asMSX

Cross-assembleur pour MSX

v.0.12g

- Edition 2007 -

[18 Mars 2007]

Traduit de l'espagnol par Granced

Index

1 Introduction....3
1.1 Description....3
1.2 Caractéristiques....4
1.3 Justification....4
1.4 Syntaxe....5
1.5 Utilisation....5
2 Langage assembleur....5
2.1 Syntaxe du langage....5
2.2 Etiquettes....6
2.3 Expressions numériques....7
2.3.1 Formats numériques....7
2.3.2 Opérateurs....8
2.4 Introduction de données....9
2.5 Directives....10
2.6 Commentaires....15
2.7 Assemblage conditionnel....15
3 Autres outils....17
3.1 RLEpack....17
3.2 binDB....19
3.3 PCX2MSX....20
4 Codes source....21
4.1 Classic Pong v.0.15....21
4.2 Classic Minesweeper v.0.10....21
4.3 Autres exemples....22
4.4 Plus d'exemples....23
5 Renseignements supplémentaires sur asMSX....23
5.1 Nouveautés de la version 0.12g....23
5.2 Nouveautés de la version 0.12f....23
5.3 Nouveautés de la version 0.12e....23
5.4 Nouveautés de la version 0.11....24
5.5 Nouveautés de la version 0.10....24
5.6 Licence d'utilisation....24
5.7 Mentions légales....24
6 Ressources disponibles....25
6.1 Sites web intéressants....25
6.2 Autres ressources sur Internet....25
6.3 Bibliographie recommandée....26
6.4 Autres assembleurs....27
6.5 Conseil gratuit....27
7 Distribution....28

1 Introduction

1.1 Description

asMSX est un assembleur pour le processeur Z80 qui intègre un certain nombre de caractéristiques qui facilitent la tâche de programmation sur MSX. Il s'agit d'un cross-assembleur qui fonctionne sur PC via un système coopératif avec Windows.

Grâce à la vitesse des ordinateurs actuels, l'assemblage est quasi-instantané, ce qui est un avantage considérable en comparaison avec les assembleurs natifs sur MSX. De même, il n'y a aucune restriction sur la taille du code source, et celui-ci peut être édité avec n'importe quel éditeur de texte.

La première version de asMSX (v.0.01) a été publiée en 2000, et n'était alors qu'une esquisse de ce qu'est dorénavant asMSX. Les erreurs de cette première version ont été corrigées, certains éléments ont été développés et des caractéristiques plus avancées mises en place.

1.2 Caractéristiques

Voici une liste de quelques caractéristiques à souligner :

- supporte toutes les instructions officielles du Z80 avec la syntaxe de Zilog
- assemble toutes les instructions non-officielles documentées par Sean Young
- comprend la syntaxe de la norme Z80 (accumulateur implicite)
- compatible avec différents systèmes de numération (décimal, hexadécimal, octal et binaire)
- opérations arithmétiques et logiques, y compris au niveau des codes source
- prise en charge des nombres décimaux à virgule flottante, converti en virgule fixe
- fonctions mathématiques réelles : trigonométriques, puissances, etc
- accepte de multiples fichiers via des inclusions
- inclusion de tout ou partie de ces fichiers binaires
- étiquettes locales ou générales pour le code source
- routines du Bios du MSX prédéfinies avec leur nom officiel
- génération automatisée de fichiers binaires chargeables depuis le Basic
- production automatisée de fichiers ROM
- supporte 4 types distincts de megaROM : Konami, Konami SCC, ASCII 8 bits et 16 bits
- utilisation de variables numériques à l'intérieur de l'assembleur
- assemblage de fichiers COM (pour MSX-DOS)
- export de la table des symboles
- impression de textes indicatifs depuis le code
- intégration avec le débogueur de BlueMSX
- amélioration de l'assemblage conditionnel
- génération de fichiers CAS ou WAV chargeables sur le MSX

1.3 Justification

Y a t-il vraiment besoin d'un autre assembleur pour Z80 ? Probablement pas. Les annexes comprennent une liste d'assembleurs pour MSX, tant cross-assembleurs qu'assembleurs natifs, ainsi que leur disponibilité. Parmi eux, il existe d'excellents outils de développement qui ont été utilisés par de nombreux programmeurs.

Le but d'asMSX est de fournir un ensemble avec le support complet des instructions du Z80, souple et confortable, fiable et largement orienté sur le développement de logiciels pour MSX. Ces objectifs sont atteints par la version actuelle. Cet assembleur est le plus commode pour le développement de ROMS et de mégaROMS MSX.

Le résultat est une application créée via console de Windows, à partir de 32 Ko de taille et assemblée à très grande vitesse.

1.4 Syntaxe

L'implémentation du langage assembleur pour le processeur Z80 qu'utilise asMSX diverge en certains aspects de la syntaxe officielle de Zilog. La différence majeure étant que les adressages indirects se font en utilisant les crochets [ ], et non pas les parenthèses ( ). La raison de ce changement apparaît évidente au vu du puissant langage mathématique qu'intègre asMSX, qui permet des calculs très complexes, les parenthèses sont ainsi utilisées dans les formules numériques.

1.5 Utilisation

Il s'agit d'une application console pour Windows, ce qui signifie que la manière la plus commode de l'utiliser est à partir de l'invite de commandes de Windows ou de MS-DOS. Pour assembler un fichier, vous avez juste à taper ceci :

```
asMSX fichier.txt
```

Si l'extension n'est pas précisée, l'extension .asm sera prise par défaut. Les fichiers du code source devront être des fichiers textes dans formatage, codés en ASCII 8 bits. Il est également possible d'assembler le code source en faisant glisser l'icône du fichier contenant le programme sur l'icône d'asMSX, mais ce n'est pas recommandé. Les noms Wiindows ne peuvent plus fonctionner correctement, ce qui laisse suggérer que les points autres que celui qui indique le début de l'extension ne sont pas reconnus.

Au cours de l'assemblage, une série de messages apparaîtra, et s'il n'y a aucune erreur dans le code, une série de fichiers sera générée :

- `fichier.sym` : un fichier texte en mode ASCII qui contient tous les symboles définis dans le programme et sa valeur en décimal. Ce fichier est compatible avec l'émulateur BlueMSX, dont le débogueur permet de télécharger la table des symboles externes afin de faciliter le débogage.
- `fichier.txt` : un autre fichier texte en mode ASCII avec les impressions que l'on désire faire depuis le programme. S'il n'est pas utilisé avec une instruction de type PRINT, il ne sera pas généré.
- `fichier[.z80/.bin/.com/.rom]` : le résultat de l'assemblage est enregistré avec le même nom que le fichier d'entrée, l'extension est déterminée par les directives d'assemblage du code source. Ainsi, l'extension Z80 est pour le code sans entête, BIN est pour les fichiers binaires à utiliser à partir du MSX-Basic avec Bload, COM pour les exécutables sous MSX-DOS, et ROM pour les ROMs et mégaROMs MSX.

2 Langage assembleur

2.1 Syntaxe du langage

Le format de ligne de code qu'accepte asMSX est le suivant :

```
[étiquette:] [directive[paramètres]] [;commentaire]
[étiquette:] [code opération[paramètres]] [;commentaire]
```

Chacun de ces blocs est optionnel et sera expliqué plus en détail dans les paragraphes suivants. Il n'y a pas de limitation en ce qui concerne les longueurs de lignes, les espaces ou tabulations, car le préprocesseur s'occupe d'ajuster ces éléments.

Exemple:

```
boucle:
ld
a,[0ffffh]
; lit le registre maître
points:
db
20,30,40,50
; table des points
Org
08000h; positionne le début du code en page 2
Nop
```

2.2 Etiquettes

Les étiquettes sont un moyen d'identifier une certaine position en mémoire. En réalité, cela revient à donner un nom à une valeur 16 bits. Leurs noms peuvent être alphanumériques, mais le premier caractère doit être une lettre ou le caractère « _ » (underscore). Pour être défini comme une étiquette, le nom doit être suivi des deux points (« : »). Attention toutefois, les majuscules et minuscules sont distinguées, ce qui signifie que etiquette: ETIQUETTE: ou EtIqUeTtE: ne sont pas identiques !

Exemple:

```
etiquette:
ETIQUETTE:
_alphanumerique:
boucle001:
```

Les deux points servent pour la définition de l'étiquette dont la valeur est égale à la position de mémoire occupée par l'instruction suivante. Pour utiliser l'étiquette, il faut la référencer sans inclure les deux points. En outre, il est possible de définir des étiquettes locales, en les précédent de @@. La particularité de ces étiquettes est qu'elles ne sont disponibles qu'entre une étiquette globale et une autre. Par conséquent, un code comme celui-ci est tout à fait utilisable :

```
ROUTINE1:
...
@@BOUCLE:
...
ROUTINE2:
...
@@BOUCLE:
...
```

2.3 Expressions numériques

Les expressions numériques peuvent être des nombres ou bien le résultat d'opérations plus ou moins complexes avec ces nombres.

2.3.1 Formats numériques

En ce qui concerne les systèmes de numérations compatibles, la liste suivante donne la bonne syntaxe et quelques exemples :

- **DECIMAL** : les nombres en base 10 se représentent normalement, comme un groupe d'un ou plusieurs chiffres. La seule restriction est qu'il ne faut pas placer de 0 à la gauche du nombre, de façon identique à lorsque l'on écrit

Exemples :

0 10 25 1 255 2048

- **DECIMAL NON ENTIER** : représentés comme les décimaux entiers, mais avec l'emploi du point pour séparer la partie entière de la partie décimale. Il est nécessaire que ce soit le point qui soit employé, dans le cas contraire, ce serait interprété comme un entier.

Exemples :

3.14 1.25 0.0 32768.0 -0.50 15.2

- **OCTAL** : les nombres en base 8 peuvent être représentés de 2 façons : la première est le format utilisé dans les langages C, C++ et Java pour l'indiquer, soit précéder d'un 0 toute la suite de chiffres octaux (0 à 7). La deuxième façon est celle utilisée habituellement par les langages d'assemblage, soit faire suivre le groupe de chiffres par la lettre o, en majuscules ou minuscules. L'octal a été inclus pour la compatibilité, bien que ce ne soit pas un système numérique commode pour une architecture telle que celle du Z80.

Exemples :

01 077 010 1o 10o

- **HEXADECIMAL** : les nombres en base 16, les plus « populaires » dans la

programmation en assembleur, peuvent être représentés de 3 façons différentes. La première est, de nouveau, compatible avec l'utilisation faite dans les langages C, C++ et Java, soit le groupe de chiffres hexadécimaux (0 à 9 et A à F) toujours précédé par 0x. La seconde méthode est celle employée par le langage Pascal : le groupe de chiffres hexadécimaux est précédée par le symbole $. La dernière est celle couramment usitée dans les assembleurs : le le groupe de chiffres hexadécimaux est suivi de la lettre h, en majuscules ou minuscules. A noter que le premier chiffre doit alors toujours être numérique, si le nombre hexadécimal commence par une lettre, il sera alors précédé d'un 0 à sa gauche.

Exemples :

0x8a 0xff 0x10 $8a $ff $10 8ah 0ffh 10h

- **BINAIRE** : les nombres en base 2 ont une seule représentation compatible avec asMSX. Il suffit de mettre la lettre b (majuscules ou minuscules) après le groupe de chiffres binaires (0 ou 1).

  Exemples :

  10000000b 11110000b 0010101b 1001b

2.3.2 Opérateurs

Il est également possible d'effectuer des opération avec ces nombres vus précédemment. La notation des opérateurs est celle utilisée conventionnellement, et pour les opérations mineures, la syntaxe du langage C/C++ a été choisie :

+	Somme
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste de la division entière)

En outre, il existe d'autres opérations :

<<	déplacement vers la gauche (shift left). Déplace vers la gauche le nombre de bits indiqué , ce qui équivaut à multiplier par 2 le même nombre de fois.
>>	déplacement vers la droite (shift right). Déplace vers la droite le nombre de bits indiqué , ce qui équivaut à diviser par 2 le même nombre de fois.

En plus des opérations arithmétiques, nous disposons des opérations logiques, qui opèrent au niveau des bits. Là encore, la syntaxe est celle du langage C. Les opérations binaires sont les suivantes :

|	OU à niveau de bits (OR)
&	ET à niveau de bits (AND)
^	OU exclusif à niveau de bits (XOR)

Il existe une opération unaire :

~	NON logique (NOT) : réalise le complément à 1

Des opérations de comparaison logique ont été également définies, utilisant aussi la même

syntaxe du langage C :

||	OU logique (OR)
&&	ET logique (AND)
==	égalité
!=	inégalité
<	infériorité
<=	infériorité ou égalité
>	supériorité
>=	supériorité ou égalité

L'ordre de préférence des opérateurs est l'ordre conventionnel, le même employé en C/C++. De même, on peut utiliser les parenthèses pour regrouper les opérations, comme on peut le faire pour les opérations arithmétiques.

Exemple :

```
((2*8)/(1+3))<<2
```

Comme pour les décimaux non entiers, on peut employer les mêmes opérations mentionnées ci-dessus et aussi d'autres fonctions réelles :

SIN(X)	Fonction sinus. Les unités sont les radians
COS(X)	Fonction cosinus
TAN(X)	Fonction tangente.
ACOS(X)	Arcsinus
ASIN(X)	Arcosinus
ATAN(X)	Arctangente.
SQR(X)	Carré
SQRT(X)	Racine carrée
EXP(X)	Fonction exponentielle
POW(X,Y)	Elever le nombre X à la puissance Y
LOG(X)	Logarithme décimal
LN(X)	Logarithme népérien
ABS(X)	valeur absolue

Il a été défini en plus une valeur par défaut PI en double précision, pour pouvoir l'utiliser directement dans les fonctions réelles la valeur π.

Exemple :

```
sin (pi*45.0/180.0)
```

Pour utiliser ces valeurs décimales dans le cadre de programmes en assembleur Z80, il est indispensable de convertir les nombres à virgule flottante en virgule fixe. Pour ce faire, voici quelques fonctions de conversion :

FIX(X)	convertit la virgule flottante en virgule fixe
FIXMUL(X,Y)	calcule le produit de 2 nombres en virgule fixe
FIXDIV(X,Y)	calcule la division en virgule fixe de X par Y
INT(X)	convertit en entier un nombre non entier

Il existe un symbole qui a une valeur changeant en permanence, c'est $. La valeur du symbole $ vaudra à tout moment l'adresse de l'assemblage actuel, ou, ce qui est identique en termes d'exécution, la valeur du registre PC.

2.4 Introduction de données

Les données peuvent être incluses par le biais de certaines directives :

db data,[data...]

defb data,[data...]

dt "texto"

deft "texto"

Ces instructions contiennent des informations telles que des octets ou des données 8 bits. DT a été inclus par souci de compatibilité avec d'autres assembleurs, mais en réalité les 4 instructions sont équivalentes.

dw dato,[dato...]

defw dato,[dato...]

C'est de cette façon que se déclarent des mots, soit des données 16 bits.

ds X

defs X

Dans ce mode, on réserve un espace de X octets au point actuel de la mémoire. Cela peut servir à identifier les variables en RAM par exemple. Pour faciliter les choses, les tailles les plus courantes, c'est à dire octet et mot, ont été prédéfinies, en utilisant éventuellement les directives suivantes :

.byte	réserve un espace d'un octet (8 bits)
.word	réserve un espace de 2 octets (16 bits) soit un mot

2.5 Directives

Les instructions prédéfinies servent à l'organisation du code et à activer les caractéristiques

additionnelles d'asMSX.

.ORG X	Sert à indiquer une position en mémoire à l'assembleur. Le code suivant sera assemblé à partir de cette adresse.
.PAGE X	Equivaut à l'instruction précédente, mais n'indique pas une adresse, mais un numéro de page mémoire. Ainsi, .PAGE 0 équivaut à .ORG 0000h, .PAGE 1 équivaut à .ORG 4000h, .PAGE 2 équivaut à .ORG 8000h et .PAGE 3 équivaut à .ORG C000h

identificateur .EQU expression	Sert à définir une constante. Les règles de nommage sont les mêmes que pour les étiquettes, à ceci près qu'il n'est pas possible de définir des identificateurs locaux.

variable = expression	asMSX permet ici l'utilisation de variables entières. Les variables doivent êtres initialisées, puis leurs valeurs numériques affectées, pour pouvoir les utiliser dans des opérations. Par exemple, vous pouvez effectuer les opérations suivantes :
	Valeur1=0
	Valeur1=Valeur1+1
	Ld a,Valeur1
	Valeur1=Valeur1*2
	Ld b,valeur1

.BIOS	Prédéfinit les adresses des routines du Bios, incluant les spécificités des standards MSX, MSX2, MSX2+ et Turbo-R. On emploie les noms usuels en majuscules.
.ROM	Définit l'en-tête pour produire une ROM. Il est impératif d'indiquer d'abord la position, ce qui peut être fait avec la directive .PAGE. Il faut également indiquer à l'aide de la directive .START l'adresse de lancement du programme.
.MEGAROM[mapper]	Définit l'en-tête pour produire une mégaROM. Par défaut, définit aussi la sous-page 0 du mapper, si bien qu'il n'est pas nécessaire d'inclure toutes les instructions .ORG, ni .PAGE ou SUBPAGE après. Les types de mappers supportés sont les suivants :

- Konami : taille de sous-page de 8 Ko, limite de 32 pages. La taille maximale de la mégaROM sera de 256 Ko (2 Mégabits). La sous-page 0 est nécessairement entre 4000h et 5FFFh , on ne peut donc pas la changer.
- KonamiSCC : taille de sous-page de 8 Ko, limite de 64 pages. La taille maximale de la mégaROM sera de 512 Ko (4 Mégabits). Prend en charge l'accès au Sound Custom Chip (SCC)

de Konami.

- ASCII8 : taille de sous-page de 8 Ko, limite de 256 pages. La taille maximale de la mégaROM sera de 2048 Ko (16 Mégabits, 2 Mo)
- ASCII16 : taille de sous-page de 16 Ko, limite de 256 pages. La taille maximale de la mégaROM sera de 4096 Ko (32 Mégabits, 4 Mo)

.BASIC : génère l'en-tête pour produire un fichier binaire qu'on pourra charger sous MSX- Basic. Il est nécessaire d'indiquer avant une position initiale avec la directive .ORG, et au cas où l'adresse d'exécution du programme ne coïncide pas avec l'adresse de départ, utiliser également la directive .START. L'extension par défaut du fichier produit est .BIN

.MSXDOS produit comme résultat un fichier COM exécutable depuis MSX-DOS. Il n'est pas nécessaire d'utiliser l'instruction .ORG parce que l'intialisation commence toujours en 0100h

.START X Indique l'adresse de début d'exécution pour que les fichiers ROM, megaROM et BIN ne lancent pas leur exécution au début du fichier.

.SEARCH Pour les ROMs et megaROMs qui démarrent à la page 1 (4000h), se charge de chercher automatiquement le slot primaire et le slot secondaire de la page 2 correspondante (8000h). Equivaut à la portion de code suivant :

```
call 0138h ;RSLREG
rrca
rrca
and 03h
; Secondary Slot
ld c,a
ld hl,0FCC1h
add a,l
ld l,a
ld a,[hl]
and 80h
or c
ld c,a
inc l
inc l
inc l
inc l
ld a,[hl]
; Define slot ID
and 0ch
or c
ld h,80h
; Enable
call 0024h ;ENASLT
```

.SUBPAGE n AT X — Cette macro est utilisée pour définir les sous-pages distinctes d'une megaROM. Dans le modèle de génération de megaROMs d'asMSX, tous les codes et données doivent être inclus en sous-pages, qui équivalent à des blocs logiques d'opérations du mapper. Vous devez indiquer le nombre de sous-pages que vous désirez créer, et en quelle adresse logique l'assembler, et en quelle adresse l'exécuter.

.SELECT n AT X/.SELECT registre AT X — Comme réponse à la précédente, cette macro va sélectionner la sous-page n dans l'adresse X. Le code concret utilisé pour cette opération dépendra du mapper sélectionné. N'altère la valeur d'aucun registre, n'affecte pas le mode d'interruption, ni les flags d'état.

.PHASE X/.DEPHASE — Ces deux routines permettent d'utiliser des adresses virtuelles de mémoire. C'est-à-dire, assembler des instructions dans une position de mémoire avant de les placer dans une autre. Il peut être utile pour introduire du code dans une ROM qui se copiera et s'exécutera depuis la RAM. Cela a pour effet que les étiquettes s'assemblent en fonction de l'adresse indiquée.

.DEPHASE revient à la normale, même si les instructions ORG, PAGE et SUBPAGE aient le même effet.

.CALLBIOS — Appelle une routine Bios depuis MSX-DOS. Equivaut au bloc de code suivant :

```
LD IY,[EXPTBL-1]
LD IX,RUTINA
CALL CALSLT
```

.CALLDOS X — Exécute une des fonctions de MSX-DOS. Equivaut au code suivant :

```
LD C, CODIGO
CALL 0005h
```

.SIZE X — Etablit la taille du fichier résultant en Kilo-octets.

.INCLUDE « fichier » — Inclut du code source en provenance d'un fichier externe

.INCBIN « fichier » [SKIP X] [SIZE Y] — Injecte un fichier binaire externe au sein du programme. On peut utiliser en plus les modificateurs SKIP et SIZE qui permettent d'indiquer le nombre d'octets à supprimer depuis le début du fichier et le nombre d'octets à garder au total.

.RANDOM(n) Génère une valeur aléatoire entre 0 et n-1. Donne un résultat aléatoire meilleur qu'en utilisant directement le registre R du Z80.

.DEBUG « texte » Etablit un texte au sein du programme de sorte qu'il puisse être vu lors du débogage mais transparent lors de l'exécution. Le debogueur de BlueMSX supporte des fonctionnalités supplémentaires.

.BREAK [X] / .BREAKPOINT [X] Définit un point de break pour Le debogueur de BlueMSX. Son exécution est transparente, mais il est recommandé de le supprimer dans le fichier final. Si l'adresse n'est pas indiquée, le break aura lieu à la position où il a été défini

REPT n / ENDR Cette macro-instruction permet de répéter un certain nombre de fois un bloc de code. Facilite la génération automatique de tableaux complexes. La seule restriction est que le nombre de répétitions doit obligatoirement être un nombre décimal, pas une expression numérique. Comme dans cet exemple :

```
REPT 16
    OUTI
ENDR
X=0
Y=0
REPT 10
    REPT 10
     DB X*Y
     X=X+1
    ENDR
    Y=Y+1
ENDR
```

.PRINTTEXT « texte » / .PRINTSTRING « texte » Imprime un texte dans le fichier texte de sortie

.PRINT expression / .PRINTDEC expression Imprime une valeur numérique en format décimal

.PRINTHEX expression Imprime une valeur numérique en format hexadécimal

.PRINTFIX expression Imprime une valeur numérique en virgule fixe

.CAS [« texte »] / .CASSETTE [« texte »]	Génère un fichier CAS avec le résultat de l'assemblage. Formats valables uniquement : BASIC, ROM en page 2 et Z80 direct. Le texte permet d'indiquer un nom pour charger depuis la cassette, avec un maximum de 6 caractères de longueur. S'il n'est pas indiqué, le nom du fichier de sortie sera utilisé.
.WAV [« texte »]	Comme la commande précédente, mais au lieu de générer un fichier CAS, elle génère un fichier WAV audio, qu'on peut alors charger directement sur un vrai MSX à travers le port cassette.

2.6 Commentaires

Il est fortement recommandé d'utiliser des commentaires dans le code source en assembleur. asMSX vous permet d'entrer des commentaires dans différents formats :

- `;commentaire`

  Commentaire sur une seule ligne. Le point-virgule est la norme pour les commentaires dans un listing en assembleur. La ligne complète est prise comme commentaire jusqu'au passage à la ligne suivante.

- `//commentaire`

  Fonctionnement identique au précédent. Le double slash est employé pour les commentaires d'un listing en C/C++

- `/* commentaire */`
- `{ commentaire}`

  Commentaires en lignes multiples. Tout le texte encadré entre les deux délimiteurs est défini comme commentaire et ne sera pas assemblé.

2.7 Assemblage conditionnel

asMSX comporte un support préliminaire pour l'assemblage conditionnel. Le format d'un bloc d'assemblage conditionnel est le suivant :

```
IF condition
    Instructions
ELSE
    Instructions
ENDIF
```

La condition peut être de n'importe quel type, à condition de suivre les règles syntaxiques de C/C++ et Java. Une condition n'est vraie que si son résultat est différent de 0.

Si la condition est vraie, le code sera assemblé selon les instructions suivant le IF.

Si la condition n'est pas vraie, les instructions suivant le IF seront ignorées, et seul le bloc suivant le ELSE sera assemblé.

Le bloc ELSE est cependant facultatif, il n'est pas nécessaire d'en avoir un dans un bloc d'assemblage conditionnel, mais il ne faut pas oublier de fermer le bloc avec un ENDIF. Dans cette nouvelle version, il est possible d'imbriquer plusieurs IF sans limite de nombre, comme dans l'exemple suivant :

```
IF (ordinateurr==MSX)
    call MSX1_Init
ELSE
    IF (ordinateur==MSX2)
          call MSX2_Init
    ELSE
          IF (ordinateur==MSX2plus)
               call MSX2plus_Init
          ELSE
               call TurboR_Init
          ENDIF
    ENDIF
ENDIF
```

En outre, tous les codes, directives et macros seront exécutées selon les conditions, ce qui peut amener à créer des structures du type suivant :

```
CARTOUCHE=1
BINAIRE=2
format=CARTOUCHE
IF (format==CARTOUCHE)
    .page 2
    .ROM
ELSE
    .org 8800h
    .Basic
ENDIF
.START Init
```

L'unique limitation est que les « phrases » IF <condition> ELSE ENDIF doivent apparaître en lignes séparées, on ne peut pas les regrouper avec d'autres instructions directives ou macros sur une seule ligne de code. Le code suivant produirait une erreur à l'assemblage :

```
IF (variable) nop ELSE inc A ENDIF
```

On doit l'écrire ainsi :

```
IF (variable)
    nop
ELSE
    inc A
ENDIF
```

En supplément, il est inclus un nouveau type de condition, IFDEF, qui ne vérifie pas la valeur de l'expression utilisée, mais si la dite étiquette a été définie ou non. Par exemple, le bloc de code suivant :

```
IFDEF preuves
    .WAV « preuves »
ENDIF
```

Ce fragment de code donnera un fichier WAV si et seulement si l'étiquette « preuves » a été définie au préalable dans le code source. Notez qu'IFDEF ne reconnaîtra une étiquette que si elle a été définie avant la « phrase » IFDEF.

3 Autres outils

3.1 RLEpack

RLEpack est un compresseur simple de données, utilisé par le système très connu RLE (Run-Length Encode).

La compression linéaire est inefficace et peu recommandée lorsqu'il s'agit de données qui varient beaucoup, mais s'il y a beaucoup de données homogènes, ce peut être une solution à considérer. Bien que le résultat de la compression soit très variable, son avantage est que la méthode est très simple et facile à comprendre.

Pour obtenir une compression bien plus efficace, il est recommandé d'utiliser BitBuster, le compresseur développée par la TEAM BOMBA.

L'idée de la compression linéaire est très simple : si on trouve une série de données égales, on les compte et on indique combien d'octets consécutifs ont cette même valeur. Le code source de l'algorithme de compression est inclus dans 2 formats distincts : l'un pour décompresser le format RLE vers la RAM, l'autre pour décompresser vers la VRAM, sans repasser par la RAM principale.

Pour compresser des fichiers binaires, il suffit de passer le nom du fichier comme paramètre unique de RLEpack. L'application engendrera des fichiers avec le même nom, suivi de l'extension .RLE (run-length encode). En plus, la taille des fichiers d'origines et des fichiers compressés ainsi que le ratio de compression seront indiqués.

Les routines de décompression sont très simples et se listent comme suit :

```
DEPACK_RAM:
; Décompresse les données compressées avec RLEpack de RAM a
RAM
; Paramètres: HL=adresse des données compressées
;DE=adresse où sont envoyées les données
@@LOOP:
ld a,[hl]
inc hl
cp 0C0h
jr c,@@SINGLE
and 3Fh
inc a
ld b,a
ld a,[hl]
inc hl
@@DUMP:
ld [de],a
inc de
djnz @@DUMP
jr @@LOOP
@@SINGLE:
ld c,a
or [hl]
ld a,c
ret z
ld b,1
jr @@DUMP
```

```
DEPACK_VRAM:
; Décompresse les données compressées avec RLEpack de RAM a
VRAM
; Paramètres: HL=adresse des données compressées
;DE=adresse où sont envoyées les données
; Note: il convient de l'exécuter juste après le tracé
ex de,hl
call SETWRT
ex de,hl
@@LOOP:
ld a,[hl]
inc hl
cp 0C0h
jr c,@@SINGLE
and 3Fh
inc a
ld b,a
ld a,[hl]
inc hl
@@DUMP:
out [98h],a
djnz @@DUMP
jr @@LOOP
@@SINGLE:
ld c,a
or [hl]
ld a,c
ret z
ld b,1
jr @@DUMP
```

Les exemples de code inclus avec cette distribution (voir annexes) utilisent ces routines pour les graphiques, qui ont été changés au préalable avec PCX2MSX, compressés avec RLEpack et ajoutés au listing du code source au moyen de binDB.

Pour compresser un fichier il suffit de passer le nom du fichier comme de paramètre au programme RLEpack, de la manière suivante :

```
RLEpack fichier.ext
```

Le fichier compressé fichier.ext.rle sera alors généré.

3.2 binDB

BinDB est une petite application qui convertit des fichiers binaires en un listing d'assembleur sous la forme de données en octets, introduites par la directive DB.

Bien que la directive INCBIN permette d'inclure des fichiers binaires à l'intérieur du code source d'un programme en assembleur, elle oblige à introduire intégralement le fichier, et les modifications sont difficiles, et l'assemblage dépend de plusieurs fichiers.

Pour changer un fichier binaire en une suite de données, il suffit de passer le nom du fichier comme paramètre au programme binDB, de la manière suivante :

```
binDB fichier.ext
```

Il en résultera un nouveau fichier, fichier.ext.asm, contenant le binaire comme une liste d'hexadécimaux en format DB, avec 8 bits par ligne, très similaires à cet exemple :

```
; fichier.ext - binary dump
; generated by binDB v.0.10
db 000h,000h,000h,000h,000h,000h,000h,000h
db 000h,000h,000h,000h,000h,000h,000h,000h
...
```

3.3 PCX2MSX

Ce programme réalise la conversion d'un fichier graphique PCX en deux fichiers séparés, un contenant l'information sur les patrons, et l'autre avec l'information des couleurs, nécessaires pour utiliser ces graphiques dans le mode graphique haute résolution du processeur TMS9918/9928 et compatibles.

Comme les outils précédents, ce programme s'utilise en passant le nom du fichier comme paramètre unique en ligne de commande. Par exemple, pour convertir le fichier EXEMPLE.PCX, on doit exécuter la commande suivante :

```
PCX2MSX EXEMPLE.PCX
```

Et on aura comme résultat deux nouveaux fichiers : EXEMPLE.PCX.CHR et EXEMPLE.PCX.CLR, contenant respectivement les données des patrons et les données des couleurs.

L'application ne réalise pas l'ajustement graphique des attributs, et ne convertira que les fichiers qui s'acquittent de la limitation de couleurs du SCREEN 2, c'est à dire 2 couleurs par bloc de 8 bits horizontaux. Dans le cas contraire, s'il y a un problème d'attributs, le programme le détectera et indiquera la position concrète où l'erreur a été localisée. Le message d'erreur serait alors le suivant :

```
Color collision at line (X,Y)
```

Notez que la conversion ne tiendra pas compte de la palette de couleur définie dans le fichier PCX, mais simplement de l'ordre. Ce qui signifie que lors de la conversion finale, l'ordre des couleurs utilisées sera celui de la palette du TMS 9918/9928.

En ce qui concerne les fichiers PCX, ils doivent être dans un format 8 bits, soit 256 couleurs et avec compression RLE, ce qui est plus habituel. Tant qu'on a ou pas une information sur les couleurs, la conversion sera faite sur la base de l'ordre des couleurs.

Note : quelques programmes modernes d'édition graphique supportent le format PCX, mais d'une forme peu orthodoxe, en inversant l'ordre normal des couleurs. Pour cela, en plus de la version normale du programme PCX2MSX, il y a une version alternative PCX2MSXi qui permet la conversion de fichiers PCX avec une palette inversée.

Un fichier PCX d'exemple est inclus (nommé SAMPLE.PCX), avec les 16 couleurs du processeur vidéo TMS 9918/9928 prédéfinies (calibrage approximatif) et avec la palette inversée.

4 Codes source

4.1 Classic Pong v.0.15

Nous avons ici une version pour MSX de l'un des jeux électroniques les plus anciens : le PONG, dont la version arcade a été initialement produite par Atari Games. Il s'agit d'une version très simplifiée, exclusivement pour 2 joueurs, mais qui donne un exemple suffisamment intéressant pour pouvoir explorer les capacités d'asMSX et voir comment un jeu complet est structuré dans un assembleur pour MSX.

Les instructions de jeu sont très simples : le premier joueur qui parvient à marquer 10 points gagne la partie. Le service est aléatoire et automatique, et chaque joueur contrôle sa raquette. Celle de gauche, du joueur 1, avec les flèches du clavier ou le joystick dans le port 1. La raquette de droite, du joueur 2, exclusivement avec le joystick dans le port 2.

Les techniques utilisées dans ce programme sont très simples et le code est truffé de commentaires et d'indications. L'idée de publier des jeux complets avec leur code source répond au souhait que les programmeurs débutants puissent accéder facilement au monde de l'assembleur pour MSX et puissent modifier le code à leur guise et y voir rapidement des résultats.

Comme programmeur original de ce jeu, je crois qu'effectivement, quelques améliorations évidentes peuvent être faites, les programmeurs intéressés peuvent faire des exercices comme

compléter le son, utiliser d'autres couleurs (cette version respecte l'original en noir et blanc), l'option de jouer contre l'ordinateur, que celui-ci ait des niveaux distincts d'intelligence, etc.

L'unique condition d'utilisation du code source est que toute version modifiée du jeu soit aussi distribuée librement et avec le code source de la version originale. Les versions qui auraient un niveau de qualité suffisant et qui peuvent amener encore d'autres améliorations sur celles-ci pourraient être incluses dans des distributions futures d'asMSX ou à la page officielle de distribution d'asMSX, si l'auteur des modifications ou extensions en donne l'accord.

4.2 Classic Minesweeper v.0.10

Un autre jeu classique, dont il n'y avait jamais eu de version pour MSX de première génération, vient grossir la liste des programmes disponibles pour le standard. De plus, il est à nouveau fait en open-source, le programme est prêt à être étendu, modifié ou amélioré par n'importe quel utilisateur, en respectant toujours les conditions pour cela : le résultat doit continuer d'être en distribution libre, et le nouveau code doit toujours être accompagné du code de la version originale. Est-il besoin de présenter encore ce jeu, qui est sans doute l'un des plus installés du monde puisqu'il accompagne toutes les versions du système d'exploitation Microsoft Windows, et a été en plus décliné pour les autres systèmes d'exploitation des PC ?

Pour la rendre compatible avec les configurations de MSX usuelles, cette version du démineur est contrôlée au moyen du clavier ou du joystick, les MSX étant dépourvus de souris. Les contrôles sont simples : on utilise les touches directionnelles ou le levier du joystick du port 1. L'équivalent du clic gauche de la souris dans la version PC est ici la barre d'espace ou le bouton 1 du joystick, l'effet est de découvrir la case pointée par le curseur. Au cas où c'est une mine, la partie se termine.

La fonction du bouton droit de la souris a été remplacée ici par la touche CTRL du clavier ou le bouton 2 du joystick, et sa fonction est de marquer la case pointée avec un drapeau, de la marquer d'un point d'interrogation, ou d'éliminer toutes les marques antérieures. Le jeu se termine avec succès quand toutes les mines du champ on été correctement balisées avec les drapeaux.

Dans la version PC, on peut faire un clic simultané avec les boutons droit et gauche de la souris sur les cellules avec un nombre dès que tous les drapeaux nécessaires autour de lui sont en place, ce qui a pour effet de faire avancer le joueur plus vite. Cette fonction est incluse de la même façon dans cette version du démineur, en appuyant sur les 2 boutons du joystick ou sur CTRL et espace simultanément.

Dans le menu principal, on peut choisir le niveau de difficulté, en choisissant de 50 à 175 mines, pour un champ de jeu de grandeur fixe, 30 cases de largeur sur 20 de hauteur. Naturellement, le niveau facile est VRAIMENT facile, alors que celui qui est dit impossible ne l'est pas réellement. En plus de savoir jouer et de montrer une esprit vif, il faudra également un peu de chance dans les niveaux plus avancés du jeu.

En ce qui concerne le jeu, il est complet et fonctionnel, il a quelques détails intéressants, tant pour le traitement graphique que pour les techniques de programmation en général (procédés récursifs, registre d'indice, génération de nombres pseudo-aléatoires, etc). Cependant, il peut toujours être un exercice pour les programmeurs qui ont de l'initiative et faim d'apprendre : l'aspect graphique est perfectible, l'aspect sonore est à travailler, on peut même penser à de nouvelles manières de jouer ou un système plus commode de menus et de configuration... Les possibilités sont presque infinies : elles dépendent de ce que chacun croît qu'il peut apporter à un tel jeu. Pour les détails de la permission d'utilisation de ce programme, référez-vous à la

permission d'utilisation d'asMSX et des commentaires inclus dans le paragraphe précédent concernant Pong v.0.15.

4.3 Autres exemples

D'autres exemples de codes sources sont disponibles, relativement plus simples, qui ont été inclus pour détailler certaines capacités d'asMSX, ou comme démonstrations d'une fonctionnalité additionnelle.

- OPCODES.ASM : exemple en assembleur qui inclut toutes les instructions documentées ou non-documentées du processeur Z80, avec les syntaxes officielles de Zilog et les alternatives (accumulateur implicite, syntaxe normalisée,etc).
- DATA.ASM : exemple en assembleur qui couvre les différents formats de données supportés par asMSX, numériques ou alphanumériques.
- MEGAROM.ASM : un exemple d'utilisation des nouvelles directives d'asMSX pour créer des mégaROMs facilement.
- FIXED.ASM : simple démonstration de l'utilisation de l'arithmétique en virgule fixe et des fonctions mathématiques incluses dans asMSX, avec des mouvements complexes de sprites multicolores.
- TV.ASM : petite démonstration dans laquelle apparaît l'image d'une télé non-syntonisée, avec la neige caractéristique, et sans répétition de patrons.

4.4 Plus d'exemples

Si quelqu'un a développé un programme avec asMSX qu'il considère comme intéressant et pédagogique, il est invité à l'adresser à la page de MSX Workshop au moyen de l'adresse mail suivante : asMSX@robsy.net

L'autre option consiste à l'envoyer directement aux forums de Karoshi Corporation, http://forum.karoshicorp.com , où tous les développeurs sont les bienvenus.

5 Renseignements supplémentaires sur asMSX

5.1 Nouveautés de la version 0.12g

- Correction d'une erreur associée à l'usage d'étiquettes locales après avoir utilisé l'instruction PHASE
- Support préliminaire pour cassette avec les directives CAS et WAV - génération de fichiers CAS (émulateurs) et WAV (MSX réels)
- Amélioration de l'assemblage conditionnel avec IFDEF

5.2 Nouveautés de la version 0.12f

- Correction d'une erreur associée au changement de version qui affectait les opérateurs binaires ^(XOR) et | (OR)
- Support préliminaire de l'assemblage conditionnel

5.3 Nouveautés de la version 0.12e

Après avoir perdu une le code source de la version 0.11, une grande partie du code a dû être reprogrammée. Voici les nouveautés spécifiques de cette version :

- Support natif pour megaROMs : MEGAROM, SUBPAGE et SELECT
- Répétitions avec la macro REPT n/ENDR
- Paramètres SKIP et SIZE pour l'instruction INCBIN
- Adaptation pour l'émulateur BlueMSX : DEBUG et BREAKPOINT
- Défintion du symbole $; équivalent à la valeur du registre PC
- Utilisation de variables numériques au niveau du code
- Génération de nombres aléatoires avec RANDOM(n)
- Localisation automatique de la seconde page de ROM de 32 Ko avec SEARCH

Comme la version est expérimentale, des problèmes d'instabilité sont possibles. Veuillez rapporter tout problème à asMSX@robsy.net

5.4 Nouveautés de la version 0.11

Peu de changements par rapport à la version 0.10 qui était très stable. Il s'agit d'une version de maintenance. La liste complète des changements est la suivante :

- Support pour les déplacements négatifs dans les instructions qui emploient les registres d'indice 16 bits IX et IY
- Normalisation du format des fichiers de symboles et de texte, passant de la syntaxe de Borland à la standard pour assembleurs respectant les nombres en format hexadécimal.
- Nouveaux outils inclus dans la distribution de l'assembleur
- Code source complet de jeux MSX commentés en profondeur
- Documentation corrigée et étoffée

5.5 Nouveautés de la version 0.10

Le code de la version 0.10 a été réécrit, parce que les changements sur la première version publiée, 0.01, sont innombrables et qu'il n'y a pas besoin d'inclure une liste exhaustive de ces changements. Comme caractéristiques propres, le support de la virgule fixe et du format de sortie ROM ont été inclus.

5.6 Licence d'utilisation

Ce programme est déclaré comme logiciel libre de copie et de distribution, chaque fois que celle-ci est gratuite. Pour sa vente ou son inclusion dans les collections de programmes qui sont commercialisés, il est indispensable d'obtenir l'autorisation de l'auteur. Dans tous les cas, l'exécutable doit être distribué avec la documentation et les exemples qui l'accompagnent.

AsMSX peut être utilisé pour tout type de projet, commercial ou de libre distribution, sans autre obligation qu'accréditer dans la documentation que cet assembleur a été utilisé et où on peut obtenir la version la plus récente. Les outils inclus avec asMSX restent soumis aux termes d'usage de la permission principale.

5.7 Mentions légales

L'auteur récuse toute responsabilité dérivée de l'usage ou du fonctionnement de ce programme. L'utilisation de ce programme ne sous-entend pas l'établissement d'aucun type de relation contractuelle entre l'auteur et l'utilisateur final. Aucune responsabilité ne sera acceptée à ce sujet.

MSX est une marque inscrite à la MSX Association au Japon.

6 Ressources disponibles

6.1 Sites web intéressants

- **MSX Info Pages** : http://www.hansotten.com : page personnelle de Hans Otten, *HansO*, où vous trouverez la plus grande collection de documentation technique, originaux scannés, des schémas, des codes sources en assembleur et un bon échantillon des autres assembleurs existants pour MSX.
- **Sean Young's MSX pages** : http://www.msxnet.org : page de Sean Young, l'homme sans lequel cet assembleur serait incomplet. Ses documentations sur le Z80 sont les plus complètes disponibles. En plus, vous pourrez trouver de la documentation sur les puces du MSX et des articles sur les megaROMs, la SCC et le FM-PAC.
- **MSX Asembler Page (MAP)** : http://map.tni.nl : une page orientée exclusivement sur l'assembleur pour MSX. Vous y trouverez une grande quantité d'informations techniques sur le hardware du MSX et des articles sur les techniques de programmation. Indispensable.
- **FUNET** : ftp://nic.funet.fi [login anonyme] : Ici vous trouverez tout (ou presque). Une multitude de documentations techniques scannées, des programmes avec code source, des utilitaires additionnels, etc.
- **Robsy's MSX Workshop** : http://www.robsy.net : l'antique page officielle d'asMSX, où vous pourrez télécharger des exemples de code et d'autres documentations exclusives.
- **MSX Banzai !** : http://banzai.tni.nl : la page web de Patriek Lesparre, plus connu sous le nom de GuyveR800. Des informations intéressantes sur le MSX en général et une brève FAQ avec des techniques de programmation très intéressantes.

6.2 Autres ressources sur Internet

En plus des pages web dédiées concrètement au développement sur le standard MSX, il existe de nombreuses autres ressources disponibles pour le MSX et qui peuvent intéresser ou aider les utilisateurs d'asMSX :

- **MSX Resource Center** : http://www/msx.org : le point de rencontre des utilisateurs du MSX au niveau mondial. En plus d'être une source inépuisable de nouvelles, et très fréquemment mis à jour, c'est le plus actif des forums MSX. Beaucoup de bons programmeurs consultent régulièrement la page, et peuvent ainsi résoudre vos problèmes de programmation ou de toute autre question relative au MSX. Les pages sont disponibles en anglais, néerlandais, portugais et espagnol. Le forum anglais est le plus international et le plus fréquenté.
- **MSX Posse** : http://www.msxposse.com : un autre forum alternatif sur le monde du MSX. Bien qu'il ne soit pas aussi grand que le MRC, il rassemble déjà un grand nombre de développeurs et de participants. De plus, ses politiques sur la participation aux forums sont plus « détendues ». Ce forum est disponible en néerlandais et en anglais.
- **Foros de Karoshi Corporation** : http://forum.karoshicorp.com : un forum centré à la base sur le développement de jeux vidéo, avec des discussions techniques très intéressantes de codage en assembleur.
- **HispaMSX** : http://www.hispamsx.org : un forum pour la communauté espagnole du MSX. Sert de support à la liste de diffusion HispaMSX, le point de rencontre de tous les utilisateurs parlant espagnol. La liste de diffusion est hébergée sur YahooGroups, et on peut y accéder via l'adresse : http://hispamsx.yahoogroups.com
- **MSX Café** : http://www.msxcafe.com : le bistrot des fous du MSX, site de la communauté française, comportant un bon lot de documentation technique et un forum où on peut avoir des dialogues intéressants avec des pointures en hardware ou en software.

6.3 Bibliographie recommandée

Sur Internet, on peut trouver tout le nécessaire pour apprendre à programmer le Z80, et par conséquent le MSX. Il existe aussi des livres excellents sur tous ces aspects, dans lesquels la liste suivante, simplement indispensable :

- **Z80 Assembler, Rodney Zaks**. C'est la bible du Z80, avec d'excellentes références (maintenant un peu dépassé, dû aux investigation de Sean Young) et avec des techniques de programmation de base et avancées dûment commentées. Il existe une édition en espagnol publiée par Anaya. Il existe d'autres livres sur le Z80, mais celui-ci est de loin le meilleur.
- **MSX Red Book, Avalon Software**. Un condensé superbe sur les MSX de première génération, avec des informations détaillées sur le hardware du MSX et son Bios. Il ne traite pas du Z80, dont la programmation est supposée connue. Il existe une version numérique de celui-ci que vous pouvez consulter ici : http://www.robsy.net/tredbook.txt
- **MSX Technical Databook, ASCII**. Un livre difficile à trouver et qui, après un gros travail de recherche et d'édition, est enfin disponible sur Internet. Il contient la définition du standard MSX (première génération), traite autant du système de base que du système étendu, de l'unité de disque, du port série, etc. Peut être téléchargé ici : http://www.robsy.net/msxtech.pdf

- **MSX Technical Book, ASCII**. La bible du MSX2. C'est aussi un livre difficile à trouver, mais grâce au travail de Nestor Soriano, *Konamiman*, c'est possible. Vous pouvez le télécharger ici : http://www.konamiman.com

En outre, il est recommandé d'obtenir les documentations techniques sur les différents processeurs programmables : VDP, PSG, PPI, FM-PAC, SCC, etc. On peut donc trouver :

- **Z80 undocumented documented**. Le document le plus complet sur le fonctionnement interne du Z80, encore meilleur que la documentation originale publiée par Zilog. Un travail de maître en ingénierie inverse fait par Sean Young. Vous pouvez le télécharger ici : http://www.robsy.net/z80undoc.pdf
- **Z80 family : CPU user manual**. C'est le manuel original de Zilog, avec des informations significatives sur le Z80, tant au niveau électronique que logique. Exemples et commentaires aux différentes instructions sont très intéressantes, encore qu'en cas de doute, le document de Sean Young soit plus fiable. Ce document est disponible sur le site de Zilog http://www.zilog.com ou plus facile à trouver sur la page de téléchargement de Rosby's Workshop : http://www.robsy.net/um0080.pdf
- **Texas Instruments TMS9918A/TMS9928A/TMS9929A Video Display Processors**. Les informations originales de Texas Instruments pour les processeurs vidéo utilisés sur les ordinateurs MSX de première génération. Il couvre avec beaucoup de détails une multitude d'aspects sur le moteur graphique commun aux MSX et à d'autres ordinateurs et consoles.

6.4 Autres assembleurs

Voici une liste non exhaustive d'autres assembleurs pour Z80. La première partie comprend des cross-assembleurs, l'autre des assembleurs natifs MSX. Il en existe beaucoup plus, ceux cités ici sont ceux qu'on retrouve fréquemment.

- **TNIasm - The New Image - Patriek Lesparre** : un cross-assembleur complet pour Z80/R800/GBZ80
- **SjAsm - XL2s - Sjoerd** : autre cross-assembleur qui s'actualise très souvent
- **Chaos Assembler 3 - Teddywarez** : environnement de développement complet basé sur l'assembleur TASM (*Table Assembler*)
- **TASM - Table Assembler** : un cross-assembleur avec la possibilité de produire du code pour une multitude de processeurs distincts. Ne doit pas être confondu avec le *Turbo Assembler* de *Borland*, même s'ils portent le même nom.
- **A22I - nocash - Martin Korth** : le cross-assembleur le plus compact qu'on puisse rencontrer, seulement 5 Ko? très très rapide.
- **DEVPAC - Hisoft** : un package très complet qui inclut un assembleur (GEN), un moniteur (MON) et un éditeur pour la version disque. Il existe une version pour disque, CP/M, et d'autres pour charger depuis une bande.
- **M80/L80 - Microsoft** : un assembleur très usuel, comprenant un assembleur et un linker.
- **ZEN/CHAMP** : Idem
- **RSC/RSC-II** - Ramon Salas : un assembleur très populaire en Espagne, qui fut distribué par les disparus MSX Club et Manhattan Transfer. A la différence des précédents, il incorpore son propre éditeur et ne requiert ni MSX-DOS ni CP/M

- **WBASS2 - Door Wilbert Berendsen** : un assembleur de bonne réputation, et des plus actualisés pour MSX
- **COMPASS - Compjoetania** : d'après beaucoup de personnes, le meilleur assembleur pour MSX. Une nouvelle version a été annoncée, mais on n'en voit pas le bout.
- **AS/LD - Egor Voznessenski** : un chef-d'œuvre en termes de programmation. Parmi les meilleurs assembleurs disponibles pour MSX-DOS. Vous aurez besoin d'une machine rapide pour que les processus d'assemblage ne durent pas trop longtemps.

6.5 Conseil gratuit

Apprendre à programmer en assembleur, malgré les apparences, n'est pas difficile. La seule chose requise est le temps et la patience en proportions égales. Les supports pour apprendre sont disponibles sur Internet et les programmes nécessaires également. La seule chose vraiment unique et indispensable est votre enthousiasme.

7 Distribution

La distribution actuelle de asMSX V.0.12g doit inclure les fichiers suivants :

- *Assembleur et documentation*
  - `asMSX.exe` - cross-assembleur
  - `asMSX.pdf` - manuel d'utilisation
- *Programmes d'exemples : répertoire EXAMPLES*
  - `opcodes.asm` - toutes les instructions du Z80, essai d'assemblage
  - `data.asm` - exemple d'introduction de constantes, essai d'assemblage
  - `fixed.asm` - exemple basique d'utilisation d'arithmétique en virgule fixe
  - `tv.asm` - effet de télévision sans syntonisation
  - `unpack.asm` - routine de décompression du format RLE
  - `pong.asm` - jeu simple complet en format ROM, avec code commenté
  - `mine.asm` - une version pour MSX du démineur classique en format ROM
  - `megarom.asm` - un exemple de création d'une mégaROM simple. Important : pour charger la ROM sur BlueMSX, il est indispensable de notifier manuellement le type de mapper : ASCII 8 KB
- *Outils supplémentaires : répertoire TOOLS*
  - `RLEpack.exe` - compresseur de fichiers utilisant un algorithme linéaire
  - `binDB.exe` - convertit les fichiers binaires en listing assembleur comprenant des DB.
  - `PCX2MSX` - convertit des fichiers graphiques en format PCX en SCREEN 2

Tous ces outils comportent leur propre documentation jointe, comprise dans les appendices de cette documentation.