Pour activer l'une des option , appuyez sur les touches dont la (ou les 2) lettre(s) figurent à gauche de la ligne du menu et concluez par ENTER . ESC retourne au menu principal A TOUT MOMENT .

Note : TOUTES LES VALEURS NUMERIQUES DOIVENT ETRE DONNEES EN HEXADECIMAL.

T : Va à l'éditeur de texte ASCII pour rédiger les codes source . (On verra ça dans la section suivante) .

B : Retourne au BASIC .

D : Désassemble .

DP : Désassemble et imprime .

Pour effectuer le désassemblage , le programme a besoin de 2 adresses exprimées en hexadécimal . Vous devrez donner :

1 : Adresse de début du désassemblage .

2 : Adresse de fin du désassemblage . Cette dernière est facultative , vous pouvez répondre juste par ENTER et faire ESC 2 fois pour quitter le désassemblage .

L : Liste la zone mémoire donnée .

LP : Imprime la zone mémoire donnée .

Il faut fournir les adresses début et fin comme pour désassembler . La mémoire est listée par lignes de 16 octets en hexadécimal et ASCII .

S : Sélectionne une ROM externe entre #C000 & #FFFF .

Donnez juste le numéro de ROM à activer . Exemple : S 7 puis D C000 C050 et vous désassemblerez la ROM DISQUE . L'état de sélection ROM/RAM est affiché en permanence en haut de l'écran .

O : Bascule entre la RAM et la ROM de l'adresse 0 à &3FFF . Exemple : O puis D O 170 pour désassembler le début de la ROM et encore O pour re7activer la RAM .

E : Editer la mémoire .

Fournir les adresses comme pour L ou D et vous voyez apparaître la page listing plus un curseur qui clignote au début de la ligne médiane . Les flèches déplacent ce curseur sur la page .

SHIFT ou CTRL plus flèche haute : Remonte d'une page .
SHIFT ou CTRL plus flèche basse : Descend d'une page .
SHIFT ou CTRL plus flèche gauche: met le curseur en haut à gauche .
SHIFT ou CTRL plus flèche droite: met le curseur en bas à droite .
TAB vous permet de passer de la zone hexadécimale à la zone ASCII .
ESC quitte l'édition .

Une fois le curseur placé sur la zone à éditer , toute autre touche modifie la valeur de l'octet pointé . Attention , en zone hexadécimale , il faut toujours donner 2 chiffres ! Pour éditer 9 tapez 09 .

F : Recherche une chaîne .

FP : Recherche une chaîne et imprime les adresses où elle est rencontrée

Comme d'habitude , il faut donner 2 adresses qui donnent le début et la fin de la zone de recherche . Ensuite , il faut préciser si on cherche de L'ASCII (A) ou de l'hexadécimal (H) . La recherche est limitée à 20 caractères ou 8 octets en hexadécimal . ESC interrompt la recherche .

La chaîne ou la série d'octets à trouver ayant été donnée , les adresses où l'occurrence est trouvée s'affichent (ou s'impriment si LP) .

NOTES : N'oubliez pas que l'adressage 16 bits du Z80 se fait à l'envers et que pour trouver CALL &BB5A il faudra rechercher : CD , 5A , BB . On peut Aussi donner un 'joker' par l'intermédiaire du "?" . La recherche sur : CD , ? , BB donnera toutes les adresses où se trouve un CALL à une adresse commençant par &BB .

M : Déplace un bloc de mémoire .

Donner les adresses de début et de fin du bloc à déplacer et conclure par l'adresse où vous voulez recopier cette zone de mémoire .

C : Compare deux blocs de mémoire .

CP : Compare deux blocs de mémoire et imprime les différences .

Donner les adresses de début et de fin du bloc à comparer et conclure par

l'adresse de début du bloc avec laquelle on effectue la comparaison . Toutes les adresses où un octet différent est trouvé s'affichent ou s'impriment selon le cas .

R : Reloge un bloc mémoire à une adresse donnée .

On peut utiliser cette commande de 2 manières : une simple et une complexe . Répondez 'Y' ou 'N' selon votre choix .

1 : Méthode simple : Vous donnez juste les adresses de début et de fin de la zone à reloger suivies de l'adresse de relocation puis vous voyez si votre programme fonctionne à sa nouvelle adresse . Si cela ne marche pas , c'est parceque les éventuelles zones de données sont traitées comme du code par cette option . D'autre part , si le programme comporte une instruction du genre LD HL,40000 , il ne peut savoir si 40000 est une valeur absolue ou un pointeur dans une zone de données qui doit être modifié en fonction de la nouvelle adresse de relocation . Donc un certain cafouillage est très possible ...

2 : Méthode complexe : Vous devrez en prime fournir 5 précisions à MAXAM.

A : Adresses de début et de fin du bloc à reloger .

B : 1ère et dernière adresse pour définir la zone des références à modifier (La zone programme) .

C : Décalage à ajouter aux adresses comprises dans la limite spécifiée.

Dans ce cas , le bloc est modifié sans être déplacé . La remarque pour les instructions du genre LD HL,40000 reste valable .

I : Initialisation d'un bloc .

La zone mémoire spécifiée est remplie avec la valeur de l'octet donné en 3ème paramètre .

X : Vous permet d'utiliser une RSX externe .

Cette commande affiche le curseur en bas de l'écran et attend votre commande . Vous pouvez indifféremment donner une commande drive (ERA , CAT , REN , Etc ...) Comme une commande MAXAM , (MODE , FIND , Etc ...) L'emploi des guillemets pour les commandes drive est facultatif sauf si la chaîne de caractère commence par un chiffre . Ex : ERA PROG.MAX , ERA "1PROG.MAX" . Les paramètres exprimés en hexadécimal doivent commencer par & . Les séparateurs (Espaces , virgules signes d'égalité) éventuels doivent être respectés . Après exécution d'une commande RSX , le curseur revient et attend une autre commande ou ESC pour quitter cette option .

Section 8 : L'éditeur de texte . (V.O. P.34/40) .

Il n'y a aucune obligation d'inclure le code source dans un programme BASIC . L'éditeur de texte vous permet de créer un fichier source tout aussi complet et de générer le programme binaire sans problème . Toutes les commandes et directives étudiées précédemment restent valables à l'exception de ASSEMBLE , GET et PUT qui ne servent qu'en BASIC .

On accède à l'éditeur de texte et à ses options par l'option T du 1er menu décrit en section 7 .

Editer un code source : Choisir l'option E du menu . Une page presque vierge s'affiche et sur la 1ère ligne vous lisez dans l'ordre :

1 : La position du curseur exprimée en lignes et colonnes .

2 : Le nombre d'octets libres pour la rédaction du programme . Attention , ne le laissez pas descendre en dessous de 1500 , vous pourriez ne plus revenir à l'éditeur après un détour par le BASIC ! Si votre fichier devient trop encombrant , coupez le en 2 ou 3 et utilisez READ pour l'assembler . La place disponible pour le fichier correspond à l'intervalle entre la fin du BASIC et HIMEM .

3 : Insertion / recouvrement . CTRL + TAB passent d'un état à l'autre . En mode insertion , le texte à droite du curseur est repoussé par les nouveaux caractères et recouvert en mode 'overwrite' .

4 : CAPS LOCK / SHIFT LOCK : Informe sur l'état du clavier . Le mode SHIFT LOCK est très déconseillé , si vous l'activez accidentellement , retournez de suite en mode normal par CTRL+CAPS LOCK .

Pour entrer du texte avec l'éditeur , il suffit de le taper . Les flèches seules déplacent le curseur à l'intérieur du texte . ENTER passe à la ligne suivante (Si mode 'Insert') . DEL et CLR ont le même rôle qu'avec l'éditeur BASIC . SHIFT + TAB insère une nouvelle ligne , SHIFT + DEL détruit la ligne en cours . TAB passe à la tabulation suivante .

SHIFT + Flèche gauche : Curseur en début de ligne .

SHIFT + Flèche droite : Curseur en fin de ligne .

SHIFT + Flèche haute : Recule la page d'une ligne .

SHIFT + Flèche basse : Avance la page d'une ligne .

CTRL + Flèche gauche : Va au début du texte .

CTRL + Flèche droite : Va à la fin du texte .

CTRL + Flèche haute : Recule d'une page .

CTRL + Flèche basse : Avance d'une page .

Une ligne de texte peut contenir jusqu'à 255 caractères , au delà des 80 caractères prévus pour l'écran , la page se décale sur la gauche .

Marquage et traitement des blocs :

Il est possible de marquer , puis de déplacer , copier , détruire des blocs de lignes comme avec un traitement de texte . Pour marquer un bloc de lignes , placez le curseur sur l'une de ses limites et appuyez sur SHIFT+COPY puis renouvellez l'opération à l'autre extrémité du bloc . Si vous essayez de placer un 3ème marqueur , un message 'Both markers in use' et un bip vous en avertissent et il faut alors appuyer sur ESC . Les marqueurs apparaissent en vidéo inverse .

Les 3 commandes qui suivent n'ont d'effet que si les 2 marqueurs sont initialisés . Dans le cas contraire , un message 'Block not defined' et un bip vous en avertissent et il faut alors appuyer sur ESC .

CTRL+CLR : Déplace le bloc marqué à l'emplacement actuel du curseur .
CTRL+COPY : Copie le bloc marqué à l'emplacement actuel du curseur .
CTRL+DEL : Détruit le bloc .

Note : Les deux lères opérations ne peuvent s'effectuer que si la position du nouveau bloc est EN DEHORS du bloc défini , sinon vous aurez droit au message 'Cursor in block , press ESC key.' . D'autre part , il faut que le nombre d'octets libres soit supérieur au nombre d'octets à déplacer ou copier sinon c'est 'Not enough memory , press ESC key' que vous obtiendrez .

ESC : Quitte l'éditeur et vous renvoie au menu principal .

Le menu de l'éditeur (V.O. Pages 36/38) .

L : Charge un fichier texte dans l'éditeur . Le précédent fichier est détruit . Un message 'Not ASCII file' vous préviendra s'il ne s'agit pas d'un fichier MAXAM .

LB : Charge un bloc de texte qui vient s'insérer dans le texte initial à la position où vous avez mis le curseur avant de quitter l'éditeur par ESC .

S : Sauve le fichier texte en entier .

SB : Sauve la section de texte que vous aurez pris la précaution de définir avec les 2 marqueurs de l'éditeur avant de le quitter par ESC . Dans le cas contraire , vous aurez droit au message 'Bloc not defined' ...

Note : Les commandes de chargement sauvegarde peuvent aussi vous renvoyer tout message d'erreur spécifique au drive .

P : Imprime tout le fichier texte . ESC pour stopper .

PB : Imprime la section de texte définie comme pour SB .

Note : Si l'imprimante n'est pas prête , le curseur se réaffiche après une brève attente .

M : Modifie texte . Cette commande permet de transformer un fichier créé depuis l'éditeur en fichier compatible avec le basic et inversement . 4 options sont disponibles :

1 : Ajoute les numéros de ligne . On peut spécifier le numéro de départ et l'incrément entre 2 lignes .

2 : Ajoute des apostrophes (REM) entre le numéro de ligne et le texte .

3 : Supprime les numéros de ligne .

4 : Supprime les apostrophes .

MB : Comme M mais n'agit que sur un bloc défini avec l'éditeur .

F : Trouver une chaîne donnée . Fonctionne exactement comme !FIND sous BASIC . Attention , la recherche se fait à partir de la position du curseur dans le texte . Si vous n'avez rien trouvé , vérifiez quand même que vous ne l'avez pas laissé sur la dernière ligne ...

Si la chaîne spécifiée est trouvée , MAXAM repasse en mode édition à l'endroit où la 1ère occurrence a été trouvée . L'appui sur COPY ira rechercher l'occurrence suivante si elle existe . L'appui sur ESC quitte le mode recherche et permet de modifier la ligne trouvée . Ceci n'annule en rien l'effet de COPY qui perdure tant qu'une autre commande F n'est pas donnée .

R : Trouver une chaîne donnée et la remplacer par la seconde . 2 options sont possibles : Global ou Partiel . Dans le 1er cas , le remplacement se fait automatiquement . Dans le second , MAXAM passe en mode édition et demande à chaque occurrence si le remplacement doit se faire .

(Y/N) . COPY et ESC ont le même effet que pour FIND .

Note : L'option 'Global' est souvent dangereuse ! Imaginez que vous ayez une routine baptisée 'EST' et que vous vouliez la remplacer par 'OUEST' alors que d'autres routines se nomment CESTUN , LESTE ... Après un remplacement global , TOUTE occurrence EST sera remplacée par OUEST et vous aurez aussi modifié , COUESTUN , LOUESTE etc ...

Note : F et R différencient majuscules et minuscules .

T : Pour établir (Jusqu'à 8) , détruire une tabulation ou simplement les lister .

A : Assemble le fichier source comme !ASSEMBLE en BASIC et attend l'appui sur une touche .

J : Exécute le code binaire précédemment assemblé . Il faut spécifier l'adresse d'exécution du code .

G : Aller en ligne N . Passe en mode édition à la ligne spécifiée . Si vous êtes déjà dans l'éditeur , CTRL+G ont le même effet .

X : Commande externe comme dans le menu utilitaires .

Q : Retourne au menu précédent .

Procédure de débogage (V.O. Page 39) .

La commande BRK fonctionne exactement comme en BASIC , insérez juste une ligne BRK aux emplacements à tester , assemblez votre programme , lancez le code par J depuis le menu ou par CALL depuis le BASIC . Etudiez le contenu des registres à chaque affichage et stoppez tout par ESC si le résultat vous surprend ou continuez l'exécution avec toute autre touche s'il ça va .

MAXAM accepte les caractères redéfinis depuis le BASIC .

L'éditeur et le BASIC (V.O. Page 40) .

Si vous utilisez un programme BASIC pour tester un code source généré à partir de l'éditeur, sachez que l'éditeur modifie les pointeurs du BASIC et qu'un SAVE tout à fait normal du basic ajoutera au fichier BASIC la longueur du texte en RAM. Ainsi la simple ligne : 10 CALL 40000 peut vous faire un fichier de 24K sur la disquette pour peu que le fichier texte en fasse 23. Pour éviter ceci, détruisez le fichier texte par !CLEAR avant de sauver votre programme BASIC.

Autre méthode : Sauvez votre programme BASIC en ASCII sous la forme : SAVE"PR0G.BAS",A .

L'appel à l'éditeur puis le retour au basic détruit TOUTES les variables précédemment initialisées par le BASIC .

NEW : Supprime le BASIC ET LE TEXTE .
DELETE : Supprime le BASIC seul .
!CLEAR : Supprime le TEXTE seul .

Le chargement d'un nouveau programme BASIC détruit le texte édité . Une seule exception : Le programme BASIC à été sauvegardé en ASCII et on le recharge par MERGE .

Les erreurs fatales (V.O. Page 42 (d)) .

- 1 : Directive ORG suivie d'une expression non définie .
- 2 : Directive EQU suivie d'une expression non définie .
- 3 : Directive DS (DEFS RMEM) suivie d'une expression non définie .
- 4 : Directive IF / IFNOT suivie d'une expression non définie .
- 5 : Directive PUT suivie d'une variable non définie .
- 6 : Erreur dans une structure conditionnelle .
- 7 : Ligne plus longue que 255 caractères .
- 8 : L'assemblage dépasse les limites de la mémoire allouée .
- 9 : Le fichier à lire par READ n'est pas en ASCII .
- 10 : Erreur dans une série de directives READ .
- 11 : Erreur disque quelconque .
- 12 : Le code objet dépasse la valeur fixée par LIMIT .

Additif à la notice originale .

Traduction des messages d'erreur les plus courants .

Bad syntax : Erreur de syntaxe dans la zone instruction du genre : CAL BB5A .

Bad addressing mode : Mauvais mode d'adressage . MAXAM conteste votre conception de l'adressage ! Evitez de lui donner des incohérences du genre LD 10000,BC .

Bad expression : Mauvaise expression . 12,30 au lieu de 12+30 c'est une mauvaise expression .

Bad IF / ELSE nesting : Votre assemblage conditionnel est bien malade .. Relisez la notice .

Bad variable reference : Mauvaise référence de variable . Il se passe quelque chose du côté de GET et PUT .

Possible missing space : Il manque peut-être un espace . Cela arrive si l'instruction n'est pas sur la même ligne que le label ou que le label est suivi de deux points . Rectifiez ou non , ce n'est pas grave .

Jump out off range : Saut hors de portée . Rien de grave , remplacez juste le JR par un JP ou le DJNZ par DEC B - JP NZ,...

Number too large : Nombre trop grand . Restez donc dans la limite de capacité des registres . 8 bits c'est de 0 à 255 et 16 de 0 à 65535 , pas plus !

Duplicate definition : Définition multiple . Vous utilisez plusieurs fois le même symbole pour définir une routine ou une constante .

Undefined symbol : Symbole non défini . La référence voulue n'apparaît pas dans votre programme , c'est peut-être une faute de frappe .

Missing quote : Il manque une valeur quelque part .

Missing label : Il manque un label (Depuis 4 ans que j'utilise MAXAM je n'ai jamais vu ce message apparaître) .

Missing ENDIF : Il manque ENDIF . Mettez le au bon endroit .

Spurious text ignored : Faux texte ignoré . MAXAM ne comprend plus , cela se produit souvent si un séparateur , virgule espace ou guillemet manque dans une série de DEFB .

Line too long : Ligne trop longue . Pas plus de 255 caractères SVP .

Code limit exceeded : Limite du code dépassée . Augmentez LIMIT ou réduisez la longueur du code .

Out off parameters : Débordement des paramètres . Il n'y a plus de place quelque part , sans doute un abus de GET ou PUT ou trop de variables transmises à ASSEMBLE .

Out off memory : Plus de place en RAM . Au choix : Réduire , la taille du texte , du basic s'il y-en a , remonter le HIMEM . Si ce message vous interdit l'accès à l'éditeur , essayez d'abord CLEAR (La commande BASIC pas la RSX MAXAM) complétez par NEW pour détruire le BASIC si CLEAR ne suffit pas puis essayez de remonter le HIMEM . Si cela ne fonctionne pas , votre texte est perdu , vous auriez dû le sauvegarder plus tôt ...

- TABLE DES MATIERES -

Chapitres	Pages
Qu'est-ce que MAXAM ? Pour commencer .	1/2
Généralités , commandes externes .	2
Exemple concret et l'assembleur en détail .	3
Correction des erreurs et débogage .	4
Messages d'erreur , location code objet , ORG .	5
Quelques directives (LIMIT , CODE Etc ...)	6
Les expressions , les symboles .	7
Placer des données dans un programme objet .	8
Les directives GET et PUT .	9
L'assemblage conditionnel .	10
Assemblage à partir de la disquette (READ) .	11
Assembler directement sur la disquette (WRITE) .	12
Les commandes de l'assembleur .	12/13
Le menu des utilitaires .	13/15
L'éditeur de texte .	15/17
Procédure de débogage , Editeur et BASIC .	17
Erreurs fatales , et additif	18/19