

# Pratique du



par GDX

Version 2018.12 (pré-version)

Basé sur le livre « Pratique du MSX2 »

de Eric Von Ascheberg

(numérisé par Metalion et Granced pour la communauté MSX)

Informations complémentaires tirées du

MSX Datapack vol. 1 à 3 de ASCII,

MSX-Turbo R Technical Hand Book

et de

diverses informations trouvées sur Internet.

# Index

<b>1 Le standard MSX.....</b>	<b>4</b>
1.1 Introduction.....	4
1.2 Utiliser ce document.....	5
1.3 Configurations minimales.....	6
1.4 Conseils au développeur de logiciels.....	8
1.5 Et maintenant.....	9
<b>2 Les Slot et le Memory Mapper.....</b>	<b>10</b>
2.1 Comment dépasser la limite des 64 Ko.....	11
2.2 Qu'est-ce qu'un Slot ?.....	11
2.3 Utiliser les Slot.....	14
2.4 Le Memory Mapper.....	17
<b>3 Le Bios (Basic Input/Output System).....</b>	<b>19</b>
3.1 Introduction au Bios.....	19
3.2 Table des sauts du Bios en Main-ROM.....	19
Routines « Math-pack ».....	68
3.3 Table des sauts du Bios de la Sub-ROM.....	80
3.4 Table des sauts du Bios de la Disk-ROM.....	112
3.5 Le Bios du MSX-Music (FM Bios).....	118
Liste des routines du Bios du MSX-Music.....	118
Format des données pour les musiques.....	120
Format des données pour la mélodie.....	121
Format des données pour les sons des instruments.....	122
3.6 Le Bios étendu.....	123
<b>4 Les interruptions.....</b>	<b>140</b>
4.1 Mode 0.....	140
4.2 Mode 1.....	140
4.3 Mode 2.....	140
<b>5 Les variables et zones de travail du système.....</b>	<b>141</b>
5.1 Introduction aux variables système.....	141
5.2 La liste des variables et des zones de travail système.....	141
5.3 Quelques exemples d'utilisation des variables système.....	156
<b>6 Les Hooks du système.....</b>	<b>158</b>
6.1 La liste des Hooks.....	159
<b>7 Le processeur vidéo (VDP).....</b>	<b>175</b>
7.1 Avertissement.....	175
7.2 Introduction au processeur vidéo.....	175
7.3 Fonctionnement du VDP.....	175
7.4 Comment accéder au VDP.....	176
Ecrire dans un registre de contrôle (0 ~ 23, 25 et 32 ~ 46).....	177
Ecrire dans un registre de la palette des couleurs.....	179
Lire un registre de statut du VDP.....	180
7.5 Lecture et écriture dans la mémoire vidéo.....	181
7.6 Les registres de contrôle du processeur video.....	182
7.7 Les registres de statut du VDP.....	198
7.8 Les commandes internes du V9938 et V9958.....	200
HMMC (High speed Move to Memory from CPU).....	203
YMMM (Y Move to Memory from Memory).....	205
HMMM (High Speed Move to Memory from Memory).....	207
HMMV (High speed Move to Memory from VDP).....	209
LMMC (Logical Move to Memory from CPU).....	211
LMCM (Logical Move to CPU from Memory).....	213
LMMM (Logical Move to Memory from Memory).....	215
LMMV (Logical Move to Memory from VDP).....	217
LINE (draw a LINE).....	219
SRCH (SeaRCH for color).....	221
PSET (Point SET).....	223
POINT (is POINT set?).....	225
7.9 Les différents modes d'affichage (SCREEN 0 à SCREEN 12).....	226
7.10 Les Sprites et leur fonctionnement.....	245

7.11 A propos de la souris et du crayon optique.....	255
<b>8 Le processeur sonore PSG.....</b>	<b>256</b>
8.1 Caractéristiques.....	256
8.2 Descriptions du PSG et accès aux registres.....	256
8.3 Registres de contrôle de fréquence.....	257
8.4 Registre de contrôle de la fréquence de bruit blanc.....	259
8.5 Registre de contrôle des voix et des ports d'E/S du PSG.....	259
8.6 Registres de contrôle de l'amplitude et du volume.....	259
8.7 Registres de contrôle la forme et de la période de l'enveloppe.....	260
8.8 Registres des ports d'entrées / sorties du PSG.....	261
<b>9 Le MSX-Music.....</b>	<b>270</b>
9.1 Caractéristiques.....	270
9.2 Registres de l'OPLL.....	271
<b>10 MSX-JE.....</b>	<b>273</b>
<b>11 Le système des disques et MSX-DOS.....</b>	<b>274</b>
11.1 Base du MSX-DOS (System Scratch Area).....	275
11.2 Zone fixe de travail, Hooks et des variables du MSX-DOS1 et du Disk-BASIC.....	276
<b>12 Le système des disques et MSX-DOS 2.....</b>	<b>283</b>
12.1 Base du MSX-DOS 2 (System Scratch Area).....	284
12.2 Zone fixe de travail, Hooks et des variables du MSX-DOS2 et du Disk-BASIC 2.....	286
<b>13 Applications types.....</b>	<b>293</b>
13.1 Revenir à l'interpréteur Basic.....	293
13.2 Quel type de MSX ?.....	293
13.3 Le « PRINT » en assembleur.....	294
13.4 Système a disquettes ou pas ?.....	294
13.5 Traiter les erreurs liées aux disquettes.....	295
13.6 Faire de la musique en assembleur.....	295
13.7 Passage de paramètres du Basic au langage machine.....	296
13.8 Ajouter une instruction au Basic avec CMD ou IPL.....	299
13.9 Les codes de <i>contrôle</i> [CTRL].....	303
13.10 Les codes escape [ESC].....	303
13.11 Utiliser le second jeu de caractères.....	305
13.12 Détourner le Reset.....	309
13.13 Ajouter des mots clés au Basic.....	310
13.14 Manipuler la souris.....	313
13.15 Développer un programme en cartouche.....	314
13.16 Trouver le Slot de la RAM depuis une ROM.....	319
<b>Annexes.....</b>	<b>324</b>
A – Liste des labels par ordre alphabétique.....	324
B – Les registres du VDP.....	333
C – Les cartes de la mémoire vidéo.....	337
D – Jeux de caractères MSX.....	341
E – Exemples de matrices de clavier.....	345
F – Codes d'erreur du Basic et du disque Basic.....	348
G – Les ports d'entrée/sortie.....	350
<b>Lexique.....</b>	<b>357</b>
- Bruit blanc.....	357
- Main-RAM.....	357
- Mapper.....	357
- Megarom.....	357
- MSX-Engine.....	357
- PPI (Programmable Port Interface).....	357
- PSG (Programmable Sound Generator).....	357
- RAM (Random Access Memory).....	357
- ROM (Read Only Memory).....	357
- Sprite.....	357
- Slot.....	358
- VRAM.....	358
- VDP (Video Display processor).....	358

# 1 Le standard MSX

## 1.1 Introduction

Le standard MSX n'a pas connu le succès escompté en Europe. Si l'on pouvait faire des reproches justifiés à la première version (prix trop élevé des premiers modèles, mémoire vive un peu juste, etc), la version 2 était un ordinateur assez puissant et convivial. Il a cependant souffert de concurrence avec les ordinateurs 16 bit qui apparurent presque aussi tôt.

En effet, le MSX 2 représentait, en toute simplicité, l'ordinateur 8 bit familial le plus puissant jamais construit ! Bien des machines professionnelles de l'époque ne possèdent pas les caractéristiques du MSX2. Citons, pour mémoire, les principales du modèle le plus vendu :

- 256 Ko de mémoire vive extensible à 4 Mo par Slot libre
- 128 Ko de mémoire vidéo extensible à 192 Ko
- 64 Ko de mémoire morte comprenant notamment un Basic Microsoft très évolué
- 2 Slot d'extension, possibilité de passer à 8 Slot
- lecteur de disquette 1 Mo (720 Ko formatée) intégré
- gestion de deux lecteurs de disquette simultanément (jusqu'à huit possible)
- compatibilité totale IBM PC au niveau des fichiers (FAT12)
- horloge interne alimentée par pile
- mémoire CMOS comprenant un système de mots de passe
- processeur graphique VLSI
- résolution 256 sur 212 pixels en 256 couleurs simultanément
- souris
- deux systèmes d'exploitation de disquettes (dont un possédant l'interface menus déroulants/icônes)
- générateur sonore sur trois voix, huit octaves
- 700 logiciels disponibles, dont une trentaine spécifique MSX2
- de très nombreux périphériques

Par la suite, le MSX2+ est sortie au Japon avec quelques évolutions mais tous les modèles proposés ont moins de mémoire que la plupart des MSX2. Puis le MSX turbo R dont la principale évolution est le premier pas franchi vers un CPU 16 bit en interne.

Il va de soi que la maîtrise du MSX ne se limite pas à une bonne connaissance du Z80 (micro-processeur qui équipe tous les MSX). Il est nécessaire, comme sur tous les ordinateurs, de pouvoir mettre en œuvre tous les composants du système. De plus, le programmeur qui travaille sur MSX doit veiller à maintenir la compatibilité, y compris avec les futurs modèles. C'est le but de ce livre qui se propose de vous guider à travers l'exploration de la programmation du système MSX.

## 1.2 Utiliser ce document

Le livre que vous tenez entre vos mains a été conçu dans une double optique. Il aspire, dans un premier temps, à apprendre à tout programmeur ayant une bonne connaissance du Basic et des notions sérieuses en Z80, à tirer le maximum de son ordinateur, qu'il s'agisse d'un MSX1 ou plus récent. A ce propos, il est précisé à chaque fois que cela est nécessaire si les explications s'adressent à une version particulière de MSX ou à différentes versions. Puis, une fois les notions assimilées, le présent ouvrage a la prétention de servir de manuel de référence complétant ainsi le manuel d'utilisation fourni avec votre machine et ce, même pour les versions de MSX qui ne sont jamais sortis en France.

### Comprendre le fonctionnement :

Pour chaque fonction, instruction, astuce, j'ai tenté de donner les explications les plus simples possibles. J'ai évité d'employer l'équivalent français de termes anglais lorsque ces derniers étaient usuels (je parle de « bitmap » plutôt que d'image matricielle, mais j'emploie le mot « octet » et non « byte »). Lorsqu'une traduction paraissait hasardeuse, j'ai toujours ajouté le terme américain entre parenthèses.

De plus, suivant le vieil adage des informaticiens « rien ne vaut la pratique », presque toutes les explications se trouvent accompagnées d'un exemple de programme Basic ou en langage assembleur l'illustrant. Lisez l'explication, si la compréhension n'est pas immédiate, essayez de taper le programme qui l'accompagne. Si vous ne voyez toujours pas, votre cas est sans espoir.

Certains points ne sont pas abordés dans ce livre. Il s'agit tout d'abord du micro-processeur. Si vous avez des problèmes avec cet élément, je ne peux que vous conseiller l'achat du « Programmation du Z80 » de Rodney Zaks chez Sybex, véritable « Bible » du Z80 de plus de six cent pages (en anglais). De même, je ne m'étends pas sur le fonctionnement du processeur sonore PSG (« Programmable Sound Generator »). Tous les manuels livrés avec les différents modèles de MSX donnent les renseignements nécessaires à la programmation de ce composant. Voyez tout de même le paragraphe 13.6 « [Faire de la musique en assembleur](#) », page 295.

Ce manuel ne comporte peu d'information sur le matériel même (« hardware »). Il n'aborde les différents éléments qu'au niveau logiciel (« software »). Il s'adresse donc avant tout aux programmeurs et non à l'électronicien.

### Retrouver un renseignement rapidement :

Lorsque vous maîtrisez l'application qui vous intéresse, évitez tout ce qui est note, remarque, etc. En général, les renseignements indispensables à la mise en œuvre de l'application (adresses mémoires, registres à charger, etc) se trouvent en début de paragraphe, les explications venant après. Il va de soi qu'il peut être utile de consulter les programmes donnés en exemple, ils permettront en effet souvent un gain de temps appréciable.

A la fin de l'ouvrage sont réunies un annexe qui renferment une grande partie des petites choses dont un programmeur a toujours besoin et qui ne se trouvent, bien entendu, jamais là où on les cherche. Essayez d'exploiter ces annexes au maximum. Les débutants ne sont pas oubliés. Il y a même un lexique.

Dans tous les cas, n'hésitez pas à corriger les erreurs et à compléter les oublis qui ne manqueront pas de se révéler. Si vous avez le temps, [envoyez-moi tous les conseils, remarques et autres critiques](#) que la lecture de cet ouvrage vous aura inspirés. Je vous en remercie d'avance.

### ***1.3 Configurations minimales***

Afin de faire un programme qui s'adresse à un maximum d'utilisateurs, il est utile de connaître la configuration minimale pour chaque version du standard MSX.

#### **MSX1 :**

- Un microprocesseur central (CPU) Z80A ou compatible, 8 bits, cadencé à 3,579545Mhz.
- 8Ko de RAM.
- 32Ko de ROM (Main-ROM incluant le Basic v.1.xx)
- Un processeur vidéo (VDP) TMS9918A/TMS9918 (NTSC) ou TMS9929 (PAL) de Texas Instrument ou compatible avec 16Ko de VRAM dédiée.
- Un processeur sonore AY-3-8910 de General Instrument ou équivalent, un PSG (Programmable Sound Generator) à 3 voix sur 8 octaves avec générateur de bruit.
- Une interface programmable pour les ports E/S (PPI) compatible avec le 8255 d'Intel.
- Un port cartouche ou un Bus d'extension de type Slot. (Les broches +12V, -12V et l'entrée sonore sont optionnelles sur le Bus d'extension).
- Un clavier doté de 10 touches de fonction plus 4 touches de déplacement du curseur.
- Différents connecteurs : sortie vidéo (RCA, RGB ou autres), magnéto cassettes, 1 général (pour manette de jeu, souris, molettes, tablette tactile, etc).

#### **MSX2 :**

- Un microprocesseur central (CPU) Z80A ou compatible, 8 bits, cadencé à 3,579545Mhz.
- 64Ko de RAM.
- 48Ko de ROM (Main-ROM incluant le Basic v.2.xx et Sub-ROM)
- Un processeur vidéo v9938 de ASCII, Microsoft et Yamaha avec 64Ko de VRAM dédiée.
- Un processeur sonore AY-3-8910 de General Instrument ou équivalent, un PSG (Programmable Sound Generator) à 3 voix sur 8 octaves avec générateur de bruit.
- Une interface programmable pour les ports E/S (PPI) compatible avec le 8255 d'Intel.
- Horloge en temps réel compatible (puce RP5C01).
- Un port cartouche ou un Bus d'extension de type Slot. (Les broches +12V, -12V et l'entrée sonore sont optionnelles sur le Bus d'extension).
- Un clavier doté de 10 touches de fonction plus 4 touches de déplacement du curseur.
- Différents connecteurs : sortie vidéo (RCA, RGB ou autres), magnéto cassettes, 1 général (pour manette de jeu, souris, molettes (Paddle Controler), Trackball, etc).

### MSX2+ :

- Un microprocesseur central (CPU) Z80A ou compatible, 8 bits, cadencé à 3,579545Mhz.
- 64 Ko de RAM (Memory Mapper).
- 96 Ko de ROM (Main-ROM incluant le Basic v.3.xx, Sub-ROM, Disk-ROM et Kanji Driver).
- 128 Ko de ROM, police de Kanji (Kanji ROM niveau 1).
- Un processeur vidéo v9958 de ASCII et Yamaha avec 128Ko de VRAM dédiée.
- Un processeur sonore AY-3-8910 de General Instrument ou équivalent, un PSG (Programmable Sound Generator) à 3 voix sur 8 octaves avec générateur de bruit.
- Une interface programmable pour les ports E/S (PPI) compatible avec le 8255 d'Intel.
- Horloge en temps réel compatible (puce RP5C01).
- Deux ports cartouche de type Slot.
- Un clavier doté de 10 touches de fonction plus 4 touches de déplacement du curseur et d'un pavé numérique .
- Différents connecteurs : sortie vidéo (RCA, RGB ou autres), magnéto cassettes, 2 général (pour manette de jeu, souris, molettes (Paddle Controler), Trackball, etc).

### MSX Turbo-R :

- Un microprocesseur (CPU) compatible Z80A de Zilog, 8 bits, cadencé à 3,579545Mhz
- Un CPU R800 de ASCII, 16 bits, cadencé à 7,15909MHz.
- 256 Ko de RAM (Memory Mapper).
- 160 Ko de ROM (Main-ROM incluant le Basic v.4.xx, Sub-ROM, Disk-ROM, MSX-JE, Kanji Driver, MSX-DOS2 ROM et MSX-Music).
- 256 Ko de ROM, police de Kanji (Kanji ROM niveau 1 et, 2 en option).
- Un processeur vidéo v9958 de ASCII et Yamaha avec 128Ko de VRAM dédiée.
- Un processeur sonore AY-3-8910 de General Instrument ou équivalent, un PSG (Programmable Sound Generator) à 3 voix sur 8 octaves avec générateur de bruit.
- Un PCM capable de numériser et restituer des sons à jusqu'à 15 Khz.
- Un processeur sonore YM2413 de Yamaha, 9 voix FM ou bien 6 + 5 de rythme. (MSX-Music)
- Une interface programmable pour les ports E/S (PPI) compatible avec le 8255 d'Intel.
- Horloge en temps réel compatible (puce RP5C01).
- Deux ports cartouche de type Slot.
- Un clavier doté de 10 touches de fonction plus 4 touches de déplacement du curseur et d'un pavé numérique.
- Un lecteur de disquette 3,5 pouce, double face.
- Différents connecteurs : sortie vidéo (RCA, RGB ou autres), imprimante, 2 ports généraux (pour manette de jeu, souris, Trackball, molettes, tablette graphique, etc).

Note : Le port du magnéto-cassette a été retiré et le Bios n'est plus compatible avec les molettes (Paddle Controller) ASCII.

## 1.4 *Conseils au développeur de logiciels*

- Les programmes que vous écrivez ne doivent en aucun cas appeler directement les routines dans la mémoire morte. Celle-ci varie d'une marque à l'autre et surtout des MSX1 aux MSX2. Il est absolument indispensable de passer par la table des sauts du Bios. Chaque Rom (Main ROM, Sub-ROM et ROM Disk) possède un Bios avec sa table des sauts.
- Pour maintenir la compatibilité entre les différents MSX, il ne faut accéder au matériel sans passer par le Bios. Ce dernier constitue une sorte de tampon entre les programmes et le matériel.
- Une des exceptions à la règle ci-dessus concerne le processeur vidéo. Pour des raisons de vitesse d'exécution, vos programmes peuvent adresser le VDP (« Video Display Processor ») sans passer par le Bios. Les adresses mémoire 00006h et 00007h contiennent l'adresse du port de lecture et du premier port d'écriture du processeur vidéo. Voir le [chapitre 6](#) pour plus de précisions à ce sujet.
- Ne pas utiliser la mémoire vive située au dessus de 0F380h comme de la RAM ordinaire. En effet, cette zone mémoire contient les variables système (voir [chapitre 4](#)) indispensables à la bonne marche de votre ordinateur.
- Il existe des différences entre les MSX commercialisés en France et dans les autres pays - ne serait-ce que le clavier - dont il est parfois important de tenir compte. Les cases mémoire 0002Bh et 0002Ch donnent des renseignements importants à ce sujet. Voir le [chapitre 3.2](#).
- Sur MSX2 uniquement, vous trouverez des renseignements complémentaires dans la mémoire CMOS de l'horloge. Voir la routine REDCLK (001F5h), [chapitre 3.3](#) « Table des sauts du Bios en Sub-ROM » page 80.
- Certains programmeurs placent la pile en haut de mémoire avec l'instruction LD SP, 00000h. Ceci ne fonctionne évidemment pas sur MSX car l'adresse 0FFFFh est utilisée pour accéder aux registres des Slot secondaires (voir le [chapitre 2](#) à propos des Slot page 10).
- Certains programmeurs partent du principe que la mémoire vive principale ou vidéo contient 0 à l'allumage. Il va de soi que ce n'est pas le cas et que la RAM peut contenir à peu près n'importe quoi si elle n'a pas été modifiée lors de l'initialisation du système.
- Si votre programme ne peut fonctionner avec la présence d'un lecteur de disquettes, il suffit de vérifier la présence du lecteur (voir le [chapitre 7](#)) puis, si un lecteur est bien présent, d'afficher à l'écran le message « Faites un RESET en laissant la touche SHIFT enfoncée jusqu'au bip sonore ».
- En effet, lors de l'initialisation du système, si le MSX détecte que la touche SHIFT est enfoncée, il n'installe pas les lecteurs de disquettes.
- Sur le même principe, la touche CTRL assigne un seul lecteur par contrôleur. Si vous avez 2 lecteurs de disquettes avec 2 contrôleurs, lors de toutes les opérations, le premier s'appelle « A » alors que le second prend la dénomination « C ». En enfonçant CTRL à l'initialisation, vous gagnerez de la place mémoire et vos lecteurs se nommeront « A » et « B ».
- Lorsque l'on écrit dans un registre « Write only » du processeur vidéo, il est préférable de sauvegarder la donnée dans la zone des variables systèmes adéquate. Ainsi, la donnée pourra être relue à tout moment, ce qui serait impossible autrement.



## ***1.5 Et maintenant...***

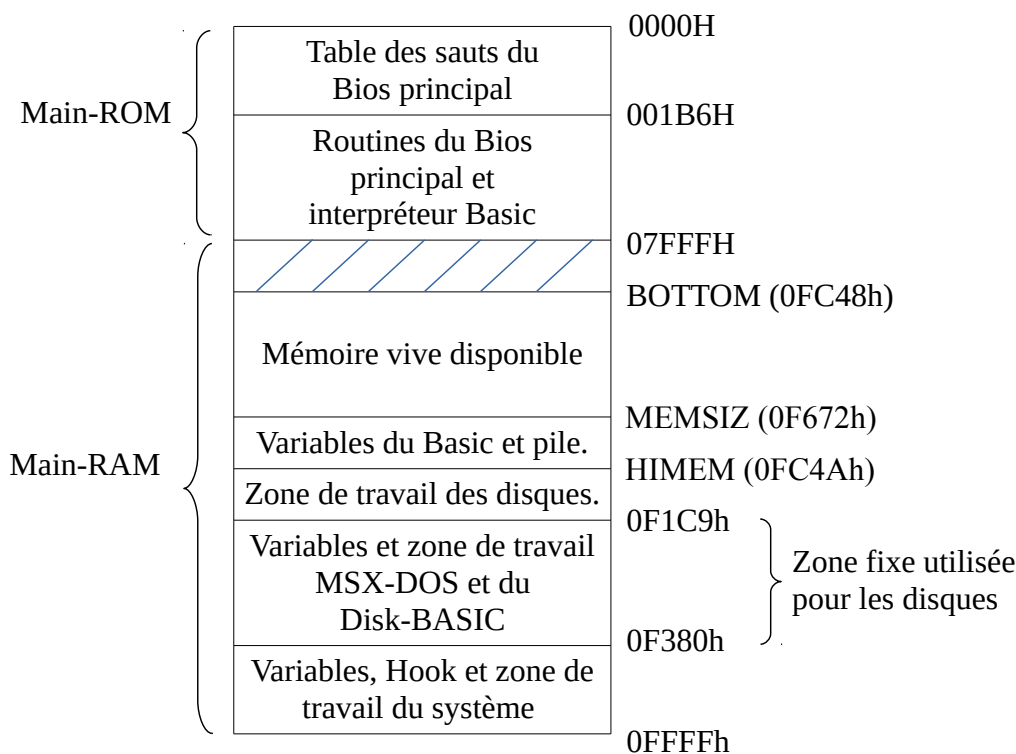
A présent, vous allez vous lancer dans la découverte des Slot, Bios, variables systèmes, et autres processeurs vidéo. Une bonne dose de patience sera, dans la plupart des cas, appréciable et bénéfique. Et maintenant à vous de jouer... et bonne chance !

## 2 Les Slot et le Memory Mapper

La mémoire vive principale (Main-RAM) est la mémoire sélectionnée au démarrage pour y installer le système. Dans l'environnement du Basic, cette mémoire est disposée de la façon suivante.

### Carte de la mémoire principale sous Basic

Cette carte doit être respectée afin que votre programme n'empiète pas sur une zone autre que celle réservée à l'utilisateur (Mémoire vive disponible).



Cette carte montre que le MSX sélectionne seulement la partie haute de la Main-Ram. La mémoire vive disponible pour l'utilisateur commence à l'adresse 0E000h ou 0C000h sur les MSX de 8 ou 16Ko. Elle commence à l'adresse 08000h sur tous les autres MSX. La variable système BOTTOM (0FC48h) indique cette adresse. Pour connaître la fin de mémoire vive disponible, il faudra lire la variable HIMEM (0FC4Ah). L'instruction CLEAR du basic permet de créer une zone « protégée » entre celle de la pile et celle de travail des disques afin d'y mettre nos propres routines en langage machine.

Sur les MSX ayant plus de 32Ko de mémoire vive, il est nécessaire de manipuler les Slot pour accéder reste de la mémoire vive qui est inaccessible au Basic. Ce chapitre explique comment manipuler les Slot.

Note : HIMEM (0FC4Ah) est spécifié par le deuxième paramètre de l'instruction CLEAR du basic et la zone entre MEMSIZ (0F672h) et la première valeur de la pile est spécifiée par le premier paramètre de CLEAR.

## 2.1 Comment dépasser la limite des 64 Ko

Tous les ordinateurs MSX sont équipés d'un Z80 ou compatible comme micro-processeur principal. Ce dernier est un processeur « 8 bits », ceci signifie qu'il manipule les données par paquets de 8. Quant aux adresses mémoires, elles se trouvent codées sur 16 bits, soit deux octets. Les adresses peuvent donc prendre n'importe quelle valeur entre 0 et 1111111111111111 en binaire, ou 0 et FFFF en hexadécimal. Ceci équivaut en décimal à une valeur entre 0 et 65535. Sachant qu'un « kilo » informatique ne vaut pas 1000 mais 2 puissance 10 soit, 1024 octets (un des petits secrets qui fait le charme de l'informatique), le Z80 qui peut accéder à 65536 adresses gère donc 64 kilo octets (65536/1024).

Ce que « voit » le processeur



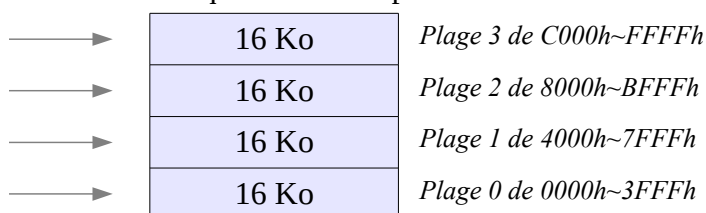
Mémoire centrale

Ainsi, quoiqu'il arrive, le Z80 ne pourra jamais adresser plus de 64 Ko de mémoire. Les concepteurs du MSX (ASCII) ont donc introduit un système, appelé « Slot », qui permet de multiplier la mémoire jusqu'à par 16, accessible par plage (Bank en anglais) de 16 Ko sur lesquelles l'utilisateur peut y changer le contenu à volonté. Le dispositif de slot se contrôle à l'aide de la puce PPI qui gère aussi le clavier.

Toutes ces opérations de manipulations de mémoire restent invisibles au Z80 qui utilise ses 64 Ko comme s'il n'y avait pas d'autre mémoire.

On numérote chaque plage de 16 Ko de 0 à 3.

Ce que « voit » le processeur



Mémoire centrale

Il existe un autre dispositif autorisant la manipulation de pages de mémoire dans les Slot, c'est le « Memory Mapper ». Les « Slot » sont utilisés sur tous les MSX. Quant au « Memory Mapper », il est apparu avec le MSX2. Seuls certains sont compatibles MSX1 (voir plus bas).

## 2.2 Qu'est-ce qu'un Slot ?

Un Slot (mot anglais que l'on traduit par fente) est une connexion commandée par un registre qui permet de relier une mémoire de 64Ko à un dispositif comprenant en quelque sorte quatre « aiguillages » à quatre positions au lieu de la relier directement au CPU. Chacun de ses « aiguillages » sont reliés à 16Ko d'un Slot différent sur la même plage mémoire. Les plages mémoire sont comprises entre les adresses 0000h et 3FFFh (page 0), 4000h et 7FFFh (page 1), 8000h et BFFFh (page 2), et C000h et FFFFh (page 3).

Il y a deux sortes de Slot, les Slot primaires et les Slot secondaires. Ces derniers sont reliés à un Slot primaire et fonctionnent de façon identique sauf que les registres sont accessibles différemment. Un MSX peut avoir jusqu'à 4 Slot primaires dont chacun peut être relié à 4 Slot secondaires au lieu d'une mémoire. Les Slot secondaires permettent de gérer 16 fois plus de mémoire.

Voici un exemple d'un MSX ayant tous ses Slot primaires étendus avec sa Main-ROM dans le Slot secondaire 0-0, une ROM quelconque dans le Slot secondaire 3-0 et ces 32Ko de RAM dans le Slot secondaire 3-2 qui a démarré sous basic en mode programmation (mode direct) :

Slot Primaire 0				Slot primaire 1				Slot primaire 2				Slot primaire 3				
SS0	SS1	SS2	SS3	SS0	SS1	SS2	SS3	SS0	SS1	SS2	SS3	SS0	SS1	SS2	SS3	
														RAM		Page 3
														RAM		Page 2
MAIN												ROM				Page 1
MAIN												ROM				Page 0

La mémoire centrale (ce que voit le CPU) sera déterminée par le numéro de Slot pour chaque plage.

Plage 3 du Z80	3-2	Mémoire du Slot 3-2.
Plage 2 du Z80	3-2	Mémoire du Slot 3-2.
Plage 1 du Z80	0-0	Mémoire du Slot 0-0.
Plage 0 du Z80	0-0	Mémoire du Slot 0-0.

Mémoire centrale

Voici un exemple de la disposition par défaut sur un MSX1, le Philips VG-8020/19 :

Plage 3							
Plage 2							
Plage 1	MAIN					RAM	
Plage 0	MAIN					RAM	
	0	1	2			3-0	3-1

- Au démarrage sous Basic, les plages 0 et 1 contiennent le Bios et l'interpréteur du Basic qu'il y a dans la mémoire morte principale (Main-ROM). Celle-ci se trouve dans le Slot primaire 0.
- Les Slot 1 et 2 correspondent aux deux ports cartouches du 8020. Le contenu d'un jeu en cartouche pourra être lu en sélectionnant le Slot 1 ou 2.
- Les plages 0 à 3 du Slot 3-2 contiennent chacune 16 Ko de mémoire vive. On comprend pourquoi on ne dispose que de 32 Ko (même un peu moins) sous Basic. Le Z80 « voit » le Bios et l'interpréteur Basic en pages 0 et 1. Il ne reste donc que les pages 2 et 3 pour de la mémoire vive.

Voici un exemple de disposition sur un MSX2, le VG-8235 de Philips :

Plage 3						MEM	
Plage 2						MEM	
Plage 1	MAIN					MEM	DISK
Plage 0	MAIN			SUB		MEM	
	0	1	2	3-0	3-1	3-2	3-3

Quelques remarques :

- Au démarrage sous Basic, les plages 0 et 1 contiennent le Bios et l'interpréteur du Basic qu'il y a dans la mémoire morte principale (Main-ROM) du Slot 0.
- Les Slot 1 et 2 correspondent aux deux ports cartouches.
- La plage 0 du Slot 3-0 renferme la ROM auxiliaire (Sub-ROM) qui contient un Bios pour manipuler les fonctions supplémentaires spécifiques au MSX2 ou plus récent.
- Les plages 0 à 3 du Slot 3-2 contiennent chacune 16 Ko de mémoire vive. (Les autres 64 Ko de la mémoire vive totale qui compte 128 Ko sur le VG-8235 sont accessibles via le Memory Mapper que nous verrons plus loin dans ce chapitre.)
- La Disk-ROM en plage 1 du Slot 3-3 contient les routines du DOS et du Disk-Basic. (Le VG-8235 est équipé d'un lecteur de disquettes intégré.)

La configuration minimum que l'on est certain de trouver sur tout ordinateur se compose de :

MSX1 :

- Une demi page de mémoire vive (située de 0E000h à 0FFFFh) dans n'importe quel Slot.
- Deux pages de mémoire morte (située de 00000h à 07FFFh) contenant le MSX-Basic version 1.0 et le Bios de la ROM principale (« Main-ROM ») dans n'importe quel Slot.
- Un port cartouche ou un Bus d'extension occupant un Slot primaire.

MSX2 :

- Quatre pages consécutives (64 Ko) de mémoire vive dans n'importe quel Slot. (En général dans un même Slot mais pas toujours)
- Trois pages (48 Ko) de mémoire morte divisée en deux parties la « Main-ROM » et la « Sub-ROM », dans n'importe quel Slot .
- Un port cartouche occupant un Slot primaire.

## 2.3 Utiliser les Slot

Toutes les routines du systèmes code le numéro de Slot sur un octet de la manière suivante (au format F000SSPP) :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
F	-	-	-	SS1	SS0	PP1	PP0

PP1 et PP0 = Ces deux bits correspondent au numéro de Slot primaire (de 0 à 3).

SS1 et SS0 = Ceux-ci correspondent au numéro de Slot secondaire (de 0 à 3).

F = Le bit 7 est un indicateur de type de Slot. Ce bit est à 1 lorsqu'il s'agit d'un Slot secondaire autrement, il s'agit d'un Slot primaire et les SS1 et SS0 ne doivent pas être pris en compte.

Les bits 6 à 4 peuvent varier suivant le contexte mais, lorsque c'est le cas, ils n'ont pas rapport direct avec les Slot.

Les registres des Slot primaires et secondaires sont codés ainsi :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Slot sélectionné sur la plage mémoire 3				Slot sélectionné sur la plage mémoire 2		Slot sélectionné sur la plage mémoire 1	
Slot sélectionné sur la plage mémoire 3				Slot sélectionné sur la plage mémoire 2		Slot sélectionné sur la plage mémoire 1	

La plage mémoire 0 va de 0000h à 3FFFh, la plage 1 de 4000h à 7FFFh, la plage 2 de 8000h à BFFFh et la plage 3 de C000h à FFFFh.

### Variables systèmes liées aux Slot :

#### - **EXPTBL** (EXPanded slot TaBLe)

Adresse : 0FCC1h~0FCC4h en Main-RAM

Rôle : Le bit 7 de ces variables est un indicateur de Slot étendu pour chacun des Slot primaires. Il est à 1 lorsque le Slot correspondant est étendu en Slot secondaires. Les autres bits sont à zéro. Ceci implique que la Main-ROM d'un MSX se trouve toujours dans le Slot primaire 0, ou le Slot secondaire 0-0. Cependant sur MSX2, la variable à 0FCC1h (appelée aussi MNROM) peut aussi indiquer le Slot de la Main-ROM au format F000SSPP. Ceci a permis de faire des extensions MSX1 vers MSX2 en remplaçant la Main-ROM interne par une autre mais avec des concessions sur la compatibilité. Par exemple, la gestion de certains matériels internes, la matrice du clavier et certains indicateurs en Main-ROM sont inévitablement remplacés par du générique.

#### - **SLTTBL** (SLoT TaBLe)

Adresse : 0FCC5h~0FCC8h en Main-RAM

Rôle : Le système y sauvegarde l'état des 4 registres des Slot secondaires de chaque Slot primaire (de 0 à 3). Attention, seuls les bits correspondants aux Slot étendus sont à prendre en compte (à l'aide des variables EXPTBL expliquées ci-dessus).

- **EXBRSA** (EXpanded Bios Rom SLoT Address)  
 Adresse : 0FAF8h en Main-RAM (MSX2 ou plus récent.)  
 Rôle : Cet octet donne le numéro du Slot qui contient la Sub-ROM.
  
- **MINROM** (ROM SLoT)  
 Adresse : 0FFF7h en Main-RAM  
 Rôle : À partir du MSX2, cet octet contient le numéro de Slot qui contient la Main-ROM mais cette variable a été considérée obsolète par la suite.
  
- **SLTATR** (SLoT ATTRibut)  
 Adresse : 0FCC9h~0FD08h en Main-RAM  
 Rôle : Les 64 octets de SLTATR informent sur le contenu des ROM de chaque Slot. Quatre plages de mémoire de 16 Ko réparties sur 16 Slot secondaires possibles, ce qui fait  $4 \times 16$  combinaisons. (Voir « [Développer un programme en cartouche](#) » au chapitre 11 pour la description)
  
- **SLTWRK** (SLoT WoRK)  
 Adresse : 0FD09h~0FD88h en Main-RAM  
 Rôle : SLTWRK est un tableau de 128 octets qui sert à réserver une zone travail en RAM pour les applications en ROM. (Voir « [Développer un programme en cartouche](#) » au chapitre 11 pour la description)

#### Routines du Bios liées aux Slot :

- **RDSLT** (ReaD SLoT)  
 Adresse : 0000Ch en Main-ROM  
 Rôle : Lecture d'une case mémoire dans un Slot.
  
- **WRSLT** (WRite SLoT)  
 Adresse : 00014h en Main-ROM  
 Rôle : Écriture dans une case mémoire dans un Slot.
  
- **CALSLT** (CALl SLoT)  
 Adresse : 0001Ch en Main-ROM  
 Rôle : Appel inter-Slot à une adresse.
  
- **ENASLT** (ENABle SLoT)  
 Adresse : 00024h en Main-ROM  
 Rôle : Sélection d'un Slot.

- **CALLF** (CALL Far)
  - Adresse : 00030h en Main-ROM
  - Rôle : Appel inter-Slot à une adresse.
  
- **RSLREG** (Read primary Slot REGister)
  - Adresse : 00138h en Main-ROM
  - Rôle : Lecture du registre des Slot primaires.
  
- **WSLREG** (Write Slot REGister)
  - Adresse : 0013Bh en Main-ROM
  - Rôle : Écriture dans le registre des Slot primaires.
  
- **SUBROM** (SUBROM call SUB-ROM)
  - Adresse : 0015Ch en Main-ROM
  - Rôle : Appel à une adresse dans la Sub-ROM.
  
- **EXTROM** (EXTROM call EXTernal ROM)
  - Adresse : 0015Fh en Main-ROM
  - Rôle : Appel à une adresse dans une ROM externe.

Ces routines n'appellent pas de commentaire particulier ([voir le Bios](#)) sauf peut-être la routine CALLF « CALL Far » :

Pour accéder à une adresse se trouvant dans un autre Slot, il suffit d'écrire les 3 instructions suivantes en assembleur :

```
rst 30
db <numéro du Slot>
dw <adresse à appeler>
```

Le RET de la routine appelée renverra l'exécution à l'octet immédiatement après l'adresse définie par le DW. Par exemple, en langage machine, la suite d'octets F7h, 8Eh, 0h, C0h, AFh générerait la séquence suivante :

```
0F7h instruction « RST 30 » du Z80
08Eh numéro de Slot secondaire 2-3
000h adresse à appeler 0C000h
0C0h
0AFh instruction « XOR A » du Z80, l' instruction RET de la routine en 0C0000h dans le
Slot 2-3 fera reprendre l'exécution du programme à ce « XOR A ».
```



## 2.4 Le Memory Mapper

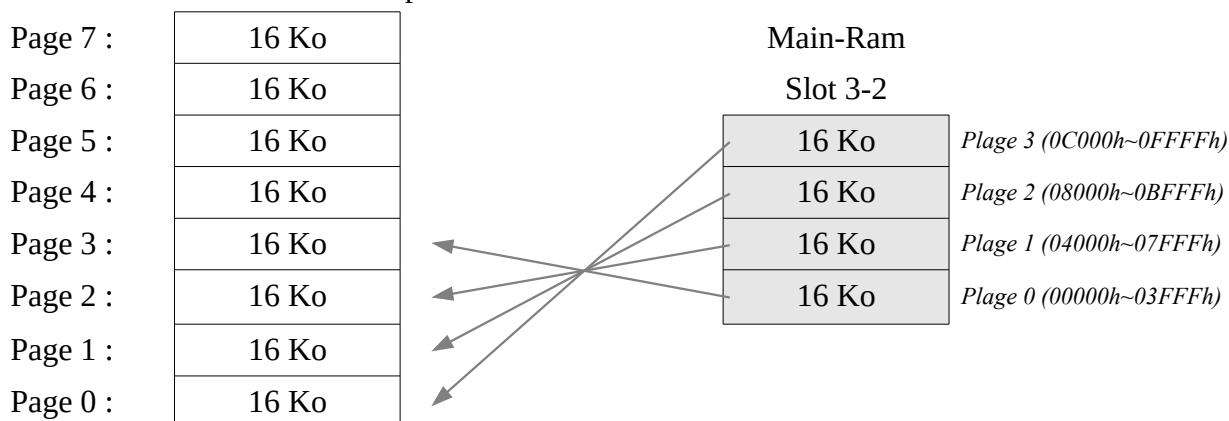
La plupart des MSX2 disponibles en France se trouvent équipés d'un dispositif perfectionné permettant de gérer des quantités assez importantes de mémoire vive pour l'époque (128 ou 256 Ko de RAM sur les modèles disponibles chez nous, mais pouvant aller en théorie jusqu'à 4 Mo par Slot).

La mémoire d'un Memory Mapper est manipulable par page de 16Ko. Chaque page d'un Memory Mapper pouvant être placée sur n'importe quelle des quatre plages mémoire d'un Slot.

Note : A l'initialisation du MSX, le système cherche la RAM dans les Slot primaires de 0 à 3 et ce sur chaque plage mémoire. Si un Slot primaire est étendu, il cherche dans les Slot secondaires avant de passer au slot primaire suivant. Ainsi le système sélectionne la première RAM qu'il trouve pour chaque plage sans tenir compte du Memory Mapper, excepté le MSX turbo R qui prend toujours sa RAM interne pour mémoire principale. De plus, les MSX1 n'ont pas de routine pour initialiser les pages du Memory Mapper dans l'ordre 3, 2, 1, 0 sur les plages 0, 1, 2, 3. Il est important de tenir compte de ces contraintes dues à l'évolution du standard.

Schéma d'une mémoire vive de 128 Ko gérée par le « Memory Mapper » dans le Slot 3-2 :

Huit blocs de 16 Ko soit 128 Ko de mémoire vive  
sur Memory Mapper comme sur le modèle VG  
8235 MSX2 de Philips



Il est possible de placer une même page sur chacune des plages. Il est également possible de mettre un autre Memory Mapper dans un autre Slot. Dans ce cas, les pages se changeront en même temps dans chacun des Slot.

Attention, la page 0 contient les variables systèmes. Veillez à la laisser toujours dans la plage 3 pour ne pas provoquer un plantage du système.

### Utilisation du Memory Mapper :

La gestion du Memory Mapper est particulièrement simple : on utilise les quatre ports d'entrée/sortie, de 0FCh à 0FFh.

En effet, le port 0FCh sert à attribuer la page de Memory Mapper pour la plage 0. Le port 0FDh pour la plage 1, le port 0FEh pour (l'aviez-vous deviné ?) la plage 2 et le port 0FFh pour la plage 3.

0FCh = Page placée sur la plage 0 (00000h~03FFFh)

0FDh = Page placée sur la plage 1 (04000h~07FFFh)

0FEh = Page placée sur la plage 2 (08000h~0BFFFh)

0FFh = Page placée sur la plage 3 (0C000h~0FFFFh)

Ces ports restent accessibles aussi bien en écriture qu'en lecture pour le Memory Mapper interne. En effet, un Memory Mapper en cartouche n'est normalement accessible qu'en lecture. Cependant, il existe des Memory Mapper en cartouche qui gèrent aussi ces ports en lecture. Ces derniers ont été prévus pour les MSX n'ayant pas de Memory Mapper interne. Le standard étant assez floue cette caractéristique, il est devenu déconseillé aux développeurs d'utiliser ces ports en lecture à cause du conflit provoqué par les bits non gérés. Certaines extensions ne gèrent pas du tout la lecture d'ailleurs.

A l'initialisation, le Bios du MSX2 (ou plus récent) place les pages de la manière suivante :

mémoire centrale	port I/O	contenu du port	page Memory Mapper
Plage 0 (00000h~03FFFh)	0FCh	003h	page 3
Plage 1 (04000h~07FFFh)	0FDh	002h	page 2
Plage 2 (08000h~0BFFFh)	0FEh	001h	page 1
Plage 3 (0C000h~0FFFFh)	0FFh	000h	page 0

Comme le Memory Mapper est une option sur MSX2, il n'existe aucune variable système ou routine du Bios, le concernant. Si vous désirez utiliser le Memory Mapper, votre programme devra déterminer seul la présence ou l'absence du Memory Mapper ainsi que sa taille mémoire.

ASCII demande à toute société commercialisant des logiciels MSX2 utilisant le Memory Mapper de porter sur l'emballage la mention « Requires XXX K MEMORY MAPPER ».

Notes :

- Étant donné que le Bios du MSX1 ne gère pas le Memory Mapper, lorsqu'une cartouche de Memory Mapper est insérée, chaque plage mémoire contiendront en général la page 0 au démarrage ou bien, une page indéterminée. Les programmes ayant besoin plus de 16Ko de RAM feront planter le MSX. Il existe des cartouches de Memory Mapper à base de puce programmable qui initialisent les pages dans le bon ordre par eux-même pour éviter ce problème MSX1 mais elles sont peu répandues.
- La Disk-ROM v2.xx, celle du MSX-DOS 2, possède des routines qui permettent de gérer les pages de plusieurs Memory Mapper. Ces routines rendent possible la création d'un gros Ramdisk ainsi que l'utilisation de diverses applications sans risquer un conflit de mémoire. Ces routines sont accessibles via le Bios étendu. (voir le [paragraphe du Bios étendu](#) pour les détails.)

## 3 Le Bios (Basic Input/Output System)

### 3.1 *Introduction au Bios*

Le Bios (Basic Input/Output System) est un ensemble de routines conçues afin de permettre au programmeur d'accéder à un matériel (« hardware ») sans que celui-ci soit toujours le même. Le Bios se compose donc de routines en ROM qui permettent d'accéder de la même façon à de plus ou moins différents matériels. Prenons l'exemple du clavier du MSX qui est géré par un circuit spécialisé, le PPI 8255. Un jour, il se peut qu'un constructeur MSX sorte un modèle avec clavier détaché à liaison infrarouge qui sera probablement contrôlé par une puce autre qu'un PPI 8255. Dans ce cas, si un programme, faisant des accès directs au PPI, ne fonctionnera pas correctement sur ce nouveau modèle. Au contraire, si le programme passe par la routine « lire l'état du clavier », il tournera très bien même avec le clavier à liaison infrarouge. Le Bios concilie compatibilité des logiciels avec évolution du matériel.

### 3.2 *Table des sauts du Bios en Main-ROM*

Les routines du Bios pouvant avoir une taille variable d'un MSX à l'autre, il a donc été nécessaire de créer une table de sauts vers chaque routine du Bios afin que tous les logiciels fonctionnent quelque soit le MSX utilisé. Sur la page suivante vous trouverez la liste de la table des sauts vers chaque routine avec son adresse mémoire, son nom, sa fonction, les paramètres devant être fournis, les résultats que vous récupèrerez en retour ainsi que les registres qui ont été modifiés par la routine et des remarques sur la routine elle-même, sur son fonctionnement ou les différences entre chaque version de MSX lorsque nécessaire.

Parmi cette liste, il y a aussi quelques données entre 00004h et 0002Fh pour fournirent des informations sur le MSX utilisé. Ces adresses sont suivies d'un astérisque pour indiquer qu'il ne s'agit pas d'un point d'entrée vers une routine du Bios.

## 00000h (Main-ROM)     **STARTUP** (appelée **CHKRAM** dans un premier temps)

Rôle :	Redémarrage du MSX. (Recherche de la Mémoire, initialisation des variables du système, etc.)
Entrée :	Rien.
Sortie :	Rien.
Modifie :	Tout.
Note :	Cette routine est la première à être exécutée à l'allumage du MSX. Voici un résumé de ce que fait la routine.

### MSX1 :

- La Main-ROM est placée sur la plage mémoire 0000h~3FFFh et 4000h~7FFFh.
- La routine cherche la RAM dans les Slot primaires de 0 à 3 sur les plages mémoire 8000h~BFFF et C000h~FFFFh. Si un Slot primaire est étendu, elle cherche dans les Slot secondaires de 0 à 3 avant de passer au slot primaire suivant. Ainsi la routine sélectionne la première RAM qu'elle trouve sur chaque plage sans tenir compte du Memory Mapper.
- Initialise la zone de travail, les variables et les Hooks.
- Cherche une éventuelle Rom dans les Slot primaires de 0 à 3 sur les plages mémoire 4000h~7FFF et 8000h~BFFFh pour l'exécuter. (Lorsqu'une Rom est trouvée sur la plage 4000h~7FFFh, la plage 8000h~BFFFh contiendra de la RAM si le MSX a 32Ko ou plus. Lorsqu'une Rom est trouvée sur la plage 8000h~BFFFh, la plage 4000h~7FFFh contiendra la Main-ROM).
- Remplace la Main-ROM sur la plage 4000h~7FFFh et démarre le Basic.

### MSX2 :

Procède de la même façon que les MSX1 sauf que les registres du Memory Mapper sont initialisés comme indiqué au paragraphe sur [le Memory Mapper](#).

### MSX2+ :

Procède de la même façon que les MSX2 sauf que la routine prend en compte si il s'agit du démarrage de l'ordinateur ou d'un redémarrage. Dans le second cas, le logo MSX ne sera pas affiché. (Le bit 5 du port d'E/S F4h est utilisé pour cela.)

### MSX turbo R :

Procède de la même façon que les MSX2+ sauf que la RAM interne sera sélectionnée d'office en mode R800 parce que la RAM interne est cadencée à 7,159 MHz au lieu de 3,579545 pour les extensions.

## 00004h\* (Main-ROM)     **CGTABL**    Character Generator TABLE

Rôle :	Les octets 00004h et 00005h contiennent l'adresse du début table de caractères au code ASCII par défaut.
--------	--

### 00006h\* (Main-ROM)    **VDP.DR**    VDP Data Read port

Rôle :            Cet octet renferme l'adresse du port d'entrée du premier port de lecture du processeur vidéo interne. Les logiciels nécessitant des accès vidéo très rapides peuvent adresser directement le VDP (sans passer par le Bios) à condition d'utiliser cette adresse comme port d'entrée.

Pour plus de précision, voir le chapitre « [Comment accéder au VDP](#) ».

### 00007h\* (Main-ROM)    **VDP.DW**    VDP Data Write port

Rôle :            Cet octet renferme l'adresse du port de sortie du premier port d'écriture du processeur vidéo interne. Les logiciels nécessitant des accès vidéo très rapides peuvent adresser directement le VDP (sans passer par le Bios) à condition d'utiliser cette adresse comme port de sortie.

Pour plus de précision, voir le chapitre « [Comment accéder au VDP](#) ».

### 00008h (Main-ROM)    **SYNCHR**    SYNtax of CHaRacter

Rôle :            Comparaison du caractère placé sur l'octet suivant le RST 8 avec celui pointé par HL. Si c'est le même, exécute la routine appelée par CHRGT (00010h en Main-ROM) sinon, exécute la routine de traitement d'erreur de l'interpréteur du Basic.

Entrée :          HL = Adresse du caractère à comparer.  
Octet suivant l'instruction RST 8 utilisée pour lancer cette routine = Caractère.

Sortie :          HL = Adresse suivante.  
A = Caractère pointé par HL.  
F = L'indicateur C passe à 1 si le caractère est un chiffre.  
l'indicateur Z passe à 1 si le caractère est 00h ou 3Ah (fin d'instruction).

Modifie :        AF, HL.

Notes :          - Cette routine est utilisé par l'interpréteur Basic. (RST 8)  
- Attention, la Main-Rom doit être sélectionnée sur la plage 4000h~7FFFh pour appeler cette routine.

### 0000Ch (Main-ROM)    **RDSL**T    ReaD SLoT

Rôle :            Lecture d'un octet dans un autre Slot.

Entrée :          HL = Adresse de l'octet à lire (de 0000h à BFFFh).  
A = Numéro du Slot sous la forme binaire F000SSPP.  
F doit être à 0 pour Slot primaire ; à 1 pour Slot secondaire.  
SS est le numéro de Slot Secondaire.  
PP est le numéro de Slot Primaire.

Sortie :          A = Octet lu.

Modifie :        AF, BC, DE.

Notes :          - Les interruptions sont désactivées par la routine.  
- Cette routine existe aussi sous MSX-DOS.

## 00010h (Main-ROM)     **CHRGTR**    CHaracteR GeTteR

- Rôle :            Récupération d'un caractère ou d'un code dans un programme Basic.
- Entrée :          HL = Adresse actuelle. (Pointeur)
- Sortie :          HL = Adresse du caractère récupéré.  
                    A = Caractère ou chiffre récupéré.  
                    F = Indicateur Z à 1 si il s'agit d'un code de fin de ligne (00h ou « : »).  
                    Indicateur C à 1 si il s'agit d'un chiffre de 0 à 9.
- Modifie :        AF, HL.
- Notes :          - Cette routine passe les codes d'espacement (020h).  
                    - - Appel au Hook H.CHRG (0FF48h).  
                    - Cette routine est utilisée par l'interpréteur Basic. (RST 10)

## 00014h (Main-ROM)     **WRSLT**     WRite SLot

- Rôle :            Ecriture d'un octet dans le Slot spécifié.
- Entrée :          HL = Adresse de l'octet à écrire (de 0000h à BFFFh).  
                    E = Donnée à écrire.  
                    A = Numéro du Slot sous la forme F000SSPP.  
                    F doit être à 0 pour Slot primaire ; à 1 pour Slot secondaire.  
                    SS est le numéro de Slot Secondaire.  
                    PP est le numéro de Slot Primaire.
- Sortie :          Rien.
- Modifie :        AF, BC, D.
- Note :            Interruptions automatiquement interdites. Cette routine peut aussi être appelée sous MSX-DOS.

## 00018h (Main-ROM)     **OUTDO**     OUT DO

- Rôle :            Sortie d'un caractère alphanumérique sur écran, imprimante ou disquette.
- Entrée :          A = Code ASCII du caractère à sortir.  
                    PRTFLG (0F416h) = Indicateur à 1 pour sortie sur imprimante ou disquette.  
                    PTRFIL (0F864h) = Adresse du fichier dans lequel le caractères sera envoyé.  
                    Si 0000h, le caractère sera envoyé à l'imprimante.
- Sortie :          Rien.
- Modifie :        Rien.
- Notes :          - Appel au Hook H.OUTD (0FEE4h).  
                    - Cette routine est utilisé par l'interpréteur Basic. (RST 18)

## 0001Ch (Main-ROM)    **CALSLT**    CALL Slot

- Rôle :            Appel inter-Slot à une adresse.
- Entrée :          IX = Adresse à appeler.  
                  IY = Numéro du Slot sous la forme F000SSPP.  
                  F doit être à 0 pour Slot primaire ; à 1 pour Slot secondaire.  
                  SS est le numéro de Slot Secondaire.  
                  PP est le numéro de Slot Primaire.
- Sortie :          Dépend de la routine appelée.
- Modifie :        AF, IX, IY, registres auxiliaires.
- Notes :          - Les interruptions sont désactivées par la routine.  
                  - L'appel à un autre Slot étendu du même Slot primaire que celui où se trouve le Bios n'est pas possible.  
                  - Cette routine existe aussi sous MSX-DOS.

Exemple en assembleur :

```
; Appeler une routine du Bios en Main-ROM sous MSX-DOS
; Attention, cette routine ne peut pas appeler de routine en SUB-ROM

CALSLT equ 001ch          ; Routine d'appel inter-slot
EXPTBL equ 0fcc1h         ; Octet indiquant le Slot de la Main-ROM
BEEP   equ 000c0h         ; Routine d'émission d'un son "Bip"

DEBUT:  ld iy,(EXPTBL-1) ; Emplacement de la Main-ROM dans IY
        ld ix,BEEP      ; Adresse de la routine BEEP dans IX
        call CALSLT     ; Appel la routine BEEP

END:
```

## 00020h (Main-ROM)    **DCOMPR**    Double register COMpare

- Rôle :            Comparaison de HL avec DE.
- Entrée :          HL = Premier nombre.  
                  DE = Deuxième nombre.
- Sortie :          F = Indicateur C passe à 1 si HL < DE ;  
                  indicateur Z passe à 1 si HL = DE.
- Modifie :        AF.
- Note :            Cette routine est utilisé par l'interpréteur Basic. (RST 20)

## 00024h (Main-ROM)    **ENASLT**    ENAble Slot

Rôle :            Sélection d'un Slot.

Entrée :          HL = Plage à sélectionner.

A = Numéro du Slot sous la forme F000SSPP.

F doit être à 0 pour Slot primaire ; à 1 pour Slot secondaire.

SS est le numéro de Slot Secondaire.

PP est le numéro de Slot Primaire.

Sortie :          Rien.

Modifie :        Tout.

Notes :          - Interruptions automatiquement interdites.  
- Cette routine existe aussi sous MSX-DOS.

Exemple d'utilisation en assembleur :

```

; Passage du MSX-DOS au Basic

MNROM      equ    0fcc1h
ENASLT      equ    00024h
RETURN      equ    0409bh

DEBUT:      org    0d000h

            ld     a, (MNROM)
            ld     hl, 0
            call   ENASLT      ; Bios en page 0
            ld     a, (MNROM)
            ld     hl, 04000h
            call   ENASLT      ; Basic page 1
            jp     RETURN      ; Saut à l'interpréteur
```

## 00028h (Main-ROM)    **GETYPR**

Rôle :            Détermination du type de DAC (« Decimal Acumulator »).

Entrée :          Rien.

Sortie :          F = S\*, C, P/V à 1 et Z à 0 si chiffre entier ;

S, Z, P/V\* à 0 et C à 1 si chiffre simple précision ;

S, Z, C\* à 0 et P/V à 1 si chiffre double précision ;

C, Z\*, P/V à 0 et S à 1 si chaine de caractères.

\* Indicateur sur lequel se baser pour connaître le type.

Modifie :        AF.

Notes :          - Cette routine est utilisé par l'interpréteur Basic. (RST 28)  
- Attention, la Main-Rom doit être sélectionnée sur la plage 4000h~7FFFh pour appeler cette routine.



## 0002Bh\* (Main-ROM) **BASRVN** BASic Rom Version Number

Rôle : Cet octet et le suivant contiennent des renseignements utiles aux programmes destinés à fonctionner dans des pays différents (« international software »).

0002Bh = Les bits 0 à 3 indiquent le types de police de caractères :

0 = japonais ; 1 = Autre.

Les bits 4 à 6 indiquent le format de la date :

0 = AA/MM/JJ ; 1 = MM/JJ/AA ; 2 = JJ/MM/AA

Le bit 7 indique la fréquence des interruptions (VSYNC) :

0 = 60 Hertz ; 1 = 50 Hertz.

0002Ch = Les bits 0 à 3 indiquent le type de clavier :

0 = Japonais ; 1 = Amérique ; 2 = Français,

3 = Anglais ; 4 = Allemand ; 6 = Espagnol.

Les bits 4 à 7 indiquent le type de Basic :

0 = Japonais ; 1 = International.

Note : Les MSX coréens ne respectent pas la norme. Ils indiquent la police de caractères et le type de clavier comme étant japonais.

Tableau des différences par pays :

Pays	Standard TV	Format de la date	Mode d'écran initial	Symbole utilisés selon le type de Basic		
				Longueur de chaine	Remplacement	Monnaie
Angleterre	PAL (50Hz)	JJ/MM/AA	SCREEN0	%	&	£
Allemagne	PAL (50Hz)	JJ/MM/AA	SCREEN0	%	&	\$
Koweït, Egypt	PAL (50Hz)	JJ/MM/AA	SCREEN0/1	%	&	\$
Argentine	PAL (50Hz)	MM/JJ/AA	SCREEN0	%	&	\$
Brésil	PAL (60Hz)	JJ/MM/AA	SCREEN0	%	&	\$
Corée	NTSC (60Hz)	AA/MM/JJ	SCREEN1	%	@	₩
Espagne	PAL (50Hz)	MM/JJ/AA	SCREEN0	%	&	\$
France	SECAM (50Hz)	JJ/MM/AA	SCREEN0	%	&	\$
Japon	NTSC (60Hz)	AA/MM/JJ	SCREEN1	%	@	₩
Pays soviétique	NTSC (60Hz)	MM/JJ/AA	SCREEN0	%	&	\$
USA	NTSC (60Hz)	MM/JJ/AA	SCREEN0	%	&	\$

## 0002Dh\* (Main-ROM) **MSXVER** MSX VERsion

Rôle : Version du MSX. Cette octet contient 0 pour un MSX1, 1 pour un MSX2, 2 pour un MSX2+, 3 pour un MSX turbo R.

Note : Les autres numéros sont réservés pour les versions futures de MSX.

### 0002Eh\* (Main-ROM)

Rôle : Permet de savoir si l'interface Midi est en interne ou pas.

Le bit 0 de cette adresse est à 1 si le MSX-Midi intégrée.

Note : Les autres numéros sont réservés pour les versions futures de MSX.

### 0002Fh\* (Main-ROM)

Rôle : Réservé.

### 00030h (Main-ROM) **CALLF** CALL Far

Rôle : Appel inter-Slot à une adresse.

Entrée : Paramètres de la routine à appeler.

Sortie : Dépend de la routine appelée.

Modifie : AF, IX, IY, registres auxiliaires, et les registres modifiés par la routine appelée.

Notes : - CALLF diffère de CALSTL dans la manière d'appeler la routine. Pour appeler CALLF, il faut utiliser l'instruction RST 30 de la façon suivante.

```
rst 30
db Numéro du Slot (F000SSPP)
dw Adresse à appeler
```

- Les interruptions sont désactivées par la routine.

- L'appel à un autre Slot étendu du même Slot primaire que celui où se trouve le Bios n'est pas possible.

- Cette routine existe aussi sous MSX-DOS.

### 00038h (Main-ROM) **KEYINT** Encode KEYboard / timer INTerrupt routine

Rôle : Routine des interruptions masquables.

Entrée : Rien.

Sortie : Rien.

Modifie : Rien.

Notes : - Cette routine appelle les Hooks H.KEYI (0FD9Ah) et H.TIMI (0FD9Fh).

- Appel à la Sub-ROM pour les conversions Roma-Kana sur MSX2 ou plus récent.

- Cette routine est appelé à chaque interruption. (RST 38)

### 0003Bh (Main-ROM) **INITIO** INITialize Input/Output

Rôle : Initialiser les périphériques.

Entrée : Rien.

Sortie : Rien.

Modifie : Tout.

**0003Eh (Main-ROM)    INIFNK    INItialize FuNction Key**

Rôle :            Ré-initialisation des touches de fonction à leur valeur de départ.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Accès au VDP.

**00041h (Main-ROM)    DISSCR    DISable SCReen display**

Rôle :            Désactive l'affichage de l'écran.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        AF, BC.

**00044h (Main-ROM)    ENASCR    ENABle SCReen display**

Rôle :            Active l'affichage de l'écran.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        AF, BC.

**00047h (Main-ROM)    WRTVDP    WRiTe VDP**

Rôle :            Écriture dans un [registre de contrôle du VDP](#).  
Entrée :          C = Numéro du registre du VDP (0~7 sur MSX1 ; 0~23 et 32~46 sur MSX2 ; 0~25 et 32~46 sur MSX2+ et Turbo R).  
                    B = Valeur à écrire.  
Sortie :          Rien.  
Modifie :        AF, B.  
Note :            Appel à la Sub-ROM sur MSX2 ou plus récent si le bit EV du registre 0 du VDP est modifié ou si le numéro de registre est supérieur à 7.

**0004Ah (Main-ROM)    RDVRM    ReaD VRaM**

Rôle :            Lecture d'un octet en VRAM (00000h~03FFFh).  
Entrée :          HL = Adresse de la case mémoire à lire.  
Sortie :          A = Contenu de la case mémoire lue.  
Modifie :        AF.  
Note :            Routine MSX1. Pour lire une case mémoire de VRAM dont l'adresse est supérieure à 03FFFh, utilisez la routine NRDVRM (00174h en Main-ROM).

## 0004Dh (Main-ROM)    **WRTVRM**    WRiTe VRaM

- Rôle :            Écriture d'un octet en VRAM.
- Entrée :        HL = Adresse de la case mémoire (00000h~03FFFh).  
                  A = Donnée à écrire.
- Sortie :        Rien.
- Modifie :      Rien.
- Note :        Routine MSX1. Pour écrire un octet dans la VRAM à une adresse supérieure à 03FFFh, utiliser la routine NWRVRM (00177h en Main-ROM).

## 00050h (Main-ROM)    **SETRD**    SET address for ReaD

- Rôle :            Définit l'adresse en VRAM où doit s'effectuer la lecture de données.
- Entrée :        HL = Adresse source en VRAM (00000h~03FFFh).
- Sortie :        Rien.
- Modifie :      AF.
- Notes :        - Une fois l'adresse définie, chaque lecture au port de lecture de donnée du VDP incrémentera automatiquement l'adresse source dans le VDP.  
                  - Routine MSX1. Pour accéder à une adresse supérieure à 03FFFh, utiliser la routine NSETRD (0016Eh en Main-ROM).

Exemple :

```
;Lire 4 octets en VRAM

VDP.DR equ 0006h

        ld hl,0           ;Adresse en VRAM
        call SETWRT

        ld hl,DATA
        ld b,7            ;Longueur du texte
        ld a,(VDP.DR)     ;Port de lecture de donnée
        ld c,a

LOOP:   ini               ;Vous pouvez remplacer ces deux
        jr nz,LOOP        ;lignes par OTIR sur MSX2/2+/turbo R
        ret

DATA:   db 0,0,0,0        ;Les données lu sont stockées ici
```

## 00053h (Main-ROM)     **SETWRT**   SET address for WRiTe

- Rôle :            Défini l'adresse en VRAM doit s'effectuer l'écriture de données.
- Entrée :          HL = Adresse de destination en VRAM (00000h~03FFFh).
- Sortie :          Rien.
- Modifie :        AF.
- Notes :          - Une fois l'adresse définie, chaque écriture au port d'écriture de donnée du VDP incrémentera automatiquement l'adresse de destination dans le VDP.
- Routine MSX1. Pour accéder à une adresse supérieure à 03FFFh, utiliser la routine NSTWRT (00171h en Main-ROM).

### Exemple :

```
;Print "Hello!" at the top left of the screen (SCREEN1)

VDP.DW equ 0007h

        ld hl,(T32NAM) ;Characters table address in SCREEN1
        call SETWRT

        ld hl,TXT
        ld b,7          ;Longueur du text
        ld a,(VDP.DW)   ;Port d'écriture de donnée
        ld c,a

HELLO:   outi            ;Vous pouvez replacer ces deux
        jr nz,HELLO     ;lignes par OTIR sur MSX2/2+/turbo R
        ret

TXT:     db "Hello!"
```

## 00056h (Main-ROM)     **FILVRM**   FILl VRaM

- Rôle :            Remplissage de la mémoire vidéo avec une donnée.
- Entrée :          HL = Adresse de début (00000h~03FFFh)
- BC = Longueur.
- A = Donnée.
- Sortie :          Rien.
- Modifie :        AF, BC.
- Note :            Routine MSX1. Pour accéder aux adresses supérieures à 03FFFh, utiliser la routine BIGFIL (0016Bh en Main-ROM).

### 00059h (Main-ROM)     **LDIRMV**    LoAD Increment Repeat Memory with Vram

Rôle :            Transfert de la mémoire vidéo vers la mémoire centrale.

Entrée :          HL = Adresse de départ en VRAM (00000h~03FFFh ou 0FFFFh).  
                    DE = Adresse de destination en RAM centrale (00000h~0FFFFh).  
                    BC = Longueur du bloc à transférer.

Sortie :          Rien.

Modifie :        Tout.

Note :            Dans les modes d'écrans autres que MSX1. La variable ACPAGE (0FAF6h) indique la page depuis laquelle le transfert doit s'effectuer.

### 0005Ch (Main-ROM)     **LDIRVM**    LoAD Increment Repeat Vram with Memory

Rôle :            Transfert de la mémoire centrale vers la mémoire vidéo.

Entrée :          HL = Adresse de départ en RAM centrale (00000h~0FFFFh).  
                    DE = Adresse de destination en VRAM (00000h~03FFFh ou 0FFFFh).  
                    BC = Longueur du bloc à transférer.

Sortie :          Rien.

Modifie :        Tout.

Note :            Dans les modes d'écrans autres que MSX1. La variable ACPAGE (0FAF6h) indique la page dans laquelle le transfert doit s'effectuer.

### 0005Fh (Main-ROM)     **CHGMOD** CHange MODE

Rôle :            Changement de mode d'écran (SCREEN X).

Entrée :          A = Mode d'écran désiré (0~3 ou 0~8 / 12, 9 seulement sur les MSX2 coréens).

Sortie :          SCRMOD (0FCAFh) = Nouveau mode d'écran.

Modifie :        Tout.

Note :            La palette n'est pas initialisée par cette routine, appelez la routine CHGMDP (001B5h en Sub-ROM) si vous avez besoin de l'initialisation de la palette.

## 00062h (Main-ROM)     **CHGCLR**   CHanGe CoLoR

- Rôle :            Change les couleurs de l'écran spécifié avec les valeurs par défaut.
- Entrée :          A = Mode d'écran.
- FORCLR (0F3E9h) = Couleur de texte.
- BAKCLR (0F3EAh) = Couleur de fond.
- BDRCLR (0F3EBh) = Couleur de bordure.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - En mode graphique, seule la couleur de bordure change.
- Même fonction que CHGCLR (00111h) de la Sub-ROM.

## 00066h (Main-ROM)     **NMI**   Non Maskable Interrupt

- Rôle :            Routine d'interruptions non-masquables.
- Entrée :          Rien.
- Sortie :          Rien.
- Modifie :        Rien.
- Notes :          - Appel au Hook H.NMI (0FDD6h).
- Interruption inutilisée par les MSX.

## 00069h (Main-ROM)     **CLRSPPR**   CLear Sprites

- Rôle :            Initialisation des Sprites.
- Entrée :          Rien.
- Sortie :          Rien.
- Modifie :        Tout.
- Note :            Les valeurs de la table des formes des Sprites sont mises à zéro, les numéros d'affichage de Sprite sont initialisés avec les valeurs de 0 à 31 (si Sprites 16x16) ou 0 à 255 (si 8x8), la couleur des Sprites prend celle défini par FORCLR (0F3E9h) et l'ordonnée des Sprites est réglée à 209 ou 217 selon la hauteur l'écran (192 ou 212 pixels).

## 0006Ch (Main-ROM)    **INITXT**    INItialize TeXT mode

- Rôle :            Initialisation du mode SCREEN 0.
- Entrée :          TXTNAM (0F3B3h) = Adresse de la table des caractères.  
                  TXTCOL (0F3B5h) = Adresse de la table des couleurs.  
                  TXTCGP (0F3B7h) = Adresse de la table des formes.  
                  TXTATR (0F3B9h) = Adresse de la table des formes.  
                  TXTCOL (0F3B5h) = Adresse de la table des couleurs.  
                  TXTPAT (0F3BBh) = Adresse de la table des caractères.  
                  LINL40 (0F3AEh) = Nombre de caractères par lignes.  
                  FORCLR (0F3E9h) = Couleur de texte.  
                  BAKCLR (0F3EAh) = Couleur de fond.  
                  BDRCLR (0F3EBh) = Couleur de bordure.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir TXTNAM, etc. Le MSX initialise leur valeur à l'allumage.  
                  - Il s'agit d'une routine MSX1. La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT (00141h) de la Sub-ROM juste après celle-ci.

## 0006Fh (Main-ROM)    **INIT32**    INItialize Text 32 mode

- Rôle :            Initialisation du mode SCREEN 1.
- Entrée :          T32NAM (0F3BDh) = Adresse de la table des caractères.  
                  T32COL (0F3BFh) = Adresse de la table des couleurs.  
                  T32CGP (0F3C1h) = Adresse de la table des formes.  
                  T32ATR (0F3C3h) = Adresse de la table des attributs de Sprites.  
                  T32PAT (0F3C5h) = Adresse de la table des formes de Sprites.  
                  LINL32 (0F3AFh) = Nombre de caractères par lignes.  
                  FORCLR (0F3E9h) = Couleur de texte.  
                  BAKCLR (0F3EAh) = Couleur de fond.  
                  BDRCLR (0F3EBh) = Couleur de bordure.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir T32NAM, etc. Le MSX initialise leur valeur à l'allumage.  
                  - La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT (00141h en Sub-ROM) juste après celle-ci.



## 00072h (Main-ROM)    **INIGRP**    INItialize GRaPhic mode

- Rôle :            Initialisation du mode SCREEN 2.
- Entrée :          GRPNAM (0F3C7h) = Adresse de la table des caractères graphiques.  
                 GRPCOL (0F3C9h) = Adresse de la table des couleurs.  
                 GRPCGP (0F3CBh) = Adresse de la table des formes.  
                 GRPATR (0F3CDh) = Adresse de la table des attributs de Sprites.  
                 GRPPAT (0F3CFh) = Adresse de la table des formes de Sprites.  
                 FORCLR (0F3E9h) = Couleur de texte ou de tracé.  
                 BAKCLR (0F3EAh) = Couleur de fond.  
                 BDRCLR (0F3EBh) = Couleur de bordure.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir GRPNAM, etc. Le MSX initialise leur valeur à l'allumage.  
                 - La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT (00141h) en Sub-ROM juste après celle-ci.

## 00075h (Main-ROM)    **INIMLT**    INItialize MuLTicolor mode

- Rôle :            Initialisation du mode SCREEN 3.
- Entrée :          MLTNAM (0F3D1h) = Adresse de la table des caractères.  
                 MLTCOL (0F3D3h) = Adresse de la table des couleurs.  
                 MLTCGP (0F3D5h) = Adresse de la table des formes.  
                 MLTATR (0F3D7h) = Adresse de la table des attributs de Sprites.  
                 MLTPAT (0F3D9h) = Adresse de la table des formes de Sprites.  
                 FORCLR (0F3E9h) = Couleur de texte ou de tracé.  
                 BAKCLR (0F3EAh) = Couleur de fond.  
                 BDRCLR (0F3EBh) = Couleur de bordure.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir MLTNAM, etc. Le MSX initialise leur valeur à l'allumage.  
                 - La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT (00141h) en Sub-ROM juste après celle-ci.

### 00078h (Main-ROM)    **SETTXT**    SET TeXT mode

- Rôle :            Passage direct en mode SCREEN 0 sans initialiser le contenu des tables.
- Entrée :          TXTNAM (0F3B3h) = Adresse de la table des caractères.  
                    TXTCGP (0F3B7h) = Adresse de la table des formes.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir TXTNAM et TXTCGP.  
                    Le MSX initialise leur valeur à l'allumage.

### 0007Bh (Main-ROM)    **SETT32**    SET Text 32 mode

- Rôle :            Passage direct en mode SCREEN 1 sans initialiser le contenu des tables.
- Entrée :          T32NAM (0F3BDh) = Adresse de la table des caractères.  
                    T32COL (0F3BFh) = Adresse de la table des couleurs.  
                    T32CGP (0F3C1h) = Adresse de la table des formes.  
                    T32ATR (0F3C3h) = Adresse de la table des attributs de Sprites.  
                    T32PAT (0F3C5h) = Adresse de la table des formes de Sprites.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir T32NAM, etc. Le MSX initialise leur valeur à l'allumage.

### 0007Eh (Main-ROM)    **SETGRP**    SET GRaPhic mode

- Rôle :            Passage direct en mode SCREEN 2 sans initialiser le contenu des tables.
- Entrée :          GRPNAM (0F3C7h) = Adresse de la table Bitmap.  
                    GRPCOL (0F3C9h) = Adresse de la table des couleurs.  
                    GRPCGP (0F3CBh) = Adresse de la table des formes.  
                    GRPATR (0F3CDh) = Adresse de la table des attributs de Sprites.  
                    GRPPAT (0F3CFh) = Adresse de la table des formes de Sprites.
- Sortie :          Rien.
- Modifie :        Tout.
- Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir GRPNAM, etc. Le MSX initialise leur valeur à l'allumage.

### 00081h (Main-ROM)     **SETMLT**     SET MuLTicolor mode

Rôle :            Passage direct en mode SCREEN3 sans initialiser le contenu des tables.

Entrée :          MLTNAM (0F3D1h) = Adresse de la table des caractères.  
                    MLTCOL (0F3D3h) = Adresse de la table des couleurs.  
                    MLTCGP (0F3D5h) = Adresse de la table des formes.  
                    MLTATR (0F3D7h) = Adresse de la table des attributs de Sprites.  
                    MLTPAT (0F3D9h) = Adresse de la table des formes de Sprites.

Sortie :          Rien.

Modifie :        Tout.

Notes :          - Dans la plupart des cas, il n'est pas nécessaire de définir MLTNAM à MLTPAT.  
                    Le MSX initialise leur valeur à l'allumage.

### 00084h (Main-ROM)     **CALPAT**     CALculate PATtern table address

Rôle :            Donne l'adresse du début de la forme d'un Sprite dans la table des formes.

Entrée :          A = Numéro du Sprite.

Sortie :          HL = Adresse cherchée.

Modifie :        AF, DE, HL.

### 00087h (Main-ROM)     **CALATR**     CALculate ATtRIBUTE table address

Rôle :            Donne l'adresse du début des attributs du Sprite indiqué.

Entrée :          A = Numéro du Sprite.

Sortie :          HL = Adresse cherchée.

Modifie :        AF, DE, HL.

### 0008Ah (Main-ROM)     **GSPSIZ**     Get Sprite SIZE

Rôle :            Indique la taille actuelle de la matrice des Sprites.

Entrée :          Rien.

Sortie :          A = Largeur de la matrice (8 ou 16).  
                    F = L'indicateur C est mis à 1 si la taille des Sprites est 16x16 ; 0 si 8x8.

Modifie :        AF.

## 0008Dh (Main-ROM)    **GRPPRT**    GRaPhic PRinT

Rôle :            Affichage d'un caractère dans les mode SCREEN 2 à 12.

Entrée :           A = Code ASCII du caractère à afficher.

                 CLOC (0F92Ah) = Dans les SCREEN 2 à 4 et 10 à 12, adresse où se trouve le curseur graphique en VRAM. Dans les SCREEN 5 à 8, abscisse actuelle du curseur graphique.

                 CMASK (0F92Ch) = Dans les SCREEN 2 à 4 et 10 à 12, masque à appliquer. Dans les SCREEN 5 à 8, ordonnée actuelle du curseur graphique.

                 LOGOPR (0FB02h) = Code d'opération logique (pour les modes graphiques de SCREEN 5 à 12).

Sortie :           Rien.

Modifie :          Rien.

Note :            Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

## 00090h (Main-ROM)    **GICINI**    GI sound Chip INitalize

Rôle :            Initialisation du PSG et des données de l'instruction PLAY.

Entrée :           Rien.

Sortie :           Rien.

Modifie :          Rien.

Notes :           - Les interruptions doivent être interdites avant l'appel de cette routine.

                 - Après initialisation, les registres auront les valeurs suivantes.

                 R#0 = 01010101    R#1 = 00000000    R#2 = 00000000

                 R#3 = 00000000    R#4 = 00000000    R#5 = 00000000

                 R#6 = 00000000    R#7 = 10111000    R#8 = 00000000

                 R#9 = 00000000    R#10 = 00000000    R#11 = 00001011

                 R#12 = 00000000    R#13 = 00000000

## 00093h (Main-ROM) **WRTPSG** WRiTe PSG

Rôle : Écriture dans les registres du PSG (« Programmable Sound Generator »).

Entrée : A = Numéro de registre (0~15).

E = Donnée à écrire.

Sortie : Rien.

Modifie : Rien.

Notes : - Voir le [chapitre 8 sur PSG](#) pour plus de détails sur les registres.

- Les interruptions sont coupées puis réactivées par la routine.

- Voici une petite descriptions des registres de contrôle du PSG :

Registre	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
R#0	8 bits de poids faible de la fréquence de la voix 1							
R#1	-	-	-	-	4 bits forts de la fréquence voix 1			
R#2	8 bits de poids faible de la fréquence de la voix 2							
R#3	-	-	-	-	4 bits forts de la fréquence voix 2			
R#4	8 bits de poids faible de la fréquence de la voix 3							
R#5	-	-	-	-	4 bits forts de la fréquence voix 3			
R#6	-	-	-	Fréquence du générateur de bruit blanc				
R#7	Ports d'E/S du PSG		Désactivation du bruit			Désactivation du son		
	B = 1	A = 0	voix 3	voix 2	voix 1	voix 3	voix 2	voix 1
R#8	-	-	-	V/A	Volume / Amplitude de la voix 1			
R#9	-	-	-	V/A	Volume / Amplitude de la voix 2			
R#10	-	-	-	V/A	Volume / Amplitude de la voix 3			
R#11	8 bits de poids faible de la période de l'enveloppe (T)							
R#12	8 bits de poids fort de la période de l'enveloppe (T)							
R#13	-	-	-	-	Forme de l'enveloppe			
R#14	Port A d'E/S du PSG (à régler toujours en entrée avec le bit 6 du R#7)							
R#15	Port B d'E/S du PSG (à régler toujours en sortie avec le bit 7 du R#7)							

## 00096h (Main-ROM) **RDPSG** ReaD PSG

Rôle : Lecture d'un registre du PSG.

Entrée : A = Numéro du registre.

Sortie : A = Contenu du registre.

Modifie : Rien.

Notes : - Voir le [chapitre 8 sur PSG](#) pour plus de détails sur les registres.

- La routine ne coupe pas les interruptions.

### 00099h (Main-ROM)    **STRTMS**    STarT background MuSic task

Rôle :            Joue le MML pour PLAY en tache de fond.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Ne joue pas si un MML est déjà en cours de lecture.

### 0009Ch (Main-ROM)    **CHSNS**

Rôle :            Mise à jour de l'affichage des touches de fonction d'une part, et vérification du cache du clavier d'autre part.  
Entrée :          Rien.  
Sortie :          F = L'indicateur Z passe à 0 si il y a un caractère dans le cache du clavier.  
Modifie :        AF.

### 0009Fh (Main-ROM)    **CHGET**    CHaracter GET

Rôle :            Attente d'un caractère tapé au clavier et retour avec son code.  
Entrée :          Rien.  
Sortie :          A = Code ASCII du caractère.  
                    CSRX (0F3DCh) = Abscisse du curseur.  
                    CSRY (0F3DDh) = Ordonnée du curseur.  
Modifie :        AF.  
Notes :          - Appel au Hook H.CHGE (0FDC2h).  
                    - Les variables SCNCNT (0F3F7h) et REPCNT (0F3F7h) permettent de régler les intervalles de scrutations du clavier.

### 000A2h (Main-ROM)    **CHPUT**    CHaracter outPUT

Rôle :            Sortie d'un caractère sur écran en mode texte.  
Entrée :          A = Code du caractère.  
                    CSRX (0F3DCh) = Abscisse du curseur.  
                    CSRY (0F3DDh) = Ordonnée du curseur.  
Sortie :          Rien.  
Modifie :        Rien.  
Note :            Appel au Hook H.CHPU (0FDA4h).

### 000A5h (Main-ROM)    **LPTOUT**    Line PrinTer OUT

Rôle :            Sortie d'un caractère sur imprimante.  
Entrée :          A = Code du caractère.  
Sortie :          F = Indicateur C sera à 1 si la routine a été interrompue par un CTRL+STOP.  
Modifie :        F.  
Notes :          - Appel au Hook H.LPTO (0FFB6h).  
                    - CTRL+STOP permet de sortir de cette routine en cas de problème avec l'imprimante.

### 000A8h (Main-ROM)    **LPTSTT**    Line PrinTer STaTus

Rôle :            Vérification de l'état de l'imprimante.  
Entrée :          Rien.  
Sortie :          A = 255 si l'imprimante est prête, 0 si elle ne l'est pas.  
                    F = Indicateur Z à si l'imprimante est prête, 1 si elle ne l'est pas.  
Modifie :        AF.  
Note :            Appel au Hook H.LPTS (0FFBBh)

### 000ABh (Main-ROM)    **CNVCHR**    CoNVert CHaRacter

Rôle :            Permet de gérer les caractères obtenus avec la routine CHGET (0009Fh) pour une éventuelle conversion des caractères graphiques étendus.  
Entrée :          A = Code d'un caractère ou 1 pour annoncer un caractère graphique étendu.  
Sortie :          GRPHED (0FCA6h) est mis à 1 lorsque le code entrée est 1.  
                    F = Si l'indicateur C est à 0, c'est que le code entrée est 1.  
                    Les indicateurs C et Z sont à 1 lorsque que le code est un caractère étendu.  
                    L'indicateur C est à 1 et Z à 0 lorsque que le code un caractère ordinaire.  
Modifie :        AF.  
Notes :          - GRPHED (0FCA6h) est pris en compte pour indiquer si il s'agit d'un caractère graphique étendu (compris entre 65 et 95) ou pas.  
                    - Voir les [jeux de caractères MSX](#) pour connaître les caractères graphiques correspondants.

### 000AEh (Main-ROM)    **PINLIN**    Program INput LINE

- Rôle : Place le curseur à la ligne suivante puis stocke les caractères entrés au clavier dans le cache BUF jusqu'à ce que la touche STOP ou RET soit pressés.
- Entrée : Rien.
- Sortie : HL = 0F55Eh  
F = Carry est mis à 1 si c'est la touche STOP qui a été pressée.  
Le cache BUF (0F55Eh) contient les caractères entrés au clavier.  
La table LINTTB (0FBB2h) est actualisée.
- Modifie : Tout.
- Notes : - Appel au Hook H.PINL (0FDDBh)  
- Si AUTFLG (0F6AAh) est à 1, appellera la routine INLIN (000B1h) à la place.

### 000B1h (Main-ROM)    **INLIN**    INput LINE

- Rôle : Stocke les caractères entrés au clavier dans le cache BUF jusqu'à ce que la touche STOP ou RET soit pressée.
- Entrée : Rien.
- Sortie : HL = 0F55Eh  
F = Carry est mis à 1 si c'est la touche STOP qui a été pressée.  
Le cache BUF (0F55Eh) contient les caractères entrés au clavier .  
La table LINTTB à FBB2h est actualisée.
- Modifie : Tout.
- Notes : - Appel au Hook H.INLI (0FDE5h).  
- Par rapport à PINLIN, cette routine ne prend en compte que les caractères placés derrière de curseur. L'invite est utilisable.

### 000B4h (Main-ROM)    **QINLIN**    Question mark and INput LINE

- Rôle : Idem INLIN sauf que cette routine affiche un point d'interrogation.
- Entrée : Rien.
- Sortie : HL = Adresse du premier octet dans le cache -1.  
F = Indicateur C à 1 si un ^C a eu lieu.
- Modifie : Tout.
- Note : Appel au Hook H.QINL (0FDE0h).



## 000B7h (Main-ROM)    **BREAKX**    BREAK X

Rôle :            Test du CTRL+STOP  
Entrée :          Rien.  
Sortie :          F = Indicateur C sera à 1 si ^C en cours.  
Modifie :        AF

Exemple d'utilisation :

```
LOOP: call BREAKX  
      jr nc, LOOP  
      ret
```

## 000BAh (Main-ROM)    **ISCNTC**    IS CoNTrol C ?

Rôle :            Test de la touche STOP ignoré par l'interpréteur Basic.  
Entrée :          BASROM (0FBB1h) = 1 si le programme Basic est en cartouche (touche STOP ignorée).  
Sortie :          Rien.  
Modifie :        Rien.  
Note :            Cette routine est utilisé par l'interpréteur Basic.

## 000BDh (Main-ROM)    **CKCNTC**    ChecK CoNtrol C

Rôle :            Test de la touche STOP ignoré par l'interpréteur Basic.  
Entrée :          BASROM (0FBB1h) = 1 si programme en cartouche (touche STOP ignorée).  
Sortie :          Rien.  
Modifie :        Rien.  
Notes :           - Cette routine est utilisé par l'interpréteur Basic.  
                  - Par rapport à ISCNTC, cette routine met le pointeur de texte du Basic TEMPPT (0F678h) à 0 afin d'interdire la reprise de l'exécution du programme avec CONT (« CONTinue »).

## 000C0h (Main-ROM)    **BEEP**        BEEP buzzer

Rôle :            Émission d'un bref « bip » sonore.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Appel à la Sub-ROM sur MSX2 ou plus récent.  
Note :            Ce son est produit par le PSG.

**000C3h (Main-ROM)    CLS    CLear Screen**

Rôle :            Effacement de l'écran dans tous les modes.  
Entrée :          F = Indicateur Z à 1.  
Sortie :           Rien.  
Modifie :        AF, BC.  
Note :            Appel à la Sub-ROM sur MSX2 ou plus récent.

**000C6h (Main-ROM)    POSIT    POSITion cursor**

Rôle :            Modification des coordonnées du curseur  
Entrée :          H = Numéro de colonne  
                    L = Numéro de ligne  
Sortie :           Rien.  
Modifie :        AF.  
Note :            Bien que n'ayant pas été modifiée sur MSX2, cette routine fonctionne parfaitement en mode 80 colonnes.

**000C9h (Main-ROM)    FNKSB    FuNction Key diSplay enaBled**

Rôle :            Active ou désactive les touches de fonction.  
Entrée :          CNSDFG (0F3DEh) = 0 pour ignorer les touches de fonction, 1 pour les activer.  
Sortie :           Rien.  
Modifie :        Tout.  
Notes :           - Appel au Hook H.DSPF (0FDB3h).  
                    - Appeler la routine DSPFNK (000CFh en Main-ROM) pour afficher le texte des touches de fonction.

**000CCh (Main-ROM)    ERAFNK    ERAse FuNction Key**

Rôle :            Efface l'affichage des touches de fonction (en mode texte).  
Entrée :          Rien.  
Sortie :           Rien.  
Modifie :        Tout.  
Note :            Appel au Hook H.ERAF (0FDB8h).

### 000CFh (Main-ROM)    **DSPFNK**    DiSPlay FuNction Key

Rôle :            Affichage des touches de fonction en mode texte.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Appel au Hook H.DSPF (0FDB3h).

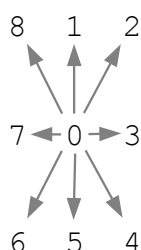
### 000D2h (Main-ROM)    **TOTEXT**    force screen TO TEXT mode

Rôle :            Retour au mode texte précédant le passage en mode graphique.  
Entrée :          OLDSCR (0FCB0h) = Mode texte dans lequel nous étions avant le passage en mode graphique. Nous retournons dans ce mode.  
Sortie :          Rien.  
Modifie :        Tout.  
Notes :          - Appel au Hook H.TOTE (0FDBDh).  
                  - Cette routine fait appel à CHGMDP (001B5h en Sub-ROM) sur MSX2 ou plus récent.

### 000D5h (Main-ROM)    **GTSTCK**    GeT joySTiCK status

Rôle :            Renvoi de la direction actuelle de la manette.  
Entrée :          A = Numéro de manette (0 ~ 2).  
                  0 = Touches du curseur du clavier.  
                  1 = manette 1.  
                  2 = manette 2.  
Sortie :          A = Direction de manette (0~8).

Format :



Lors de l'utilisation du clavier comme joystick 0, il faut appuyer simultanément sur deux flèches afin d'obtenir une diagonale. Par exemple, les touches **BAS** et **DROITE** donnent 4 lorsqu'elles sont pressées.

Modifie :        Tout.

000D8h (Main-ROM)    **GTTRIG**    GeT TRIGger button status

Rôle :            Renvoi de l'état du bouton de manette sélectionné.

Entrée :          A = Numéro du bouton (0~4).

0 = Barre d'espacement du clavier ;

1 = Bouton A de la manette 1 ;

2 = Bouton A de la manette 2 ;

3 = Bouton B de la manette 1 ;

4 = Bouton B de la manette 2.

Sortie :          A = 255 si le bouton est enfoncé, 0 si ce n'est pas le cas.

Modifie :        AF, BC.

## 000DBh (Main-ROM)    **GTPAD**    GeT touch PAD

- Rôle : Lecture du rayon optique, de la souris, de la tablette graphique ou du Track-Ball.
- Entrée : A = Numéro de l'opération à effectuer (0~7 sur MSX1 ou 0~19 sur les autres). Pour chaque périphérique, la première opération (0, 4, 8, 12 ou 16) est indispensable car c'est elle qui lit les coordonnées fournies par le périphérique et les sauvegarde dans les variables systèmes XSAVE (0FAFEh) et YSAVE (0FB00h). Les opérations de lecture de coordonnées ne font que lire ces variables.
- Sortie : A = Valeur résultante.
- Modifie : Tout.
- Notes :  
 - La routine GTPAD fait appel à la routine NEWPAD de la Sub-ROM. (MSX2~)  
 - Voici la liste des opérations possibles :

Entrée	Port	Valeur résultante
<b><u>0</u></b>	Général n°1	A = 0FFh si le stylet est en contact avec la tablette graphique, les coordonnées sont donc lisible. Sinon A = 0.
<b><u>1</u></b>		Abscisse de la tablette.
<b><u>2</u></b>		Ordonnée de la tablette.
<b><u>3</u></b>		0FFh si le stylet appuie sur la tablette, sinon 0.
<b><u>4</u></b>	Général n°2	A = 0FFh si le stylet est en contact avec la tablette graphique, les coordonnées sont donc lisible. Sinon A = 0.
<b><u>5</u></b>		Abscisse de la tablette.
<b><u>6</u></b>		Ordonnée de la tablette.
<b><u>7</u></b>		0FFh si le stylet appuie sur la tablette, sinon 0.
<b><u>8</u></b>	Crayon optique	A = 0FFh si le crayon optique touche l'écran, les coordonnées sont donc lisible. Sinon A = 0.
<b><u>9</u></b>		Abscisse du crayon optique.
<b><u>10</u></b>		Ordonnée du crayon optique.
<b><u>11</u></b>		0FFh si le bouton du crayon optique est pressé, sinon 0.
<b><u>12</u></b>	Général n°1	Toujours 0FFh. Lire les coordonnées juste après cet opération.
<b><u>13</u></b>		Abscisse de la souris ou du Track-Ball.
<b><u>14</u></b>		Ordonnée de la souris ou du Track-Ball.
<b><u>15</u></b>		Toujours 0. (inutilisé)
<b><u>16</u></b>	Général n°2	Toujours 0FFh. Lire les coordonnées juste après cet opérations.
<b><u>17</u></b>		Abscisse de la souris ou du Track-Ball.
<b><u>18</u></b>		Ordonnée de la souris ou du Track-Ball.
<b><u>19</u></b>		Toujours 0. (inutilisé)

- Pour tester l'état des boutons de la souris ou du Track-Ball, veuillez utiliser la routine GTTRIG (000DBh en Main-ROM).
- Le crayon optique n'est pas géré par le MSX turbo R. Si on entre A = 8~11 en entrée, on aura A = 0 en sortie.

- La tablette graphique doit être une NEC PC-6051 ou compatible.

## 000DEh (Main-ROM)    **GTPDL**    GeT PaDdLe

Rôle :            Lecture d'une molette (« Paddle Controler ») ASCII.

Entrée :        A = Paramètre d'entrée (1~12).

	Port général 1	Port général 2
Molette A	1	2
Molette B	3	4
Molette C	5	6
Molette D	7	8
Molette E	9	10
Molette F	11	12

Sortie :        A = Valeur en fonction de l'angle de rotation (0~255).

Modifie :      Tout.

Notes :        - Les molettes du jeu Arkanoïd ne sont pas prises en charge par cette routine.  
                  - Cette routine n'a pas d'effet sur MSX turbo R. (A = 0 en sortie)

### 000E1h (Main-ROM)    **TAPION**    TAPe In ON

- Rôle :            Démarrage du moteur du magnétophone et lecture d'un signal d'entête ou d'un signal séparateur.
- Entrée :          Rien.
- Sortie :          F = Indicateur C à 1 si opération interrompue (après une erreur par exemple).
- Modifie :        Tout.
- Notes :          - Voir TAPoon (000EAh en Main-ROM) pour les explications du signal d'entête et du signal séparateur.
- La vitesse (1200 ou 2400 bauds) est déterminée lors de la lecture du signal.
- Cette routine n'a pas d'effet sur MSX turbo R. (l'indicateur C = 1 en sortie)

### 000E4h (Main-ROM)    **TAPIN**        TAPe IN

- Rôle :            Lecture d'un octet depuis la cassette.
- Entrée :          Rien.
- Sortie :          A = Octet lu.
- F = Indicateur C à 1 si opération interrompue.
- Modifie :        Tout.
- Note :            Cette routine n'a pas d'effet sur MSX turbo R. (l'indicateur C = 1 en sortie)

### 000E7h (Main-ROM)    **TAPIOF**    TAPe In OFf

- Rôle :            Fin de lecture, arrêt moteur.
- Entrée :          Rien.
- Sortie :          Rien.
- Modifie :        Rien.
- Note :            Cette routine n'a pas d'effet sur MSX turbo R. (l'indicateur C = 1 en sortie)

## 000EAh (Main-ROM) **TAPOON** TAPe Out ON

- Rôle : Démarrage du moteur du magnétophone et écriture d'un signal d'entête ou d'un signal séparateur.
- Entrée : A = 0 pour écrire un signal un séparateur, 1 pour un signal d'entête.
- Sortie : F = Indicateur C à 1 si l'opération a été interrompue (après un CTRL+STOP par exemple).
- Modifie : Tout.
- Notes : - Le signal d'entête sert à marquer le début d'un fichier alors que l'autre signal sert à séparer les différentes parties d'un fichier.

Le MSX reconnaît trois types de fichiers sur cassettes : Les fichiers Basic, les fichiers de texte ASCII et les binaires. Voici le détail de chacun d'eux :

Format d'un fichier Basic :

Début du fichier	Signal d'entête (de 6.7 sec)							
Entête du fichier	0D3h	0D3h	0D3h	0D3h	0D3h	0D3h	0D3h	0D3h
	0D3h	0D3h	Nom du fichier (6 octets)					
	Espace vide de durée quelconque							
	Signal séparateur (de 1.7 sec)							
.	Début...  Programme Basic condensé  ...fin							
.								
.								
.								
Fin du fichier	000h	000h	000h 000h 000h 000h 000h					

Le fichier Basic commence par un signal d'entête suivi de dix octets 0D3h pour indiquer qu'il contient un programme Basic condensé et du nom du fichier. Après cette entête, il doit y avoir un séparateur qui annonce l'emplacement du programme. Le fichier se finit par une suite de sept 000h.

Format d'un fichier binaire :

Début du fichier	Signal d'entête (de 6.7 sec)							
Entête du fichier	0D0h	0D0h	0D0h	0D0h	0D0h	0D0h	0D0h	0D0h
	0D0h	0D0h	Nom du fichier (6 octets)					
	Espace vide de durée quelconque							
	Signal séparateur (de 1.7 sec)							
	Adrs. début		Adrs. de fin		Adrs. Execut.		Début...	
. . . . . .	Données binaires (Programme machine ou autre)							
Fin du fichier			...fin					

Le fichier binaire commence par un signal d'entête suivi de dix octets 0D0h indiquant qu'il s'agit d'un programme machine et du nom du fichier. Après cette entête, il doit y avoir un séparateur qui annonce l'adresse de début du programme, l'adresse de fin, l'adresse d'exécution. Ensuite, viennent les données.

Format d'un fichier ASCII :



Début du fichier	Signal d'entête (de 6.7 sec)							
Entête du fichier	0EAh	0EAh	0EAh	0EAh	0EAh	0EAh	0EAh	0EAh
	0EAh	0EAh	Nom du fichier (6 octets)					
	Espace vide de durée quelconque							
	Signal séparateur (de 1.7 sec)							
.	Début...							
	256 caractères de texte ASCII							
	Signal séparateur (de 1.7 sec)							
	256 caractères de texte ASCII							
	.							
	.							
Fin du fichier	Signal séparateur (de 1.7 sec)							
	Reste des caractères				...fin			

Le fichier ASCII commence par un signal d'entête suivi de dix octets 0EAh pour indiquer qu'il contient de texte ASCII puis, du nom du fichier. Après cette entête, il doit y avoir un séparateur qui annonce l'emplacement de 256 caractères de texte ASCII. Si le texte dépasse les 256 caractères, il devra y avoir un signal séparateur entre chaque 256 caractères.

- Cette routine n'a pas d'effet sur MSX turbo R. (l'indicateur C = 1 en sortie)

## 000EDh (Main-ROM) **TAPOUT** TAPe OUT

Rôle : Écriture d'un octet sur la cassette.  
 Entrée : A = Octet à écrire.  
 Sortie : F = Indicateur C à 1 si l'opération a été interrompue. (toujours à 1 sur MSX Turbo R.)  
 Modifie : Tout.  
 Note : Cette routine n'a pas d'effet sur MSX turbo R. (l'indicateur C = 1 en sortie)

## 000F0h (Main-ROM) **TAPOOFF** TAPe Out OFF

Rôle : Fin d'écriture, arrêt du moteur.  
 Entrée : Rien.  
 Sortie : Rien.  
 Modifie : Rien.  
 Notes: - Rétabli les interruptions.  
 - Cette routine n'a pas d'effet sur MSX turbo R. (l'indicateur C = 1 en sortie)

**000F3h (Main-ROM)    STMOTR    SeT MOTor**

Rôle :            Changement d'état du moteur du magnétophone.  
Entrée :          A = 0 pour arrêter le moteur.  
                    1 pour démarrer le moteur.  
                    255 pour inverser l'état actuel.  
Sortie :          Rien.  
Modifie :        AF.  
Note :            Cette routine n'a pas d'effet sur MSX turbo R. Elle modifie rien.

**000F6h (Main-ROM)    LFTQ          LeFT in Queue**

Rôle :            Donne le nombre d'octets restants dans une queue.  
Entrée :          A = Numéro de queue.  
Sortie :          A = Nombre d'octets.  
Modifie :        AF, BC, HL.

**000F9h (Main-ROM)    PUTQ          PUT in Queue**

Rôle :            Empilage d'un octet sur une queue.  
Entrée :          A = Numéro de queue.  
                    E = Donnée à mettre.  
Sortie :          F = Indicateur zéro à 1 si la queue est entièrement remplie.  
Modifie :        AF, BC, HL.

### 000FCh (Main-ROM)    **RIGHTC**    RIGHT Current

Rôle : Déplacement du curseur graphique d'un pixel vers la droite.  
Entrée : Rien.  
Sortie : Rien.  
Modifie : AF, CLOC (0F92Ah) et éventuellement CMASK (0F92Ch).  
Note : Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

Exemple en Basic :

```
10 SCREEN 2
20 PSET (0,0),1
30 X=PEEK(&HF92A)+PEEK(&HF92B)*256: Y=PEEK(&HF92C)
40 DEFUSR=&HFC: A=USR(0) ' Appelle RIGHTC
40 X=PEEK(&HF92A)+PEEK(&HF92B)*256:Y=PEEK(&HF92C)
50 SCREEN 0
60 PRINT"X=";X,"Y=";Y,"X2=";X2,"Y2=";Y2
```

Ce programme affiche : « X=0    Y=128    X2=0    Y2=64 »  
(en binaire, 128 = 10000000B et 64 = 01000000B)

### 000FFh (Main-ROM)    **LEFTC**    LEFT Current

Rôle : Déplacement du curseur graphique d'un pixel vers la gauche.  
Entrée : Rien.  
Sortie : Rien.  
Modifie : AF, CLOC (0F92Ah) et éventuellement CMASK (0F92Ch).  
Notes : - Passe à la fin de la ligne précédente sur l'abscisse du curseur était 0.  
          - Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 00102h (Main-ROM)    **UPC**    UP Current

Rôle : Déplacement du curseur graphique d'un pixel vers le haut.  
Entrée : Rien.  
Sortie : Rien.  
Modifie : AF et éventuellement CLOC (0F92Ah).  
Note : Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 00105h (Main-ROM)    **TUPC**    Test and UP Current

Rôle : Déplacement du curseur d'un pixel vers le haut avec test de dépassement d'écran.  
Entrée : Rien.  
Sortie : F = L'indicateur C passe à 1 si le curseur n'a pu être déplacé (sortie d'écran).  
Modifie : AF et éventuellement CLOC (0F92Ah).  
Note : Appel à la Sub-ROM si le mode d'écran est supérieur au Screen 5.

### 00108h (Main-ROM)    **DOWNC**    DOWN Current

Rôle : Déplacement du curseur graphique d'un pixel vers le bas.  
Entrée : Rien.  
Sortie : Rien.  
Modifie : AF et éventuellement CLOC (0F92Ah).  
Note : Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 0010Bh (Main-ROM)    **TDOWNC**    Test and DOWN Current

Rôle : Déplacement du curseur graphique d'un pixel vers le bas avec test de dépassement d'écran.  
Entrée : Rien.  
Sortie : F = Indicateur C à 1 si si le curseur n'a pu être déplacé (sortie d'écran).  
Modifie : AF et éventuellement CLOC (0F92Ah).  
Note : Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 0010Eh (Main-ROM)    **SCALXY**    SCALE X & Y

Rôle : Vérifie si les coordonnées contenues dans BC et DE dépassent ou pas de l'écran. Les coordonnées seront ajustées en cas de dépassement.  
Entrée : BC = Abscisse (0 ~ 0FFFFh).  
DE = Ordonnée (0 ~ 0FFFFh).  
Sortie : BC = Abscisse valide.  
DE = Ordonnée valide.  
F = Indicateur C à 0 si BC ou DE était hors de l'écran.  
Modifie : AF.  
Notes : - Lorsque l'une des deux coordonnées est négative, cette routine prend 0 comme nouvelle coordonnée. De même, si une coordonnée dépasse sa valeur maximale autorisée (255 ou 511 pour X, 191 ou 211 pour Y), la routine prend la valeur maximale autorisée comme nouvelle coordonnée.  
- Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 00111h (Main-ROM)    **MAPXYC**    MAP X&Y to Current

Rôle : Convertie les coordonnées graphiques contenues dans BC et DE en une adresse mémoire vidéo et un masque. (Pas utile dans les SCREEN 5 à 8).

Entrée : BC = Abscisse.  
DE = Ordonnée.

Sortie : CLOC (0F92Ah) = Dans les SCREEN 2 à 4 et 10 à 12, adresse en VRAM correspondante aux coordonnées. Dans les SCREEN 5 à 8, abscisse.  
CMASK (0F92Ch) = Dans les SCREEN 2 à 4 et 10 à 12, masque à appliquer. Dans les SCREEN 5 à 8, ordonnée.

Note : Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

#### 00114h (Main-ROM) **FETCHC** FETCH Current

Rôle : Récupération de l'adresse VRAM actuelle et du masque actuel (ou des coordonnées du pixel pour les SCREEN 5 à 8).

Entrée : Rien.

Sortie : HL = Dans les SCREEN 2 à 4, adresse VRAM du curseur graphique.  
Dans les SCREEN 5 à 8, abscisse du pixel.  
A = Dans les SCREEN 2 à 4, masque du curseur graphique.  
Dans les SCREEN 5 à 8, ordonnée du pixel.

Modifie : F.

#### 00117h (Main-ROM) **STOREC** STORE Current

Rôle : Sauvegarde de l'adresse VRAM actuelle et du masque actuel (ou des coordonnées).

Entrée : HL = Dans les SCREEN 2 à 4, adresse VRAM du curseur graphique.  
Dans les SCREEN 5 à 8, abscisse du pixel.  
A = Dans les SCREEN 2 à 4, masque du curseur graphique.  
Dans les SCREEN 5 à 8, ordonnée du pixel.

Sortie : Rien.

Modifie : Rien.

### 0011Ah (Main-ROM)    **SETATR**    SET ATtRIBUTE byte

Rôle :            Modification de l'octet d'attribut ATRBYT (0F3F2h).  
Entrée :          A = Valeur du nouvel attribut.  
Sortie :          F = Indicateur C à 1 si l'attribut n'est pas valide (>15).  
Modifie :        F.  
Note :            Cette routine n'est valable que dans les SCREEN 0 à 4.

### 0011Dh (Main-ROM)    **READC**    READ Current

Rôle :            Lecture de la couleur du pixel au curseur graphique actuel.  
Entrée :          CLOC (0F92Ah) = Dans les SCREEN 2 à 4 et 10 à 12, adresse où se trouve le  
   curseur graphique en VRAM. Dans les SCREEN 5 à 8,  
   abscisse actuelle du curseur graphique.  
   CMASK (0F92Ch) = Dans les SCREEN 2 à 4 et 10 à 12, masque à appliquer. Dans  
   les SCREEN 5 à 8, ordonnée actuelle du curseur graphique.  
Sortie :          A = Couleur du pixel (0~15 dans les SCREEN 5 à 7 ; 0 à 255 dans les SCREEN 8  
   à 12)  
Modifie :        F.  
Note :            Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 00120h (Main-ROM)    **SETC**    SET Current

Rôle :            Placer un pixel à l'écran au curseur graphique actuel.  
Entrée :          CLOC (0F92Ah) = Dans les SCREEN 2 à 4 et 10 à 12, adresse où se trouve le  
   curseur graphique en VRAM. Dans les SCREEN 5 à 8,  
   abscisse actuelle du curseur graphique.  
   CMASK (0F92Ch) = Dans les SCREEN 2 à 4 et 10 à 12, masque à appliquer. Dans  
   les SCREEN 5 à 8, ordonnée actuelle du curseur graphique.  
   ATRBYT (0F3F2h) = Couleur du pixel (0~3 en SCREEN 6 ; 0~15 dans les  
   SCREEN 5 et 7 ; 0~255 dans les SCREEN 8 à 12).  
Sortie :          Rien.  
Modifie :        AF  
Note :            Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

00123h (Main-ROM)	<b>NSETCX</b>	Next SET Current X
-------------------	---------------	--------------------

Rôle : Affichage de plusieurs pixels à partir du curseur graphique actuel.

Entrée :      HL = Nombre de pixels à afficher.

CLOC (0F92Ah) = Dans les SCREEN 2 à 4 et 10 à 12, adresse où se trouve le curseur graphique en VRAM. Dans les SCREEN 5 à 8, abscisse actuelle du curseur graphique.

CMASK (0F92Ch) = Dans les SCREEN 2 à 4 et 10 à 12, masque à appliquer. Dans les SCREEN 5 à 8, ordonnée actuelle du curseur graphique.

ATRBYT (0F3F2h) = Couleur du pixel (0~3 en SCREEN 6 ; 0~15 dans les SCREEN 5 et 7 ; 0~255 dans les SCREEN 8 à 12).

Sortie : Rien.

Modifie : Tout.

00126h (Main-ROM)      **GTASPC**      GeT ASPeCt ratio

Rôle : Renvoi de l'aspect du cercle.

Entrée : Rien.

Sortie : DE = Contenu de ASPCT1 (0F40Bh)

HL = Contenu de ASPCT2 (0F40Dh)

Modifie : Rien.

00129h (Main-ROM)      **PNTINI**      PaiNT INItialize

Rôle : Initialisation de la procédure de peinture.

Entrée : Rien.

Sortie : Rien.

Modifie : AF.

0012Ch (Main-ROM)      **SCANR**      SCAN Right

Rôle : Recherche vers la droite du premier pixel d'une autre couleur

Entrée : DE = Nombre de pixels sur lesquels la recherche doit s'effectuer

Sortie :            DE = Position du pixel recherché

Modifie : Tout.

Notes : - Un exemple, si vous chargez DE avec 100h et que le premier pixel d'une autre couleur se trouve à 4 pixels, la routine renverra 0FCh dans DE

- Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 0012Fh (Main-ROM)     **SCANL**     SCAN Left

- Rôle : Recherche vers la gauche du premier pixel d'une autre couleur.
- Entrée : DE = Nombre de pixels sur lesquels la recherche doit s'effectuer.
- Sortie : DE = Position du pixel recherché.
- Modifie : Tout.
- Notes :  
- Un exemple, si vous chargez DE avec 120h et que le premier pixel d'une autre couleur se trouve à 9 pixels, la routine renverra 117h dans DE  
- Appel à la Sub-ROM si le mode d'écran est supérieur au SCREEN 4.

### 00132h (Main-ROM)     **CHGCAP**     CHanGe CAPs lamp status

- Rôle : Allume ou éteint le voyant CAPS LOCK.
- Entrée : A = 0 pour éteindre le voyant ; 1 pour l'allumer.
- Sortie : Rien.
- Modifie : AF.

### 00135h (Main-ROM)     **CHGSND**     CHanGe SouND port status

- Rôle : Mettre ou couper le son.
- Entrée : A = 0 pour couper : 1 pour mettre le son.
- Sortie : Rien.
- Modifie : AF.



### 00138h (Main-ROM) **RSLREG** Read primary Slot REGister

Rôle : Lecture du registre des Slot primaires.

Entrée : Rien.

Sortie : A = Valeur du registre.

Modifie : Rien.

Note : Format du registre :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Slot de la plage 3 (C000h~FFFFh)		Slot de la plage 2 (8000h~BFFFh)		Slot de la plage 1 (4000h~7FFFh)		Slot de la plage 0 (0000h~3FFFh)	

### 0013Bh (Main-ROM) **WSLREG** Write Slot REGister

Rôle : Écriture dans le registre des Slot primaires.

Entrée : A = Donnée à écrire.

Sortie : Rien.

Modifie : Rien.

Note : Format du registre :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Slot de la plage 3 (C000h~FFFFh)		Slot de la plage 2 (8000h~BFFFh)		Slot de la plage 1 (4000h~7FFFh)		Slot de la plage 0 (0000h~3FFFh)	

### 0013Eh (Main-ROM) **RDVDP** ReaD VDP status register

Rôle : Lecture du registre de statut du VDP.

Entrée : Rien.

Sortie : A = Contenu du registre.

Modifie : Rien.

Note : Sur V9938 ou plus récent, cette routine lit le registre défini par le registre 15.

### 00141h (Main-ROM) **SNSMAT** ScaN Specified row in keyboard MATrix

Rôle : Lecture d'une ligne dans la matrice clavier

Entrée : A = Numéro de la ligne (0~8 ou 0~10 pour les claviers avec pavé numérique)

Sortie : A = état de la ligne, les bits correspondants aux touches enfoncées sont à 0

Modifie : AF, C

Note : Les matrices de clavier varient d'un MSX à l'autre. Veuillez voir les [exemples donnés dans l'annexe](#) pour vous faire une idée.

## 00144h (Main-ROM)    **PHYDIO**    PHYsical Disk I/O

- Rôle : Appeler la routine de lecture / écriture de secteur(s) sur un disque logique du la Disk-ROM.
- Entrée : A = Numéro du disque logique (0 ~ 7).  
B = Nombre de secteur.  
C = Identifiant du type de média utilisé. (0F8h~0FFh).  
0F8h pour les disquettes 1DD, 9 secteurs par pistes ;  
0F9h pour les disquettes 2DD, 9 secteurs par pistes ;  
0FAh pour les disquettes 1DD, 9 secteurs par pistes ;  
0FBh pour les disquettes 2DD, 9 secteurs par pistes ;  
0FCh pour les disquettes 2D, 8 secteurs par pistes ;  
0FDh pour les disquettes 2D, 8 secteurs par pistes ;  
0FEh pour les disquettes 1D, 8 secteurs par pistes ;  
0FFh pour les disquettes 1D, 8 secteurs par pistes.  
DE = Numéro du premier secteur.  
HL = Adresse de destination en mémoire vive.  
F = Indicateur C à 1 pour écrire, 0 pour lire.
- Sortie : F = Indicateur C à 1 si erreur rencontrée, sinon 0.  
A = Code d'erreur.  
B = Nombre de secteur non lus.
- Modifie : Tout.
- Notes : - Appel au Hook H.PHYD (0FFA7h).  
- Aucun effet si il n'a y pas de Disk-ROM (sauf l'appel au Hook H.PHYD).

Exemple en assembleur :

; Lire le secteur 0 d'une disquette double face 3.5".

PHYDIO    equ    0114h

org    0d000h

RDSECT:

```
ld    a,0                    ; Lecteur "A:"
ld    b,1                    ; Nombre de secteur à lire = 1
ld    c,0f9h                ; Identifiant disquette double face 3.5"
ld    de,0                   ; Premier secteur
ld    hl,(0f351h)           ; Vers la cache de secteur du Disk-Basic
or    a                      ; Lecture (mettre SCF pour écriture)
call  PHYDIO                ; Lecture du secteur 0
```

### 00147h (Main-ROM)    **FORMAT**    disk FORMATter

- Rôle :            Formater une disquette.
- Entrée :          Rien.
- Sortie :           Rien.
- Modifie :        Tout.
- Notes :           - Appel au Hook H.FORM (0FFACh).  
                     - Aucun effet si il n'a y pas de Disk-ROM (sauf l'appel au Hook H.FORM).

### 0014Ah (Main-ROM)    **ISFLIO**    IS FiLe I/O

- Rôle :            Vérification du fonctionnement des lecteurs.
- Entrée :          Rien.
- Sortie :           A = 0 si accès en cours.
- Modifie :        AF.
- Notes :           - Appel au Hook H.ISFL (0FEDFh).  
                     - Aucun effet si il n'a y pas de Disk-ROM (sauf l'appel au Hook H.ISFL).

### 0014Dh (Main-ROM)    **OUTDLP**    OUT Do Line Printer

- Rôle :            Sortie d'un caractère sur imprimante.
- Entrée :          A = Code ASCII du caractère à sortir.
- Sortie :           Rien.
- Modifie :        F.
- Note :            Par rapport à LPTOUT, cette routine transforme les tabulations en espaces. Lors de l'utilisation d'une imprimante non-standard MSX, les caractères graphiques changent et les caractères japonais Hiragana sont convertis en Katakana sur un MSX japonais. De plus, CTRL+STOP provoque un « device I/O error ».

### 00150h (Main-ROM)    **GETVCP**    GET VoiCe Pointer

- Rôle :            Obtention du pointeur pour une queue musicale.
- Entrée :          A = Numéro de voix (1 ~ 3).
- Sortie :           HL = Pointeur de la voix désiré.
- Modifie :        AF.
- Note :            Utilisé uniquement pour jouer de la musique en arrière-plan.

### 00153h (Main-ROM)    **GETVC2**    GET VoiCe 2

Rôle :            Renvoie le pointeur à VOICEN (0FB38h).  
Entrée :          L = Pointeur de l'instruction PLAY.  
Sortie :          HL = Nouvelle valeur du pointeur.  
Modifie :        AF.  
Note :            Utilisé uniquement pour jouer de la musique en arrière-plan.

### 00156h (Main-ROM)    **KILBUF**    KILL Buffer

Rôle :            Vide le cache du clavier.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        HL.

### 00159h (Main-ROM)    **CALBAS**    CALL BASic

Rôle :            Appel à une routine dans la ROM Basic (appel inter-Slot) depuis un autre Slot.  
                    (Utilisé par l'instruction CALL du Basic.)  
Entrée :          IX = Adresse à appeler.  
Sortie :          Rien.  
Modifie :        AF, IX, IY, registres auxiliaires.

### Routines ajoutées aux MSX2 :

### 0015Ch (Main-ROM)    **SUBROM**    call SUB-ROM

Rôle :            Appel à une adresse dans la Sub-ROM.  
Entrée :          Mettre IX dans la pile puis, IX = Adresse à appeler.  
Sortie :          Rien.  
Modifie :        Tout sauf IY et registres auxiliaires du Z80.  
Notes :          - Routine absente sur MSX1.  
                    - Méthode pour utiliser cette routine :

```
SUBROM        equ    015ch
```

```
EXPL:        push ix  
              ld     ix,<adresse>        ; routine de la Sub-ROM  
              jp    SUBROM
```

Une fois la routine de la Sub-ROM exécutée, on aura un retour à la suite du programme EXPL derrière le JP SUBROM. Attention, cette routine ne fonctionne pas sous MSX-DOS ! Pour cela, vous pouvez utiliser la méthode suivante.

Les conditions requises pour appeler une routine de la Sub-ROM sont :

1/ Charger l'adresse de la routine à appeler dans le registre IX.

2/ Assurez-vous que la pile ne trouve pas sur plage 0000h~3FFFh)

3/ Il n'est pas nécessaire de désactiver les interruptions.

4/ Le Hook H.NMI peut rester tel quel après un appel.

Comme il n'est pas possible de transmettre directement l'adresse avec IX et que la routine d'appel inter-Slot ne peut appeler de routine en Sub-ROM, utilisez la routine suivante. Celle-ci a été fournie par ASCII. Elle fonctionne sous toutes les versions de MSX-DOS.

```
; Entrée : IX = Adresse à appeler dans la Sub-ROM
;          AF, HL, DE, BC = Paramètres de la routine
;
; Sortie : AF, HL, DE, BC = Dépendent de la routine
;
CALSLT    equ    001ch
EXTROM    equ    015fh          ; Non-maskable interrupt
NMI       equ    0066h          ; Hook for NMI
EXPTBL    equ    0fcc1h
H.NMI     equ    0fdd6h
;
CALSUB:
    exx                    ; Préserve les arguments
    ex    af,af'          ; des registers

    ld     hl,EXTROM       ; Place la routine suivante
    push  hl               ; dans la pile (longueur = 10)
    ld     hl,0c300h       ;
    push  hl               ; +0    inc sp
    push  ix               ; +1    inc sp
    ld     hl,021ddh       ; +2    ld ix,<adresse à appeler>
    push  hl               ; +6    nop
    ld     hl,03333h       ; +7    jp EXTROM
    push  hl
    ld     hl,0
    add   hl,sp
    ld     a,0c3h
    ld     (H.NMI),a
    ld     (H.NMI+1),hl
    ex     af,af'          ; Restore les arguments
    exx                    ; des registers
;
    ld     ix,NMI
    ld     iy,(EXPTBL-1)
    call  CALSLT
    ei
;
    ld     hl,10 ; Throw away the interface routine
    add   hl,sp
    ld     sp,hl
    ret
;
end
```

**0015Fh (Main-ROM)    EXTROM    call EXTernal ROM**

Rôle :            Appel à une adresse dans une ROM externe.  
Entrée :          IX = Adresse à appeler.  
Sortie :          Rien.  
Modifie :        Tout sauf IY et les registres auxiliaires du Z80.  
Note :            Routine absente sur MSX1.

**00162h (Main-ROM)    CHKSLZ    CHeCK Slot Z**

Rôle :            Recherche du Slot de la Sub-ROM. (Routine du système appelée pendant l'initialisation du MSX.)  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Routine absente sur MSX1.

**00165h (Main-ROM)    CHKNEW    CHeCK NEW screen mode**

Rôle :            Vérification si le mode d'écran est supérieur à SCREEN 4.  
Entrée :          Rien.  
Sortie :          F = Indicateur C à 0 si le mode d'écran actuel est entre SCREEN 5 et 12.  
Modifie :        AF.  
Note :            Routine absente sur MSX1.

**00168h (Main-ROM)    EOL            erase to End Of Line**

Rôle :            Effacement de caractères d'un point donné jusqu'à la fin de la ligne (équivalent au CTRL+E du Basic).  
Entrée :          H = Numéro de la colonne à partir de laquelle il faut effacer.  
                    L = Numéro de la ligne à effacer.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Routine absente sur MSX1.

**0016Bh (Main-ROM)    **BIGFIL**    BIG FILl**

Rôle :            Remplissage de la mémoire vidéo.  
Entrée :          HL = Adresse de début (00000h~0FFFFh).  
                    BC = Longueur.  
                    A = Donnée.  
Sortie :          Rien.  
Modifie :        AF, BC.  
Note :            Routine absente sur MSX1.

**0016Eh (Main-ROM)    **NSETRD**    New SET address for ReaD**

Rôle :            Transfert dans le VDP de l'adresse de la case mémoire où doit s'effectuer la lecture.  
Entrée :          HL = Adresse de lecture (00000h~0FFFFh)  
Sortie :          Rien.  
Modifie :        AF.  
Note :            Routine absente sur MSX1.

**00171h (Main-ROM)    **NSTWRT**    New SeT address for WRiT**

Rôle :            Transfert dans le VDP de l'adresse de la case mémoire où doit s'effectuer l'écriture.  
Entrée :          HL = Adresse d'écriture (00000h~0FFFFh).  
Sortie :          Rien.  
Modifie :        AF.  
Note :            Routine inexistante sur MSX1.

**00174h (Main-ROM)    **NRDVRM**    New ReaD VRaM**

Rôle :            Lecture d'une case mémoire de VRAM.  
Entrée :          HL = Adresse de la case mémoire à lire (00000h~0FFFFh).  
Sortie :          A = Contenu de la case mémoire lue.  
Modifie :        F.  
Note :            Routine inexistante sur MSX1.

## 00177h (Main-ROM)    **NWRVRM**    New WRite VRaM

Rôle :            Écriture dans une case mémoire de VRAM.  
Entrée :        HL = Adresse de la case mémoire (00000h~0FFFFh).  
                  A = Donnée à écrire.  
Sortie :        Rien.  
Modifie :      AF.  
Note :        Routine existante qu'à partir du MSX1.

### Routines ajoutées aux MSX2+ :

## 0017Ah (Main-ROM)    **RDRES**    ReaD RESet

Rôle :            Indique le type initialisation du MSX paramétré.  
Entrée :        Rien.  
Sortie :        A = Le bit 7 à 1 indique si la routine STARUP (0000h) effectue un « Soft Reset » sinon ce sera un « Hard Reset ». Le bit 5 indique, sur MSX turbo R, quel CPU utiliser à l'initialisation (1 pour le R800).  
Modifie :      AF.  
Note :        - Routine existante qu'à partir du MSX2+.  
                  - Un « Hard Reset » initialise toute la RAM. Le logo du MSX s'affiche pour marquer la différence.

## 0017Dh (Main-ROM)    **WRRES**    WRite RESet

Rôle :            Choix du type initialisation du MSX.  
Entrée :        A = Le bit 7 permet de choisir la façon d'initialiser le MSX. 1 pour « Soft Reset » sinon « Hard Reset ». Sur MSX turbo R, mettre le bit 5 à 1 pour initialiser avec le R800.  
Sortie :        Rien.  
Modifie :      AF.  
Note :        - Routine existante qu'à partir du MSX2+.  
                  - Un « Hard Reset » initialise toute la RAM. Le logo du MSX s'affiche pour marquer la différence.  
Exemple :      Programme en assembleur pour faire une initialisation logicielle (Soft Reset).

```
call 017ah  
or   80h           ; Bit 7 à 1  
call 017dh  
rst  0
```



## Routines ajoutées aux MSX turbo R :

### 00180h (Main-ROM)    **CHGCPU**   CHanGe CPU

Rôle :            Changer le CPU du MSX turbo R. (Z80A à 3,579Mhz ou R800 à 7,159Mhz.)

Entrée :          A = Paramètres.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LED	0	0	0	0	0	MODE	

MODE = 00 pour Z80 ; 01 pour R800 en mode ROM et 10 pour R800 en mode DRAM.

LED = 1 allume la LED si changement vers R800 sinon, éteint la LED. 0 pour laisser tel quel.

Sortie :          Rien.

Modifie :        Rien.

Note :           - Routine pour MSX turbo R seulement.

                 - En mode RAM, le contenu des ROM du système (Main-ROM, Sub-ROM et Kanji Driver ROM) est recopié en RAM. Cette zone de mémoire est protégée en écriture. Ce mode permet d'accélérer l'exécution des programmes mais il fait perdre les quatre dernières pages du Memory Mapper à l'utilisateur.

                 - Le changement de CPU peut s'effectuer à tout moment du Z80A au R800 en mode ROM et vice versa. Cependant, le passage en mode RAM doit s'effectuer au démarrage du MSX.

                 - L'utilisation du R800 sous MSX-DOS1 peut endommager les fichiers manipulés.

### 00183h (Main-ROM)    **GETCPU**   GET CPU

Rôle :            Indique quel est le CPU en fonctionnement.

Entrée :          Rien.

Sortie :          A = 0 si Z80 ; 1 si R800 en mode ROM et 2 si R800 en mode DRAM.

Modifie :        F.

Note :           Routine pour MSX turbo R seulement.

## 00186h (Main-ROM)    **PCMPPLY**    PCM PLaY

Rôle :            Jouer un son numérisé.

Entrée :        A = Paramètres.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
SOR	0	0	0	0	0	FREQ	

FREQ = Fréquence de l'échantillonnage.

00 = 15,75kHz ; 01 = 7,875kHz ; 10 = 5,25kHz et 11 = 3,9375kHz.

SOR = Emplacement des données à lire.

0 pour RAM principale ; 1 pour VRAM.

BC = Longueur des données à lire.

HL = Adresse du début des données à lire. (08000h~ si RAM principale)

E = 3 bits de point fort de l'adresse des données à lire. (VRAM seulement)

Sortie :        A = 1 si la fréquence de l'échantillonnage est erronée ;

2 si la lecture a été stoppée en pressant la touche « STOP ».

F = Indicateur C à 1 si la lecture s'est déroulée normalement.

E = 3 bits de point fort de l'adresse où la lecture a été stoppée. (VRAM)

HL = Adresse où la lecture a été stoppée.

Modifie :      Tout.

Notes :        - Routine pour MSX turbo R seulement.

- Cette routine utilise toujours le R800. Si le Z80A est utilisé lors de l'appel à cette routine, le R800 sera utilisé en mode ROM puis remplacera le Z80A avant de rendre la main.

- Les interruptions sont coupées pendant l'exécution de cette routine.

## 00189h (Main-ROM)    **PCMREC**    PCM REcOrd

Rôle :            Numériser un son.

Entrée :        A = Paramètres.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
DEST	Trigger level				COMP	FREQ	

FREQ = Fréquence d'échantillonnage.

00 = 15,75kHz, 01 = 7,875kHz, 10 = 5,25kHz et 11 = 3,9375kHz.

COMP = 1 pour compression des données.

Trigger level = Niveau de déclenchement (sensibilité).

DEST = Emplacement des données à écrire.

0 pour RAM principale. 1 pour VRAM.

BC = Longueur des données à écrire.

HL = Adresse du début des données à écrire. (08000H~ si RAM principale)

E = 3 bits de point fort de l'adresse des données à écrire. (VRAM seulement)

Sortie :        A = 1 si la fréquence de l'échantillonnage est erronée.

2 si la lecture a été stoppée en pressant la touche « STOP ».

F = Indicateur C à 1 si la lecture s'est déroulée normalement.

E = 3 bits de point fort de l'adresse où la lecture a été stoppée. (VRAM)

HL = Adresse où la lecture a été stoppée.

Modifie :     Tout.

Notes :

- Routine pour MSX turbo R seulement.
- Cette routine utilise toujours le R800. Si le Z80A est utilisé lors de l'appel à cette routine, le R800 sera utilisé en mode ROM puis remplacera le Z80A avant de rendre la main.
- Les interruptions sont coupées pendant l'exécution de cette routine.
- Un enregistrement à 15,75kHz en mode R800 ROM provoque une erreur (indicateur C à 0).
- Presser la touche STOP pour interrompre un enregistrement.

## Routines « Math-pack »

En plus des routines du Bios, voici des routines internes du Basic appelées « Math-pack ». Celles-ci se trouvent en Main-Rom et ont été garanties par ASCII de rester aux mêmes adresses.

Les routines « Math-pack » sont des routines mathématiques mais aussi de déplacement, de comparaison et de conversion de nombres codés sur 2 octets pour un entier, 3 pour une chaîne de caractères, 4 pour une valeur en précision simple ou 8 pour une valeur en double précision. Ces nombres sont codés comme ci-dessous.

Nombre entier :

1er octet : 8 bits les moins significatif du nombre

2ème octet : Signe (bit 7) et les 7 bits les plus significatifs du nombre

Les nombres codés en simple et double précision sont stockés au format BCD.

### Nombre codés en simple/double précision :

1<sup>er</sup> octet : Le signe (bit 7) et exposant

2<sup>ème</sup> octet : Le chiffre avant et après la virgule

3<sup>ème</sup> octet : 2<sup>ème</sup> et 3<sup>ème</sup> chiffre après la virgule

4<sup>ème</sup> octet : 4<sup>ème</sup> et 5<sup>ème</sup> chiffre après la virgule ← Jusqu'ici en simple précision

5<sup>ème</sup> octet : 6<sup>ème</sup> et 7<sup>ème</sup> chiffre après la virgule

6<sup>ème</sup> octet : 8<sup>ème</sup> et 9<sup>ème</sup> chiffre après la virgule

7<sup>ème</sup> octet : 10<sup>ème</sup> et 11<sup>ème</sup> chiffre après la virgule

8<sup>ème</sup> octet : 12<sup>ème</sup> et 13<sup>ème</sup> chiffre après la virgule

### Nombre codés sur une chaîne de caractères :

1er octet : La longueur de la chaîne

2<sup>ème</sup> octet : 8 bits les moins significatif de l'adresse de chaîne

3<sup>ème</sup> octet : 8 bits les plus significatif de l'adresse de chaîne

La variable VALTYP (0F7F6h) spécifie le type de codage des nombres.

2 = Nombre entier

3 = Nombre dans une chaîne de caractères

4 = Nombre en simple précision

8 = Nombre en double précision

## Routines d'opérations de base :

0268Ch (Main-ROM)    **DECSUB**    DECimal SUBstraction

Rôle :            Effectuer l'opération  $DAC \leftarrow DAC - ARG$

Modifie :        Tous les registres.

Note :            DAC et ARG doivent être en double précision.

**0269Ah (Main-ROM)    DECADD    DECimal ADDition**

Rôle :            Effectuer l'opération  $DAC \leftarrow DAC + ARG$   
Modifie :        Tous les registres.  
Note :            DAC et ARG doivent être en double précision.

**026FAh (Main-ROM)    DECNRM    DECimal NoRMalisation**

Rôle :            Normaliser DAC  
Modifie :        Tous les registres.  
Notes :          - DAC et ARG doivent être en double précision.  
                    - Cette routine retire les zéros inutiles après la virgule. (ex : 0.00123 -> 0.123E-2)

**0273Ch (Main-ROM)    DECROU    DECimal ROUnd**

Rôle :            Arrondir DAC  
Modifie :        Tous les registres.  
Note :            DAC et ARG doivent être en double précision.

**027E6H (Main-ROM)    DECMUL    DECimal MULTiplication**

Rôle :            Effectuer l'opération  $DAC \leftarrow DAC * ARG$   
Modifie :        Tous les registres.  
Note :            DAC et ARG doivent être en double précision.

**0289FH (Main-ROM)    DECDIV    DECimal DIVision**

Rôle :            Effectuer l'opération  $DAC \leftarrow DAC / ARG$   
Modifie :        Tous les registres.  
Note :            DAC et ARG doivent être en double précision.

**0314Ah (Main-ROM)    UMULT    Unsigned MULTiplication**

Rôle :            Effectuer l'opération  $DE \leftarrow BC * DE$   
Modifie :        Tous les registres.  
Note :            Le résultat est également placé dans DAC+2 (2 octets)

**03167h (Main-ROM)    ISUB        Integer SUBstraction**

Rôle :            Effectuer l'opération  $HL \leftarrow DE - HL$   
Modifie :        Tous les registres.  
Note :            Le résultat est également placé dans DAC+2 (2 octets)

**03172h (Main-ROM)    IADD    Integer ADDition**

Rôle :            Effectuer l'opération  $HL \leftarrow DE + HL$   
Modifie :        Tous les registres.  
Note :            Le résultat est également placé dans DAC+2 (2 octets)

**03193h (Main-ROM)    IMULT    Integer MULTiplication**

Rôle :            Effectuer l'opération  $HL \leftarrow DE * HL$   
Modifie :        Tous les registres.  
Note :            Le résultat est également placé dans DAC+2 (2 octets)

**031E6h (Main-ROM)    IDIV    Integer DIVision**

Rôle :            Effectuer l'opération  $HL \leftarrow DE / HL$   
Modifie :        Tous les registres.  
Note :            Le résultat est également placé dans DAC+2 (2 octets)

**0323Ah (Main-ROM)    IMOD    Integer MOD**

Rôle :            Effectuer l'opération  $HL \leftarrow DE \bmod HL$   
Modifie :        Tous les registres.  
Note :            Le résultat est également placé dans DAC+2 (2 octets)

**Puissance :**

**037C8h (Main-ROM)    SNGEXP    SiNGle presision EXPonentiation**

Rôle :            Effectuer l'opération  $DAC \leftarrow DAC \wedge ARG$   
Entrée :          VALTYP (0F7F6h) à 4  
Modifie :        Tous les registres.

**037D7h (Main-ROM)    DBLEXP    DouBLE presision EXPonentiation**

Rôle :            Effectuer l'opération  $DAC \leftarrow DAC \wedge ARG$   
Entrée :          VALTYP (0F7F6h) à 8  
Modifie :        Tous les registres.

**0383Fh (Main-ROM)    **INTEXP**    INTeger EXPonentiation**

Rôle :            Effectuer l'opération  $DAC \leftarrow DE \wedge HL$  (integer)

Entrée :        VALTYP (0F7F6h) à 2

Modifie :      Tous les registres.

**Fonctions trigonométriques :**

**02993h (Main-ROM)    **COS**        COSine**

Rôle :            Effectuer l'opération  $DAC \leftarrow \cos(DAC)$

Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8

Modifie :      A, BC, DE, HL

**029ACh (Main-ROM)    **SIN**        SINE**

Rôle :            Effectuer l'opération  $DAC \leftarrow \sin(DAC)$

Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8

Modifie :      A, BC, DE, HL

**029FBh (Main-ROM)    **TAN**        TANGent**

Rôle :            Effectuer l'opération  $DAC \leftarrow \tan(DAC)$

Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8

Modifie :      A, BC, DE, HL

**02A14h (Main-ROM)    **ATN**        ArcTANGent**

Rôle :            Effectuer l'opération  $DAC \leftarrow \arctan(DAC)$

Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8

Modifie :      A, BC, DE, HL

**Autres fonctions :**

**02A72h (Main-ROM)    **LOG**        LOGarithmic**

Rôle :            Effectuer l'opération  $DAC \leftarrow \log(DAC)$

Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8

Modifie :      A, BC, DE, HL

**02AFFh (Main-ROM)    SQR        SQuare Root**

Rôle :            Effectuer l'opération  $DAC \leftarrow SQR(DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       A, BC, DE, HL

**02A72h (Main-ROM)    EXP        EXPonential**

Rôle :            Effectuer l'opération  $DAC \leftarrow EXP(DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       A, BC, DE, HL

**02AFFh (Main-ROM)    RND        RaNDom**

Rôle :            Effectuer l'opération  $DAC \leftarrow RND(DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       A, BC, DE, HL

**02E71h (Main-ROM)    SIGN       SIGN**

Rôle :            Indique le signe de DAC  
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Sortie:          A  
Modifie :       A

**02E82h (Main-ROM)    ABSFN       ABSolute**

Rôle :            Effectuer l'opération  $DAC \leftarrow ABS(DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       A, BC, DE, HL

**02E8Dh (Main-ROM)    NEG        NEGative base**

Rôle :            Effectuer l'opération  $DAC \leftarrow NEG(DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       A, HL

**02E97h (Main-ROM)    SGN        SiGNum**

Rôle :            Effectuer l'opération  $DAC \leftarrow SNG(DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       A, HL  
Note :           Le résultat est un nombre entier.



## Déplacement de nombre :

### 02C4Dh (Main-ROM)    **MAF**

Rôle :            Effectuer l'opération ARG  $\leftarrow$  DAC  
Entrée :        VALTYP (0F7F6h) à 8  
Modifie :       A, BC, DE, HL

### 02C50h (Main-ROM)    **MAM**

Rôle :            Effectuer l'opération ARG  $\leftarrow$  (HL)  
Entrée :        VALTYP (0F7F6h) à 8  
Modifie :       A, BC, DE, HL

### 02C53h (Main-ROM)    **MOV8DH**

Rôle :            Effectuer l'opération (DE)  $\leftarrow$  (HL)  
Entrée :        VALTYP (0F7F6h) à 8  
Modifie :       A, BC, DE, HL

### 02C59h (Main-ROM)    **MFA**

Rôle :            Effectuer l'opération DAC  $\leftarrow$  ARG  
Entrée :        VALTYP (0F7F6h) à 8  
Modifie :       A, BC, DE, HL

### 02C5Ch (Main-ROM)    **MFm**

Rôle :            Effectuer l'opération DAC  $\leftarrow$  (HL)  
Entrée :        VALTYP (0F7F6h) à 8  
Modifie :       A, BC, DE, HL

### 02C67h (Main-ROM)    **MMF**

Rôle :            Effectuer l'opération (HL)  $\leftarrow$  DAC  
Entrée :        VALTYP (0F7F6h) à 8  
Modifie :       A, BC, DE, HL

### 02C6Ah (Main-ROM)    **MOV8HD**

Rôle :            Effectuer l'opération (HL)  $\leftarrow$  (DE)  
Entrée :        VALTYP (0F7F6h) à 8  
Modifie :       A, BC, DE, HL

## 02C6Fh (Main-ROM) **XTF**

Rôle : Effectuer l'opération (SP)  $\leftrightarrow$  DAC  
Entrée : VALTYP (0F7F6h) à 8  
Modifie : A, BC, DE, HL

## 02CC7h (Main-ROM) **PHA** PusH the Argument

Rôle : Effectuer l'opération ARG  $\rightarrow$  (SP)  
Entrée : VALTYP (0F7F6h) à 8  
Modifie : A, BC, DE, HL

## 02CDCh (Main-ROM) **PHF** PusH the Floating-point number

Rôle : Effectuer l'opération DAC  $\rightarrow$  (SP)  
Entrée : VALTYP (0F7F6h) à 8  
Modifie : A, BC, DE, HL

## 02CE1h (Main-ROM) **PPA** PoP the Argument

Rôle : Effectuer l'opération (SP)  $\rightarrow$  ARG  
Entrée : VALTYP (0F7F6h) à 8  
Modifie : A, BC, DE, HL

## 02CE1h (Main-ROM) **PPF** PoP the Floating-point number

Rôle : Effectuer l'opération (SP)  $\rightarrow$  DAC  
Entrée : VALTYP (0F7F6h) à 8  
Modifie : A, BC, DE, HL

## 02EB1h (Main-ROM) **PUSHF** PUSH the Floating-point number

Rôle : Effectuer l'opération DAC  $\rightarrow$  (SP)  
Entrée : VALTYP (0F7F6h) à 4  
Modifie : DE

## 02EBEh (Main-ROM) **MOVFM**

Rôle : Effectuer l'opération DAC  $\leftarrow$  (HL)  
Entrée : VALTYP (0F7F6h) à 4  
Modifie : BC, DE, HL

### 02EC1h (Main-ROM)    **MOVFR**

Rôle :            Effectuer l'opération  $DAC \rightarrow (CBED)$   
Entrée :        VALTYP (0F7F6h) à 4  
Modifie :       DE

### 02ECCh (Main-ROM)    **MOVRF**

Rôle :            Effectuer l'opération  $(CBED) \leftarrow DAC$   
Entrée :        VALTYP (0F7F6h) à 4  
Modifie :       BC, DE, HL

### 02ED6h (Main-ROM)    **MOVRFMI**

Rôle :            Effectuer l'opération  $(CBED) \leftarrow (HL)$   
Entrée :        VALTYP (0F7F6h) à 4  
Modifie :       BC, DE, HL

### 02EDFh (Main-ROM)    **MOVRFM**

Rôle :            Effectuer l'opération  $(BCDE) \leftarrow (HL)$   
Entrée :        VALTYP (0F7F6h) à 4  
Modifie :       BC, DE, HL

### 02EE8h (Main-ROM)    **MOVFMF**

Rôle :            Effectuer l'opération  $(HL) \leftarrow DAC$   
Entrée :        VALTYP (0F7F6h) à 4  
Modifie :       BC, DE, HL

### 02EEBh (Main-ROM)    **MOVE**                    **MOVE**

Rôle :            Effectuer l'opération  $(HL) \leftarrow (DE)$   
Entrée :        VALTYP (0F7F6h) à 4  
Modifie :       BC, DE, HL

### 02EEFh (Main-ROM)    **VMOVAM**

Rôle :            Effectuer l'opération  $ARG \leftarrow (HL)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       BC, DE, HL

## 02EF2h (Main-ROM)    **MOVVFM**

Rôle :            Effectuer l'opération  $(DE) \leftarrow (HL)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       BC, DE, HL

## 02EF3h (Main-ROM)    **VMOVE**

Rôle :            Déplacer le nombre pointée par DE vers celui pointé par HL.  $((HL) \leftarrow (DE))$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       BC, DE, HL

## 02F05h (Main-ROM)    **VMOVFA**

Rôle :            Déplacer l'argument vers l'accumulateur décimal.  $(DAC \leftarrow ARG)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       BC, DE, HL

## 02F08h (Main-ROM)    **VMOVFM**

Rôle :            Déplacer l'accumulateur décimal vers l'adresse pointée par HL.  $(DAC \leftarrow (HL))$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       BC, DE, HL

## 02F0Dh (Main-ROM)    **VMOVAF**

Rôle :            Déplacer l'accumulateur décimal vers l'argument.  $(ARG \leftarrow DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       BC, DE, HL

## 02F10h (Main-ROM)    **VMOVMF**

Rôle :            Déplacer le nombre pointée par HL vers DAC.  $((HL) \leftarrow DAC)$   
Entrée :        VALTYP (0F7F6h) à 2, 4 ou 8  
Modifie :       BC, DE, HL

## Comparaison de nombre :

02F21h (Main-ROM)     **FCOMP**     Floating-point number COMParison

Rôle :            Comparer le nombre placé dans les registres CBED avec DAC. (C contient le premier octet)

Entrée :        VALTYP (0F7F6h) à 4.

Modifie :       HL

02F4Dh (Main-ROM)     **ICOMP**     Integer COMParison

Rôle :            Comparer le nombre entier placé dans DE avec HL.

Entrée :        VALTYP (0F7F6h) à 2.

Modifie :       HL

02F5Ch (Main-ROM)     **XDCOMP**     XD COMParison

Rôle :            Comparer l'argument avec l'accumulateur décimal. (double précision)

Entrée :        VALTYP (0F7F6h) à 8.

Modifie :       Tous les registres.

## Conversion de nombre :

030CFh (Main-ROM)     **INT**     convert to INTeger

Rôle :            Convertir l'accumulateur décimal en un nombre entier.

Entrée :        DAC

                    VALTYP (0F7F6h) à 4 or 8.

Sortie :        DAC

                    VALTYP (0F7F6h) = 2.

Modifie :       Tous les registres.

03225h (Main-ROM)     **FOUT**     Floating-point number OUT

Rôle :            Convertir un nombre réel simple précision dans DAC vers un nombre simple précision dans une chaîne de caractère. (format non spécifié)

Entrée :        DAC = Nombre simple précision

Sortie :        HL = Adresse de la chaîne résultante.

                    DE = Adresse de fin de la chaîne.

Modifie :       Tous les registres.

Note :           Ne peut être appelé lorsque les interruptions sont désactivées.

## 03299h (Main-ROM)      **FIN**      Floating-point number IN

Rôle :            Convertir un nombre simple précision dans une chaîne de caractère vers un nombre réel simple précision dans DAC.

Entrée :        HL = Adresse de la chaîne  
                 VALTYP (0F7F6h) à 4.

Sortie :        DAC = Nombre simple précision  
                 C = 0 si la chaîne contient bien un nombre simple précision, sinon 255.  
                 B = Nombre de chiffres après le virgule.  
                 D = Nombre total de chiffres.

## 03426h (Main-ROM)      **PUFOUT**

Rôle :            Convertir un nombre numérique stocké dans DAC en une chaîne. (format spécifié)

Entrée :        DAC = Nombre  
                 A = Format désiré  
                 Bit7 = 0 pour aucun format spécifié, 1 pour spécifier le format  
                 Bit6 = 1 pour séparer chaque 3 chiffres avec une virgule  
                 Bit5 = 1 pour remplacer l'espace en tête par \*  
                 Bit4 = 1 pour mettre \$ en tête  
                 Bit3 = 1 pour mettre le signe + si la valeur est positive  
                 Bit2 = 1 pour placer le signe derrière  
                 Bit1 = Inutilisé  
                 Bit0 = 0 pour virgule fixe, 1 virgule flottante  
                 B = Nombre de chiffres avant le virgule  
                 C = Nombre de chiffres après le virgule

Sortie :        HL = Adresse de la chaîne résultante.  
                 DE = Adresse de fin de la chaîne.

Modifie :      Tous les registres.

Note :          Ne peut être appelé lorsque les interruptions sont désactivées.

## 0371Ah (Main-ROM)      **FOUTB**

Rôle :            Convertit un entier sur 2 octets à DAC +2 en une chaîne d'expression binaire.

Entrée :        VALTYP (0F663h) doit être 2.

Sortie :        HL = Adresse de la chaîne résultante.

Modifie :      Tous les registres.

Note :          Ne peut être appelé lorsque les interruptions sont désactivées.

### 0371Eh (Main-ROM)    **FOUTO**

Rôle :            Convertit un entier sur 2 octets à DAC +2 en une chaîne d'expression octale.  
Entrée :        VALTYP (0F663h) doit être 2.  
Sortie :        HL = Adresse de la chaîne résultante.  
Modifie :      Tous les registres.  
Note :        Ne peut être appelé lorsque les interruptions sont désactivées.

### 03722h (Main-ROM)    **FOUTH**

Rôle :            Convertit un entier sur 2 octets à DAC +2 en une chaîne d'expression hexadécimale.  
Entrée :        VALTYP (0F663h) doit être 2.  
Sortie :        HL = Adresse de la chaîne résultante.  
Modifie :      Tous les registres.  
Note :        Ne peut être appelé lorsque les interruptions sont désactivées.

### 02F8Ah (Main-ROM)    **FRCINT**

Rôle :            Convertit un entier sur 2 octets à DAC +2.  
Entrée :        VALTYP (0F663h) à 2.  
Sortie :        HL = Adresse de la chaîne résultante.  
Modifie :      Tous les registres.

### 02FB2h (Main-ROM)    **FRCSNG**

Rôle :            Convertit le DAC en un seul nombre de simple précision.  
Entrée :        VALTYP (0F663h) = 2 ou 8.  
Sortie :        VALTYP (0F663h) = 4.  
Modifie :      Tous les registres.

### 0303Ah (Main-ROM)    **FRCDBL**

Rôle :            Convertit le DAC en un seul nombre de double précision.  
Entrée :        VALTYP (0F663h) = 2 ou 4.  
Sortie :        VALTYP (0F663h) = 8.  
Modifie :      Tous les registres.

### 030BEh (Main-ROM)    **FIXER**

Rôle :             $DAC \leftarrow SGN(DAC) * INT(ABS(DAC))$ .  
Sortie :        VALTYP (0F663h) = 2, 4 ou 8.  
Modifie :      Tous les registres.

### 3.3 Table des sauts du Bios de la Sub-ROM

Avec l'évolution du standard vers le MSX2, la ROM contenant le Bios étant trop petite, il a fallu ajouter une ROM auxiliaire afin d'étendre le Bios. Cette Rom s'appelle « Sub-ROM ». Celle-ci contient aussi une table des sauts qui permet d'appeler les routines supplémentaires spécifiques au MSX2 et aux générations supérieures plus efficacement qu'en passant par la Main-ROM. Pour appeler une de ces routines, vous pouvez passer la routine EXTROM (0015Fh) ou SUBROM (0015Ch) de la Main-ROM ou sélectionner vous-même la Sub-ROM et passer par la table des sauts. Vous pouvez connaître le Slot de la Sub-ROM en lisant la variable système EXBRSA (0FAF8h).

Liste des routines de la mémoire morte auxiliaire (« Sub-ROM ») :

#### 00069h (Sub-ROM)      **PAINT**      PAINT

Rôle :            Remplissage d'une surface à l'écran.  
Entrée :          HL = Pointeur sur le texte de l'interpréteur du Basic.  
Sortie :          HL = Pointeur réactualisé.  
Modifie :        Tout.  
Notes :          - Valide dans les SCREEN 5 à 8 / 12.  
                    - Utilisé par l'interpréteur du Basic.

Exemple d'utilisation :

```
EXTROM equ 0015Fh
PAINT   equ 00069h

        org 0c000h
DEBUT:
        ld hl, DATA
        ld ix, PAINT
        call EXTROM
        ret

DATA:   db '(100,100),5,8',0
```

#### 0006Dh (Sub-ROM)      **PSET**      Pixel SET

Rôle :            Modifier un pixel à l'écran.  
Entrée :          HL = Pointeur sur le texte Basic.  
Sortie :          HL = Pointeur réactualisé.  
Modifie :        Tout.  
Notes :          - Valide dans les SCREEN 5 à 8 / 12.  
                    - Utilisé par l'interpréteur du Basic.  
                    - Voir l'exemple de PAINT ci-dessus pour le fonctionnement.



### 00071h (Sub-ROMp)     **ATRSCN**     ATRibute SCaN

Rôle : Recherche d'une couleur.  
Entrée : HL = Pointeur sur le texte Basic.  
GXPOS (0FCB3h) = Abscisse du pixel de la couleur trouvée.  
GYPOS (0FCB5h) = Ordonnée du pixel de la couleur trouvée.  
Sortie : HL = Pointeur réactualisé.  
Modifie : Tout.  
Notes : - Valide dans les SCREEN 5 à 8 / 12.  
- Utilisé par l'interpréteur du Basic.

### 00075h (Sub-ROM)     **GLINE**     Graphic LINE

Rôle : Tracé d'une ligne.  
Entrée : HL = Pointeur de texte du Basic.  
Sortie : HL = Pointeur réactualisé.  
GXPOS (0FCB3h) = Abscisse du pixel d'arrivée.  
GYPOS (0FCB5h) = Ordonnée du pixel d'arrivée.  
Modifie : Tout.  
Notes : - Valide dans les SCREEN 5 à 8 / 12.  
- Utilisé par l'interpréteur du Basic.

### 00079h (Sub-ROM)     **DOBOXF**     DO BOX Fill

Rôle : Tracé d'un rectangle plein.  
Entrée : HL = Pointeur vers le texte Basic.  
Sortie : HL = Pointeur réactualisé.  
GXPOS (0FCB3h) = Abscisse du pixel d'arrivée.  
GYPOS (0FCB5h) = Ordonnée du pixel d'arrivée.  
Modifie : Tout.  
Notes : - Valide dans les SCREEN 5 à 8 / 12.  
- Utilisé par l'interpréteur du Basic.

0007Dh (Sub-ROM)      **DOLINE**    DO LINE

Rôle :            Tracé de ligne.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
                  GXPOS (0FCB3h) = Abscisse du pixel d'arrivée.  
                  GYPOS (0FCB5h) = Ordonnée du pixel d'arrivée.  
Modifie :      Tout.  
Notes :        - Valide dans les SCREEN 5 à 8 / 12.  
                  - Utilisé par l'interpréteur du Basic.

00081h (Sub-ROM)      **BOXLIN**

Rôle :            Tracé d'un rectangle.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
                  GXPOS (0FCB3h) = Abscisse du pixel d'arrivée.  
                  GYPOS (0FCB5h) = Ordonnée du pixel d'arrivée.  
Modifie :      Tout.  
Notes :        - Valide dans les SCREEN 5 à 8 / 12.  
                  - Utilisé par l'interpréteur du Basic.

## 00085h (Sub-ROM)      **DOGRPH** DO GRaPHic line

Rôle :            Tracé de ligne.

Entrée :        BC = Abscisse du pixel de départ.

DE = Ordonnée du pixel de départ.

GXPOS (0FCB3h) = Abscisse du pixel d'arrivée.

GYPOS (0FCB5h) = Ordonnée du pixel d'arrivée.

ATRBYT (0F3F2h) = Couleur du tracé (0 ~ 3 en SCREEN 6 ; 0 ~ 15 dans les  
SCREEN 5 et 7 ; 0 ~ 255 dans les SCREEN 8 à 12).

LOGOPR (0FB02h) = Opérateur logique à appliquer.

Sortie :        Rien.

Modifie :      AF.

Note :        Valide dans les SCREEN 5 à 8 / 12.

LOGOPR (« logical operation ») peut être :

0 (0000B) = IMP	8 (1000B) = TIMP
1 (0001B) = AND	9 (1001B) = TAND
2 (0010B) = OR	10 (1010B) = TOR
3 (0011B) = XOR	11 (1011B) = TXOR
4 (0100B) = NOT	12 (1100B) = TNOT

Les opérateurs de droite sont identiques à ceux de gauche sauf qu'ils n'agissent pas quand la couleur de fond est 0.

Exemple d'utilisation :

```
EXTROM      equ    0015fh
DOGRPH      equ    00085h
GXPOS       equ    0fcb3h
GYPOS       equ    0fcb5h
ATRBYT      equ    0f3f2h
LOGOPR      equ    0fb02h

              org    0c000h
DEBUT:
              ld     bc, 10h
              ld     de, 50h
              ld     hl, 90h
              ld     (GXPOS), hl
              ld     (GYPOS), hl
              ld     a, 7
              ld     (ATRBYT), a
              xor    a
              ld     (LOGOPR), a
              ld     ix, DOGRPH
              call   EXTROM
              ret
```

00089h (Sub-ROM) **GRPPRT** GRaPhic PRinT

Rôle : Affichage d'un caractère sur écran graphique.

Entrée :      A = Code ASCII du caractère.

GRPACX (0FCB7h) = Abscisse du pixel où doit s'afficher le caractère.

GRPACY (0FCB9h) = Ordonnée du pixel où doit s'afficher le caractère.

ATRBYT (0F3F2h) = Couleur du caractère (0 ~ 3 en SCREEN 6 ; 0 ~ 15 en SCREEN 5 et 7 ; 0 ~ 255 dans les SCREEN 8 à 12).

LOGOPR (0FB02h) = Opérateur logique.

Sortie : Rien.

Modifie : Rien.

Note : Valide dans les SCREEN 5 à 8 / 12.

Voir DOGRPH (00085h en Sub-ROM) pour l'explication de LOGOPR.

0008Dh (Sub-ROM)      **SCALXY**    SCALe X&Y

Rôle :	Vérification et éventuellement ajustement des valeurs du curseur graphique.
--------	---

Entrée : BC = Abscisse (0 ~ 0FFFFh).

DE = Ordonnée (0 ~ 0FFFFh).

Sortie : BC = Abscisse valide.

DE = Ordonnée valide.

F = Indicateur C à 1 si une des coordonnées était hors de l'écran.

Modifie : AF.

Note : Lorsque l'une des deux coordonnées est négative, cette routine prend 0 comme nouvelle coordonnée. De même, si une coordonnée dépasse la valeur maximale autorisée (255 ou 511 pour l'abscisse, 191 ou 211 pour l'ordonnée), la routine prend la valeur maximale comme nouvelle coordonnée.

00091h (Sub-ROM)      **MAPXYC**   MAP X&Y to Current

Rôle :	Convertie les coordonnées graphiques contenues dans BC et DE en une adresse mémoire vidéo et un masque. (Pas utile dans les SCREEN 5 à 8).
--------	--

Entrée : BC = Abscisse.

DE = Ordonnée.

Sortie : HL et CLOC (0F92Ah) = En SCREEN 3, adresse correspondante aux coordonnées.  
Dans les SCREEN 5 à 8, abscisse.

A et CMASK (0F92Ch) = En SCREEN 3, masque à appliquer à l'octet ci-dessus pour placer le pixel qui nous intéresse. Dans les SCREEN 5 à 8, ordonnée du pixel à placer.

Modifie : F.

Note : Valide dans les SCREEN 3 et 5 à 8.

### 00095h (Sub-ROM)      **READC**      READ Current

Rôle :            Lecture de l'attribut du pixel actuel.

Entrée :        CLOC (0F92Ah) = En SCREEN 3, adresse où se trouve le curseur graphique en VRAM. Dans les SCREEN 5 à 8, abscisse actuelle du curseur graphique.

                  CMASK (0F92Ch) = En SCREEN 3, masque à appliquer. Dans les SCREEN 5 à 8, ordonnée actuelle du curseur graphique.

Sortie :        A = Attribut. (Numéro de la couleur du pixel.)

Modifie :      AF.

Note :        Valide dans les SCREEN 3 et 5 à 8.

### 00099h (Sub-ROM)      **SETATR**      SET ATRibute

Rôle :            Modification de l'octet d'attribut ATRBYT (0F3F2h).

Entrée :        A = Valeur du nouvel attribut. (Couleur de tracé des routines graphiques.)

Sortie :        F = Indicateur C à 1 si attribut non valide (>3 en SCREEN 6 ; >15 en SCREEN 5 et 7).

Modifie :      F.

### 0009Dh (Sub-ROM)      **SETC**      SET Current

Rôle :            Allumage du pixel actuel.

Entrée :        CLOC (0F92Ah) = En SCREEN 3, adresse où se trouve le curseur graphique en VRAM. Dans les SCREEN 5 à 8, abscisse actuelle du curseur graphique.

                  CMASK (0F92Ch) = En SCREEN 3, masque à appliquer. Dans les SCREEN 5 à 8, ordonnée actuelle du curseur graphique.

                  ATRBYT (0F3F2h) = Couleur du pixel à tracer (0 ~ 3 en SCREEN 6 ; 0 ~ 15 dans les SCREEN 3 à 5 et 7 ; 0 ~ 255 dans les SCREEN 8 à 12).

Sortie :        Rien.

Modifie :      AF.

Note :        Valide dans les SCREEN 3 et 5 à 8 / 12.

### 000A1h (Sub-ROM)      **TRIGHTC** Test RIGHT Current

Rôle :            Déplacement d'un pixel vers la droite après vérification de la validité des coordonnées.

Entrée :        CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.

Sortie :        CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.

Modifie :      AF.

Note :        Valide en SCREEN 3 uniquement.

#### 000A5h (Sub-ROM)     **RIGHTC**    RIGHT Current

Rôle :            Déplacement d'un pixel vers la droite  
Entrée :        CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.  
Sortie :        CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.  
Modifie :      AF.  
Note :        Valide en SCREEN 3 uniquement.

#### 000A9h (Sub-ROM)     **TLEFTC**    Test LEFT Current

Rôle :            Déplacement d'un pixel vers la gauche après vérification de la validité des coordonnées.  
Entrée :        CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.  
Sortie :        CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.  
Modifie :      AF.  
Note :        Valide en SCREEN 3 et 5 à 8.

#### 000ADh (Sub-ROM)     **LEFTC**     LEFT Current

Rôle :            Déplacement d'un pixel vers la gauche  
Entrée :        CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.  
Sortie :        CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.  
Modifie :      AF  
Note :        Valide en SCREEN 3 uniquement.

#### 000B1h (Sub-ROM)     **TDOWNC** Test DOWN Current

Rôle :            Déplacement d'un pixel vers le bas après vérification de la validité des coordonnées.  
Entrée :        CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.  
Sortie :        CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.  
Modifie :      AF.  
Note :        Valide en SCREEN 3 et 5 à 8.

#### 000B5h (Sub-ROM)     **DOWNC**    DOWN Current

Rôle :            Déplacement d'un pixel vers le bas.  
Entrée :        CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.  
Sortie :        CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.  
Modifie :      AF.  
Note :        Valide en SCREEN 3 uniquement.

### 000B9h (Sub-ROM)      **TUPC**      Test UP Current

Rôle : Déplacement d'un pixel vers le haut après vérification de la validité des coordonnées.  
Entrée : CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.  
Sortie : CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.  
Modifie : AF.  
Note : Valide en SCREEN 3 et 5 à 8.

### 000BDh (Sub-ROM)      **UPC**      UP Current

Rôle : Déplacement d'un pixel vers le haut.  
Entrée : CLOC (0F92Ah) et CMASK (0F92Ch) = Coordonnées du pixel actuel.  
Sortie : CLOC (0F92Ah) et CMASK (0F92Ch) = Nouvelles coordonnées.  
Modifie : AF.  
Note : Valide en SCREEN 3 uniquement.

### 000C1h (Sub-ROM)      **SCANR**      SCAN Right

Rôle : Recherche vers la droite du premier pixel d'une autre couleur.  
Entrée : DE = Nombre de pixels sur lesquels la recherche doit être effectuée.  
Sortie : DE = Position du pixel recherché.  
Modifie : Tout.  
Note : Voir SCANR en Main-ROM pour plus d'explications.

### 000C5h (Sub-ROM)      **SCANL**      SCAN Left

Rôle : Recherche vers la gauche du premier pixel d'une autre couleur.  
Entrée : DE = Nombre de pixels sur lesquels la recherche doit être effectuée.  
Sortie : DE = Position du pixel recherché.  
Modifie : Tout.  
Note : Voir SCANL en Main-ROM pour plus d'explications.

## 000C9h (Sub-ROM)      **NVBXLN**

- Rôle :            Tracé d'un rectangle à l'écran.
- Entrée :        BC = Abscisse du pixel de départ.  
                 DE = Ordonnée du pixel de départ.  
                 GXPOS (0FCB3h) = Abscisse du pixel d'arrivée.  
                 GYPOS (0FCB5h) = Ordonnée du pixel d'arrivée.  
                 ATRBYT (0F3F2h) = Couleur de tracé (0 ~ 3 en SCREEN 6 ; 0 ~ 15 en SCREEN 5 et 7 ; 0 ~ 255 dans les SCREEN 8 à 12).  
                 LOGOPR (0FB02h) = Opérateur logique à appliquer.
- Sortie :        Rien.
- Modifie :      AF.
- Notes :        - Valide dans les SCREEN 5 à 8 / 12.  
                 - Voir DOGRPH (00085h en Sub-ROM) pour l'explication de LOGOPR.

## 000CDh (Sub-ROM)      **NVBXFL**

- Rôle :            Tracé d'un rectangle plein à l'écran.
- Entrée :        BC = Abscisse du pixel de départ.  
                 DE = Ordonnée du pixel de départ.  
                 GXPOS (0FCB3h) = Abscisse du pixel d'arrivée.  
                 GYPOS (0FCB5h) = Ordonnée du pixel d'arrivée.  
                 ATRBYT (0F3F2h) = Couleur de tracé (0 ~ 3 en SCREEN 6 ; 0 ~ 15 en SCREEN 5 et 7 ; 0 ~ 255 dans les SCREEN 8 à 12).  
                 LOGOPR (0FB02h) = Opérateur logique à appliquer.
- Sortie :        Rien.
- Modifie :      AF
- Notes :        - Valide dans les SCREEN 5 à 8 / 12.  
                 - Voir DOGRPH (00085h en Sub-ROM) pour l'explication de LOGOPR

## 000D1h (Sub-ROM)      **CHGMOD** CHanGe MODe

- Rôle :            Changement de mode d'écran (SCREEN X).
- Entrée :        A = SCREEN désiré. (0 ~ 8 / 12)
- Sortie :        SCRMOD (0FCAfh) = Nouveau mode d'écran.
- Modifie :      Tout.
- Note :          La palette n'est pas initialisée par cette routine, utilisez la routine CHGMDP (001B5h en Sub-ROM) si vous avez besoin de l'initialisation de la palette.



## 000D5h (Sub-ROM)      **INITXT**      INItialize TeXT mode

- Rôle :            Initialisation du mode SCREEN 0.
- Entrée :        TXTNAM (0F3B3h) = Adresse de la table des caractères.  
                  TXTCOL (0F3B5h) = Adresse de la table des couleurs.  
                  TXTCGP (0F3B7h) = Adresse de la table des formes.  
                  TXTATR (0F3B9h) = Adresse de la table des formes.  
                  TXTCOL (0F3B5h) = Adresse de la table des couleurs.  
                  TXTPAT (0F3BBh) = Adresse de la table des caractères.  
                  LINL40 (0F3AEh) = Nombre de caractères par lignes.  
                  FORCLR (0F3E9h) = Couleur de texte.  
                  BAKCLR (0F3EAh) = Couleur de fond.  
                  BDRCLR (0F3EBh) = Couleur de bordure.
- Sortie :        Rien.
- Modifie :      Tout.
- Note :        Il n'est pas obligatoire de définir TXTNAM, etc car le MSX initialise leur valeur à l'allumage.

## 000D9h (Sub-ROM)      **INIT32**      INItialize Text 32 mode

- Rôle :            Initialisation du mode SCREEN 1.
- Entrée :        T32NAM (0F3BDh) = Adresse de la table des caractères.  
                  T32COL (0F3BFh) = Adresse de la table des couleurs.  
                  T32CGP (0F3C1h) = Adresse de la table des formes.  
                  T32ATR (0F3C3h) = Adresse de la table des attributs de Sprites.  
                  T32PAT (0F3C5h) = Adresse de la table des formes de Sprites.  
                  LINL32 (0F3AFh) = Nombre de caractères par lignes.  
                  FORCLR (0F3E9h) = Couleur de texte.  
                  BAKCLR (0F3EAh) = Couleur de fond.  
                  BDRCLR (0F3EBh) = Couleur de bordure.
- Sortie :        Rien.
- Modifie :      Tout.
- Note :        Il n'est pas obligatoire de définir T32NAM, etc car le MSX initialise leur valeur à l'allumage.

## 000DDh (Sub-ROM)     **INIGRP**     INItialize GRaPhic mode

Rôle :            Initialisation du mode graphique 256x192 (SCREEN 2).

Entrée :          GRPNAM (0F3C7h) = Adresse de la table des motifs.  
                    GRPCOL (0F3C9h) = Adresse de la table des couleurs.  
                    GRPCGP (0F3CBh) = Adresse de la table des formes.  
                    GRPATR (0F3CDh) = Adresse de la table des attributs de Sprites.  
                    GRPPAT (0F3CFh) = Adresse de la table des formes de Sprites.  
                    FORCLR (0F3E9h) = Couleur de texte.  
                    BAKCLR (0F3EAh) = Couleur de fond.  
                    BDRCLR (0F3EBh) = Couleur de bordure.

Sortie :          Rien.

Modifie :        Tout.

Note :            Il n'est pas obligatoire de définir GRPNAM, etc car le MSX initialise leur valeur à l'allumage.

## 000E1h (Sub-ROM)     **INIMLT**     INItialize MuLTicolor mode

Rôle :            Initialisation du mode SCREEN 3.

Entrée :          MLTNAM (0F3D1h) = Adresse de la table des caractères.  
                    MLTCOL (0F3D3h) = Adresse de la table des couleurs.  
                    MLTCGP (0F3D5h) = Adresse de la table des formes.  
                    MLTATR (0F3D7h) = Adresse de la table des attributs de Sprites.  
                    MLTPAT (0F3D9h) = Adresse de la table des formes de Sprites.  
                    FORCLR (0F3E9h) = Couleur de texte.  
                    BAKCLR (0F3EAh) = Couleur de fond.  
                    BDRCLR (0F3EBh) = Couleur de bordure.

Sortie :          Rien.

Modifie :        Tout.

Note :            Il n'est pas obligatoire de définir MLTNAM, etc car le MSX initialise leur valeur à l'allumage.

## 000E5h (Sub-ROM)     **SETTXT**     SET TeXT mode

Rôle :            Passage direct en mode texte 40x24.

Entrée :          TXTNAM (0F3B3h) = Adresse de la table des caractères.  
                    TXTCGP (0F3B7h) = Adresse de la table des formes.

Sortie :          Rien.

Modifie :        Tout.

Note :            Il n'est pas obligatoire de définir TXTNAM et TXTCGP car le MSX initialise leur valeur à l'allumage.

### 000E9h (Sub-ROM)      **SETT32**    SET Text 32 mode

Rôle :            Passage direct en mode texte 32x24.

Entrée :        T32NAM (0F3BDh) = Adresse de la table des caractères.  
                  T32COL (0F3BFh) = Adresse de la table des couleurs.  
                  T32CGP (0F3C1h) = Adresse de la table des formes.  
                  T32ATR (0F3C3h) = Adresse de la table des attributs de Sprites.  
                  T32PAT (0F3C5h) = Adresse de la table des formes de Sprites.

Sortie :        Rien.

Modifie :      Tout.

Note :        Il n'est pas obligatoire de définir T32NAM, etc car le MSX initialise leur valeur à l'allumage.

### 000EDh (Sub-ROM)      **SETGRP**    SET GRaPhic mode

Rôle :            Passage direct en mode SCREEN 2.

Entrée :        GRPNAM (0F3C7h) = Adresse de la table des caractères.  
                  GRPCOL (0F3C9h) = Adresse de la table des couleurs.  
                  GRPCGP (0F3CBh) = Adresse de la table des formes.  
                  GRPATR (0F3CDh) = Adresse de la table des attributs de Sprites.  
                  GRPPAT (0F3CFh) = Adresse de la table des formes de Sprites.

Sortie :        Rien.

Modifie :      Tout.

Note :        Il n'est pas obligatoire de définir GRPNAM, etc car le MSX initialise leur valeur à l'allumage.

### 000F1h (Sub-ROM)      **SETMLT**    SET MuLTicolor mode

Rôle :            Passage direct en mode SCREEN 3.

Entrée :        MLTNAM (0F3D1h) = Adresse de la table des caractères.  
                  MLTCOL (0F3D3h) = Adresse de la table des couleurs.  
                  MLTCGP (0F3D5h) = Adresse de la table des formes.  
                  MLTATR (0F3D7h) = Adresse de la table des attributs de Sprites.  
                  MLTPAT (0F3D9h) = Adresse de la table des formes de Sprites.

Sortie :        Rien.

Modifie :      Tout.

Note :        Il n'est pas obligatoire de définir MLTNAM, etc car le MSX initialise leur valeur à l'allumage.

### 000F5h (Sub-ROM)      **CLRSPR**    CLear Sprites

Rôle :            Initialisation des Sprites.  
Entrée :        SCRMOD (0FCAFh) = Mode d'écran actuel.  
Sortie :        Rien.  
Modifie :      Tout.  
Note :        La table des formes de Sprites est effacée, les numéros de Sprites sont mis aux numéros de plans, la couleur des Sprites est mise à celle de la couleur définie par FORCLR (0F3E9h) et la position des Sprites est réglée à 217.

### 000F9h (Sub-ROM)      **CALPAT**    CALculate address of PATtern table

Rôle :            Calcul de l'adresse du début des formes d'un Sprite.  
Entrée :        A = Numéro de plan du Sprite.  
Sortie :        HL = Adresse cherchée.  
Modifie :      AF, DE, HL.  
Note :        Idem CALPAT en Main-ROM.

### 000FDh (Sub-ROM)      **CALATR**    CALculate address of ATRibute table

Rôle :            Calcul de l'adresse du début des attributs d'un Sprite.  
Entrée :        A = Numéro de plan du Sprite  
Sortie :        HL = Adresse cherchée.  
Modifie :      AF, DE, HL.  
Note :        Idem CALATR en Main-ROM.

### 00101h (Sub-ROM)      **GSPSIZ**    Get Sprite SIZE

Rôle :            Indique la taille actuel des Sprites.  
Entrée :        Rien.  
Sortie :        A = Taille d'un Sprite. (8 ou 16).  
                  F = L'indicateur C est mis à 1 si la taille des Sprites est 16x16 ; 0 si 8x8.  
Modifie :      AF.  
Note :        Idem GSPSIZ en Main-ROM.

### 00105h (Sub-ROM)      **GETPAT**    GET PATtern

Rôle :            Obtenir la forme d'un caractère.  
Entrée :        A = Code ASCII du caractère.  
Sortie :        PATWRK (0FC40h) = Les huit octets qui définissent la forme du caractère.  
Modifie :      Tout.

### 00109h (Sub-ROM)      **WRTVRM** WRiTe VRaM

Rôle :            Écriture dans une case mémoire de la VRAM  
Entrée :        HL = Adresse de case mémoire. (0 ~ 0FFFFh)  
                  A = Donnée à écrire.  
Sortie :        Rien.  
Modifie :      AF.  
Note :        Cette routine permet l'accès à toute la VRAM, au contraire du WRTVRM de la Main-ROM.

### 0010Dh (Sub-ROM)      **RDVRM**    ReaD VRaM

Rôle :            Lecture d'une case mémoire de la VRAM  
Entrée :        HL = Adresse de case mémoire (0 ~ 0FFFFh)  
Sortie :        A = Contenu de la case mémoire lue.  
Modifie :      AF.  
Note :        Cette routine permet l'accès à toute la VRAM, au contraire du RDVRM de la Main-ROM.

### 00111h (Sub-ROM)      **CHGCLR** CHanGe CoLoR

Rôle :            Change les couleurs de l'écran spécifié avec les valeurs par défaut.  
Entrée :        A = Mode d'écran.  
                  FORCLR (0F3E9h) = Couleur de texte ou de tracé.  
                  BAKCLR (0F3EAh) = Couleur de fond.  
                  BDRCLR (0F3EBh) = Couleur de bordure.  
Sortie :        Rien.  
Modifie :      Tout.  
Notes :        - En mode graphique, seule la couleur de bordure change.  
                  - Même fonction que CHGCLR (00062h) de la Main-ROM.

### 00115h (Sub-ROM)      **CLS**            CLear Screen

Rôle :            Effacement de l'écran.  
Entrée :        Rien.  
Sortie :        Rien.  
Modifie :      Tout.

**00119h (Sub-ROM)      CLRTXT    CLeaR TeXT screen**

Rôle :            Effacement de l'écran en mode texte.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        Tout.

**0011Dh (Sub-ROM)      DSPFNK    DiSPlay FuNction Keys**

Rôle :            Affichage des touches de fonction.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Valide en SCREEN 0 et 1 uniquement.

**00121h (Sub-ROM)      DELLN0    DELeTe LiNe mode 0**

Rôle :            Effacement d'une ligne en mode texte.  
Entrée :          L = Numéro de la ligne  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Valide en SCREEN 0 uniquement.

**00125h (Sub-ROM)      INSLN0    INSert LiNe mode 0**

Rôle :            Insertion d'une ligne en mode texte.  
Entrée :          L = Numéro de la ligne.  
Sortie :          Rien.  
Modifie :        Tout.  
Note :            Valide en SCREEN 0 uniquement.

**00129h (Sub-ROM)      PUTVRM    PUT character in VRaM**

Rôle :            Affichage d'un caractère en mode texte.  
Entrée :          C = Code ASCII du caractère à afficher.  
                    H = Numéro de colonne.  
                    L = Numéro de ligne.  
Sortie :          Rien.  
Modifie :        AF.

**0012Dh (Sub-ROM)      WRTVDP   WRiTe VDP**

Rôle :            Écriture dans un registre du VDP.  
Entrée :        C = Numéro de registre (0 ~ 23 et 32 ~ 46).  
                    B = Donnée à écrire.  
Sortie :        Rien.  
Modifie :      AF, BC

**00131h (Sub-ROM)      VDPSTA   VDP STATus**

Rôle :            Lecture d'un des registres de statut du VDP.  
Entrée :        A = Numéro de registre (0 ~ 9).  
Sortie :        A = Contenu du registre lu.  
Modifie :      F

**00135h (Sub-ROM)      KYKLOK   KeY Kana LOcK**

Rôle :            Traitement du voyant et des touches KANA.  
Entrée :        Rien.  
Sortie :        Rien.  
Modifie :      AF  
Note :           Sans grand intérêt pour les MSX disponibles en France.

**00139h (Sub-ROM)      PUTCHR   PUT kana CHaRacter**

Rôle :            Détection de l'appui sur une touche du clavier et transformation en code KANA.  
Entrée :        F = Indicateur Z à 1 si pas de conversion.  
Sortie :        Rien.  
Modifie :      Tout.  
Note :           Sans grand intérêt pour les MSX disponibles en France.

**0013Dh (Sub-ROM)      SETPAG   SET PAGe**

Rôle :            Changement de page graphique.  
Entrée :        ACPAGE = Page affichée à l'écran.  
                    DPPAGE = Page sur laquelle on travaille.  
Sortie :        Rien.  
Modifie :      AF.

**00141h (Sub-ROM)      INIPLT      INItialize PaLeTte**

Rôle :            Initialisation de la palette de couleurs.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        AF, BC, DE.  
Note :            Équivalent de l'instruction Basic « COLOR=NEW ».

**00145h (Sub-ROM)      RSTPLT      ReSTore PaLeTte**

Rôle :            Récupération de la palette depuis la VRAM.  
Entrée :          Rien.  
Sortie :          Rien.  
Modifie :        AF, BC, DE.  
Note :            Équivalent de l'instruction Basic « COLOR=RESTORE ».

**00149h (Sub-ROM)      GETPLT      GET PaLeTte**

Rôle :            Récupération des valeurs la palette d'une couleur.  
Entrée :          A = Numéro de la couleur.  
Sortie :          B = 4 bits de poids fort pour le rouge, 4 bits de poids faible pour le bleu.  
                    C = 4 bits de poids faible pour le vert.  
Modifie :        AF, DE.

**0014Dh (Sub-ROM)      SETPLT      SET PaLeTte**

Rôle :            Modification d'une couleur dans la palette.  
Entrée :          D = Numéro de la couleur (0 ~ 15).  
                    A = 4 bits de poids fort pour le rouge, 4 bits de poids faible pour le bleu.  
                    E = 4 bits de poids faible pour le vert.  
Sortie :          Rien.  
Modifie :        AF.

**00151h (Sub-ROM)      PUTSPR      PUT Sprite**

Rôle :            Affichage d'un Sprite.  
Entrée :          HL = Pointeur de texte du Basic.  
Sortie :          HL = Pointeur réactualisé.  
Modifie :        Tout.  
Note :            Utilisé par l'interpréteur du Basic.



### 00155h (Sub-ROM)      **COLOR**    COLOR

Rôle :            Changement de couleurs, de la palette, de couleurs de Sprite.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

### 00159h (Sub-ROM)      **SCREEN**    SCREEN

Rôle :            Modification de tous les paramètres définis par l'instruction Basic. « SCREEN ».  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

### 0015Dh (Sub-ROM)      **WIDTHS**    WIDTH Screen

Rôle :            Modification de la largeur de l'écran.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

### 00161h (Sub-ROM)      **VDP**            VDP

Rôle :            Écriture dans un registre du VDP.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

### 00165h (Sub-ROM)      **VDPF**            VDP Fetch

Rôle :            Lecture d'un registre du VDP.  
Entrée :        HL = Pointeur de texte du Basic  
Sortie :        HL = Pointeur réactualisé  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

**00169h (Sub-ROM)      BASE      BASE**

Rôle :            Écriture dans un registre « base » du VDP.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

**0016Dh (Sub-ROM)      BASEF      BASE Fetch**

Rôle :            Lecture d'un registre « base » du VDP.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

**00171h (Sub-ROM)      VPOKE      Video POKE**

Rôle :            Écriture dans la mémoire vidéo.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

**00175h (Sub-ROM)      VPEEK      Video PEEK**

Rôle :            Lecture de la mémoire vidéo.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

**00179h (Sub-ROM)      SETS      SET Screen**

Rôle :            Traitement des instructions « SET » (SCREEN, ADJUST, TIME, etc).  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé.  
Modifie :      Tout.  
Note :        Utilisé par l'interpréteur du Basic.

**0017Dh (Sub-ROM)      BEEP      BEEP**

Rôle :            Émission d'un bref « bip » sonore.  
Entrée :        Rien.  
Sortie :        Rien.  
Modifie :      Tout.  
Note :        Ce son est produit par le PSG.

**00181h (Sub-ROM)      PROMPT      PROMPT**

Rôle :            Affichage de OK ou du message défini par l'utilisateur.  
Entrée :        Rien.  
Sortie :        Rien.  
Modifie :      Tout.

**00185h (Sub-ROM)      SDFSCR      Set DeFault SCReen**

Rôle :            Restaure l'écran initial avec les paramètres sauvegardés dans la SRAM de l'horloge (largeur d'écran, couleurs, etc).  
Entrée :        L'indicateur Carry doit être à 1 pour afficher le texte des touches de fonction, sinon 0.  
Sortie :        Rien.  
Modifie :      Tout.

**00189h (Sub-ROM)      SETSCR      SET start SCReen**

Rôle :            Similaire à SDFSCR avec en plus l'affichage du message de départ à l'allumage.  
Entrée :        Rien.  
Sortie :        Rien.  
Modifie :      Tout.

**0018Dh (Sub-ROM)      SCOPY      Screen COPY**

Rôle :            Traitement de l'instruction « COPY » vidéo.  
Entrée :        HL = Pointeur de texte du Basic.  
Sortie :        HL = Pointeur réactualisé  
Modifie :      Tout.

## 00191h (Sub-ROM)      **BLTVV**      BLock Transfer Vram to Vram

Rôle :            Transfert d'un bloc d'un endroit de mémoire vidéo à un autre.

Entrée :        HL = 0F562h (pointeur).

Sortie :        Rien.

Modifie :      Tout.

Exemple d'utilisation :

Faire « COPY (32,16)-(128,96) TO (192,128) » en assembleur.

```
EXTROM equ 0015fh
BLTVV   equ 00191h
SX      equ 0f562h      ; abscisse du pixel de départ
SY      equ SX+2        ; ordonnée du pixel de départ
DX      equ SY+2        ; abscisse pixel de destination
DY      equ DX+2        ; ordonnée pixel de destination
NX      equ DY+2        ; largeur du bloc à copier
NY      equ NX+2        ; hauteur du bloc à copier
ARG      equ NY+3
LOGOP   equ ARG+1      ; opérateur logique

DEBUT:   org 0c000h

        ld hl,20h
        ld (SX),hl
        ld hl,10h
        ld (SY),hl
        ld hl,0c0h
        ld (DX),hl
        ld hl,80h
        ld (DY),hl
        ld hl,80h-20h+1 ; calcul de la largeur
        ld (NX),hl
        ld HL,60h-10h+1 ; calcul de la hauteur
        ld (NY),hl
        xor a           ; ARG doit toujours être
        ld (ARG),a      ; mis à zéro !
        ld a,3
        ld (LOGOP),a    ; au même format que LOGOPR

        ld hl,SX
        ld ix,BLTVV     ; adresse de la routine à appeler
        call EXTROM
        ret
```

Le VDP travaille comme s'il n'y avait qu'un écran géant dans lequel on déplace une fenêtre suivant le schéma :

SCREEN 5		VRAM	SCREEN 6	
(0,0)	(255,0)	00000h	(0,0)	(511,0)
page 0			page 0	
(0,255)	(255,255)	07FFFh	(0,255)	(511,255)
(0,256)	(255,256)	08000h	(0,256)	(511,256)
page 1			page 1	
(0,511)	(255,511)	0FFFFh	(0,511)	(511,511)
(0,512)	(255,512)	10000h	(0,512)	(511,512)
page 2			page 2	
(0,767)	(255,767)	17FFFh	(0,767 )	(511,767)
(0,768)	(255,768)	18000h	(0,768)	(511,768)
page 3			page 3	
(0,1023)	(255,1023)	1FFFFh	(0,1023)	(511,1023)

SCREEN 7		VRAM	SCREEN 8 ~ 12	
(0,0)	(511,0)	00000h	(0,0)	(255,0)
page 0			page 0	
(0,255)	(511,255)	0FFFFh	(0,255)	(255,255)
(0,256)	(511,256)	10000h	(0,256)	(255,256)
page 1			page 1	
(0,511)	(511,511)	1FFFFh	(0,511)	(255,511)

## 00195h (Sub-ROM)      **BLTVM**      BLock Transfert Vram from Memory

Rôle :            Transfert de la mémoire centrale vers la mémoire vidéo

Entrée :        HL = 0F562h (pointeur).

Sortie :        Rien.

Modifie :      Tout.

Exemple d'utilisation :

Faire « COPY &HD000 TO (16,32) » en assembleur.

```
EXTROM equ 0015fh
BLTVM  equ 00195h
DPTR   equ 0f562h
DX      equ DPTR+4      ; Abscisse pixel de destination
DY      equ DX+2        ; Ordonnée pixel de destination
ARG     equ DY+7
LOGOP   equ ARG+1       ; Opérateur logique

      org 0c000h
DEBUT:
      ld  hl,0d000h
      ld  (DPTR),hl      ; Adresse du début des données
      ld  hl,10h
      ld  (DX),hl
      ld  hl,20h
      ld  (DY),hl
      xor  a              ; ARG doit toujours être
      ld  (ARG),a         ; mis à zéro !!!
      ld  a,2
      ld  (LOGOP),a       ; Au même format que LOGOPR

      ld  hl,DPTR
      ld  ix,BLTVM        ; Adresse de la routine à appeler
      call EXTROM
      ret

;
      org 0d000h
P_HORZ: dw 030h           ; Nombre de pixels horizontaux
P_VERT: dw 00bh           ; Nombre de pixels verticaux
NBOCT:  ds 210h           ; Nombre d'octets nécessaires
```

Pour calculer NBOCT :

en SCREEN 5 - (NBOCT) = (P. HORZ)/2 \* (P.VERT) +1

en SCREEN 6 - (NBOCT) = (P. HORZ)/4 \* (P.VERT) +1

en SCREEN 7 - (NBOCT) = (P. HORZ)/2 \* (P.VERT) +1

en SCREEN 8 - (NBOCT) = (P. HORZ) \* (P.VERT)

Pour plus de précisions, voir la routine BLTVV.

## 00199h (Sub-ROM)      **BLTMV**      BLock Transfert Memory from Vram

Rôle :            Transfert de la mémoire vidéo vers la mémoire centrale.

Entrée :        HL = 0F562h (pointeur).

Sortie :        Rien.

Modifie :      Tout.

Exemple d'utilisation :

Faire « COPY (16,32) TO &HD000 » en assembleur.

```
EXTROM equ 0015Fh
BLTMV  equ 00199h
SX      equ 0F562h      ; abscisse du pixel de départ
SY      equ SX+2        ; ordonnée du pixel de départ
DPTR    equ SY+2        ; adresse en mémoire centrale
NX      equ SY+6        ; abscisse du pixel de destination
NY      equ NX+2        ; ordonnée du pixel de destination
ARG     equ NY+3
LOGOP   equ ARG+1      ; opérateur logique

        org 0c000h
DEBUT:
        ld hl,0d000h
        ld (DPTR),hl
        ld hl,10h
        ld (SX),hl
        ld hl,20h
        ld (SY),hl
        ld hl,60h-10h+1 ; calcul de la largeur
        ld (NX),hl
        ld hl,40h-20h+1 ; calcul de la hauteur
        ld (NY),hl
        xor a           ; ARG doit toujours être
        ld (ARG),a      ; mis à zéro !!!
        ld a,2
        ld (LOGOP),a    ; au même format que LOGOPR

        ld hl,SX
        ld ix,BLTMV     ; Adresse de la routine à appeler
        call EXTROM
        ret
```

Vous trouverez en 0D000h et 0D001h la largeur du bloc, en 0D002h et 0D003h la hauteur du bloc. Pour calculer le nombre d'octets utilisés à partir de 0D004h pour stocker le bloc, appliquez la formule adéquate :

en SCREEN 5 - nb octets = largeur/2 \* hauteur +1

en SCREEN 6 - nb octets = largeur/4 \* hauteur +1

en SCREEN 7 - nb octets = largeur/2 \* hauteur +1

en SCREEN 8 - nb octets = largeur \* hauteur

Pour plus de précisions, voir la routine BLTVV.

## 0019Dh (Sub-ROM)      **BLTVD**      BLock Transfert Vram from Disk

Rôle :            Transfert de la disquette vers la mémoire vidéo.

Entrée :        HL = 0F562h (pointeur).

Sortie :        F = Carry est mis à 0.

Modifie :      Tout.

Exemple d'utilisation :

Faire « COPY "A:ESSAI.SC7" to (16,32) » en assembleur.

```
EXTROM equ 0015fh
BLTVD equ 0019dh
FNPTR equ 0f562h ; Nom du fichier dans BUF
DX equ FNPTR+4 ; Abscisse de destination
DY equ DX+2 ; Ordonnée de destination
ARG equ DY+7
LOGOP equ ARG+1 ; Opérateur logique

db 0feh ; Entête pour les
dw DEBUT,END,DEBUT ; fichiers binaires

org 0c000h
DEBUT:
ld hl,NOM
ld (FNPTR),hl
ld hl,10h
ld (DX),hl
ld hl,20h
ld (DY),hl
xor a ; Mise à zéro !
ld (ARG),a
ld (LOGOP),a ; Au même format que LOGOPR
;
; Ne pas oublier ce qui suit.
;
ld hl,FNPTR
ld ix,BLTVD
call EXTROM
ret
NOM:
db 022h,'A:ESSAI.SC7',022h,0
```

END:

Assemblez la routine sous le nom "BLTVD.BIN", puis lancer la avec le programme Basic suivant :

```
10 CLEAR,&HC000: SCREEN7
20 BLOAD"BLTVD.BIN": DEFUSR=&HC000: A$=USR(0)
```

Vous aurez le nom du fichier en sortie dans la variable A\$.

Pour plus de précisions, voir les routines BLTVV et BLTDV.



## 001A1h (Sub-ROM)      **BLTDV**      BLock Transfert Disk from Vram

Rôle :            Transfert de la mémoire vidéo vers la disquette.

Entrée :        HL = 0F562h (pointeur).

Sortie :        Rien.

Modifie :      Tout.

Exemple d'utilisation :

Faire « COPY (16,32)-(96,128) TO "A:ESSAI.SC7" » en assembleur.

```
EXTROM equ 0015fh
BLTDV   equ 001a1h
SX      equ 0f562h      ; Abscisse du pixel de départ
SY      equ SX+2        ; Ordonnée du pixel de départ
FNPTR   equ SY+2        ; Pointeur sur le nom de fichier
NX      equ SY+6        ; Abscisse du pixel de destination
NY      equ NX+2        ; Ordonnée du pixel de destination
ARG     equ NY+3

        db 0feh          ; Entête pour les
        dw DEBUT,END,DEBUT ; fichiers binaires

        org 0c000h
DEBUT:
        ld hl,NOM
        ld (FNPTR),hl
        ld hl,10h
        ld (SX),hl
        ld hl,20h
        ld (SY),hl
        ld hl,60h-10h+1 ; calcul de la largeur
        ld (NX),hl
        ld hl,80h-20h+1 ; calcul de la hauteur
        ld (NY),hl
        xor a           ; ARG doit toujours être
        ld (ARG),a      ; mis à zéro !!!

        ld hl,SX
        ld ix,BLTDV     ; Adresse de la routine à appeler
        call EXTROM
        ret

NOM:    db 022h,'A:ESSAI.SC7',022h,0
END:
```

Assemblez la routine sous le nom "BLTDV.BIN", puis lancer la avec le programme Basic suivant :

```
10 CLEAR,&HC000: SCREEN7: CIRCLE(128,106),105,6
20 BLOAD"BLTDV.BIN": DEFUSR=&HC000: A$=USR(0)
```

Vous aurez le nom du fichier en sortie dans la variable A\$.

Pour plus de précisions, voir les routines BLTVV et BLTVD.

## 001A5h (Sub-ROM)     **BLTMD**     BLock Transfert Memory from Disk

Rôle :            Transfert de la disquette vers la mémoire centrale.

Entrée :        HL = 0F562h (pointeur).

Sortie :        Rien.

Modifie :      Tout.

Exemple d'utilisation :

Faire « COPY "A:ESSAI.SC7" TO <0D000h> » en assembleur.

```
EXTROM equ 0015fh
BLTMD  equ 001a5h
FNPTR  equ 0f562h      ; pointeur du nom de fichier
SPTR   equ FNPTR+4     ; adresse de départ en mémoire
EPTR   equ FNPTR+6     ; adresse de fin en mémoire

      org 0c000h
DEBUT:
      ld  hl,NOM
      ld  (FNPTR),hl
      ld  hl,0d000h
      ld  (SPTR),hl
      ld  hl,0d020h
      ld  (EPTR),hl

      ld  hl,FNPTR
      ld  ix,BLTMD      ; Adresse de la routine à appeler
      call EXTROM
      ret

;
NOM:   db  022h,'A:ESSAI.SC7',022h,0
END:
```

Si vous appelez cette routine depuis le Basic, faites-le par un  
DEFUSR=&HC000 : A\$=USR(0). N'oubliez pas le dollar. Dans A\$,  
vous aurez le nom du fichier.

## 001A9h (Sub-ROM)      **BLTDM**      BLock Transfert Disk from Memory

Rôle :            Transfert de la mémoire centrale vers la disquette.

Entrée :        HL = 0F562h (pointeur).

Sortie :        Rien.

Modifie :      Tout.

Exemple d'utilisation :

```
; Faire un COPY <0D000h> TO "A:ESSAI.SC7" en assembleur
```

```
EXTROM equ 0015fh
BLTDM  equ 001a9h
SPTR   equ 0f562h      ; Adresse du début des données
EPTR   equ SPTR+2      ; Adresse de fin des données en mémoire
FNPTR  equ SPTR+4      ; Pointeur sur le nom de fichier

DEBUT:  org 0c000h
        ld hl,NOM
        ld (FNPTR),hl
        ld hl,0d000h
        ld (SPTR),hl
        ld hl,0d020h
        ld (EPTR),hl

        ld hl,SPTR
        ld ix,BLTDM    ; Adresse de la routine à appeler
        call EXTROM
        ret

NOM:
        db 022h,'A:ESSAI.SC7',022h,0

END:
```

Si vous appelez cette routine depuis le Basic, faites-le par un  
DEFUSR=&HC000 : A\$=USR(0). N'oubliez pas le dollar. Dans A\$,  
vous aurez le nom du fichier.

### 001ADh (Sub-ROM)     **NEWPAD**   NEW get touch PAD

Rôle :            Lecture de la tablette graphique, du stylo optique, de la souris ou du Track-ball (Cat).

Entrée :          A = Numéro de l'opération à effectuer (0 ~ 19).

Sortie :          A = résultat de l'opération.

Modifie :        Tout.

Notes :          - Sur MSX2 ou plus récent, il est possible d'appeler la routine **GTPAD** en Main-ROM à place. C'est juste un peu plus rapide d'appeler **NEWPAD** directement.

                     - Voir la routine **GTPAD** (000DBh) en Main-ROM pour connaître les opérations possibles.

### 001B1h (Sub-ROM)     **GETPUT**   GET time/date & PUT kanji

Rôle :            Récupération de l'heure et de la date et traitement des caractères kanji.

Entrée :          HL = Pointeur de texte du Basic.

Sortie :          Rien.

Modifie :        Tout.

### 001B5h (Sub-ROM)     **CHGMDP**   CHanGe Mode DisPlay

Rôle :            Changement de mode d'écran (SCREEN X).

Entrée :          A = Numéro du mode d'écran (0 ~ 8 / 12).

Sortie :          Rien.

Modifie :        Tout.

Note :            Par rapport à CHGMOD (000D1h), cette routine initialise la palette.

### 001B9h (Sub-ROM)     **RESV1**     RESerVed 1

Rôle :            Réserve aux versions futures de MSX.

Entrée :          Rien.

Sortie :          Rien.

Modifie :        Rien.

### 001BDh (Sub-ROM)     **KNJPRT**   KaNJi PRinT

Rôle :            Affichage d'un caractère kanji à l'écran.

Entrée :          BC = Code du caractère kanji.

                     A = Type d'affichage.

Sortie :          Rien.

Modifie :        AF.

Note :            Routine présente sur les MSX japonais ayant une Kanji-ROM.

## 001F5h (Sub-ROM) **REDCLK** REaD CLoCK

Rôle : Lecture d'un registre de l'horloge interne (RTC). L'état des registres sont conservés grâce à une pile ou une batterie selon le MSX.

Entrée : C = Numéro du bloc et du registre à lire.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
-	-	Bloc		Registre (0 ~ 12)			

Sortie : A = Contenu de la case mémoire lue. (quatre bits de poids faible)

Modifie : F.

Note : L'adresse mémoire se décompose de la façon. Les cases grises sont inutilisées.

Bloc 0 :

Registre	bit 3	bit 2	bit 1	bit 0
0	Compteur de l'unité des secondes			
1	-	Compteur de la dizaine des secondes		
2	Compteur de l'unité des minutes			
3	-	Compteur de la dizaine des minutes		
4	Compteur de l'unité des heures			
5	-	-	Compteur de la dizaine des heures	
6		Compteur du jour des semaines		
7	Compteur de l'unité des jours des mois			
8	-	-	Compteur de la dizaine des jours des mois	
9	Compteur de l'unité des mois			
10 (Ah)	-	-	-	Dizaine des mois
11 (Bh)	Compteur de l'unité des années			
12 (Ch)	Compteur de la dizaine des années			

Bloc 1 (01 en binaire) :

Registre	bit 3	bit 2	bit 1	bit 0
0	-			
1	-			
2	Unité des minutes de l'alarme			
3	-	Dizaine des minutes de l'alarme		
4	Unité de l'heure de l'alarme			
5	-	-	Dizaine de l'heure de l'alarme	
6		Jour de semaine de l'alarme		
7	Unité du jour du mois de l'alarme			
8	-	-	Dizaine du jour du mois de l'alarme	
9	-			
10 (Ah)	-	-	-	12 / 24 heure
11 (Bh)	-	-	Compteur pour année bissextile	
12 (Ch)	-			

Note : Ce bloc n'est pas utilisé par le système mais l'interface SCSI Gouda (Novaxis) utilise certains de ces registres pour préserver quelques paramètres comme l'ID hôte et l'ID cible.

Bloc 2 (10 en binaire) :

Registre	bit 3	bit 2	bit 1	bit 0
0	Identificateur (1010 si la RAM est initialisée)			
1	Décalage horizontale du SET ADJUST (-8 à 7)			
2	Décalage verticale du SET ADJUST (-8 à 7)			
3	-	-	-	SCREEN 0 / 1
4	4 bits de poids faible de la valeur de WIDTH			
5	-	3 bits de poids fort de la valeur de WIDTH		
6	Couleur du texte par défaut (0 ~ 15)			
7	Couleur de fond par défaut (0 ~ 15)			
8	Couleur de bordure par défaut (0 ~ 15)			
9	Vitesse de transfert 0 = 1200 baud 1 = 2400 baud	0 = Imprimante MSX 1 = Imprimante Non MSX	Cliquetis des touches du clavier	KEY OFF / ON
10 (Ah)	Type de son BEEP		Volume du BEEP	
11 (Bh)	-	-	Couleur de l'écran de démarrage	
12 (Ch)	Code de la zone : 0 = Japon ; 1 = USA ; 2 = International ; 3 = Grande Bretagne ; 4 = France ; 5 = Allemagne ; 6 = Italie ; 7 = Espagne ; 8 = Emirats arabes ; 9 = Corée ; 10 = URSS ; 11 ~ 15 = Indéfinis (au 05/02/1986)			

Note : Le code de la zone n'est pas toujours le bon par défaut.

Bloc 3 (11 en binaire) :

Registre	bit 3	bit 2	bit 1	bit 0
0	identificateur (0 ~ 15) : 0 = Titre ; 1 = Mot de passe ; 2 = Invite (Prompt) ; 3 à 15 = <i>Indéfinis</i> au 05/02/1986			
1	4 bits de point faible du premier caractère du mot de passe, titre ou l'invite			
2	4 bits de point fort du premier caractère du mot de passe, titre ou l'invite			
3	4 bits de point faible du second caractère du mot de passe, titre ou l'invite			
4	4 bits de point fort du second caractère du mot de passe, titre ou l'invite			
5	4 bits de point faible du troisième caractère du mot de passe, titre ou l'invite			
6	4 bits de point fort du troisième caractère du mot de passe, titre ou l'invite			
7	4 bits de point faible du quatrième caractère du mot de passe, titre ou l'invite			
8	4 bits de point fort du quatrième caractère du mot de passe, titre ou l'invite			
9	4 bits de point faible du cinquième caractère du mot de passe, titre ou l'invite			
10 (Ah)	4 bits de point fort du cinquième caractère du mot de passe, titre ou l'invite			
11 (Bh)	4 bits de point faible du sixième caractère du mot de passe, titre ou l'invite			
12 (Ch)	4 bits de point fort du sixième caractère du mot de passe, titre ou l'invite			

Exemple d'utilisation :

```

EXTROM    equ    0015fh
REDCLK    equ    001f5h

          org    0c000h
DEBUT:
          ld     c, 02ch                ; mode 2 - adresse 12
          ld     ix, REDCLK
          call   EXTROM
          ret                          ; lors du retour A contient 4
  
```

## 001F9h (Sub-ROM)      **WRTCLK** WRiTe CLoCK

Rôle :            Écriture dans un registre de l'horloge interne (RTC). L'état des registres sont conservés grâce à une pile ou une batterie selon le MSX.

Entrée :        C = Numéro du bloc et du registre  
                   A = Donnée à écrire. (quatre bits de poids faible)

Sortie :        Rien.

Modifie :      F.

Note :          Voir REDCLK (ci-dessus) pour le détail du fonctionnement de cette routine.

### 3.4 Table des sauts du Bios de la Disk-ROM

La mémoire Morte du contrôleur de disques (« Disk-ROM ») a aussi une table des sauts. Cette ROM est présente dans tous les MSX ayant un lecteur de disquette interne ou dans n'importe quelle interface de disque. La Disk-ROM contient toutes les instructions d'accès aux disques du Disk Basic, le noyau (Kernel) du MSX-DOS 1 ou 2 et le pilote du contrôleur de disque.

Comme il peut y avoir jusqu'à quatre Disk-ROM installées, le numéro du Slot qui contient la Disk-ROM peut s'obtenir de la façon.

Le numéro du Slot qui contient la Disk-ROM maitre est indiqué par la variable système MASTERS (0F348h). La Disk-ROM la plus récente est choisie comme maitre. Les variables DRVINF (0FB22h) indiqueront le Slot de la Sub-ROM pour chaque disques installés. Il ne peut y avoir que huit disques d'installés au total.

#### 04010h (Disk-ROM)      **DSKIO**      DiSK I/O

Rôle :	Lire ou écrire de 1 à 255 secteur sur un disque. (Incompatible FAT16.)
Entrée :	A = Numéro de disque physique de l'interface correspondante à la Disk-ROM. F = Mettre l'indicateur Carry à 0 pour lire ou 1 pour écrire. B = Nombre de secteur à lire / écrire. C = Identifiant du type de Média. DE = Numéro du premier secteur. HL = Adresse de destination en RAM.
Sortie :	A= Code d'erreur (voir la note ci-dessous) si l'indicateur Carry = 1 B = Nombre de secteur à lu / écrit.
Modifie :	Tout.
Note :	Voici les codes d'erreur possible. 0 = Le disque est protégé contre l'écriture. (Write protected) 2 = Le disque n'est pas inséré ou pas prêt. (Disk offline) 4 = Erreur détectée à la vérification. (Data/CRC error) 6 = Erreurs de recherche d'une piste sur le disque. (Seek error) 8 = Le secteur d'amorçage ne contient pas d'entête. Le disque nécessite un formatage. (Header not found) 0Ah = Il y a eu une anomalie pendant l'écriture. (Write fault / Write current error) 0Ch = Autres erreurs (Other error). 12h = Le disque n'est pas formaté pour le DOS. (Not a DOS disk) (MSX-DOS2) 14h = Le disque n'est pas compatible. (Incompatible disk) (MSX-DOS2) 16h = Le disque n'est pas formaté. (Unformatted disk) (MSX-DOS2)



### Exemple en assembleur :

```
DAC      equ    0f7f6h
DRVINV   equ    0fb21h
DSKIO     equ    04010h
ENASLT    equ    00024h
EXPTBL    equ    0fcc1h
VALTYP    equ    0f663h

; --> Entête du fichier
        db      0feh          ; Code pour fichier binaire
        dw      START        ; Adresse de destination du programme
        dw      END          ; Adresse de fin du programme
        dw      START        ; Adresse d'execution du programme

; <--
        org     0c000h

START:   ld      a,2           ; Lecteur C:
        ld      hl,DRVINV     ; Table des interfaces de disque
RPT:     sub     (hl)
        jr      c,CONT
        inc     hl
        inc     hl
        jr      RPT
CONT:    add     a,(hl)
        push    af
        inc     hl
        ld      a,(hl)
        ld      h,040h
        call    ENASLT        ; Selectionne la Disk-ROM du disque C:
        ei
        pop     af
READ:    ld      b,7
        ld      c,0fbh        ;
        ld      de,5
        ld      hl,0c100h
        or      a              ; Met carry à 0 (pour lecture)
        call    DSKIO
        ld      hl,255
        jr      nc,RD_OK
        ld      l,a
RD_OK:   ld      (DAC+2),hl    ; Code d'erreur -> variable du Basic
        ld      a,2
        ld      (VALTYP),a
        ld      a,(EXPTBL)
        ld      h,040h
        call    ENASLT        ; Sélectionne la Main-Rom
        ei
        ret                  ; Retour au Basic

END:
```

Une fois assemblé et sauvegardé sous le nom « RDSECT5.BIN », exécuter la routine avec le programme Basic suivant.

```

10 CLEAR200,&HBFFF: DEFUSR=&HC000
20 BLOAD"RDSECT5.BIN"
30 A%=USR(0): IF A%<255 THEN 90
40 FOR I=&HC100 TO I+7*512-1 STEP 32
50 FOR J=0 TO 10: C=PEEK(I+J)
60 IF C=0 THEN END ELSE IF C=&HE5 THEN 80
70 PRINT CHR$(C): NEXT J: PRINT
80 NEXT I: END
90 BEEP: ON A%/2 GOTO 100,110,120,130,140
100 PRINT"DISK OFF LINE!": END
110 PRINT"DISK READ ERROR!": END
120 PRINT"DISK SEEK ERROR!": END
130 PRINT"SECTOR NOT FOUND!": END
140 PRINT"UNDEFINED ERROR!": END

```

## 04013h (Disk-ROM)      **DSKCHG** DiSK CHanGe

- Rôle :            Vérifie si le disque a été changé.
- Entrée :          A = Numéro de disque physique de l'interface correspondante à la Disk-ROM.  
                     B = 0.  
                     C = Identifiant du type de Média.  
                     HL = Adresse du FCB.
- Sortie :          A = Code d'erreur (voir la routine DSKIO ci-dessus) si l'indicateur Carry = 1  
                     B = 1 (le disque est le même), 255 (disque différent) ou 0 (autre).
- Modifie :        AF, BC, DE, HL, IX et IY.
- Notes :          - Si le disque a été changé (B=255) ou probablement modifié (B=0), lire le secteur d'amorçage (boot sector) ou le premier octet du secteur de la FAT et transférer un nouveau DPB avec GETDPB comme décrit ci-dessous.  
                     - Le DOS2 ne tient pas compte des changements du FCB.

## 04016h (Disk-ROM)     **GETDPB**    GET DPB

Rôle :            Obtenir le bloc de paramètres d'un disque (Drive Parameter Block).

Entrée :          A = Numéro de disque physique de l'interface correspondante à la Disk-ROM.  
                    B = Premier secteur de la FAT.  
                    C = Identifiant du type de Média.  
                    HL = Adresse du DPB à obtenir.

Sortie :          A = Code d'erreur (voir la routine DSKIO plus haut) si l'indicateur Carry = 1  
                    HL = Adresse du DPB obtenu. (Longueur = 37 octets)

Modifie :        AF, BC, DE, HL, IX et IY.

Description du DPB obtenu :

HL	Long.	Nom	Description
+0	1	MEDIA	Type de Média (0F8h~0FFh)
+1	2	SECSIZ	Taille des secteurs (doit être 2^n)
+3	1	DIRMSK	(SECSIZE/32)-1
+4	1	DIRSHFT	Nombre de bits dans DIRMSK
+5	1	CLUSMSK	(Secteurs par cluster)-1
+6	1	CLUSSHFT	(Nombre de bits dans CLUSMSK)+1
+7	2	FIRFAT	Numéro du secteur logique de la première FAT
+8	1	FATCNT	Nombre de FAT
+10	1	MAXENT	Nombre de fichier ou dossier possible (254 maxi.)
+11	2	FIRREC	Numéro du premier secteur des données.
+13	2	MAXCLUS	(Nombre de clusters (sans inclure les secteurs réservés de la FAT et dossier)) + 1
+15	1	FATSIZ	Nombre de secteur par FAT
+16	2	FIRDIR	Numéro de secteur logique du début du dossier de la FAT

## 04019h (Disk-ROM)     **CHOICE**    CHOICE

Rôle :            Indique l'emplacement du texte des types de format possible qui sont affichés pour formater un disque.

Entrée :          Rien.

Sortie :          HL = Adresse de la zone comprenant le texte des choix à afficher pour la routine DSKFMT. Cette zone se termine par zéro. S'il n'y a pas de choix possible (seulement un format) alors HL = 0.

Modifie :        AF, BC, DE, HL, IX et IY.

Note :            Le texte est au format ASCII.

## 0401Ch (Disk-ROM)     **DSKFMT**   DiSK ForMaT

- Rôle :            Formater un disque.
- Entrée :          A = Choix spécifié par l'utilisateur (1~9).  
                    D = Numéro de lecteur (0 = Premier lecteur)  
                    HL = Adresse de la zone de travail.  
                    BC = Longueur de la zone de travail.
- Sortie :          A= Code d'erreur (voir la note ci-dessous) si l'indicateur Carry = 1
- Modifie :        Tous les registres sauf SP.
- Notes :          - Ecrit également un secteur d'amorçage sur le secteur 0 pour MSX, efface toutes les FAT (descripteur de média au premier octet, 0FFh au deuxième / troisième octet et reste à zéro) et efface les répertoires (en le remplissant de zéros)  
                    - Beaucoup d'interfaces utilisent un logiciel externe pour formater leurs disques.  
                    - Codes d'erreurs possibles :  
                            0 = Le disque est protégé contre l'écriture. (Write protected)  
                            2 = Le disque n'est pas inséré ou pas prêt. (Disk offline/Not ready)  
                            4 = Erreur détectée à la vérification. (Data/CRC error)  
                            6 = Erreurs de recherche d'une piste sur le disque. (Seek error)  
                            8 = Secteur d'amorçage ne contient pas d'entête. Le disque nécessite un formatage. (Header/Record not found)  
                            0Ah = Il y a une anomalie pendant l'écriture. (Write fault / Write current error)  
                            0Ch = Un mauvais paramètre a été spécifié (Bad parameter)  
                            0Eh = La mémoire sur le disque est insuffisante (Insufficient memory)  
                            10h = Autres erreurs (Other error)

## 0401Fh (Disk-ROM)

- Rôle :            Stopper le moteur des disques de l'interface correspondante.
- Entrée :          Rien.
- Sortie :          Rien.
- Modifie :        AF, BC, DE, HL, IX et IY.
- Note :            Cette routine n'existe que si l'interface gère les disques amovibles. Dans le cas contraire, 0401Fh contiendra l'octet 00h (ou parfois C9h).

## 04029h (Disk-ROM)     **MTOFF**     MoTor OFF

Rôle :           Stopper le moteur de tous les disques du système.

Entrée :          Rien.

Sortie :          Rien.

Modifie :        AF, BC, DE, HL, IX et IY.

Note :           Cette routine n'existe que si l'interface gère les disques amovibles. Dans le cas contraire, 04029h contiendra l'octet zéro (00h).

Exemple :        Routine en assembleur.

```
RDSLT            equ    000ch
CALSLT           equ    001ch
MTOFF            equ    4029h
MASTERS          equ    0f348h

DiskOFF:
               ld      a, (MASTERS)
               ld      hl, MTOFF
               call    RDSLT
               and     a                    ; Routine MTOFF présente?
               ret     z                    ; Retour si pas de MTOFF
               ld      iy, (MASTERS)
               ld      ix, MTOFF
               jp      CALSLT
```

## 0402Dh (Disk-ROM)     **GTSLT1**     GET SLot bank 1

Rôle :           Donne le numéro de Slot sélectionné sur la plage 04000h~07FFFh.

Entrée :          Rien.

Sortie :          A = Numéro de Slot.

Modifie :        Tout.

Note:

### 3.5 Le Bios du MSX-Music (FM Bios)

Le système n'a pas de variable pour le MSX-Music. Vous devez donc trouver le Slot vous-même pour appeler une routine du Bios d'un MSX-Music. Le standard conseille d'utiliser le Bios plutôt que les ports d'E/S bien que dans certains cas avancés, vous serez certainement obligé de passer par les ports d'E/S.

Le Bios du MSX-Music est appelé « FM Bios ». Il est séparé en deux zones, une qui contient des données pour créer les sons et une pour la table des sauts comme tous les autres Bios.

4100h~4BFFh	FM Bios
4C00h~4DFFh	Données pour les sons des instruments

#### Liste des routines du Bios du MSX-Music

##### 04110h      **WRTOPL**    WRiTe to OPLL

- Rôle :            Écrit un paramètre dans un registre de l'OPLL.
- Entrée :          A = Numéro de registre (0 ~ 7, 14 ~ 24, 32 ~ 40 et 48 ~ 56).  
                     E = Valeur à écrire.
- Sortie :           Rien.
- Modifie :        Rien.
- Note :            La routine ne coupe pas les interruptions. Veillez à que les interruptions soient coupées pour appeler cette routine.

##### 04113h      **INIOPL**      INItializes the OPLL

- Rôle :            Prépare l'environnement du FM Bios et initialise tous les registres de l'OPLL.
- Entrée :          HL = Pointeur (chiffre pair) vers la zone de travail à utiliser pour le FM Bios. Cette zone doit être de la RAM. Sinon plusieurs routines ne fonctionneront pas.
- Sortie :           Rien.
- Modifie :        AF, BC, DE, HL, IX et IY.
- Notes :          - Il faut appeler cette routine avant l'utilisation de toutes autres routines du MSX-Music. L'adresse du pointeur est stocké dans le tableau de variables système SLTWRK (0FD09H~).  
                     - La routine coupe et rétablit les interruptions.

04116h	<b>MSTART</b> Music START
Rôle :	Joue la musique.
Entrée :	HL = Pointeur vers les données de la musique à jouer. A = Paramètre. 0 pour jouer la musique en boucle ; 1 ~ 254 pour le nombre de fois à jouer la musique ; 255 ne doit pas être utilisé.
Sortie :	Rien.
Modifie :	AF, BC, DE, HL, IX et IY.
Notes :	- Avant d'appeler cette routine, il faut paramétrer la zone de travail du MSX-Music en fonction de l'entête des données de la musique pointé par HL. Assurez-vous aussi que OPLDRV soit bien appelé par le Hook H.TIMI (0FD9FH). - La routine coupe et rétablie les interruptions.
04119h	<b>MSTOP</b> Music STOP
Rôle :	Stoppe la musique jouée par l'OPLL et initialise la zone de travail du MSX-Music.
Entrée :	Rien.
Sortie :	Rien.
Modifie :	AF, BC, DE, HL, IX et IY.
Note :	La routine coupe et rétablie les interruptions.
0411Ch	<b>RDDATA</b> Read DATA
Rôle :	Lit les données sonores d'un instrument en ROM.
Entrée :	HL = Pointeur vers la zone de travail qui recevra les données lues. A = Numéro du son d'un instrument. (0~63)
Sortie :	Rien.
Modifie :	F.
Note :	Les données sonores de instrument sont copiées dans la zone de travail.
0411Fh	<b>OPLDRV</b> OPLL DRiVer
Rôle :	Routine de l'interpréteur du pilote de OPLL.
Entrée :	Rien.
Sortie :	Rien.
Modifie :	F.
Note :	Cette routine permet de jouer les musiques. Elle doit être appelée à chaque interruption via le Hook H.TIMI (0FD9Fh). Il faut doc modifier ce Hook en appelant une fois la routine INIOPL (04113h) avant tout appel aux autres routines du FM Bios.

04122h

**TSTBGM** TeST BGM

- Rôle : Vérifie si la musique est finie ou pas.
- Entrée : Rien.
- Sortie : A = 0 si la musique est finie, autre valeur dans le cas contraire.
- Modifie : AF.
- Notes :  
 - Vérifie si la musique jouée par MSTART est finie ou pas.  
 - La routine ne coupe pas les interruptions.

**Format des données pour les musiques**

Il y a un format pour chaque mode.

Contenu	Longueur	
0Eh, 00h	2	Mode 6 voix + boîte à rythme. Le pointeur des données de la musique doit pointer sur 0Eh.
Adresse	2	Pointeur pour la voix FM 1.
Adresse	2	Pointeur pour la voix FM 2.
Adresse	2	Pointeur pour la voix FM 3.
Adresse	2	Pointeur pour la voix FM 4.
Adresse	2	Pointeur pour la voix FM 5.
Adresse	2	Pointeur pour la voix FM 6.
Données	~	Contient les données qui composent la musique à jouer. (voir plus bas pour plus de détail.)

Contenu	Longueur	
12h, 00h	2	Mode 9 voix. Le pointeur des données de la musique doit pointer sur 0Eh.
Adresse	2	Pointeur pour la voix FM 2.
Adresse	2	Pointeur pour la voix FM 3.
Adresse	2	Pointeur pour la voix FM 4.
Adresse	2	Pointeur pour la voix FM 5.
Adresse	2	Pointeur pour la voix FM 6.
Adresse	2	Pointeur pour la voix FM 7.
Adresse	2	Pointeur pour la voix FM 7.
Adresse	2	Pointeur pour la voix FM 7.
Données	~	Contient les données qui composent la musique à jouer. Les données pour la voix FM 1 se trouve au début de cette zone. (voir plus bas pour plus de détail.)



## Format des données pour la mélodie

### Adresse

00h~5Fh	Ces valeurs indiquent l'intervalle.
60h~6Fh	Ces valeurs indiquent le volume sonore.
70h~7Fh	Ces valeurs indiquent l'instrument à utiliser.
80h, 81h	80h désactive le sustain et 81h l'active.
82h	Cette valeur indique l'instrument à utiliser dans la ROM. Cet octet est toujours suivi par le numéro de l'instrument à utiliser. (0~63)
83h	Cette valeur indique l'instrument défini par l'utilisateur. Les deux octets suivants contiennent l'adresse qui pointe sur les données de l'instrument.
84h	Désactive le legato.
85h	Active le legato. (Lie les notes entre elles.)
86h	Cette valeur indique Q. L'octet qui suit est une valeur entre 0 et 8. Ce paramètre est ignoré lorsque le legato est activé.
87h~0FEh	Inutilisés
0FFh	Code de fin des données de cette voix.

## Format des données pour les sons des instruments

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Pointeur	AM	VIB	EGTYP	KSR	MULTIPLE				(opérateur 0)
Pointeur+1	AM	VIB	EGTYP	KSR	MULTIPLE				(opérateur 1)
Pointeur+2	KSL (op. 0)		Total Level Modulator						(opérateur 0) (opérateur 1)
Pointeur+3	KSL (op. 1)		-	DC	DM	Feedback			
Pointeur+4	AR				DR				
Pointeur+5	AR				DR				
Pointeur+6	SL				RR				
Pointeur+7	SL				RR				

**MULTIPLE** = Contrôle de l'échantillon de l'onde - Relation harmonique.

**KSR** (Key Scale Rate) = Taux de .

**EGTYP** = 0 pour une tonalité de percussion, 1 pour une tonalité soutenue.

**VIB** (Vibrato) = Active / Désactive le vibrato.

**AM** (Amplitude Modulation) = Active / Désactive la modulation d'amplitude.

**Total Level Modulator** = Niveau total de l'onde modulée. Contrôle de l'index de modulation.

**KSL** (Key Scale Level) = Niveau de « Key scale ».

**FR** (Feedback Rate) = Constante de retour FM.

**DM** (Distortion Module) = Distorsion de l'onde modulée de l'enveloppe. (Rectification par ondes planes.)

**DC** (Distortion Carrier) = Distorsion de l'onde porteuse de l'enveloppe. (Rectification par ondes planes.)

**DR** (Decay Rate) = Contrôle de la fréquence de changement d'enveloppe de décomposition.

**AR** (Attack Rate) = Contrôle du changement du taux d'enveloppe d'attaque.

**RR** (Release Rate) = Contrôle du niveau de changement d'enveloppe de sortie.

**SL** (Sustain Level) = Indication de décroissance - niveau de maintient.

L'opérateur 0 est un opérateur modulateur et l'opérateur 1 est un opérateur de type porteur.

### 3.6 Le Bios étendu

La routine EXTBIO a été ajoutée au standard MSX afin de connaître combien et quelles extensions sont connectées au système. Elle permet aussi de les gérer plus aisément en langage machine ou en C.

L'adresse de cette routine se trouve à 0FFCAh dans le Slot des variables système. La plupart des appels à EXTBIO provoquera un appel au Bios de l'extension correspondante.

Cette routine étant optionnelle, il est nécessaire de vérifier sa présence avant de l'appeler en lisant la variable HOKVLD (0FB20h). Si le bit 0 de cette variable est à zéro, c'est qu'il n'y a pas d'extension compatible.

#### 0FFCAh                      **EXTBIO**

Entrée :                  D = Identifiant de l'extension.

E = Numéro de la fonction de la routine à appeler.

Note :                    De manière générale, la fonction 0 a pour but de créer une table pour connaître combien et quelles extensions sont connectées au système.

Il est possible d'ajouter ou créer une routine pour une extension en procédant de la façon suivante.

1. Lire le bit 0 de l'octet à l'adresse 0FB20h (HOKVLD). Si il est à 0, mettez-le à 1 puis, écrivez 0C9h (instruction RET) sur les 29 octets à partir de 0FFCAh et passez à l'étape 3.
2. Copier les 5 octets à partir de 0FFCAh vers une zone de votre choix.
3. Ensuite écrire les instructions de saut vers la routine de votre Bios étendu à 0FFCAh. Votre routine devra se terminer par un saut vers la zone copiée auparavant (à l'étape 2) ou bien un RET si l'étape 2 a été sautée.
4. Créez les routines DISINT et ENAINT si elles ne sont pas présentes.

#### Liste des périphériques avec leur fonctions :

#### 000h (0)                      **Broad-cast**

Fonction 0

Rôle :                    Création d'une table contenant les numéros de périphériques ayant un Bios étendus.

Entrée :                  B = Slot dans lequel créer la table.

HL = Adresse de la table à créer.

Sortie :                   B = Slot dans lequel se trouve le premier octet suivant la table.

HL = Adresse du premier octet suivant la table.

Modifie :                Tout.

Note :                    La table résultante contient les numéros des périphériques disponibles chacun espacé d'un octet réservé (00h). Broad-cast (0) et System (255) n'apparaissent pas dans la table.

Voici un exemple pour obtenir la table des périphériques :

```

RAMAD0      equ      0f341h      ; Slot de la Main-Ram plage 0~3FFFh
EXTBIO      equ      0ffc0h      ; Adresse d'appel à un Bios étendu

```

Getdev:

```

      ld      hl,Table
      call    Getslt      ; B = numéro le slot de la table
      ld      d,0         ; Broad-cast device
      ld      e,0         ; Fonction 'get device number'
      jp      EXTBIO

```

```

; Routine getslt
; Entrée : HL = Adresse en Ram.
; Sortie : B = Numéro de Slot correspondant à l'adresse HL.
; Modifie : A et B.

```

Getslt:

```

      push    hl
      ld      a,h
      rla
      rla
      rla          ; Bit 6 et 7 vers bit 1 et 0
      and     3          ; mise à zéro des bits inutilisées
      ld      c,a
      ld      b,0
      ld      hl,RAMAD0
      add     hl,bc
      ld      b,(hl)     ; B = Numéro du Slot de la Main-RAM
      pop     hl
      ret

```

Table:

```

      ds      32          ; Réserve 32 octets pour la table

```

## Fonction 1

Rôle : Obtenir le numéro de trap à placer sur les instructions ON ... GOSUB du Basic afin d'insérer une routine personnelle en langage machine qui sera exécutée lors d'un saut effectué par un ON ... GOSUB.

Entrée : A = 0

Sortie : A = Numéro de trap

- 0 ~ 9 pour ON KEY GOSUB.
- 10 pour ON STOP GOSUB.
- 11 pour ON SPRITE GOSUB.
- 12 ~ 16 pour ON STRIG GOSUB.
- 17 pour ON INTERVAL GOSUB.
- 18 ~ 23 pour les périphériques étendus.
- 24 ~ 25 Réservés pour le système.

Modifie : F et DE.

## Fonction 2

Rôle : Désactiver les interruptions des périphériques.

Entrée : Rien.

Sortie : Rien.

Note : Cette routine permet d'éviter les erreurs de transfert des périphériques qui envoient

un signal d'interruption. Cela arrive, par exemple, avec la RS-232C lorsqu'on coupe les interruptions du CPU trop longtemps.

### Fonction 3

Rôle : Activer les interruptions des périphériques.  
 Entrée : Rien.  
 Sortie : Rien.

## 004h (4) **Memory Mapper**

### Fonction 0

Rôle : Obtenir la table d'information sur le Memory Mapper primaire.  
 Entrée : B = Slot dans lequel créer la table.  
 HL = Adresse de la table à créer.  
 Sortie : B = Slot dans lequel se trouve le premier octet suivant la table.  
 HL = Adresse du premier octet suivant la table.  
 Modifie : Tout.  
 Note : La table d'information sur le Memory Mapper primaire a le format suivant.

Adresse entrée	
HL	Numéro du Slot du Memory Mapper primaire
HL+01h	8 bits de poids faible de l'adresse de la table des sauts
HL+02h	8 bits de poids fort de l'adresse de la table des sauts
HL+03h	Nombre de pages libres
HL+04h	Nombre de pages
HL+05h	Octet réservé
HL+06h	Octet réservé
HL+07h	Octet réservé
Adresse dans HL en sortie :	

### Fonction 1

Rôle : Obtenir la table des variables des Memory Mapper.  
 Entrée : A = 0  
 Sortie : A = Numéro de Slot du Memory Mapper principal  
 HL = Adresse de la table de variables  
 Modifie : Tout.  
 Note : Exemple de table de variables avec deux Memory Mapper connectés.

Adresse en sortie	
HL	Numéro du Slot du Memory Mapper
HL+01h	Nombre de pages
HL+02h	Nombre de pages libres
HL+03h	Nombre de pages réservées par le système
HL+04h	Nombre de pages réservées par l'utilisateur

HL+05h	Octet réservé
HL+06h	Octet réservé
HL+07h	Octet réservé
.	Numéro du Slot du second Memory Mapper
.	Nombre de pages
.	Nombre de pages libres
	Nombre de pages réservées par le système
	Nombre de pages réservées par l'utilisateur
	Octet réservé
	Octet réservé
	Octet réservé
	00h

Sur MSX turbo R, le Memory Mapper primaire sera toujours le Memory Mapper interne.

## Fonction 2

Rôle : Obtenir la table des sauts vers les routines du Memory Mapper.

Entrée : A = 0

Sortie : A = Nombre de page du Memory Mapper primaire  
 B = Numéro de Slot du Memory Mapper primaire  
 C = Nombre de pages libres du Memory Mapper primaire  
 HL = Adresse de la table des sauts

Modifie : Tout sauf B.

Table des sauts :

HL + 00h **ALL\_SEG**

Rôle : Allouer une page.

Entrée : A = 0 pour l'utilisateur / 1 pour le système

B = Slot du mapper sous la forme FxxxSSPP.

Si xxx = 000, alloue une page au Slot indiqué.

Si xxx = 001, alloue une page dans un autre Slot que celui indiqué.

Si xxx = 010, alloue une page au Slot indiqué ou un autre si le mapper est déjà pris.

Si xxx = 011, alloue une page dans un autre Slot que celui indiqué. En cas d'échec, essaie dans un autre Slot.

B = 0 si Memory Mapper principal.

Sortie : F = Carry à 1 si aucune page libre.

Si Carry à 0, alors A = Numéro de page, B = Numéro de Slot du Memory Mapper (B = 0 si Memory Mapper principal.)

HL + 03h **FRE\_SEG**

Rôle : Libérer une page.

Entrée : A = Numéro de page.

B = Slot du Mapper sous la forme FxxxSSPP (B = 0 pour le Memory Mapper principal.)

Sortie : F = Carry à 1 si aucune n'a été libérée.

HL + 06h	<b>RD_SEG</b>
	Rôle : Lire un octet dans une page. Entrée : A = Numéro de page. HL = Adresse à lire dans la page. Sortie : A = Octet lu. Modifie : AF. Note : Cette routine désactive les interruptions.
HL + 09h	<b>WR_SEG</b>
	Rôle : Ecrire un octet dans une page. Entrée : A = Numéro de page. HL = Adresse. E = Octet à écrire. Modifie : AF. Note : Cette routine désactive les interruptions.
HL + 0Ch	<b>CAL_SEG</b>
	Rôle : Appel inter-page. Entrée : IYh = Numéro de page. IX = Adresse. Modifie : Tout sauf AF, BC, DE et HL. Note : AF, BC, DE et HL peuvent servir de paramètres pour la routine à appeler.
HL + 0Fh	<b>CALLS</b>
	Rôle : Appel inter-page. Entrée : Les trois octets suivants l'instruction call CALLS. call CALLS db PAGE dw ADRESSE Modifie : Tout sauf AF, BC, DE et HL. Note : AF, BC, DE et HL peuvent servir de paramètres pour la routine à appeler.
HL + 12h	<b>PUT_PH</b>
	Rôle : Sélectionner une page sur la plage mémoire correspondante à l'adresse indiquée. Entrée : HL = Adresse (ce sont en fait les bits 7 et 6 qui indiquent la plage 0~3FFFh, 4000h~7FFFh, 8000h~BFFFh ou C000h~FFFFh). A = Numéro de page. Modifie : Rien.
HL + 15h	<b>GET_PH</b>
	Rôle : Obtenir le numéro de page sélectionnée sur la plage mémoire correspondante à l'adresse indiquée. Entrée : HL = Adresse (ce sont en fait les bits 7 et 6 qui indiquent la plage 0~3FFFh, 4000h~7FFFh, 8000h~BFFFh ou C000h~FFFFh). A = Numéro de page. Modifie : Rien.
HL + 18h	<b>PUT_P0</b>
	Rôle : Sélectionner une page sur la plage 0000h~3FFFh. Entrée : A = Numéro de page.
HL + 1bh	<b>GET_P0</b>
	Rôle : Obtenir le numéro de page de la plage 0000h~3FFFh. Sortie : A = Numéro de page.
HL + 1eh	<b>PUT_P1</b>

Rôle : Sélectionner une page sur la plage 4000h~7FFFh.  
 Entrée : A = Numéro de page.

HL + 21h **GET\_P1**

Rôle : Obtenir le numéro de page de la plage 4000h~7FFFh.  
 Sortie : A = Numéro de page.

HL + 24h **PUT\_P2**

Rôle : Sélectionner une page sur la plage 8000h~BFFFh.  
 Entrée : A = Numéro de page.

HL + 27h **GET\_P2**

Rôle : Obtenir le numéro de page de la plage 8000h~BFFFh.  
 Sortie : A = Numéro de page.

HL + 2ah **PUT\_P3**

Rôle : Sélectionner une page sur la plage C000h~FFFFh.  
 Entrée : A = Numéro de page.

HL + 2dh **GET\_P3**

Rôle : Obtenir le numéro de page de la plage C000h~FFFFh.  
 Sortie : A = Numéro de page.

## 008h (8) **MSX-Modem and RS-232C**

### Fonction 0

Rôle : Obtenir une table d'informations sur les MSX-Modem et RS-232C installés.

Entrée : B = Slot dans lequel créer la table.

HL = Adresse de la table à créer.

Sortie : B = Slot dans lequel se trouve le premier octet suivant la table.

HL = Adresse du premier octet suivant la table.

Modifie : Tout.

Note : Voici un exemple de table avec deux interfaces installées.

Adresse indiquée par HL en entrée :

Numéro du Slot de l'interface
Octet de poids faible de l'adresse des sauts
Octet de poids fort de l'adresse des sauts
Octet réservé (00h par défaut)
Numéro du Slot de la seconde interface
Octet de poids faible de l'adresse des sauts
Octet de poids fort de l'adresse des sauts
Octet réservé (00h par défaut)

Adresse indiquée par HL en sortie :

## 00Ah (10) **MSX-Audio**

### Fonction 0

Rôle : Obtenir une table d'informations sur les MSX-Audio installés.



Entrée : B = Slot dans lequel créer la table.  
HL = Adresse de la table à créer.

Sortie : B = Slot dans lequel se trouve le premier octet suivant la table.  
HL = Adresse du premier octet suivant la table.

Modifie : F.

Note : Voici un exemple de table avec deux MSX-Audio installées.

Adresse indiquée par HL en entrée :	
	Numéro du Slot du MSX-Audio
	Octet de poids faible de l'adresse des sauts
	Octet de poids fort de l'adresse des sauts
	Numéro du Slot du second MSX-Audio
	Octet de poids faible de l'adresse des sauts
	Octet de poids fort de l'adresse des sauts
Adresse indiquée par HL en sortie :	Octet réservé (00h par défaut)

## Fonction 1

Rôle : Obtenir la table des sauts vers les routines du MSX-Audio.

Entrée : A = 0.  
B = Slot dans lequel créer la table.  
HL = Adresse de la table à créer.

Sortie : A = Nombre de MSX-Audio installés  
HL = Adresse de la table des sauts.

Modifie : Tout sauf B.

Table des sauts :

### HL+00h **VERSION**

Rôle : Ces 3 octets servent à indiquer la version de la ROM.

### HL+03h **MBIOS**

Rôle : Appel aux routines du MBIOS.

Entrée : HL = Adresse de la routine du MBIOS à appeler.

Pour les routines qui utilise IX et IY comme paramètre, veuillez placer ces paramètres dans BUF (0F55Eh) comme suit.

BUF = 8 bits de poids faible de IX.

BUF+1 = 8 bits de poids fort de IX.

BUF+2 = 8 bits de poids faible de IY.

BUF+3 = 8 bits de poids fort de IY.

Sortie : Voir les routines MBIOS.

### HL+06h **AUDIO**

Rôle : Initialiser le MSX-Audio. Les routines du MSX-Audio ne seront utilisable qu'après une initialisation.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Nombre de chaîne MML par voix FM pour PLAY. (0~9)

BUF+1 = Mode switch.

bit 0 pour activer la boîte à rythme

bit 1 pour activer l'ADPCM du PLAY

bit 2 pour activer le mode CMS du MSX-Audio

BUF+2 = Nombre de voix FM pour les instruments. (0~9)

BUF+3 = Nombre de voix FM de la première chaîne MML. (0~9)

BUF+4 = Nombre de voix FM de la seconde chaîne MML. (0~8)

BUF+5 = Nombre de voix FM de la troisième chaîne MML. (0~7)

BUF+6 = Nombre de voix FM de la quatrième chaîne MML. (0~6)

BUF+7 = Nombre de voix FM de la cinquième chaîne MML. (0~5)

BUF+8 = Nombre de voix FM de la sixième chaîne MML. (0~4)

BUF+9 = Nombre de voix FM de la septième chaîne MML. (0~3)

BUF+10 = Nombre de voix FM de la huitième chaîne MML. (0~2)

BUF+11 = Nombre de voix FM de la neuvième chaîne MML. (0~1)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

Note : Lorsque la boîte à rythme est activé le nombre de voix FM passe de 9 à 6 maximum.

#### HL+09h **SYNTHE**

Rôle : Transfère le contrôle au programme d'application fourni. Cette routine n'aura aucun effet lorsque la routine AUDIO est appelé.

Entrée : Rien.

Sortie : Rien.

Note : Cette routine ne fonctionnera plus correctement après l'exécution de l'instruction CLEAR du Basic. (Il y a risque de plantage)

#### HL+0Ch **PLAYF**

Rôle : Examine de l'état de la lecture des MML.

Entrée : A = Numéro de voix. (0 pour toutes les voix)

Sortie : HL = FFFFh si lecture en cours sinon HL = 0000h.

#### HL+0Fh **BGM**

Rôle : Activer / désactiver le mode BGM (lecture en tâche de fond).

Entrée : A = Activer / désactiver le mode BGM. (1 / 0)

Sortie : Rien.

Note : Voici les fonctions qui peuvent fonctionner en mode BGM

- Jouer une chaîne MML.

- Lecture d'un enregistrement ADPCM du mode local.

- Lecture d'un enregistrement du clavier musical à l'adresse indiquée.

#### HL+12h **MKTEMPO**

Rôle : Réglage du tempo pour jouer ou enregistrer un air du clavier musical.

Entrée : DE = Nombre de noires par minute. (25 ~ 360)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

#### HL+15h **PLAYMK**

Rôle : Jouer un air enregistré au clavier musical.

Entrée : BC = Adresse de début des données enregistrées au clavier musical.

DE = Adresse de fin des données enregistrées au clavier musical.

Sortie : Rien.

#### HL+18h **RECMK**

Rôle : Enregistrer un air au clavier musical.

Entrée : BC = Adresse de début des données à enregistrer au clavier musical.

DE = Adresse de fin des données à enregistrer au clavier musical.

Sortie : Rien.

#### HL+1bh **STOPM**

Rôle : Stopper la lecture d'un enregistrement au clavier musical, de d'un enregistrement ADPCM / PCM, ou d'une chaîne MML jouée.

Entrée : Rien.

Sortie : Rien.

Note : Il est possible de reprendre la lecture d'un enregistrement au clavier musical en appelant CONTMK.

**HL+1eh CONTMK**

Rôle : Continuer la lecture d'un enregistrement au clavier musical, ADPCM / PCM, ou d'une chaîne MML jouée.

Entrée : Rien.

Sortie : Rien.

**HL+21h RECMOD**

Rôle : Régler le mode du clavier musical.

Entrée : A = Mode.

0 pour muet (N'enregistre pas).

1 pour enregistrement.

2 pour jouer l'air.

3 pour jouer tout en enregistrant.

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

**HL+24h STPPLY**

Rôle : Stopper la lecture d'une chaîne MML.

Entrée : Rien.

Sortie : Rien.

**HL+27h SETPCM**

Rôle : Initialiser le fichier audio ADPCM/PCM.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Numéro de l'enregistrement. (0 ~ 15)

BUF+1 = Numéro de matériel à employer. (0 ~ 3 ou 5)

BUF+2 = Mode. (0 ou 1)

BUF+3 = Si le numéro de matériel est 1 ou 3 alors mettre ici un numéro de son en ROM. Si le numéro de matériel est 5, indiquer ici les 8 bits de poids faible de l'adresse en VRAM.

BUF+4 = Si le numéro de matériel est 1 ou 3 alors mettre 0. Si c'est 5, indiquer les 8 bits de poids fort de l'adresse en VRAM.

BUF+5 = 8 bits de poids faible de la longueur de l'enregistrement.

BUF+6 = 8 bits de poids fort de la longueur de l'enregistrement.

BUF+7 = 8 bits de poids faible de la fréquence d'échantillonnage.

BUF+8 = 8 bits de poids fort de la fréquence d'échantillonnage.

BUF+9 = Canal. (0 ou 1)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

Note : Le matériel 4 (le CPU) n'est pas utilisable.

**HL+2ah RECPCM**

Rôle : Enregistrer un son dans un fichier.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Numéro de l'enregistrement. (0~15)

BUF+1 = Synchro. (0 ou 1)

BUF+2 = Les 8 bits de poids faible du décalage.

BUF+3 = Les 8 bits de poids fort du décalage.

BUF+4 = 8 bits de poids faible de la longueur de l'enregistrement.

BUF+5 = 8 bits de poids fort de la longueur de l'enregistrement.

BUF+6 = 8 bits de poids faible de la fréquence d'échantillonnage.

BUF+7 = 8 bits de poids fort de la fréquence d'échantillonnage.

BUF+8 = Canal. (0 ou 1)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

Note : La fréquence d'échantillonnage peut être définie avec SETPCM. Si c'est le cas, mettre 0FFh à BUF+6 et BUF+7.

## HL+2dh **PLAYPCM**

Rôle : Lire un enregistrement.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Numéro de l'enregistrement. (0 ~ 15)

BUF+1 = Lecture en boucle. (0 ou 1)

BUF+2 = 8 bits de faible du décalage.

BUF+3 = 8 bits de poids fort du décalage.

BUF+4 = 8 bits de poids faible de la longueur de l'enregistrement.

BUF+5 = 8 bits de poids fort de la longueur de l'enregistrement.

BUF+6 = 8 bits de poids faible de la fréquence d'échantillonnage.

BUF+7 = 8 bits de poids fort de la fréquence d'échantillonnage.

BUF+8 = Canal. (0 ou 1)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

Note : La fréquence d'échantillonnage peut être définie avec SETPCM. Si c'est le cas, mettre 0FFh à BUF+6 et BUF+7.

## HL+30h **PCMFREQ**

Rôle : Changer la fréquence de lecture.

Entrée : BC = Fréquence du premier canal en Hz. (1800~49716)

DE = Fréquence du second canal en Hz. (1800~49716)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

## HL+33h **MKPCM**

Rôle : Paramétrer le numéro de fichier ADPCM joué par le clavier musical.

Entrée : A = Numéro de l'enregistrement. (0~15 ou 0FFh pour aucun)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

Note : Seul les fichiers en mode local sont jouable.

## HL + 36h **PCMVOL**

Rôle : Réglage du volume pour jouer un son ADPCM / PCM.

Entrée : BC = Volume du premier canal. (0~63)

DE = Volume du second canal. (0~63)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

Note : Lorsqu'il n'y a pas de deuxième canal, le registre DE doit contenir la même valeur que celle de BC. Par défaut, le volume ADPCM est 63, celui PCM est 32.

## HL+39h **SAVEPCM**

Rôle : Sauvegarder un enregistrement ADPCM / PCM sur disque.

Entrée : A = Numéro de l'enregistrement.

HL = Adresse pointant sur le nom de fichier entre guillemets.

Sortie : F = Carry passe à 1 en cas de mauvais de numéro d'enregistrement.

Notes :

- En assembleur, on définit le nom de fichier de la façon suivante.

FILENAME: DB 22H, "A: TRACK. PCM", 22H, 0

- En cas d'erreur disque (nom de fichier erroné, disque non inséré, etc), la gestion sera transférée à la routine de traitement d'erreur de l'interpréteur BASIC. Vous pouvez l'intercepter grâce au Hook H.ERRO (0FFB1h).

- Le fichier créé sera constituer de la façon suivante.

En premier, il y a l'entête qui comprends 7 octets comme pour les fichiers binaires sauvegardés par l'instruction BSAVE du Basic suivi de 8 octets comprenant des informations sur les données ADPCM / PCM enregistrées. Le reste sera les données du son numérisé.

Entête :

Octet du  
fichier

Entête au format d'un fichier binaire

0	Indicateur de fichier binaire. (0FEh)
---	---------------------------------------

1~2	0000h
3~4	Longueur des données du fichier -1. (Bloc d'informations compris)
5~6	0000h pour ADPCM, 0001h pour PCM.

Bloc d'informations

7~8	Longueur des données de l'enregistrement. (Unité = 256 octets)
9~10	Fréquence d'échantillonnage en Herz.
11~12	8000h pour ADPCM (valeur par défaut), 0000h pour PCM.
13~14	007Fh pour ADPCM (valeur par défaut), 0000h pour PCM.

Bloc contenant les données du son numérisé

15~	Données du son numérisé
-----	-------------------------

### HL+3Ch **LOADPCM**

Rôle : Chargement d'un enregistrement ADPCM / PCM sur disque effectué avec SAVEPCM.

Entrée : A = Numéro de l'enregistrement.

HL = Adresse pointant sur le nom de fichier entre guillemets.

Sortie : Rien.

Notes :

- En assembleur, on définit le nom de fichier de la façon suivante.  
FILENAME: DB 22H, "A:TRACK.PCM", 22H, 0
- En cas d'erreur disque (nom de fichier erroné, disque non inséré, etc), la gestion sera transférée à la routine de traitement d'erreur de l'interpréteur BASIC. Vous pouvez l'intercepter grâce au Hook H.ERRO (0FFB1h).

### HL+3Fh **COPYPCM**

Rôle : Copie les données d'un enregistrement ADPCM / PCM.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Numéro de l'enregistrement source. (0~15)

BUF+1 = Numéro de l'enregistrement de destination. (0~15)

BUF+2 = 8 bits de faible de l'adresse de l'enregistrement source.

BUF+3 = 8 bits de fort de l'adresse de l'enregistrement source.

BUF+4 = 8 bits de poids faible de la longueur de l'enregistrement.

BUF+5 = 8 bits de poids fort de la longueur de l'enregistrement.

BUF+6 = 8 bits de faible de l'adresse de l'enregistrement de destination.

BUF+7 = 8 bits de faible de l'adresse de l'enregistrement de destination.

BUF+8 = Canal source. (0 ou 1)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

### HL+42h **CONVP**

Rôle : Convertir un enregistrement PCM au format ADPCM.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Numéro de l'enregistrement source. (0~15)

BUF+1 = Numéro de l'enregistrement de destination. (0~15)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

### HL+45h **CONVA**

Rôle : Convertir un enregistrement ADPCM au format PCM.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Numéro de l'enregistrement source. (0~15)

BUF+1 = Numéro de l'enregistrement de destination. (0~15)

Sortie : F = Carry passe à 1 en cas d'erreur de paramètre.

### HL+48h **VOICE**

Rôle : Définir la tonalité pour chaque voix sonore FM.

Entrée : Définir les paramètres suivants dans BUF (0F55Eh).

BUF = Bloc de paramètres de la Voix 1.

BUF+4 = Bloc de paramètres de la Voix 2.

BUF+8 = Bloc de paramètres de la Voix 3.

BUF+12 = Bloc de paramètres de la Voix 4.

BUF+16 = Bloc de paramètres de la Voix 5.

BUF+20 = Bloc de paramètres de la Voix 6.

BUF+24 = Bloc de paramètres de la Voix 7.

BUF+28 = Bloc de paramètres de la Voix 8.

BUF+32 = Bloc de paramètres de la Voix 9.

BUF+36 = Fin (0FFh)

Chaque bloc est constitué de 4 octets de la façon suivante.

+0 = Numéro de voix sonore FM. (0~8)

+1 = Mettre à 00h.

+2 = Numéro du son de la librairie. (0~63)

+3 = Mettre à 00h.

Certains blocs peuvent être constitués de la façon suivante.

+0 = Numéro de voix sonore FM. (0~8)

+1 = Mettre à FFh.

+2 = 8 bits de poids faible de l'adresse d'un enregistrement.

+3 = 8 bits de poids fort de l'adresse d'un enregistrement.

Sortie : Rien.

#### HL+4Bh **VOICECOPY**

Rôle : Copier les données d'une voix FM.

Entrée : Définir les paramètres dans BUF (0F55Eh) selon une des cinq possibilités suivantes.

1. Copie un son de la librairie vers un autre son de la librairie.

BUF+0 = 0

BUF+1 = Numéro de son de la librairie source. (0~63)

BUF+2 = 0

BUF+3 = 0

BUF+4 = 0

BUF+5 = 0

BUF+6 = Numéro du son de la librairie de destination. (32~63)

BUF+7 = 0

BUF+8 = 0

BUF+9 = 0

2. Copie un son de la librairie vers un son de l'utilisateur.

BUF+0 = 0

BUF+1 = Numéro de son de la librairie source. (0~63)

BUF+2 = 0

BUF+3 = 0

BUF+4 = 0

BUF+5 = 0FFh

BUF+6 = 8 bits de poids faible de l'adresse de destination.

BUF+7 = 8 bits de poids fort de l'adresse de destination.

BUF+8 = 0

BUF+9 = 0

3. Copie un enregistrement vers un son de la librairie.

BUF+0 = 0FFh

BUF+1 = 8 bits de poids fort de l'adresse de l'enregistrement source.

BUF+2 = 8 bits de poids fort de l'adresse de l'enregistrement source.

BUF+3 = 0

BUF+4 = 0

BUF+5 = 0

BUF+6 = Numéro du son de la librairie de destination. (32~63)

BUF+7 = 0  
 BUF+8 = 0  
 BUF+9 = 0  
 4. Copie les sons de 32 à 63 de la librairie vers un enregistrement.  
 BUF = 0  
 BUF+1 = 0FFh  
 BUF+2 = 0  
 BUF+3 = 0  
 BUF+4 = 0  
 BUF+5 = 0FFh  
 BUF+6 = 8 bits de poids faible de l'adresse de destination.  
 BUF+7 = 8 bits de poids fort de l'adresse de destination.  
 BUF+8 = 8 bits de poids faible de la longueur de l'enregistrement.  
 BUF+9 = 8 bits de poids fort de la longueur de l'enregistrement.  
 5. Copie un enregistrement vers les sons de 32 à 63 de la librairie.  
 BUF = 0FFh  
 BUF+1 = 8 bits de poids faible de l'adresse de l'enregistrement source.  
 BUF+2 = 8 bits de poids fort de l'adresse de l'enregistrement source.  
 BUF+3 = 8 bits de poids faible de la longueur de l'enregistrement.  
 BUF+4 = 8 bits de poids fort de la longueur de l'enregistrement.  
 BUF+5 = 0  
 BUF+6 = 0FFh  
 BUF+7 = 0  
 BUF+8 = 0  
 BUF+9 = 0

Sortie : Rien.

## 010h (16) **MSX-JE**

### Fonction 0

Rôle : Définir la zone de travail d'entrée de caractères japonais. (Disponible sur certains MSX japonais ou, cartouches de Kanji et traitements de texte japonais avec MSX-JE.)  
 Entrée : B = Numéro du Slot de la zone de travail des caractères entrés par RETURN.  
 HL = Pointeur sur la zone de travail de 64 octets.  
 Sortie : B = Numéro du Slot suivant la zone de travail.  
 HL = Adresse suivant la zone de travail.  
 Modifie : F.  
 Note : La pile doit être sur la plage 0C000h~0FFFFh. Par conséquent, le point d'accès recherche l'adresse d'entrée comme suit.

Exemple de programme pour obtenir l'adresse d'entrée du BIOS étendue du MSX-JE:

```

HOKVLD    equ    0fb20h                ;Extended BIOS flag
EXTBIO    equ    0ffcah                ;Extended BIOS hook
;
          ld      a, (HOKVLD)
          rra
  
```

```

jr    nc,NO_MJE
ld    hl,MJETBL
call  GETSLT          ;Get slot address of MJETBL into [B]
ld    de,16*256+0
push  hl
call  EXTBIO
pop   de
or    a
sbc   hl,de
jr    z,NO_MJE
.
.
.

```

En sortie de EXTBIO, une table est crée comme suit à l'adresse MJETBL et HL pointe l'octet derrière la table. Lorsque la fonction MSX-JE n'existe pas, HL ne change pas. La table qui suit est un exemple avec deux MSX-JE.

Valeur de MJETBL :	Vecteur de capacité	} Descripteur du premier MSX-JE
	Numéro du Slot	
	Octet de poids faible de l'adresse de la table des sauts.	
	Octet de poids fort de l'adresse d'entrée	
	Vecteur de capacité	} Descripteur du second MSX-JE
	Numéro du Slot	
	Octet de poids faible de l'adresse d'entrée	
	Octet de poids fort de l'adresse d'entrée	
Adresse indiquée par HL en sortie :	.	
	.	
	.	

#### Fonction 1 (Inquiry)

Rôle : Renvoie la taille de la zone de travail.

Entrée : A = 1

Sortie : BC = Limite supérieure de la zone de travail utilisée par MSX-JE (MAX).

DE = Limite inférieure de la zone de travail utilisée par MSX-JE (MIN).

HL = Taille minimale requise pour utiliser la fonction d'apprentissage (MIN2).

Modifie : .

Notes : - Une application peut déterminer la taille de la zone de travail requise pour MSX-JE avec n de la façon suivante.

Par rapport aux valeurs renvoyées	Taille de la zone de travail
$n < \text{MIN}$	(L'application ne peut pas utiliser MSX-JE)
$\text{MIN} \leq n < \text{MIN2}$	MIN (l'application ne peut pas utiliser la fonction d'apprentissage)



MIN  $\leq$  n < MAX

n

MAX  $\leq$  n

MAX

---

Il est possible qu'une application ne puisse pas utiliser la fonction d'apprentissage. De plus, si vous pouvez toujours sécuriser 2560 octets de travail, vous n'avez pas besoin d'appeler cette fonction.

- Avec cette fonction, l'adresse de la zone de travail n'est pas transmise au MSX-JE en tant que paramètre. Par conséquent, MSX-JE ne peut pas utiliser la zone de travail.

#### Fonction 2 (Invoke)

Rôle : Démarrer MSX-JE. (Initialise une zone de travail sécurisée pour une application.)

Entrée : A = 2

DE = Taille de la zone de travail

HL = Adresse de la zone de travail

Sortie : Rien.

Modifie : .

Notes :

#### Fonction 3 (Release)

Rôle : Quitter MSX-JE. (Libérer une zone de travail sécurisée par une application.)

Entrée : A = 3

HL = Adresse de la zone de travail

Sortie : Rien.

Modifie : .

Notes :

#### Fonction 4 (Clear)

Rôle : Efface la cache pour la conversion Kana-Kanji. (Efface le texte en cours de saisi.)

Entrée : A = 4

HL = Adresse de la zone de travail.

Sortie : Rien.

Modifie : .

Notes :

#### Fonction 5 (Set TTB)

Rôle : .

Entrée : A = 5

Sortie : BC = Adresse TTB.

DE = Adresse des données du texte à reconvertir.

HL = Adresse de la zone de travail.

Modifie : .

Notes :

## 011h (17) **Kanji driver**

Fonction 0

Rôle : ?

Entrée : ?

Sortie : ?

Modifie : ?

Note : ?

## 0FFh (255) **System**

Fonction 0

Rôle : Obtenir une table d'informations sur les Bios étendus ajoutées au système.

Entrée : A = 0.

B = Slot dans lequel créer la table.

HL = Adresse de la table à créer.

Sortie : B = Slot dans lequel se trouve le premier octet suivant la table.

HL = Adresse de l'octet derrière la table.

Modifie : Tout.

Note : Format de la table d'information sur les Bios étendus.

Adresse indiquée par HL en entrée :

Numéro du Slot du périphérique
Octet de poids faible de l'adresse des sauts
Octet de poids fort de l'adresse des sauts
Identifiant du fabricant (Voir liste ci-dessous)
Octet réservé
Numéro du Slot du périphérique
Octet de poids faible de l'adresse des sauts
Octet de poids fort de l'adresse des sauts
Identifiant du fabricant (Voir liste ci-dessous)
Octet réservé

Table du périphérique suivant :

Adresse indiquée par HL en sortie :

Liste des identifiants de fabricant :

0 = ASCII	9 = Mitsubishi	18 = Spectravideo
1 = Microsoft	10 = Nippon Denki	19 = Toshiba
2 = Canon	11 = Yamaha	20 = Mitsumi
3 = Casio	12 = Victor	21 = Telematika
4 = Fujitsu	13 = Philips	22 = Gradiente
5 = General	14 = Pioneer	23 = Sharp Epcom
6 = Hitachi	15 = Sanyo	24 = Goldstar
7 = Kyocera	16 = Sharp	25 = Daewoo
8 = National/Panasonic	17 = Sony	26 = Samsung

#### Fonction 1

Rôle : Nombre de lecteurs installés.

Entrée : Rien.

Sortie : B = Numéro du Slot suivant.  
HL = Pointeur sur la zone de travail suivante.

Modifie : ?.

#### Fonction 2

Rôle : Désactivation des interruptions.

Entrée : Rien.

Sortie : Rien.

Modifie : Rien.

#### Fonction 3

Rôle : Activation des interruptions.

Entrée : Rien.

Sortie : Rien.

Modifie : Rien.

## 4 Les interruptions

Le Z80 possède deux broches qui sont des entrées destinées à recevoir un signal extérieur qui provoque immédiatement l'appel à une routine dédiée. Ces deux broches se nomment  $\overline{\text{NMI}}$  et  $\overline{\text{INT}}$ . La broche  $\overline{\text{INT}}$  provoque un appel à l'adresse 0038h. Cet appel est désactivable et ré-activable avec les instructions DI et EI du Z80. Cette broche est généralement reliée au processeur vidéo qui envoie par défaut un signal cinquante ou soixante fois par seconde à chaque fin d'affichage de l'écran (de l'avant-plan). La broche  $\overline{\text{NMI}}$  provoque un appel à l'adresse 0066h. Cet appel n'est pas désactivable mais, malgré que la routine est présente dans le Bios et le DOS, cette broche n'est pas utilisée sur MSX, elle est reliée au +5V. Vous pouvez l'ignorer.

Le Z80 possède trois modes d'interruptions.

### 4.1 Mode 0

Dans ce mode, lorsque qu'un signal d'interruption est envoyée au Z80, celui-ci termine l'instruction en cours puis exécute l'instruction dont le premier octet est envoyé sur le bus de données par le périphérique qui envoie la requête d'interruption. Ce mode ne peut être utilisé qu'en ajoutant un dispositif matériel adéquate.

### 4.2 Mode 1

Dans ce mode, lorsque qu'un signal d'interruption est envoyée au Z80, celui-ci termine l'instruction en cours puis provoque un appel à l'adresse 0038h lors d'une interruption  $\overline{\text{INT}}$  ou 0066h d'une interruption  $\overline{\text{NMI}}$ . Les MSX fonctionnent dans ce mode. C'est le Bios qui active ce mode au démarrage.

### 4.3 Mode 2

Ce mode est appelé le mode d'interruption vectorisé. Lorsque qu'un signal d'interruption est envoyée au Z80, celui-ci termine l'instruction en cours puis provoque un appel à l'adresse formée à l'aide du registre I (octet de poids fort) et de l'octet placé sur le bus de données (octet de poids faible) par le périphérique ayant provoqué l'interruption. Ce mode ne peut être utilisé qu'en ajoutant un dispositif matériel adéquate.

## 5 Les variables et zones de travail du système

### 5.1 Introduction aux variables système

L'utilité des variables système n'est plus à démontrer. La plupart des trucs et astuces sont basés sur une bonne connaissance de celles-ci. Il devient ainsi facile de combler les quelques lacunes du Basic. Quant au programmeur en langage machine, il peut parfois ignorer les variables système et utiliser cette zone pour constituer une zone de travail appréciable pour un très gros programme mais il ne faut pas négliger que les routines du Bios utilisent cette zone et ce, particulièrement si le programmeur souhaite développer des routines en langage machine qui coexistent avec l'interpréteur Basic. Dans ce cas, il devra être particulièrement attentif à cette zone mémoire et ne pas écraser n'importe quoi.

### 5.2 La liste des variables et des zones de travail système

Adresse	Nom	Long.	Fonction
0F380h	RDPRIM	5	Sous-routine de RDSLT (Main-ROM) qui sert à lire dans un Slot primaire.
0F385h	WRPRIM	7	Sous-routine de WRSLT (Main-ROM) qui sert à écriture dans un Slot primaire.
0F38Ch	CLPRIM	14	Sous-routine de CALSLT (Main-ROM) qui sert à faire un appel dans un Slot primaire.
0F39Ah	USRTAB	20	Adresses des routines définies par l'instruction DEFUSR X=. USRTAB = Adresse de la routine définie par DEFUSR0= USRTAB+2 = Adresse de la routine définies par DEFUSR1= USRTAB+4 = Adresse de la routine définies par DEFUSR2= USRTAB+6 = Adresse de la routine définies par DEFUSR3= USRTAB+8 = Adresse de la routine définies par DEFUSR4= USRTAB+10 = Adresse de la routine définies par DEFUSR5= USRTAB+12 = Adresse de la routine définies par DEFUSR6= USRTAB+14 = Adresse de la routine définies par DEFUSR7= USRTAB+16 = Adresse de la routine définies par DEFUSR8= USRTAB+18 = Adresse de la routine définies par DEFUSR9= Note :Contient l'adresse de la routine d'erreur du Basic par défaut.
0F3AEh	LINL40	1	Longueur d'une ligne définie par l'instruction WIDTH en mode SCREEN 0. (37 par défaut.)
0F3AFh	LINL32	1	Longueur d'une ligne définie par l'instruction WIDTH en mode SCREEN 1. (29 par défaut.)
0F3B0h	LINLEN	1	Longueur de la ligne actuelle.
0F3B1h	CRTCNT	1	Nombre de ligne de la page actuelle. (24 par défaut.)
0F3B2h	CLMLST	1	Valeur utilisée par PRINT. (LINLEN-(LINLEN MOD 14)-14)
0F3B3h	TXTNAM	2	Adresse de la table des caractères en mode SCREEN 0.

0F3B5h	TXTCOL	2	Adresse de la table des couleurs en SCREEN 0. (MSX2~)
0F3B7h	TXTCGP	2	Adresse de la table des formes en mode SCREEN 0.
0F3B9h	TXATTR	2	Inutilisée.
0F3BBh	TXTPAT	2	Inutilisée.
0F3BDh	T32NAM	2	Adresse de la table des caractères en mode SCREEN 1.
0F3BFh	T32COL	2	Adresse de la table des couleurs en mode SCREEN 1.
0F3C1h	T32CGP	2	Adresse de la table des formes en mode SCREEN 1.
0F3C3h	T32ATR	2	Adresse de la table des attributs de Sprites en mode SCREEN 1.
0F3C5h	T32PAT	2	Adresse de la table des formes de Sprites en mode SCREEN 1.
0F3C7h	GRPNAM	2	Adresse de la table des motifs en mode SCREEN 2.
0F3C9h	GRPCOL	2	Adresse de la table des couleurs en mode SCREEN 2.
0F3CBh	GRPCGP	2	Adresse de la table des formes en mode SCREEN 2.
0F3CDh	GRPATR	2	Adresse de la table des attributs de Sprites en mode SCREEN 2.
0F3CFh	GRPPAT	2	Adresse de la table des formes de Sprites en mode SCREEN 2.
0F3D1h	MLTNAM	2	Adresse de la table des caractères en mode SCREEN 3.
0F3D3h	MLTCOL	2	Adresse de la table des couleurs en mode SCREEN 3.
0F3D5h	MLTCGP	2	Adresse de la table des formes en mode SCREEN 3.
0F3D7h	MLATTR	2	Adresse de la table des attributs de Sprites en mode SCREEN 3.
0F3D9h	MLTPAT	2	Adresse de la table des formes de Sprites en mode SCREEN 3.
0F3DBh	CLIKSW	1	Cliquetis des touches. 0 pour désactiver ; autre pour activer.
0F3DCh	CSRY	1	Ordonnée du curseur.
0F3DDh	CSRX	1	Abscisse du curseur.
0F3DEh	CNSDFG	1	Indicateur pour savoir si les touches de fonction sont affichées (255) ou pas (0).
0F3DFh	RG0SAV	1	Contenu du registre 0 du VDP.
0F3E0h	RG1SAV	1	Contenu du registre 1 du VDP.
0F3E1h	RG2SAV	1	Contenu du registre 2 du VDP.
0F3E2h	RG3SAV	1	Contenu du registre 3 du VDP.
0F3E3h	RG4SAV	1	Contenu du registre 4 du VDP.
0F3E4h	RG5SAV	1	Contenu du registre 5 du VDP.
0F3E5h	RG6SAV	1	Contenu du registre 6 du VDP.
0F3E6h	RG7SAV	1	Contenu du registre 7 du VDP.
0F3E7h	STATFL	1	Contenu du registre de statut 0 du VDP.

0F3E8h	TRGFLG	1	État des boutons des manettes (0 = Bouton pressé) bit 0 = Barre d'espace ; bit 1~3 = <i>Inutilisés</i> ; bit 4 = Bouton 1 de la manette 1 ; bit 5 = Bouton 2 de la manette 1 ; bit 6 = Bouton 1 de la manette 2 ; bit 7 = Bouton 2 de la manette 2.
0F3E9h	FORCLR	1	Couleur du texte ou du tracé par défaut.
0F3EAh	BAKCLR	1	Couleur de fond par défaut. (Couleur donné à l'avant-plan lors d'un effacement d'écran sauf en SCREEN 0. Dans ce mode, BAKCLR correspond à la couleur de l'arrière-plan)
0F3EBh	BDRCLR	1	Couleur de la bordure par défaut. (Correspond à la couleur de l'arrière-plan sauf en SCREEN 0.)
0F3ECh	MAXUPD	3	Zone de travail de l'instruction CIRCLE. Contient JP 0000h, saut en 0000h, par défaut.
0F3EFh	MINUPD	3	Zone de travail de l'instruction CIRCLE. Contient JP 0000h, saut en 0000h, par défaut.
0F3F2h	ATRBYT	1	Octet d'attribut graphique. (Couleur de tracé des routines graphiques.)
0F3F3h	QUEUES	2	Adresse de la table des queues QUETAB (0F959h par défaut) de l'instruction PLAY.
0F3F5h	FRCNEW	1	Taille de la mémoire de travail de l'instruction PLAY. (255 par défaut)
0F3F6h	SCNCNT	1	Intervalle entre deux scrutations du clavier. Réglé en permanence par le Basic.
0F3F7h	REPCNT	1	Intervalle avant de commencer la fonction de répétition automatique. Réglé en permanence par le Basic. (1 par défaut)
0F3F8h	PUTPNT	2	Adresse du premier emplacement libre dans le cache du clavier. (0FBF0h par défaut)
0F3FAh	GETPNT	2	Adresse de la prochaine donnée issue du clavier (dans le cache du clavier).
0F3FCh	CS120	5	Zone de travail du lecteur de cassette. (Jusqu'au MSX2+) CS120 = Longueur du signal LOW du bit 0. (53h par défaut) CS120+1 = Longueur du signal HIGH du bit 0. (5Ch par défaut) CS120+2 = Longueur du signal LOW du bit 1. (26h par défaut) CS120+3 = Longueur du signal HIGH du bit 1. (2Dh par défaut) CS120+4 = Longueur du signal d'entête calculé par HEADLEN*2/256 (HEADLEN = 2000 par défaut)

0F401h	CS240	5	Zone de travail du lecteur de cassette. (Jusqu'au MSX2+) CS240 = Longueur du signal LOW du bit 0. (25h par défaut) CS240+1 = Longueur du signal HIGH du bit 0. (2Dh par défaut) CS240+2 = Longueur du signal LOW du bit 1. (0Eh par défaut) CS240+3 = Longueur du signal HIGH du bit 1. (16h par défaut) CS240+3 = Longueur du signal d'entête calculé par HEADLEN*4/256 (HEADLEN = 2000 par défaut)
0F406h	LOW	2	Paramètres pour le lecteur de cassette. (Jusqu'au MSX2+) LOW = Longueur du signal LOW qui représente le bit 0 à la vitesse de transmission actuelle. (53h par défaut) LOW+1 = Longueur du signal HIGH qui représente le bit 0 à la vitesse de transmission actuelle. (5Ch par défaut)
0F408h	HIGH	2	Paramètres pour le lecteur de cassette. (Jusqu'au MSX2+) HIGH = Longueur du signal LOW qui représente le bit 1 à la vitesse de transmission actuelle. (26h par défaut) HIGH+1 = Longueur du signal HIGH qui représente le bit 1 à la vitesse de transmission actuelle. (2Dh par défaut)
0F40Ah	HEADER	1	Paramètres pour le lecteur de cassette. (Jusqu'au MSX2+) Longueur du signal d'entête actuel calculé par HEADLEN*2/256 ou HEADLEN*4/256 (HEADLEN = 2000 par défaut)
0F40Bh	ASPCT1	2	Rapport d'aspect du cercle tracé par l'instruction CIRCLE.
0F40Dh	ASPCT2	2	Rapport d'aspect du cercle tracé par l'instruction CIRCLE.
0F40Fh	ENDPRG	5	Leurre de fin de programme pour l'instruction RESUME NEXT. (« : » suivit de 4 zéro par défaut.)
0F414h	ERRFLG	1	Numéro de la dernière erreur.
0F415h	LPTPOS	1	Position de la tête de l'imprimante. 0 au départ.
0F416h	PRTFLG	1	Indicateur de sortie sur imprimante. 0 pas de sortie.
0F417h	NTMSXP	1	Indicateur de type d'imprimante. 0 = Imprimante MSX (les Hiragana sont convertis en Katakana sur les MSX japonais.) ; Autre valeur = Pas de conversion Hiragana vers Katakana.
0F418h	RAWPRT	1	Mode d'impression. 0 = Conversion des tabulations en espaces ; Autre valeur = Imprimer tel quel.
0F419h	VLZADR	2	Adresse du caractère remplacé par l'instruction VAL du Basic.
0F41Bh	VLZDAT	1	Caractère remplacé par 0 avec l'instruction VAL du Basic.
0F41Ch	CURLIN	2	Numéro de la ligne en cours d'exécution.
0F41Eh	KBFMIN	1	Utilisé lorsqu'une erreur de déclaration directe survient.
0F41Fh	KBUF	318	« Crunch Buffer » utilisé par le Basic.
0F55Dh	BUFMIN	1	Petit cache intermédiaire.
0F55Eh	BUF	258	Cache du mode direct et zone de travail de plusieurs instructions du Basic en court d'exécution. 0F562h (FNPTR) = Paramètres de l'instruction.
0F660h	ENDBUF	1	Indicateur de dépassement du cache BUF.



0F661h	TTYPOS	1	Position finale du curseur stockée par l'instruction PRINT.
0F662h	DIMFLG	1	Indicateur de l'instruction DIM pour savoir si un tableau est en cours.
0F663h	VALTYP	1	Type de variable contenue dans DAC (0F7F6h). 2 = Entier ; 3 = Chaîne de caractères ; 4 = Simple précision ; 8 = Double précision.
0F664h	OPRTYP	1	Le numéro de l'opérateur est stocké momentanément ici lorsqu'une instruction utilise un opérateur.
"	DORES	1	Indicateur de l'interpréteur Basic pour indiquer si il peut comprimer des mots-clés ou pas. (Par exemple : les données de DATA ou PRINT ne doivent pas être comprimées.)
0F665h	DONUM	1	Indicateur utilisé pour la compression des nombres.
0F666h	CONXTT	2	Sauvegarde temporaire du pointeur de texte.
0F668h	CONSAV	1	Sauvegarde temporaire du code de l'instruction en cours.
0F669h	CONTYP	1	Type de constante trouvé par l'interpréteur BASIC. Utilisé aussi par la routine CHRGT (00010h en Main-ROM).
0F66Ah	CONLO	8	Valeur de la constante trouvée par l'interpréteur BASIC. Utilisé aussi par la routine CHRGT (00010h en Main-ROM).
0F672h	MEMSIZ	2	Adresse de la dernière case mémoire disponible sous Basic. Modifié par l'instruction CLEAR.
0F674h	STKTOP	2	Adresse de la fin de la zone réservée pour la pile. Modifié par CLEAR du Basic.
0F676h	TXTTAB	2	Adresse de début du programme Basic chargé. Contient initialement la valeur de BOTTOM+1 (0FC48h+1).
0F678h	TEMPPT	2	Pointeur du Basic.
0F67Ah	TEMPST	30	Stockage provisoire pour NUMTEMP de taille de trois UMTMP.
0F698h	DSCTMP	3	Adresse du premier octet de la table des chaînes de caractères.
0F69Bh	FRETOP	2	Adresse de fin de la table de chaînes de caractères.
0F69Dh	TEMP3	2	Mémoire de travail temporaire.
0F69Fh	TEMP8	2	Mémoire de travail temporaire.
0F6A1h	ENDFOR	2	Sauvegarde de la position derrière l'instruction FOR d'une boucle afin de pouvoir revenir avec NEXT.
0F6A3h	DATLIN	2	Numéro de ligne du DATA en cours. Utilisé par READ du Basic.
0F6A5h	SUBFLG	1	Indicateur pour un tableau de l'instruction DEFUSR du Basic.
0F6A6h	FLGINP	1	Indicateur pour les instructions INPUT et READ du Basic.
0F6A7h	TEMP	2	Mémoire de travail pour les instructions INPUT, FOR...NEXT et LET mais aussi ^C.
0F6A9h	PTRFLG	1	Indicateur de conversion de numéro de ligne en pointeur pour l'interpréteur Basic. 0 = Ne pas convertir ; 1 = Convertir.
0F6AAh	AUTFLG	1	Indicateur pour l'instruction AUTO du Basic. 0 = Manuelle ; 1 = Numérotation des lignes automatique en cours.
0F6ABh	AUTLIN	2	Numérotation de ligne actuelle en mode AUTO.
0F6ADh	AUTINC	2	Valeur à incrémenter en mode AUTO.

0F6AFh	SAVTXT	2	Sauvegarde du pointeur de texte après une erreur pour l'instruction RESUME du Basic.
0F6B1h	SAVSTK	2	Sauvegarde de l'adresse de la pile après une erreur afin d'être dans la bonne situation au moment du RESUME (Basic).
0F6B3h	ERRLIN	2	Numéro de la ligne à laquelle la dernière erreur s'est produite.
0F6B5h	DOT	2	Numéro de ligne actuelle affichée par l'instruction LIST du Basic.
0F6B7h	ERRTXT	2	Pointeur de texte utilisé par l'instruction RESUME du Basic.
0F6B9h	ONELIN	2	Numéro de ligne définie par l'instruction ON ERROR GOTO.
0F6BAh	ONEFLG	1	Indicateur d'erreur pour l'instruction ON ERROR GOTO. 1 si traitement d'erreur en cours, autrement 0.
0F6BCh	TEMP2	2	Mémoire de travail pour le programme d'évaluation de formules.
0F6BEh	OLDLIN	2	Numéro de ligne gardé après l'exécution de l'instruction STOP ou END, ou en pressant CTRL+STOP.
0F6C0h	OLDTXT	2	Ancien pointeur de texte. Le pointeur est dirigé sur l'instruction suivant celle où s'est produit l'arrêt.
0F6C2h	VARTAB	2	Pointeur sur le début de la zone des variables simples.
0F6C4h	ARYTAB	2	Pointeur sur le début de la zone des tableaux de variables.
0F6C6h	STREND	2	Adresse de la fin de la zone des variables.
0F6C8h	DATPTR	2	Pointeur de la donnée à lire de l'instruction DATA. Modifié par l'instruction RESTORE.
0F6CAh	DEFTBL	26	Table de déclaration des variables définies par les instructions DEFSTR, DEFINT, DEFSNG et DEFDBL du Basic pour chacune des 26 lettres de l'alphabet. 2 = Entier ; 3 = Chaîne ; 4 = Simple précision ; 8 = Double précision (valeur par défaut).
0F6E4h	PRMSTK	2	Pointeur vers le bloc défini précédemment. (Afin de nettoyer les déchets.)
0F6E6h	PRMLN	2	Nombre d'octets actuellement utilisés dans PARM1.
0F6E8h	PARM1	100	Définition des paramètres.
0F74Ch	PRMPRV	2	Pointeur vers le bloc de paramètres précédent.
0F74Eh	PRMLN2	2	Taille du bloc de paramètres en cours d'élaboration.
0F750h	PARM2	100	Zone pour sauvegarder les blocs en cours de création.
0F7B4h	PRMFLG	1	Indicateur pour savoir si une recherche dans PARM1 est en cours.
0F7B5h	ARYTA2	2	Point d'arrêt pour une recherche simple.
0F7B7h	NOFUNS	1	Indicateur de fonction. 0 si aucune fonction n'est active.
0F7B8h	TEMP9	2	Pointeur de stockage temporaire. (Afin de nettoyer les déchets.)
0F7BAh	FUNACT	2	Nombre de fonctions actives.
0F7BCh	SWPTMP	8	Mémoire de travail pour l'instruction SWAP du Basic.
0F7C4h	TRCFLG	1	Indicateur de tracé. 0 = Pas de traçage en cours ; Autre valeur = Traçage en cours.
0F7C5h	FBUFFR	43	Cache utilisé par les routines mathématiques.

0F7F0h	DECTMP	2	Mémoire de travail utilisé pour transformer un nombre entier décimal en nombre à virgule flottante.
0F7F2h	DECTM2	2	Mémoire de travail pour les divisions.
0F7F4h	DECCNT	1	Mémoire de travail pour les divisions.
0F7F6h	DAC	16	Accumulateur décimal. Le contenu varie selon que le nombre soit un entier, un simple précision ou un double précision.
0F806h	HOLD8	48	Sauvegarde temporaire pendant les multiplications décimales.
0F836h	HOLD2	8	Zone de travail pour l'exécution d'opérateurs numériques.
0F83Eh	HOLD	8	Idem que ci-dessus.
0F847h	ARG	16	Argument. Le contenu varie selon que le nombre soit un entier, un simple précision ou un double précision.
0F857h	RNDX	8	Dernier nombre aléatoire généré.
0F85Fh	MAXFIL	1	Nombre maximum de fichiers autorisés. Modifié par l'instruction MAXFILES du Disk-Basic.
0F860h	FILTAB	2	Pointeur sur l'adresse des données du fichier.
0F862h	NULBUF	2	Pointeur du cache des instructions SAVE et LOAD du Disk-Basic.
0F864h	PTRFIL	2	Pointeur sur les données du fichier sélectionné.
0F866h	RUNFLG	1	Indicateur d'exécution pour l'instruction LOAD. 0 si le programme a été exécuté par l'option R.
0F866h	FILNAM	11	Nom du fichier d'une instruction du Disk-Basic.
0F871h	FILNM2	11	Nom du second fichier des instructions du Disk-Basic (NAME, COPY, MOVE, etc).
0F87Ch	NLONLY	1	Indicateur différent de 0 lors d'un chargement de programme. Utilisé par les instructions BSAVE et CREATE.
0F87Dh	SAVEND	2	Adresse de fin de l'instruction BSAVE du Disk-Basic.
0F87Fh	FNKSTR	160	Contient les caractères des touches de fonction à afficher. (16 octets par touche.)
0F91Fh	CGPNT	3	Emplacement de la forme des caractères pour initialiser l'écran. CGPNT = Slot ; CGPNT+1 = Adresse.
0F922h	NAMBAS	2	Adresse de la table des caractères ou Bitmap actuelle en VRAM.
0F924h	CGPBAS	2	Adresse de la table des formes actuelle en VRAM.
0F926h	PATBAS	2	Adresse de la table des formes de Sprites actuelle en VRAM.
0F928h	ATRBAS	2	Adresse de la table des attributs de Sprites actuelle en VRAM.
0F92Ah	CLOC	2	Adresse en VRAM du curseur graphique (SCREEN 2 à 4) ou, abscisse du curseur graphique (SCREEN 5 à 8)
0F92Ch	CMASK	1	Masque du curseur graphique (SCREEN 2 à 4) ou, ordonnée du curseur graphique (SCREEN 5 à 12).
0F92Dh	MINDEL	2	Mémoire de travail pour l'instruction LINE du Basic.
0F92Fh	MAXDEL	2	Fin de la mémoire de travail pour l'instruction LINE du Basic.

0F931h	ASPECT	2	Aspect du cercle à tracer. Défini par l'angle de l'instruction CIRCLE du Basic.
0F933h	CENCNT	2	Compteur utilisé par l'instruction CIRCLE du Basic.
0F935h	CLINEF	1	Indicateur pour tracer ou non une ligne vers le centre. Défini par l'angle de l'instruction CIRCLE du Basic.
0F936h	CNPNTS	2	Nombre de points à placer dans un segment de 45°. Utilisé par l'instruction CIRCLE du Basic.
0F938h	CPLOTF	1	Indicateur de polarité. Utilisé par l'instruction CIRCLE du Basic.
0F939h	CPCNT	2	Nombre de points dans 1/8 du cercle. Utilisé par l'instruction CIRCLE du Basic.
0F93Bh	CPCNT8	2	Nombre de points dans le cercle. Utilisé par l'instruction CIRCLE.
0F93Dh	CRCSUM	2	Somme du cercle. Utilisé par l'instruction CIRCLE du Basic.
0F93Fh	CSTCNT	2	Variable pour maintenir le nombre de points de l'angle de départ. Utilisé par l'instruction CIRCLE du Basic.
0F941h	CSCLXY	1	Rapport entre la largeur et la hauteur. (CIRCLE)
0F942h	CSAVEA	2	Adresse du premier pixel de couleur différente. Utilisé par l'instruction PAINT du Basic.
0F944h	CSAVEM	1	Masque du premier pixel de couleur différente. Utilisé par l'instruction PAINT du Basic.
0F945h	CXOFF	2	Décalage horizontal depuis la position sauvegardée. Utilisé par l'instruction PAINT du Basic.
0F947h	CYOFF	2	Décalage vertical depuis la position sauvegardée. Utilisé par l'instruction PAINT du Basic.
0F949h	LOHMSK	1	Position la plus à gauche d'une excursion de LH. Utilisé par l'instruction PAINT du Basic.
0F94Ah	LOHDIR	1	Nouvelle direction de peinture requise par une excursion de LH. Utilisé par l'instruction PAINT du Basic.
0F94Bh	LOHADR	2	Position la plus à gauche d'un LH. Utilisé par l'instruction PAINT du Basic.
0F94Dh	LOHCNT	2	Taille d'une excursion de LH. Utilisé par l'instruction PAINT du Basic.
0F94Fh	SKPCNT	2	Nombre de saut retournés par la routine SCANR du Bios. Utilisé par l'instruction PAINT du Basic.
0F951h	MOVCNT	2	Garde le nombre de déplacement retourné par la routine SCANR du Bios. Utilisé par l'instruction PAINT du Basic.
0F953h	PDIREC	1	Direction dans laquelle on peint. Utilisé par l'instruction PAINT. 40h = Vers le bas, C0h = Vers le haut, 00h = Terminé.
0F954h	LFPROG	1	Mis à 1 lors d'une progression vers la gauche. Utilisé par l'instruction PAINT du Basic.
0F955h	RTPROG	1	Mis à 1 lors d'une progression vers la droite. Utilisé par l'instruction PAINT du Basic.
0F956h	MCLTAB	2	Pointeur vers la chaîne de caractères du Macro langage de l'instruction PLAY ou DRAW du Basic.

0F958h	MCLFLG	1	Indicateur indiquant si la chaîne de caractères du langage macro appartient à PLAY (>0) ou à DRAW (0).
0F959h	QUETAB	24	Table des queues de PLAY et de l'interface RS-232C pour MSX1. QUETAB: ; Queue voix A db 0 ; Position de départ db 0 ; Indicateur de position db 0 ; Indicateur de remplacement db 7Fh ; Taille dw VOICAQ ; Adresse ; Queue voix B db 0 ; Position de départ db 0 ; Indicateur de position db 0 ; Indicateur de remplacement db 7Fh ; Taille db VOICBQ ; Adresse ; Queue voix C db 0 ; Position de départ db 0 ; Indicateur de position db 0 ; Indicateur de remplacement db 7Fh ; Taille dw VOICCQ ; Adresse ; Queue de la RS-232C db 0 ; Position de départ db 0 ; Indicateur de position db 0 ; Indicateur de remplacement db 0 ; Taille dw 0000h ; Adresse Les trois tables de contrôle de la musique sont initialisées par la routine GICINI (00090h) et ensuite géré par la routine d'interruption et la routine PUTQ (000F9h). Le standard stipule que <b>cette zone ne doit plus être utilisée pour l'RS-232C à partir du MSX2.</b>
0F971h	QUEBAK	4	Table de caractères de remplacement des queues. QUEBAK: db 0 ; Caractère de remplacement (voix A) db 0 ; Caractère de remplacement (voix B) db 0 ; Caractère de remplacement (voix C) db 0 ; Caractère de remplacement (RS-232)
0F975h	VOICAQ	128	Queue musicale pour la voix A.
0F9F5h	VOICBQ	128	Queue musicale pour la voix B
0FA75h	VOICCQ	128	Queue musicale pour la voix C
0FAF5h	RS2IQ	64	Queue de la RS-232C. (Retiré du standard à partir du MSX2)
0FAF5h	DPPAGE	1	Page affichée. (MSX2~)
0FAF6h	ACPAGE	1	Page active. (MSX2~)
0FAF7h	AVCSAV	1	Sauvegarde du port (F7h) de contrôle AV. (MSX2+~)
0FAF8h	EXBRSA	1	Slot dans lequel se trouve la Sub-ROM. (MSX2~)
0FAF9h	CHRCNT	1	Compteur de caractère dans le cache de conversion Roma-Kana pour le système. (MSX2~)
0FAFAh	ROMA	2	Adresse de sauvegarde le caractères par le système lors d'une conversion Roma-Kana. (MSX2~)

0FAFCh	MODE	1	Indicateurs du type de masque à appliquer pour tracer à l'écran en Screen 5 ~ 12, de la taille de la VRAM disponible et pour la conversion Romaji vers Katakana/ Hiragana. bit 0 = 1 si la conversion des Romaji vers Kana est possible. bit 1~2 = Taille de la VRAM. 00 pour 16Ko ; 01 pour 64Ko ; 10 pour 128Ko ; 11 pour 192Ko. bit 3 = 1 pour appliquer le masque en SCREEN 0 ~ 3. bit 4 = 0/1 pour limiter la coordonnée Y à 211/255. (MSX2+~) bit 5 = 0/1 pour tracer en mode RGB/YJK en SCREEN 10 ou 11. bit 6 = 1 si la ROM Kanji est de niveau 2. (MSX2+~) bit 7 = 1 conversion vers Katakana ; 0 conversion vers Hiragana.
0FAFDh	NORUSE	1	Utilisé par le pilote de Kanji (Kanji Driver). bit 0~2 = Opérateur logique utilisé pour l'affichage des Kanji. 000 pour PSET ; 001 pour AND ; 010 pour OR ; 011 pour XOR ; 100 pour NOT. bit 3 = 1 pour couleur 0 transparente. bit 4 = Inutilisé. bit 5 = Désactive bit 6 = Permet le défilement avec SHIFT+Haut/Bas (en mode Kanji). bit 7 = 1 pour interdire le retour au mode texte.
0FAFEh	XSAVE	2	Sauvegarde de l'abscisse du curseur graphique lue avec la routine GTPAD ou NEWPAD lors de l'utilisation du crayon optique, de la souris, de la tablette graphique, etc. Le bit 15 indique une demande d'interruption du crayon optique. Les bits 14~8 restent à zéro sauf avec le crayon optique car ces bits servent au décalage de la calibration.
0FB00h	YSAVE	2	Sauvegarde de l'ordonnée du curseur graphique lue avec la routine GTPAD ou NEWPAD lors de l'utilisation du crayon optique, de la souris, de la tablette graphique, etc. Le bit 15 n'est pas utilisé. Les bits 14~8 restent à zéro sauf avec le crayon optique car ces bits servent au décalage de la calibration.
0FB02h	LOGOPR	1	Opérateur logique à effectuer lors d'un tracé ou d'une copie graphique : 0 = IMP ; 1 = AND ; 2 = OR ; 3 = XOR ; 4 = NOT.
0FB03h	TOCNT	1	Compteur utilisée par l'interface RS-232C.
0FB04h	RSFCB	2	Adresse du FCB (« File Control Block ») de la RS-232C.
0FB06h	RSIQLN	1	Donnée temporaire utilisée par l'interface RS-232C.
0FB07h	MEXBIH	5	Hook appelé par l'interface RS-232C. MEXBIH : RST 30h (0F7h) MEXBIH+1 : numéro du Slot MEXBIH+2 : adresse MEXBIH+4 : RET (0C9h)

0FB0Ch	OLDSTT	5	Hook appelé par l'interface RS-232C. OLDSTT : RST 30h (0F7h) OLDSTT+1 : numéro du Slot OLDSTT+2 : adresse OLDSTT+4 : RET (0C9h)
0FB11h	OLDINT	5	Hook appelé par l'interface RS-232C. OLDINT : RST 30h (0F7h) OLDINT+1 : numéro du Slot OLDINT+2 : adresse OLDINT+4 : RET (0C9h)
0FB16h	DEVNUM	1	Utilisée par l'interface RS-232C.
0FB17h	DATCNT	3	RS-232C. DATCNT : numéro du Slot DATCNT+1 : pointeur (adresse)
0FB1Ah	ERRORS	1	Code d'erreur utilisée par l'interface RS-232C.
0FB1Bh	FLAGS	1	Indicateurs utilisés par l'interface RS-232C.
0FB1Ch	ESTBLS	1	Utilisée par l'interface RS-232C.
0FB1Dh	COMMSK	1	Utilisée par l'interface RS-232C.
0FB1Eh	LSTCOM	1	Utilisée par l'interface RS-232C.
0FB1Fh	LSTMOD	1	Utilisée par l'interface RS-232C.
0FB20h	HOKVLD	1	Le bit 0 de cet octet indique la présence d'un Bios étendu. 0 = Pas de Bios ; 1 = Il y a au moins un Bios qui peut être appelé à l'adresse 0FFCAh (EXTBIO).
0FB35h	PRSCNT	1	Mémoire de travail pour l'instruction PLAY du Basic afin de compter les commandes effectuées.
0FB36h	SAVSP	2	L'instruction PLAY du Basic y sauvegarde le pointeur de pile (contenu du registre SP) lorsqu'elle est exécuter.
0FB38h	VOICEN	1	Nombre de voix jouées actuellement par l'instruction PLAY.
0FB39h	SAVVOL	2	Sauvegarde du volume lors d'un pause.
0FB3Bh	MCLLEN	1	Longueur de la macro en cours d'analyse.
0FB3Ch	MCLPTR	2	Adresse de la macro en cours d'analyse.
0FB3Eh	QUEUEN	1	Numéro de la queue actuelle.
0FB3Fh	MUSICF	1	Indicateur d'interruption musicale.
0FB40h	PLYCNT	1	Nombre de chaines macro dans la file d'attente de PLAY.
0FB41h	VCBA	37	Données pour la voix 0.
0FB66h	VCBB	37	Données pour la voix 1.
0FB8Bh	VCBC	37	Données pour la voix 2.
0FBB0h	ENSTOP	1	Indicateur différent de 0 lorsqu'il est possible de reprendre l'exécution d'un programme Basic.

0FBB1h	BASROM	1	Programme Basic en ROM si différent de 0. (L'exécution du programme ne peut être interrompue par CTRL+STOP dans ce cas.)
0FBB2h	LINTTB	24	Table de 24 indicateurs de fin de ligne pour chaque ligne physiques. (Utilisé par l'interpréteur Basic.) 0 = La ligne correspondante contient une ligne de programme qui continue sur la ligne suivante ; Autre valeur = La ligne correspondante contient une ligne de programme qui se termine ici.
0FBCAh	FSTPOS	2	Pointeur vers le premier caractère entré par les routines INLIN (00B1h) et QINLIN (00B4h) en Main-ROM.
0FBCCh	CODSAV	1	Contient le code du caractère placé sous le curseur.
0FBCDh	FNKSWI	1	Indique quelle série de touches de fonction est affichée. 0 = Touches F1 à F6 ; Autre valeur = F6 à F10.
0FBCEh	FNKFLG	10	Dix indicateurs pour l'instruction KEY(x) ON/OFF/STOP du Basic. 0 = KEY(x) OFF/STOP ; Autre valeur= KEY(x) ON.
0FBD8h	ONGSBF	1	Indicateur pour l'instruction ON... GOSUB du Basic.
0FBD9h	CLIKFL	1	Indicateur pour savoir si le click a déjà eu lieu.
0FBDAh	OLDKEY	11	Statut précédent de chaque ligne de la matrice du clavier.
0FBE5h	NEWKEY	11	Nouveau statut de chaque ligne de la matrice du clavier. Le statut est actualisé par la routine d'interruption KEYINT.
0FBF0h	KEYBUF	40	Cache du clavier par défaut. Contient la valeur des dernières touches pressées.
0FC18h	LINWRK	40	Zone de travail pour la gestion de l'écran.
0FC40h	PATWRK	8	Zone de travail pour le convertisseur noms vers formes.
0FC48h	BOTTOM	2	Adresse du début de la zone de RAM disponible.
0FC4Ah	HIMEM	2	Adresse de la fin de la zone de RAM disponible.
0FC4Ch	TRPTBL	78	Tables pour chacune des instructions suivantes : TRPTBL (longueur 3 × 10 octets) = ON KEY GOSUB TRPTBL+30 (longueur 3 × 1 octet) = ON STOP GOSUB TRPTBL+33 (longueur 3 × 1 octet) = ON SPRITE GOSUB TRPTBL+36 (longueur 3 × 5 octets) = ON GOSUB STRIG TRPTBL+51 (longueur 3 × 1 octet) = ON INTERVAL GOSUB TRPTBL+54 (longueur 3 × 8 octet) = Réserve Le premier octet sert d'indicateur. 0 = OFF ; 1 = ON ; 2 = STOP ; 3 = Appel en cours. 7 = Appel en attente. Les 2 autres octets contiennent l'adresse du numéro de ligne de la routine à appeler par le GOSUB dans le programme Basic.
0FC9Ah	RTYCNT	1	Contrôle d'interruption.
0FC9Bh	INTFLG	1	Indicateur d'interruption. 4 si STOP est pressé, 3 si CTRL+STOP, autrement 0.
0FC9Ch	PADX	1	Valeur en X de la molette (« paddle controller »).
0FC9Dh	PADY	1	Valeur en Y de la molette (« paddle controller »).
0FC9Eh	JIFFY	2	Compteur croissant de l'instruction TIME du Basic.



0FCA0h	INTVAL	2	Valeur de l'intervalle de l'instruction ON INTERVAL=... GOSUB du Basic.
0FCA2h	INTCNT	2	Compteur décroissant de l'instruction ON INTERVAL=... GOSUB.
0FCA4h	LOWLIM	1	Utilisé durant la lecture cassette.
0FCA5h	WINWID	1	Idem ci-dessus.
0FCA6h	GRPHED	1	Entête pour la sortie des caractères graphiques.
0FCA7h	ESCCNT	1	Compteur pour une séquence d'escape.
0FCA8h	INSFLG	1	Indicateur de mode insertion.
0FCA9h	CSRSW	1	Indicateur pour afficher ou pas le curseur. 0 = Ne pas afficher.
0FCAAh	CSTYLE	1	Taille du curseur. 0 = Curseur entier ; Autre valeur = Petit curseur (pour le mode insertion).
0FCABh	CAPST	1	Indicateur CAPS d'entrée de caractère. 0 = Caractère en minuscules ; Autre valeur = Majuscules.
0FCACH	KANAST	1	Indicateur de mode kana. (MSX Japonais)
0FCADh	KANAMD	1	Indicateur pour connaître le type de clavier. (MSX Japonais) Contient 0 si l'agencement des touches du clavier est « Kana » sinon, il est « JIS ».
0FCAEh	FLBMEM	1	Indicateur de chargement : 0 si un programme Basic se charge.
0FCAFh	SCRMOD	1	Mode d'écran actuel.
0FCB0h	OLDSCR	1	Ancien mode d'écran de texte. Utilisé pour savoir dans quel mode texte il faut revenir après un mode graphique.
0FCB1h	CASPRV	1	Mémoire de travail pour la cassette. Sur les MSX turbo R, le bit 0 de cet octet indique l'état de la LED « Pause » et le bit 7 indique l'état de la LED «R800».
0FCB2h	BRDATR	1	Couleur de la frontière pour l'instruction PAINT du Basic.
0FCB3h	GXPOS	2	Abscisse du curseur graphique.
0FCB5h	GYPOS	2	Ordonnée du curseur graphique.
0FCB7h	GRPACX	2	Accumulateur graphique X.
0FCB9h	GRPACY	2	Accumulateur graphique Y.
0FCBBh	DRWFLG	1	Indicateur pour l'instruction DRAW du Basic.
0FCBCh	DRWSCL	1	Taille du zoom pour l'instruction DRAW du Basic.
0FCBDh	DRWANG	1	Angle pour l'instruction « DRAW » du Basic
0FCBEh	RUNBNF	1	Indicateur d'Entrée/Sortie : 255 = E/S binaire en cours
0FCC1h	EXPTBL	4	Indicateur de Slot étendu pour chaque Slot primaire. Le bit 7 de ces variables est à 1 lorsque le Slot correspondant est étendu en Slot secondaires. (Voir le chapitre « <a href="#">Utiliser les Slot</a> » page 14 pour plus de précision.)

0FCC5h	SLTTBL	4	<p>Sauvegarde l'état des 4 registres des Slot secondaires de chaque Slot primaire étendu après que le système est fini de scruter les Slot pendant le démarrage.</p> <p>SLTTBL+0 = État des Slot secondaires du Slot primaire 0</p> <p>SLTTBL+1 = État des Slot secondaires du Slot primaire 1</p> <p>SLTTBL+2 = État des Slot secondaires du Slot primaire 2</p> <p>SLTTBL+3 = État des Slot secondaires du Slot primaire 3</p> <p>Format de la valeur :</p> <p>bit 0 et 1 = Slot secondaire sélectionné sur la plage mémoire 0</p> <p>bit 2 et 3 = Slot secondaire sélectionné sur la plage mémoire 1</p> <p>bit 4 et 5 = Slot secondaire sélectionné sur la plage mémoire 2</p> <p>bit 6 et 7 = Slot secondaire sélectionné sur la plage mémoire 3</p>
0FCC9h	SLTATR	64	Attributs de chaque Slot. (Voir « <a href="#">Développer un programme en cartouche</a> » au chapitre 13 page 314)
0FD09h	SLTWKR	128	Tableau pour réserver une zone de travail en RAM pour les application en ROM. (Voir « <a href="#">Développer un programme en cartouche</a> » au chapitre 13 page 314 pour la description)
0FD89h	PROCNM	16	Nom de l'instruction exécutées par CALL.
0FFE7h	RG8SAV	1	Contenu du registre 8 du VDP. (MSX2~)
0FFE8h	RG9SAV	1	Contenu du registre 9 du VDP. (MSX2~)
0FFE9h	RG10SAV	1	Contenu du registre 10 du VDP. (MSX2~)
0FFEAh	RG11SAV	1	Contenu du registre 11 du VDP. (MSX2~)
0FFEBh	RG12SAV	1	Contenu du registre 12 du VDP. (MSX2~)
0FFEC	RG13SAV	1	Contenu du registre 13 du VDP. (MSX2~)
0FFEDh	RG14SAV	1	Contenu du registre 14 du VDP. (MSX2~)
0FFEEh	RG15SAV	1	Contenu du registre 15 du VDP. (MSX2~)
0FFEFh	RG16SAV	1	Contenu du registre 16 du VDP. (MSX2~)
0FFF0h	RG17SAV	1	Contenu du registre 17 du VDP. (MSX2~)
0FFF1h	RG18SAV	1	Contenu du registre 18 du VDP. (MSX2~)
0FFF2h	RG19SAV	1	Contenu du registre 19 du VDP. (MSX2~)
0FFF3h	RG20SAV	1	Contenu du registre 20 du VDP. (MSX2~)
0FFF4h	RG21SAV	1	Contenu du registre 21 du VDP. (MSX2~)
0FFF5h	RG22SAV	1	Contenu du registre 22 du VDP. (MSX2~)
0FFF6h	RG23SAV	1	Contenu du registre 23 du VDP. (MSX2~)
0FFF7h	MINROM	1	Contient aussi le numéro de Slot de la Main-ROM mais, utilisez MNROM (0FCC1H) dans vos programme pour connaître l'emplacement de la Main-ROM. (MSX2~)
0FFF8h		2	Réservé.
0FFFAh	RG25SAV	1	Contenu du registre 25 du VDP. (MSX2+~)
0FFFBh	RG26SAV	1	Contenu du registre 26 du VDP. (MSX2+~)

0FFFCh    RG27SAV    1    Contenu du registre 27 du VDP. (MSX2+~)

0FFFDh                    2    Réserve.

0FFFFh    SLTSL    1    Adresse d'accès au registre de sélection des Slot secondaires (du Slot primaire sélectionné). La valeur lue correspond aux Slot secondaires dont les bits sont inversés afin de pouvoir déterminer si le Slot est étendu ou pas.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
$\overline{SS3}$	$\overline{SS2}$	$\overline{SS1}$	$\overline{SS0}$				

Attention : Étant donné qu'il faut sélectionner le Slot primaire correspondant avant de pouvoir sélectionner un Slot secondaire, prenez soin de désactiver les interruptions jusqu'à ce que les variables du système soit remises sur sa plage mémoire.

### 5.3 Quelques exemples d'utilisation des variables système

- Sous Basic, vous désirez couper un bout d'écran afin de réserver quelques lignes (sur l'exemple des touches de fonction). Il vous suffit de faire :

```
POKE &HF3B1,24-nn          (nn étant le nombre de lignes à réserver)
```

- Il est parfois utile de connaître la largeur de l'écran (réglé avec « WIDTH ») dans un programme d'application. Essayez :

```
L=PEEK (&HF3B0)
```

- Dans le même ordre d'idées, le nombre maximum de fichiers ouvrables (défini par l'instruction « MAXFILES ») s'obtient par :

```
M=PEEK (&HF85F)
```

- Qui n'a jamais connu le problème de cache du clavier qui renvoie des caractères au mauvais moment. Effacer ce cache est souvent utile :

En Basic :

```
POKE &HF3FA,PEEK (&HF3F8)
POKE &HF3FB,PEEK (&HF3F9)
```

En assembleur :

```
ld    hl,(PUTPNT)
ld    (GETPNT),hl
```

- Voici un truc un peu plus sophistiqué : l'interdiction de l'action combinée des touches CTRL+STOP peut facilement être obtenue en faisant croire au MSX que votre programme se trouve en cartouche de mémoire morte.

```
POKE &HFBB1,1 fera parfaitement l'affaire.
```

- Au contraire, pour ceux qui désirent accéder à des programmes Basic protégés contre le CTRL + STOP, il leur suffit de taper :

```
POKE &HFBB0,1 avant de charger le programme.
```

Puis, le programme ayant démarré, il faut presser simultanément les touches CTRL, SHIFT, GRAPH et CODE. Le MSX rendra alors la main.

- Pour ceux que la présence du curseur rassure, ils peuvent essayer en mode direct :

```
POKE &HFCA9,1
```

Pour se rendre compte de l'effet obtenu, il suffit de rentrer le petit programme suivant :

```
10 PRINT"Hello"
20 GOTO 20
```

et d'entrer RUN.

- Le dernier exemple est un peu plus conséquent que les autres. Il s'agit d'utiliser la zone réservée à la définition des touches fonction au mieux de manière à pouvoir assigner n'importe quelle chaîne de caractères à une touche. Notons qu'on peut mettre au maximum 39 caractères par touche, sachant que l'on ne dispose de toutes manières que de 160 octets et que si l'on étend le contenu d'une touche de fonction, ce ne peut être qu'aux dépens de la suivante.

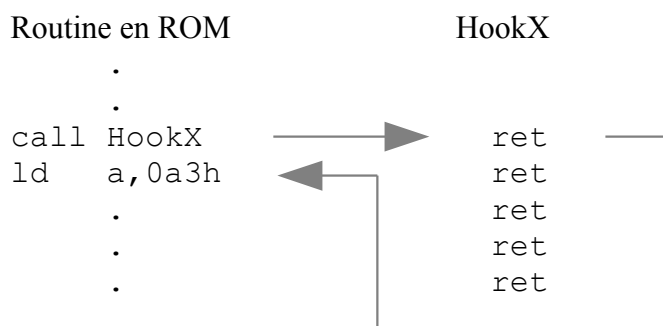
Le petit programme Basic suivant illustre cet exemple :

```
10 AD=&HF87F:CLS
```

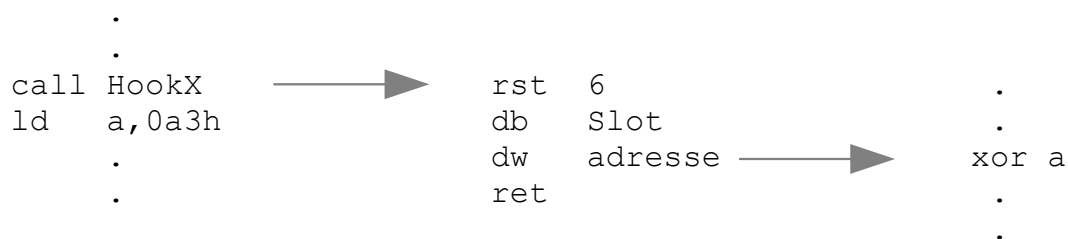
```
20 READ A$:IF LEN(A$)>39 THEN PRINT"Impossible!":END
30 FOR I=1 TO LEN(A$)
40 I$=MID$(A$,I,1)
50 POKE AD-1+I,ASC(I$)
60 NEXT I
70 POKE AD-1+I,0:END
100 DATA "C'est un peu long jeune homme ..."
```

## 6 Les Hooks du système

Un Hook est une zone mémoire de cinq octets en mémoire vive (RAM) qui permet d'étendre ou dérouter les routines en mémoire morte, ainsi que les instructions au Basic. De base, les cinq octets contiennent tous 0C9h (RET), comme dans l'exemple suivant :



Il est facile de modifier les 5 octets du Hook afin de détourner une fonction en ROM vers une autre dans un Slot différent, comme ceci :



Pour illustrer le fonctionnement des Hooks, je vous propose un exemple en Basic qui utilise le Hook H.LIST pour empêcher l'accès à la liste du programme.

```

10 POKE &HFF8C,&H40 ' met dans H.LIST :
20 POKE &HFF8B,&H9B ' 0FF89h: POP HL
30 POKE &HFF8A,&HC3 ' 0FF8Ah: JP 0409Bh
40 POKE &HFF89,&HE1 ' <- remplacer le &HC9 en dernier

```

Il suffit de faire POKE&HFF89, &HC9 pour restituer son pouvoir à la commande LIST.

Note : Dans la pratique avant de modifier un Hook votre programme devra déplacer le contenu du Hook si il est déjà utilisé pour que votre routine y fasse un saut avant ou après exécution.

## 6.1 La liste des Hooks

Voici la liste des Hooks pour tous les MSX. Elle comprend l'adresse, le nom du Hook, le nom de la routine qui l'appelle, ainsi que son utilité :

0FD9AH	<b>H.KEYI</b> (« KEY Interruption »)
Appel :	Au début de la routine des interruptions KEYINT (Main-ROM à 0038h).
Rôle :	Permet de tester si l'interruption a été provoquée par un dispositif autre que le VDP. (La RS-232C, le MSX-Midi, etc.)
0FD9Fh	<b>H.TIMI</b> (« TIMer Interruption »)
Appel :	Appelé par la routine des interruptions KEYINT (Main-ROM à 0038h) juste après avoir lu le registre de statut 0 du VDP.
Rôle :	Permet d'ajouter un Timer, de gérer les interruptions ligne du VDP, la collision de sprites, etc. L'appel à ce Hook se fait juste après la lecture du registre de statut 0 du VDP (le registre de statut spécifié par le registre de contrôle 15 sur MSX2 et plus récent). Ce Hook est appelé lorsqu'il s'agit de l'interruption provoquée en fin d'affichage (VBI). Le registre A contiendra la valeur lue.
0FDA4h	<b>H.CHPU</b> (« Character outPUt »)
Appel :	Au début de la routine CHPUT du Bios (Main-ROM à 00A2h) qui sert à afficher un caractère à l'écran en mode texte.
Rôle :	Permet de sortir le caractère sur un périphérique autre que l'écran.
0FDA9h	<b>H.DSPC</b> (« DiSPlay Cursor »)
Appel :	Au début de la routine interne DSPCSR qui sert à afficher le curseur.
Rôle :	Permettre l'accès à d'autres périphériques que l'écran.
0FDAEh	<b>H.ERAC</b> (« ERAsE Cursor »)
Appel :	Au début de la routine interne ERACSR d'effacement du curseur.
Rôle :	Permettre l'accès à d'autres périphériques que l'écran.
0FDB3h	<b>H.DSPF</b> (« DiSplay Fonctions »)
Appel :	Au début de la routine du Bios DSPFNK (Main-ROM à 00CFh) qui sert à afficher le contenu des touches de fonction.
Rôle :	Permettre l'accès à d'autres périphériques que l'écran.
0FDB8h	<b>H.ERAF</b> (« ERAsE Fonctions »)
Appel :	Au début de la routine ERAFNK du Bios (Main-ROM à 00CCh) qui sert à effacer le contenu des touches de fonction.

Rôle :	Permettre l'accès à d'autres périphériques que l'écran.
0FDBDh	<b>H.TOTE</b>
Appel :	Au début de la routine du Bios TOTEXT (Main-ROM à 00D2h) de passage en mode texte.
Rôle :	Permettre l'accès à d'autres périphériques que l'écran.
0FDC2h	<b>H.CHGE</b>
Appel :	Au début de la routine du Bios CHGET (Main-ROM à 009Fh) de lecture d'un caractère au clavier.
Rôle :	Permettre l'accès à d'autres périphériques d'entrée que le clavier du MSX.
0FDC7h	<b>H.INIP</b>
Appel :	Au début de la routine interne d'initialisation des caractères INIPAT qui copient la police des caractères vers la VRAM.
Rôle :	Permet de modifier le jeu de caractères lorsque l'on revient en mode texte.
0FDCCh	<b>H.KEYC</b> (« KEY Code »)
Appel :	Au début de la routine interne KEYCOD de lecture de clavier.
Rôle :	Permet d'intercepter la lecture du clavier. Lorsque le Hook est appelé, l'accumulateur contient 8 fois le numéro de ligne plus le numéro de colonne (numéro de bit) de la touche enfoncée dans <a href="#">la matrice du clavier</a> .
0FDD1h	<b>H.KYEA</b> (« KEY eAsy »)
Appel :	Au début de la routine interne KYEASY de conversion d'un caractère lu au clavier.
Rôle :	Permet de modifier la manière dont une touche est interprétée sous Basic.
0FDD6h	<b>H.NMI</b> (« Non-Masquable Interruption »)
Appel :	Au début de la routine NMI du Bios (Main-ROM à 0066h) d'interruptions non masquables.
Rôle :	Permet le traitement des interruptions non masquables. (Réservé mais jamais utilisé sauf peut-être en option par le Turbo R en mode R800)
0FDDb	<b>H.PINL</b>
Appel :	Au début de la routine PINLIN du Bios (Main-ROM à 00AEh) qui gère l'entrée d'une ligne de programme au clavier.
Rôle :	Permet d'utiliser un autre périphérique d'entrée que le clavier ou gérer le mode 80 colonnes.
0FDE0h	<b>H.QINL</b>



Appel : Au début de la routine QINLIN du Bios (Main-ROM à 00B4h) de l'entrée au clavier avec affichage d'un point d'interrogation.

Rôle : Permet d'utiliser le mode 80 colonnes, ou un autre périphérique d'entrée que le clavier.

**0FDE5h H.INLI**

Appel : Au début de la routine INLIN du Bios (Main-ROM à 000B1h) d'entrée au clavier.

Rôle : Permet d'utiliser les 80 colonnes, ou un autre périphérique d'entrée que le clavier.

**0FDEAh H.ONGO (« ON GOto procedure »)**

Appel : Au début de la routine interne ONGOTP, pour les instructions Basic de type ON GOTO, ON GOSUB.

Rôle : Permet de détourner ou modifier ce type d'instructions.

**0FDEFh H.DSKO (« DiSK sector Output »)**

Appel : Au début de l'instruction DSKO\$ du Basic.

Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.

Note : La Disk-ROM utilise ce Hook pour écrire le secteur d'un disque.

**0FDF4h H.SETS**

Appel : Au début de l'instruction SET du Basic.

Rôle : Détourner ou étendre l'instruction SET.

Note : Sur MSX1, l'instruction SET n'a pas d'autre effet que d'appeler ce hook et renvoyer une erreur. Sur MSX2 ou plus récent, les instructions SET SCREEN, SET ADJUST, SET TIME, etc appellent ce hook.

**0FDF9h H.NAME**

Appel : Au début de l'instruction NAME du Basic.

Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.

Note : La Disk-ROM utilise ce Hook pour renommer des fichiers.

**0FDFEh H.KILL**

Appel : Au début de l'instruction KILL du Basic.

Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.

Note : La Disk-ROM utilise ce Hook pour effacer des fichiers.

**0FE03h H.IPL**

Appel : Au début de l'instruction IPL du Basic.

Rôle : L'instruction IPL du Basic ne fait qu'appeler ce Hook car il s'agit d'une instruction réservée pour une utilisation future mais jamais utilisée.

Note : Un programmeur peut créer sa propre instruction. Voir le chapitre 13.8 « [Ajouter une instruction au Basic avec CMD ou IPL](#) » page 299 pour plus de précision.

#### 0FE08h **H.COPY**

Appel : Au début de l'instruction COPY du Basic.

Rôle : Ajouter l'instruction sur un système avec disque. Permet aussi d'étendre l'instruction. Par exemple, pour d'autres mode d'écran. Peut servir à empêcher la copie de fichier sur disque.

Note : Utilisé par la Disk-ROM.

#### 0FE0Dh **H.CMD** (« CoMmand »)

Appel : Au début de l'instruction CMD du Basic.

Rôle : L'instruction CMD du Basic ne fait qu'appeler ce Hook car il s'agit d'une instruction réservée pour une utilisation future mais jamais utilisée.

Note : Un programmeur peut créer sa propre instruction. Voir le chapitre 13.8 « [Ajouter une instruction au Basic avec CMD ou IPL](#) » page 299 pour plus de précision.

#### 0FE12h **H.DSKF** (« DiSK Free »)

Appel : Au début de la routine DSKF, utilisée l'instruction DSKF\$ du Basic.

Rôle : Ajouter l'instruction sur un système avec disque. Empêcher de connaître la mémoire disponible sur les disques. Utilisé par défaut par la Disk-ROM pour obtenir un accès aux disques.

Note : Utilisé par la Disk-ROM.

#### 0FE17h **H.DSKI** (« DiSK Input»)

Appel : Au début de la routine DSKI , utilisée par l'instruction DSKI\$ du Basic.

Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.

Note : Utilisé par la Disk-ROM.

#### 0FE1Ch **H.ATTR** (« ATTRibute »)

Appel : Au début de l'instruction ATTR\$ du Basic.

Rôle : Instruction réservée.

Note : Cette instruction ne fait qu'appeler ce Hook car il s'agit d'une instruction réservée pour une utilisation future mais jamais utilisée.

#### 0FE21h **H.LSET** (« Left SET »)

Appel : Au début de la routine LSET, utilisée par l'instruction LSET du Basic.  
 Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.  
 Note : Utilisé par la Disk-ROM.

**0FE26h H.RSET (« Right SET »)**

Appel : Au début de la routine RSET, utilisée par l'instruction RSET du Basic.  
 Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.  
 Note : Utilisé par la Disk-ROM.

**0FE2Bh H.FIEL**

Appel : Au début de la routine FIELD, utilisée par l'instruction FIELD du Basic.  
 Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.  
 Note : Utilisé par la Disk-ROM.

**0FE30h H.MKI\$ (« MaKe Integer »)**

Appel : Au début de la routine MKI, utilisée par l'instruction MKI\$ du Basic.  
 Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.  
 Note : Utilisé par la Disk-ROM.

**0FE35h H.MKS\$ (« MaKe Simple precision »)**

Appel : Au début de la routine MKS, utilisée par l'instruction MKS\$ du Basic.  
 Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.  
 Note : Utilisé par la Disk-ROM.

**0FE3Ah H.MKD\$ (« MaKe Double precision »)**

Appel : Au début de la routine MKD, utilisée par l'instruction MKD\$ du Basic.  
 Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.  
 Note : Utilisé par la Disk-ROM.

**0FE3Fh H.CVI (« ConVert Integer »)**

Appel : Au début de la routine CVI, utilisée par l'instruction CVI du Basic.  
 Rôle : Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.  
 Note : Utilisé par la Disk-ROM.

0FE44h	<b>H.CVS</b> (« ConVert Single precision »)
Appel :	Au début de la routine interne CVS, utilisée par l'instruction CVS du Basic.
Rôle :	Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.
Note :	Utilisé par la Disk-ROM.
0FE49h	<b>H.CVD</b> (« ConVert Double precision »)
Appel :	Au début de la routine interne CVD, utilisé par l'instruction CVD du Basic.
Rôle :	Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.
Note :	Utilisé par la Disk-ROM.
0FE4Eh	<b>H.GETP</b> (« GET file Pointer »)
Appel :	Au début de la routine interne GETPTR, positionnement sur un fichier.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.
0FE53h	<b>H.SETF</b> (« SET File »)
Appel :	Au début de la routine interne SETFIL, positionnement d'un pointeur sur un fichier ouvert au préalable.
Rôle :	Traiter la routine.
0FE58h	<b>H.NOFO</b> (« NO FOr close »)
Appel :	Au début de la routine interne NOFOR, utilisée par l'instruction OPEN du Basic.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.
0FE5Dh	<b>H.NULO</b> (« NULL file Open »)
Appel :	Au début de la routine NULOPN de la Disk-ROM, utilisée par les instructions LOAD, KILL, MERGE, etc du Basic.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.
0FE62h	<b>H.NTFL</b> (« NoT FiLe number 0 »)
Appel :	Au début de la routine NTFL0 de la Disk-ROM, utilisée par l'instruction CLOSE du Basic.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.

0FE67h	<b>H.MERG</b> (« MERGE program files »)
Appel :	Au début de l'instruction MERGE du Basic.
Rôle :	Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.
0FE6Ch	<b>H.SAVE</b> (« SAVE program files »)
Appel :	Au début de l'instruction SAVE du Basic.
Rôle :	Ajouter l'instruction sur un système avec disque ou la détourner/modifier lorsqu'elle est présente.
0FE71h	<b>H.BINS</b> (« BINary Save »)
Appel :	Au début de la routine interne BINSAV, sauvegarde d'une zone mémoire en binaire, utilisée par l'instruction SAVE du Basic.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.
0FE76h	<b>H.BINL</b> (« BINary Load »)
Appel :	Au début de la routine interne BINLOD, chargement d'une zone mémoire en binaire, utilisée par l'instruction LOAD du Basic.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.
0FE7Bh	<b>H.FILE</b>
Appel :	Au début de l'instruction FILES du Basic.
Rôle :	Traiter l'instruction.
Note :	Utilisé par la Disk-ROM.
0FE80h	<b>H.DGET</b> (« Disk GET »)
Appel :	Au début de la routine interne DGET, utilisée par l'instruction GET et PUT du Basic.
Rôle :	Traiter l'instruction.
Note :	Utilisé par la Disk-ROM.
0FE85h	<b>H.FILO</b> (« FILE Output 1 »)
Appel :	Au début de la routine interne FILOU1, sortie dans un fichier.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.

0FE8Ah	<b>H.INDS</b> (« INput DiSk character »)
Appel :	Au début de la routine interne INDSKC d'entrée d'attribut du disque.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.
0FE8Fh	<b>H.RSLF</b>
Appel :	Au début de la routine interne qui sélectionne l'ancien lecteur de disquettes comme lecteur actuel.
Rôle :	Traiter la routine.
0FE94h	<b>H.SAVD</b> (« SAVe to Disk »)
Appel :	Au début de la routine interne de sauvegarde du lecteur actuel, utilisée par les instructions LOF, LOC, EOF, FPOS, etc.
Rôle :	Ajouter ces instructions sur un système avec disque. Permet aussi de détourner ou modifier ces instructions lorsqu'elles sont présentes.
0FE99h	<b>H.LOC</b> (« LOCation »)
Appel :	Au début de la routine interne LOC, utilisée par l'instruction LOC du Basic.
Rôle :	Traiter la routine.
Note :	Utilisé par la Disk-ROM.
0FE9Eh	<b>H.LOF</b> (« Length Of File »)
Appel :	Au début de la routine interne LOF, utilisée par l'instruction LOF du Basic.
Rôle :	Détourner l'instruction.
Note :	Utilisé par la Disk-ROM.
0FEA3h	<b>H.EOF</b> (« End Of File »)
Appel :	Au début de la routine interne EOF, utilisée lors de l'instruction EOF du Basic.
Rôle :	Détourner l'instruction. Utilisé par défaut par la Disk-ROM pour obtenir un accès aux disques.
Note :	Utilisé par la Disk-ROM.
0FEA8h	<b>H.FPOS</b> (« File POSition »)
Appel :	Au début de l'instruction FPOS.
Rôle :	Instruction réservée.
Note :	Cette instruction ne fait qu'appeler ce Hook car il s'agit d'une instruction réservée pour une utilisation future mais jamais utilisée.
0FEADh	<b>H.BAKU</b> (« BAcK Up »)

Appel : Au début de la routine interne BAKUPT.

Rôle : Traiter la routine.

Note : Utilisé par la Disk-ROM.

**0FEB2h H.PARD (« PARse Device name »)**

Appel : Au début de la routine interne PARDEV qui analyse le nom du périphérique.

Rôle : Permet de changer le nom du disque logique («device device name »).

Note : Utilisé par la Disk-ROM.

**0FEB7h H.NODE (« NO DEvice name »)**

Appel : Au début de la routine interne NODEVN qui est appelé lorsque aucun nom n'a été trouvé dans la table des noms de périphérique.

Rôle : Permet de mettre un nom au disque dont le nom a été omis.

Note : Utilisé par la Disk-ROM.

**0FEBCh H.POSD (« POSSible Disk »)**

Appel : Au début de la routine interne POSDSK.

Rôle : Permet de raccorder un disque.

Note : Utilisé par la Disk-ROM.

**0FEC1h H.DEVN (« DEvice NAME »)**

Appel : Au début de la routine interne DEVNAM pour traiter le nom de périphérique.

Rôle : Permet d'étendre un nom de disque logique.

**0FEC6h H.GEND (« GENeral device Dispatcher »)**

Appel : Au début de la routine interne GENDSP pour assign le nom de périphérique.

Rôle : Permet d'étendre un nom du disque logique.

**0FECBh H.RUNC (« RUN Clear »)**

Appel : Au début de la routine interne RUNC, utilisée par les instructions NEW et RUN du Basic.

Rôle : Permet le détournement des instructions NEW et RUN.

**0FED0h H.CLEA (« CLEAR clear »)**

Appel : Au début de la routine interne CLEARC qui initialise les variables, utilisée par l'instruction CLEAR du Basic.

Rôle : Permet d'éviter l'effacement des variables Basic, par exemple.

0FED5h	<b>H.LOPD</b> (« LOoP and set Default »)
Appel :	Au début de la routine interne LOPDFT, initialisation de la table des variables.
Rôle :	Traiter la routine.
0FEDAh	<b>H.STKE</b> (« STAcK Error »)
Appel :	Au début de la routine interne STKERR, erreur de pile.
Rôle :	Lors de l'initialisation ou du MSX-DOS, le MSX vérifie si ce Hook a été modifié par une éventuelle cartouche utilisant le lecteur de disquette (contenu de 0FEDAh différent de 0C9h).
0FEDFh	<b>H.ISFL</b> (« IS FiLe I/O »)
Appel :	Au début de la routine interne ISFLIO qui teste si le fichier à écrire ou à lire.
Rôle :	Traiter la routine.
0FEE4h	<b>H.OUTD</b> (« OUT Do »)
Appel :	Au début de la routine du Bios OUTDO (Main-ROM à 0018h), sortie d'un caractère sur écran ou imprimante.
Rôle :	Faire une routine pour un autre périphérique.
0FEE9h	<b>H.CRDO</b> (« CaRriage left DO »)
Appel :	Au début de la routine interne CRDO qui émet un code CR (« Carriage Return », 0Dh) et d'un LF (« Line Feed », 0Ah).
Rôle :	Permet, par exemple, d'adapter l'impression des lignes sur une imprimante possédant un line-feed automatique.
0FEEeh	<b>H.DSKC</b> (« DiSK Character input »)
Appel :	Au début de la routine interne DSKCHI, pour l'entrée de l'attribut d'un disque.
Rôle :	Traiter la routine.
0FEF3h	<b>H.DOGR</b>
Appel :	Au début de la routine interne DOGRPH, utilisée par les instructions du Basic de traçage graphiques .
Rôle :	Traiter la routine.
0FEF8h	<b>H.PRGE</b> (« PRoGram End »)
Appel :	Au début de la routine PRGEND, appelé à la fin de l'exécution d'un programme Basic.
Rôle :	Permet n'importe quelle action avant de rendre la main à l'utilisateur.
0FEFDh	<b>H.ERRP</b> (« ERRor Print »)



Appel : Au début de la routine ERRPRT pour l'affichage du message d'erreur sous Basic.  
 Rôle : Permet de changer ou ajouter un message d'erreur.  
 Note : Utilisé par la Disk-ROM.

**0FF02h H.ERRF**

Appel : À la fin de la routine qui affiche le message d'erreur sous Basic.  
 Rôle : Permet une action supplémentaire après l'affichage de l'erreur.

**0FF07h H.READ (« READy »)**

Appel : Au début de la routine READY qui affiche le message « Ok » (ou celui défini par l'utilisateur sur MSX2~).  
 Rôle : Permet, par exemple, de provoquer d'un bip sonore à chaque affichage de « OK ».

**0FF0Ch H.MAIN (« MAIN entry »)**

Appel : Au début de la routine MAIN, utilisée à chaque accès à l'interpréteur Basic.  
 Rôle : Permet de programmer une interruption par exemple.

**0FF11h H.DIRD (« DIRection Do »)**

Appel : Au début de la routine DIRDO qui est appelée lors de l'exécution d'instructions en mode direct.  
 Rôle : Permet toutes les manipulations sur le mode direct du Basic.

**0FF16h H.FINI (« Function INterpretation Initialize »)**

Appel : Au début de la routine FININT, utilisée lors de la traduction d'une instruction Basic par l'interpréteur.  
 Rôle : Permet de détourner le traitement des instructions Basic.

**0FF1Bh H.FINE (« Function INterpretation End »)**

Appel : Au début de la routine FINEND, utilisée à la fin de l'interprétation d'une instruction Basic.  
 Rôle : Ajouter une routine.

**0FF20h H.CRUN**

Appel : Au début de la routine CRUNCH, transformation d'une ligne Basic en mots-clefs.  
 Rôle : Traiter la routine.

**0FF25h H.CRUS**

Appel : Au début de la routine CRUSH qui recherche un mot-clef dans la liste alphabétique en Rom.

Rôle :	Traiter la routine.
0FF2Ah	<b>H.ISRE</b> (« IS token REserved »)
Appel :	Au début de la routine ISRESV lorsqu'un mot-clef est trouvé par la routine CRUSH.
Rôle :	Traiter la routine.
0FF2Fh	<b>H.NTFN</b>
Appel :	Au début de la routine NTFN2 lorsqu'un mot-clef est suivi par un numéro de ligne.
Rôle :	Traiter la routine.
0FF34h	<b>H.NOTR</b> (« token NOT Reserved »)
Appel :	Au début de la routine NOTRSV lorsque la suite de caractères examinée par la routine CRUNCH ne constitue pas un mot-clef.
Rôle :	Traiter la routine.
0FF39h	<b>H.SNGF</b>
Appel :	Au début de la routine SNGFOR, accès aux fonctions mathématiques BCD
Rôle :	Permet d'installer de nouvelles routines mathématiques.
0FF3Eh	<b>H.NEWS</b> (« NEW SStatement »)
Appel :	Au début de la routine NEWSTT, passage à une nouvelle instruction Basic.
Rôle :	Traiter la routine.
0FF43h	<b>H.GONE</b>
Appel :	Au début de la routine GONE2, utilisée par les instructions de saut (GOTO, THEN, ...)
Rôle :	Traiter la routine.
0FF48h	<b>H.CHRG</b> (« CHaracter GeTteR »)
Appel :	Au début de la routine CHRGET (00010h) de récupération d'un caractère.
Rôle :	Traiter la routine.
0FF4Dh	<b>H.RETU</b> (« RETUrN from sub-routine »)
Appel :	Au début de l'instruction RETURN du Basic.
Rôle :	Changer le traitement de l'instruction.
0FF52h	<b>H.PRTF</b>

Appel :	Au début de l'instruction Basic PRINT
Rôle :	Changer le traitement de l'instruction.
0FF57h	<b>H.COMP</b>
Appel :	Au début de la routine COMPRT, boucle interne à l'instruction Basic PRINT
Rôle :	Changer le traitement de l'instruction.
0FF5Ch	<b>H.FINP</b> (« FINal Print »)
Appel :	A la fin de l'affichage d'un texte sous Basic.
Rôle :	Ajouter une routine après l'affichage d'un texte.
0FF61h	<b>H.TRMN</b>
Appel :	Au début de la routine TRMNOK, traitement d'une mauvaise donnée entrée pour les instructions Basic READ et INPUT.
Rôle :	Traiter l'erreur.
0FF66h	<b>H.FRME</b> (« FoRMula EVaLuator »)
Appel :	Au début de la routine interne FRMEVL de reconnaissance d'une expression dans le texte d'un programme Basic.
Rôle :	Permet d'ajouter de nouvelles fonctions mathématiques au Basic
Note :	Voici la description de la routine FRMEVL. Entrée : HL = Pointeur du texte. Sortie : HL = Pointeur vers l'expression trouvée. VALTYP (F663h) = Type de valeur de l'expression. DAC (F7F6h) = Valeur trouvée.
0FF6Bh	<b>H.NTPL</b>
Appel :	Au début de la routine utilisée lors de FRMEVL.
Rôle :	Permet d'ajouter de nouvelles fonctions mathématiques au Basic.
0FF70h	<b>H.EVAL</b> (« EVALuate statement »)
Appel :	Au début de la routine EVALST, évaluation d'une expression.
Rôle :	Permet d'ajouter de nouvelles fonctions mathématiques au Basic
0FF75h	<b>H.OKNO</b> (« OK or NO ») ou <b>H.MDIN</b> (« MiDi IN »)
Appel :	Au début de la routine de calcul de fonctions transcendantes (MSX1/2/2+) ou au début de la routine d'interruption de l'entrée du MSX-Midi (MSX Turbo R).
Rôle :	Traiter la routine de calcul de fonctions transcendantes ou de gérer les interruptions de l'entrée Midi.

0FF7Ah	<b>H.FING</b>
Appel :	A la fin de la routine de calcul de fonctions transcendantes.
Rôle :	Traiter la routine.
0FF7Fh	<b>H.ISMI</b> (« IS Mid\$ »)
Appel :	Au début de la routine ISMID\$, utilisée lors de l'instruction MID\$ du Basic.
Rôle :	Traiter l'instruction.
0FF84h	<b>H.WIDT</b> (« WIDTH of screen »)
Appel :	Au début de la routine WIDTHS, utilisée lors de l'instruction Basic WIDTH.
Rôle :	Traiter l'instruction.
0FF89h	<b>H.LIST</b>
Appel :	Au début de l'instructions Basic LIST (et LLIST).
Rôle :	Traiter l'instruction.
0FF8Eh	<b>H.BUFL</b> (« Buffer Line »)
Appel :	Par la routine de l'instruction Basic LIST (et LLIST) qui convertie le code d'une instruction en nom.
Rôle :	Traiter la routine.
0FF93h	<b>H.FRQI</b> ou <b>H.MDTM</b> (si l'MSX avec Midi interne)
Appel :	Au début de la routine interne FRQINT (MSX1/2/2+) ou au début de la routine d'interruptions HMDTM du timer du MSX-Midi interne (MSX Turbo R) .
Rôle :	Traiter la routine ou, de gérer l'interruption du timer Midi qui se produit toutes les 5 ms.
0FF98h	<b>H.SCNE</b>
Appel :	Au début de la routine SCNEX2 de l'interpréteur Basic, conversion d'un numéro de ligne en adresse mémoire et inversement.
Rôle :	Traiter la routine.
0FF9Dh	<b>H.FRET</b> (« FREe up Tempories »)
Appel :	Au début de la routine FRETMP de l'interpréteur Basic, recherche d'un emplacement libre pour stocker une variable alphanumérique.
Rôle :	Traiter la routine.
0FFA2h	<b>H.PTRG</b> (« PoinTeR Get »)

Appel : Au début de la routine PTRGET de l'interpréteur Basic, recherche d'une variable.  
Rôle : Permet d'utiliser d'autres type de variables que celle par défaut.

**0FFA7h H.PHYD (« PHYsical Disk I/O »)**

Appel : Au début de la routine PHYDIO du Bios (Main-ROM à 0144h).  
Rôle : Traiter la routine. Normalement, si 0FFA7h contient 0C9h, c'est qu'il n'y a pas de disque installé.  
Note : Utilisée par la Disk-ROM.

**0FFACh H.FORM (« disk FORMatter »)**

Appel : Au début de la routine FORMAT du Bios (Main-ROM à 0147h).  
Rôle : Traiter la routine.  
Note : Utilisée par la Disk-ROM.

**0FFB1h H.ERRO (« ERROr »)**

Appel : Au début de la routine ERROR de traitement d'une erreur sous Basic. Le registre A contient le numéro d'erreur (voir le [tableau des codes d'erreurs du Basic](#) page 348).  
Rôle : Permet de traiter des erreurs avec sa propre routine.

**0FFB6h H.LPTO (« Line PrinTer Output »)**

Appel : Au début de la routine LPTOUT, sortie d'un caractère sur imprimante.  
Rôle : Permet d'utiliser une imprimante non-MSX.

**0FFBBh H.LPTS (« Line PrinTer Status »)**

Appel : Au début de la routine LPTSTT, test de l'état de l'imprimante.  
Rôle : Permet d'utiliser une imprimante non-MSX.

**0FFC0h H.SCRe (« SCREen »)**

Appel : Au début de l'instruction SCREEN du Basic.  
Rôle : Traiter l'instruction.

**0FFC5h H.PLAY**

Appel : Au début de l'instruction PLAY du Basic.  
Rôle : Traiter l'instruction.

**0FFCFh H.BGFD**

Appel : Juste avant chaque accès à un disque.  
Rôle : Traiter l'instruction.

Note : Obsolète depuis l'introduction du Bios étendu. Remplacé par la routine DISINT ([voir Bios étendu](#) page 123).

0FFD4h

**H.ENFD**

Appel : Juste après chaque accès à un disque.

Rôle : Traiter l'instruction.

Note : Obsolète depuis l'introduction du Bios étendu. Remplacé par la routine ENAINT ([voir Bios étendu](#) page 123).

## 7 Le processeur vidéo (VDP)

### 7.1 *Avertissement*

Avec le processeur vidéo, nous abordons un domaine plus spécifique aux MSX. Ce livre contient tous les renseignements nécessaires pour faire fonctionner correctement le processeur vidéo de chaque version de MSX en ne tenant compte que de chaque mode d'écran du MSX1 à MSX turbo R. Quant aux registres, ils seront entièrement détaillés afin de pouvoir y accéder directement sans avoir de passer par le Bios.

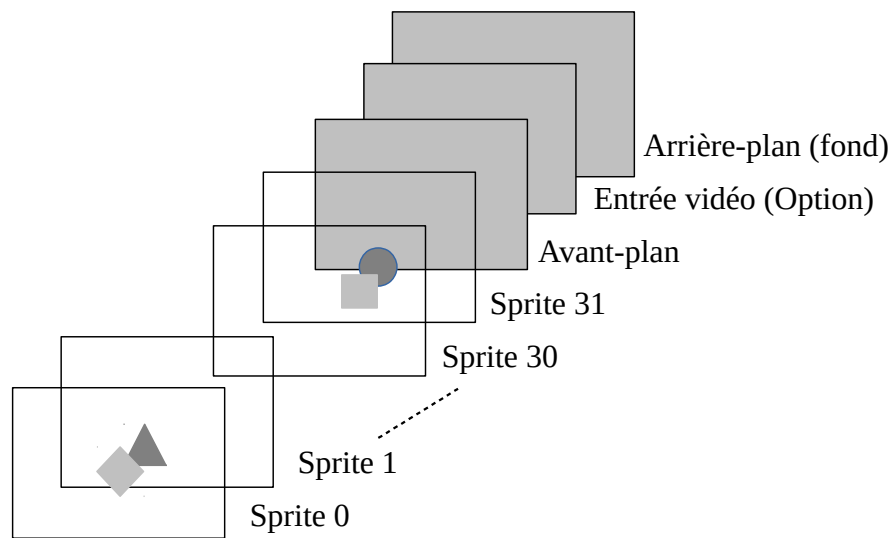
### 7.2 *Introduction au processeur vidéo*

Le processeur vidéo ou VDP (« Video Display Processor ») est sans aucun doute l'élément le plus excitant du système MSX. Surtout sur MSX2 avec sa mémoire vidéo énorme (128 Ko en général) qui offre une résolution graphique extraordinaire pour du matériel grand public de l'époque. Il autorise l'affichage de deux cent cinquante six couleurs simultanément à l'écran sur MSX2 et plusieurs milliers sur MSX2+. Le VDP comporte par ailleurs un jeu complet de commandes graphiques permettant notamment le tracé de lignes, de rectangles pleins ou vides, de scrolling pour ne citer que quelques fonctions. La puissance du V99x8 vient du fait que c'est un processeur VLSI (Very Large Scale Integration), c'est-à-dire qu'il atteint un degré d'intégration encore inégalé en micro-informatique. Le V9938 restera dans l'histoire comme le premier processeur VLSI présent en informatique grand public.

### 7.3 *Fonctionnement du VDP*

Le principe d'un processeur vidéo est relativement simple. Il s'agit de soulager le micro-processeur central de toutes les tâches liées au graphisme en lui adjoignant un second processeur dédié à l'affichage. Au niveau de la programmation, on peut ignorer totalement la présence du processeur et n'utiliser que les routines du Bios (tracé de lignes, chargement de pages, Sprites, etc). Cette solution, si elle offre l'avantage de la simplicité, prive le programmeur de toute une série de fonctions non-disponibles avec le Bios (scrolling, clignotement pour ne citer que deux exemples). Il est donc intéressant de pouvoir accéder directement au processeur vidéo. Ceci s'opère par l'intermédiaire de registres, huit sur MSX1 et quarante sept sur MSX2 - ce qui vous donne une idée de la différence de puissance entre le processeur du MSX1 et celui du MSX2. La plupart des registres définissent des paramètres précis, la couleur du texte par exemple, alors que quelques registres ont une fonction spéciale, écrire dans le registre 46 déclenche automatiquement une opération, le tracé d'une ligne par exemple. Vous trouverez tout au long de ce chapitre l'explication du contenu de chaque registre ainsi qu'un exposé des différents modes graphiques et des Sprites.

Avant de voir tout ça, voici un petit schéma qui montre les priorités d'affichage du VDP pour se donner une image de ce quoi on parle par la suite.



### 7.4 Comment accéder au VDP

Il existe deux moyens d'accéder au processeur vidéo. Tout dépend de ce que vous cherchez à faire et du MSX utilisé.

1. Soit vous appelez simplement les routines du Bios en Main-ROM ou en Sub-ROM.
2. Soit vous décidez d'y accéder directement pour une application qui nécessite une vitesse la plus élevée possible. L'utilisation du Bios ralentit légèrement les accès au VDP.

Dans le premier cas, la solution est évidente : utilisez les routines WRTVDP, RDVDP, SETPLT, GETPLT et VDPSTA du Bios.

Dans le second cas, la situation est moins simple, sans passer par le Bios, point de salut, vous risquez de perdre la compatibilité dès que vous utilisez l'instruction OUT ou IN du Z80. Heureusement, Microsoft et ASCII ont prévu ce cas de figure lors de la conception du système MSX. Voici la marche à suivre pour accéder directement au VDP par les ports E/S :

Dans le Bios en Main-ROM, il y a 2 octets qui servent à indiquer l'adresse du port d'E/S pour accéder au VDP. Le premier port de lecture est indiqué par ce que contient la ROM à l'adresse 0006h et le premier port d'écriture par le contenu de l'adresse suivante. Les autres ports se trouvent à la suite comme indiqué dans le tableau suivant.

Port de lecture	Adresse du port d'entrée du CPU	Fonction
0	Contenu de VDP.DR (0006h en Main-ROM)	Lecture d'une donnée en VRAM
1	Contenu de VDP.DR +1	Lecture du registre de statut sélectionné



Port d'écriture	Adresse du port de sortie du CPU	Fonction
0	Contenu de VDP.DW (0007h en Main-ROM)	Écrire en VRAM
1	Contenu de VDP.DW +1	Écrire dans les registres
2	Contenu de VDP.DW +2	Écrire dans la palette de
3	Contenu de VDP.DW +3	Envoi de données successives dans un ou une suite de registre de contrôle

Les MSX1 sont équipés en général d'un VDP de première génération (TMS9918 à TMS9929 selon le modèle). Ces VDP ont deux ports d'écriture et deux de lecture (ports 0 et 1).

Les MSX de générations suivantes sont équipés d'un VDP V9938 (MSX2) ou d'un V9958 (MSX2+ et MSX turbo R). Ces VDP ont deux ports d'écriture (ports 2 et 3) supplémentaires.

Notes : - Certains MSX1 sont équipés d'un V9938 (le Yamaha CX5MII et le Spectravideo SVI-738). Il y a même certains MSX2 équipés d'un V9958. Leur Bios ne contient cependant que les routines correspondantes à leur génération.

- Dans les faits, tous les MSX utilisent les ports de lecture 098h et 099h et les ports d'écriture 098h à 09Bh pour accéder au VDP interne. Ça a d'ailleurs été officialisé à la sortie du MSX2+. Seuls les VDP externes utilisent des ports différents. Et le seul VDP externe dont les ports E/S sont standardisés est le V9938 utilisé pour l'extension MSX1 vers MSX2 ou pour le mode 80 colonnes. Ce V9938 utilise les ports de lecture 088h et 089h et les ports d'écriture 088h à 08Bh.

### Écrire dans un registre de contrôle (0 ~ 23, 25 et 32 ~ 46)

1. Lire le contenu de la case mémoire à l'adresse 00007h en Main-ROM. Vous aurez ainsi l'adresse d'accès au port 0 du VDP. L'adresse du port 1 sera la suivante. (Si vous ne comprenez pas la signification de Main-ROM, voyez le chapitre concernant les Slot.)
1. Envoyer la donnée à écrire sur le port 1. (« OUT »).
2. Envoyer, toujours sur le port 1, le numéro du registre dans lequel vous voulez écrire en gardant le bit de poids fort à 1, ce qui revient à ajouter 80h au numéro du registre. Si le registre de statut 0 est lu avant l'écriture du numéro du registre (à cause d'une interruption par exemple), la donnée sera perdue.

Un exemple :

```
< Mettre le bit 2, du registre de contrôle 9 du VDP, à 0. >

        org      0c000h
;
; Le programme suivant active le mode d'affichage
; à 60 Hertz. Sur un téléviseur ou un moniteur
; en 50 Hz, l'image "sautera". Il faut remettre
; le bit 2 à 0 dans le registre neuf afin de
; rétablir une image normale.
;
DEBUT:   ld      a, (7)          ; C = Numéro du port E/S correspondant
        ld      c, a           ; au port 0 d'écriture du VDP.
        inc     c              ; Port E/S du port 1 d'écriture.
;
        ld      a, (0ffe8h)     ; Met le contenu de RG09SAV dans A
```

```

and    0fdh          ; Met le bit 2 à 0
di          ; Interruptions interdites
out     (c),a        ; Envoi sur le port 1
ld      (0ffe8h),a    ; Actualise la variable RG09SAV
ld      a,80h+9       ; Numéro de registre avec bit 7 à 1
out     (c),a        ; Envoi sur le port 1
ei          ; Interruptions autorisées
ret

;
; NOTE: Ce programme lit le contenu de RG09SAV afin de préserver
; l'état des autres bits du registre
;
END:

```

L'équivalent en Basic de ce programme serait :

```
C=PEEK(7):C=C+1:OUT C,(PEEK(&HFFE8)AND &HFD):OUT C,&H80+9
```

ou mieux :

```
VDP(10)=VDP(10)AND &HFD
```

Dans certains cas, il peut être utile d'écrire dans une suite de registres ou d'écrire plusieurs fois dans le même registre. Pour cela, l'accès indirect est possible grâce au port 3 du processeur vidéo :

1. Écrire le numéro du registre dans lequel vous désirez écrire dans le registre 17 de contrôle avec la méthode précédente.
1. Envoyez vos données, les unes à la suite des autres sur le port 3 du VDP.

L'opération d'auto-incrémentation a normalement lieu, à savoir que la valeur dans le registre 17 augmente à chaque accès au port 3. Il est possible d'annuler cette fonction de manière à toujours écrire dans le même registre. Il suffit de mettre à 1 le bit 7 du registre 17, ce qui revient à ajouter 80h.

Exemple d'écriture indirecte :

< Mettre le bit 2 du registre de contrôle 9 à 0, attendre un peu puis, remise du bit 2 à 1. >

```

DEBUT:    org    0c000h

ld      a,(7)          ; C = Numéro du port E/S correspondant
ld      c,a            ; au port 0 d'écriture du VDP.
inc     c              ; Port E/S du port 1 d'écriture.

ld      a,80h+9        ; Registre 9 sans auto-incrémentation...
di          ; Interruptions interdites
out     (c),a          ;
ld      a,80h+17       ; ...dans registre 17 (+80h)
out     (c),a          ;
ei          ; Interruptions autorisées

inc     c              ; Port E/S du port 2 dans C
inc     c              ; Port E/S du port 3 dans C
ld      a,(0ffe8h)     ; Contenu de RG09SAV dans A
and     0fdh          ; Mettre le bit 2 à 0 (mode 60 Hz)
out     (c),a          ; Envoi sur le port 3
ld      (0ffe8h),a     ; Actualise RG09SAV

```

```

call ATTEND      ; Appel le sous-programme d'attente

or    2          ; Met le bit 2 à 1 (mode 50 Hz)
ld    (0ffe8h),a ; Actualise RG09SAV
out   (c),a      ; Envoi sur le port 3
ret

ATTEND:
ld    b,a        ; Préserve A
push  bc         ; et C
ld    bc,0       ; Boucle d'attente

BCL:
dec   bc         ; Décremente BC
nop   ; Petit temps mort
ld    a,b
or    c
cp    0
jr    nz,BCL     ; Saute si BC > 0
pop   bc         ; Restitue C
ld    a,b        ; et A
ret

```

## Ecrire dans un registre de la palette des couleurs

1. Écrire le numéro de la couleur à modifier (0 ~ 15) dans le registre de contrôle 16.
1. Envoyez sur le port 2 du VDP, deux octets qui codent la nouvelle teinte au format suivant.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Premier octet :	0	rouge (0~7)			0	bleu (0~7)		

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Deuxième octet :	0	0	0	0	0	vert (0~7)		

Exemple :

< Changer la couleur n°1 (noir), en vert. >

```

org    0c000h
DEBUT:
ld     a,(7)      ; C = Numéro du port E/S correspondant
ld     c,a        ; au port 0 d'écriture du VDP.
inc    c          ; Port E/S du port 1 d'écriture.
;
ld     a,1        ; A=1
di     ; Interruptions interdites.
out    (c),a      ; Envoie de la donnée 1.
ld     a,80h+16   ; A=16 (avec le bit 7 à 1)
out    (c),a      ; Ecriture de la donnée 1 dans le registre 16.
ei     ; Interruptions autorisées
;
inc    c          ; Port E/S du port 2 d'écriture
ld     a,42h      ; rouge=4 bleu =2
out    (c),a      ; Envoie de la donnée 42h.
ld     a,5        ; vert=5
out    (c),a      ; Envoie de la donnée 5.
ret

```

## Lire un registre de statut du VDP

Sur un MSX1, il n'y a qu'un seul registre de statut donc il suffit de lire le port correspondant. Sur MSX2 ou plus récent, par défaut vous pouvez procéder comme sur MSX1 seulement pour lire le registre de statut 0. Autrement, il faut procéder de la façon suivante.

1. Écrire le numéro de registre de statut que vous désirez lire (0 ~ 9) dans le registre de contrôle
2. Lire le port 1 du VDP pour obtenir le contenu du registre. (Sur MSX1, seule cette étape est nécessaire.)
3. Remettre 0 dans le registre de contrôle 15. Cette étape est nécessaire à partir du MSX2 car à chaque interruption la routine du Bios lit le registre de statut 0 sans tenir compte du registre de contrôle 15.

Exemple :      < lire le registre de statut 1 >

```
org 0c000h
DEBUT:
    ld    a,(7)        ; C = Numéro du port E/S correspondant
    ld    c,a          ; au port 0 d'écriture du VDP.
    inc   c            ; Port E/S du port 1 d'écriture.
    ld    b,c          ; Préserve le Numéro de port dans B

    ld    a,0
    di                    ; Interruptions interdites
    out   (c),a         ; Numéro du registre de statut...
    ld    a,80h+15      ;
    out   (c),a         ; ...dans le registre 15

    ld    a,(6)        ; C = Numéro du port E/S correspondant
    ld    c,a          ; au port 0 de lecture du VDP
    inc   c            ; Port E/S du port 1 d'écriture
    in    (c),a         ; Lecture du registre

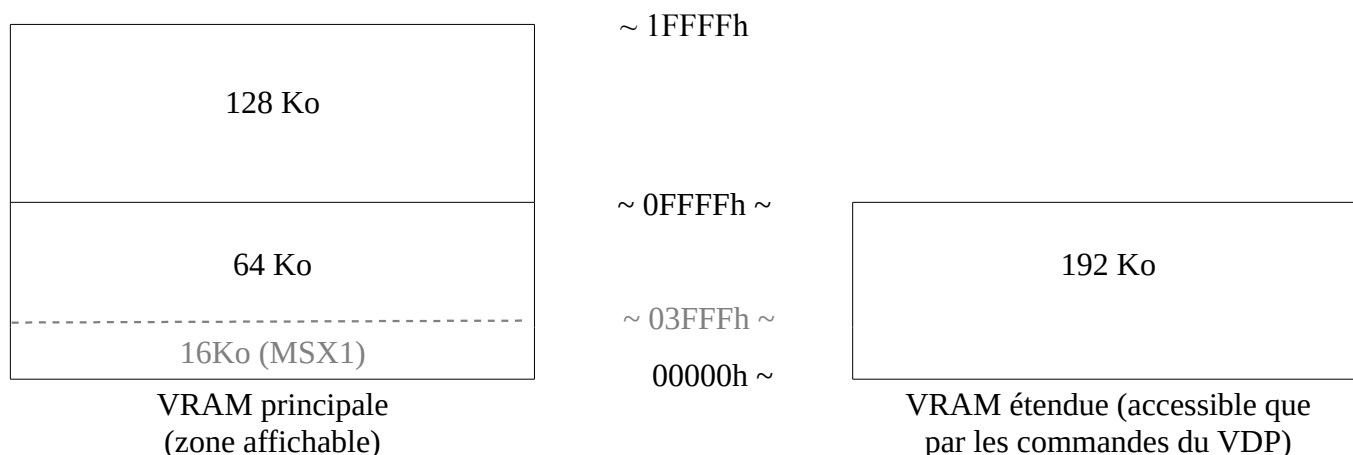
    ld    b,c          ; Restitue le Numéro de port 1
    ld    a,0
    out   (c),a         ; Registre de statut 0
    ld    a,80h+15      ;
    out   (c),a         ; dans le registre 15
    ei                    ; Interruptions autorisées
    ret
```

Note : À chaque interruption le registre de statut 0 est lu par un simple IN A, (099h) donc sur MSX2 ou plus récent, il faut remettre le registre 15 à 0 avant qu'une interruption se produise après la lecture d'un autre registre de statut pour assurer le bon fonctionnement du système.

## 7.5 Lecture et écriture dans la mémoire vidéo

Un MSX peut être équipé de 16, 64, 128 ou 192 Ko de mémoire vive réservée à la vidéo (VRAM). Tous les MSX2 commercialisés en France comportent 128 Ko. Cette mémoire ne se trouve pas dans un Slot, ce faisant, le Z80 ne peut en aucun cas l'adresser, seul le processeur vidéo peut le faire.

Carte de la VRAM



Il existe trois méthodes permettant de manipuler la mémoire vidéo. La première consiste à utiliser le Bios, puisqu'on y trouve les routines WRTVRM, RDVRM qui respectivement écrivent et lisent la mémoire vidéo. Les deux autres méthodes nécessitent des accès directs au processeur vidéo, l'une passe par l'envoi de l'adresse à laquelle on désire accéder puis lecture/écriture de la donnée, alors que l'autre passe par les commandes du processeur vidéo. Pour ce qui est de cette dernière solution, elle est détaillée dans la partie concernant les commandes propres au VDP.

Nous allons à présent détailler en 5 étapes la méthode d'accès à la VRAM par envoie de l'adresse afin de lire ou écrire une donnée.

1. Choisir la VRAM principale ou la VRAM secondaire en agissant sur le bit 6 du registre 45 du VDP (0=VRAM principale, 1=VRAM étendue). Cette opération n'est nécessaire que lors du premier accès vidéo. En général, vous n'avez pas à vous occuper de cette étape. Aucun MSX officiel ont 192 Ko de VRAM.
2. Sur MSX2 ou plus récent, l'adressage de la VRAM est codée sur 17 bits (0 à 1FFFFh). Le registre 14 sert à manipuler les 3 bits de poids fort de l'adresse qui vous intéresse. Charger ces 3 bits (A16, A15 et A14) dans le registre 14. (Sur MSX1, vous n'avez pas à vous occuper de cette étape.)
3. Envoyer sur le port 1 du VDP les 8 bits de poids faible de l'adresse (bits A7 à A0).
4. Envoyer à nouveau sur le port 1 les bits manquants de l'adresse (bits A13 à A8) ainsi que l'instruction d'écriture ou de lecture en VRAM.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Octet :	00=lecture / 01=écriture	A13	A12	A11	A10	A9	A8	

5. Lire ou écrire la donnée sur le port 0 du VDP. L'auto-incrémentation de l'adresse ayant lieu, il n'est pas nécessaire de redéfinir l'adresse pour accéder à chaque octet suivant.

Exemple :

< en SCREEN 0 / 40 colonnes, envoyer des caractères dans la table des caractères afin d'afficher un message en haut à droite de l'écran >

```

org      0c000h
DEBUT:
    ld     a,(7)      ; C = Numéro du port E/S relié
    ld     c,a        ; au port 0 d'écriture du VDP.
    inc    c          ; Port E/S relié au port 1 d'écriture.
;
    ld     a,0        ; 3 bits de poids fort      } Supprimez
    di     ; Interruptions interites
    out    (c),a      ;                          } ces quatre
    ld     a,80h+14   ; 3 bits de poids fort      } lignes sur
    out    (c),a      ; dans le registre 14      } MSX1
;
    ld     a,01fh     ; 8 bits de
    out    (c),a      ; poids faible = 31
;
    ld     a,040h     ; Bits 6~7 mis à 01 pour une
    out    (c),a      ; écriture + autres bits de l'adresse
    ei     ; Interruptions autorisées
;
    ex     (sp),hl    ; Temps d'attente
    ex     (sp),hl    ; pour VDP MSX1
;
    dec    c          ; Numéro du port E/S relié au port 0 dans C
    ld     hl,DATA    ; Position des données
    ld     b,9        ; Neuf lettres
ENVOIS:
    outi           ; Transfert des données sur le port 0
    djnz   ENVOIS    ; (Peut-être remplacé par OTIR sur MSX2~)
    ret
DATA:      db      'CA MARCHE'
END:

```

## 7.6 Les registres de contrôle du processeur video.

Ces registres permettent de modifier le comportement du VDP. Ils ne sont accessibles qu'en écriture. Il est donc conseillé, voir nécessaire, d'actualiser les variables système juste après chaque écriture à un registre de 0 à 25 par accès direct.

Voici la liste complète de tous les registres de contrôle et la signification des informations qu'ils contiennent :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 0 :	0	DG	IE2	IE1	M5	M4	M3	EV

EV = Entrée vidéo externe. 1 pour l'activer ; 0 pour la désactiver (valeur par défaut).

M3 = Bit de mode graphique.

M4 = Bit de mode graphique. (Inutilisé sur MSX1.)

M5 = Bit de mode graphique. (Inutilisé sur MSX1.)

IE1 = Autorisation des interruptions qui se produisent juste avant le balayage de la ligne horizontale indiquée au registre 19. (Inutilisé sur MSX1.)

IE2 = Autorisation des interruptions du crayon optique. (Inutilisé sauf les MSX2 coréens Daewoo CPC- 300 et CPC-400/400S, mettre à 0 sinon.) (N'existe pas sur le V9958)

DG = 1 pour mettre le bus de couleur en mode « entrée » afin de récupérer les données en VRAM sur les MSX équipé d'un digitaliseur. (Inutilisé sur MSX1.)

Note : Pour plus de détails sur les bits de mode graphique, voir le registre 1 du VDP ci-dessous.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 1 :	4/16k	BL	IE0	M1	M2	0	SI	MAG

MAG = Doublement de la taille des Sprites : 1 = Double taille ; 0 = Taille ordinaire.

SI = Taille des Sprites : 1 pour 16x16 ; 0 pour 8x8.

M2 = Bit de mode graphique.

M1 = Bit de mode graphique.

IE0 = Autorise l'interruption qui se produit à chaque fin d'affichage de l'écran (avant-plan).

BL = Affichage de l'écran et lecture de la VRAM. 1 pour activer (valeur par défaut) ; 0 pour désactiver. (Les commandes du VDP travaillent un peu plus vite ainsi).

4/16k = Configuration de la VRAM. 1 pour 16Ko ; 0 pour 4Ko. Mettre à 1 si le VDP est un TMS-9918/28/29 !

Note : Le mode graphique est déterminé par les bits M1 à M5 de la manière suivante.

	M5	M4	M3	M2	M1	
SCREEN 0	0	0	0	0	1	40 colonnes, 2 couleurs (4 couleurs possible sur MSX2~)
SCREEN 0	0	1	0	0	1	80 colonnes, 2 couleurs (4 couleurs possible)
SCREEN 1	0	0	0	0	0	32 colonnes, 16 couleurs (2 couleurs par caractères)
SCREEN 2	0	0	1	0	0	256x192, 16 couleurs (2 couleurs chaque 8 pixels)
SCREEN 3	0	0	0	1	0	64x48, 16 couleurs (sans contrainte)
SCREEN 4	0	1	0	0	0	Equivalent au SCREEN 2 mais avec Sprites MSX2
SCREEN 5	0	1	1	0	0	256x212, 16 couleurs (sans contrainte)
SCREEN 6	1	0	0	0	0	512x212, 4 couleurs (sans contrainte)
SCREEN 7	1	0	1	0	0	512x212, 16 couleurs (sans contrainte)
SCREEN 8	1	1	1	0	0	256x212, 256 couleurs (sans contrainte)
SCREEN 9	Idem au SCREEN 6					40 colonnes en Hangeul, 4 couleurs (MSX2 coréen seulement)
SCREEN 10	Idem au SCREEN 8					256x212, 12599 couleurs (avec contrainte)
SCREEN 11	Idem au SCREEN 8					Idem au SCREEN 10
SCREEN 12	Idem au SCREEN 8					256x212, 19268 couleurs (avec contrainte)

- Le mode SCREEN 9 est en fait le même mode que le SCREEN 6 mais géré à la façon d'un mode texte par le Basic. Ce mode n'existe que sur certains MSX2 coréens.
- Le mode SCREEN 10 et 11 est obtenu en mettant les bits YJK et YAE du registre 25 à 1. La différence entre les deux se situe dans la gestion des couleurs sous Basic (RGB ou YJK).
- Le mode SCREEN 12 est obtenu en mettant le bits YJK à 1 et YAE à 0 (registre 25).

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 2 :	0	N16	N15	N14	N13	N12	N11	N10

Ce registre détermine l'emplacement de la table qui constitue l'avant-plan.

Modes d'écran MSX1 :

N10 à N13 = Ces bits correspondent aux quatre bits de poids fort de l'adresse du début de la table des caractères en VRAM. L'adresse véritable s'obtient en multipliant la valeur de ces 4 bits par 400h. Par exemple, si les bits sont 1001 (soit 9 en hexadécimal), la table des caractères se trouve en  $9 \times 400h = 2400h$ .

N14 à N16 = N'existent pas. L'adresse peut varier dans ces conditions entre 0 et 3C00h.

Autres modes d'écran :

N10 à N14 = Ces bits doivent toujours être à 1.

N15 et N16 = Ces bits déterminent le numéro de page (Bits de poids fort de la table bitmap).

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 3 :	C13	C12	C11	C10	C9	C8	C7	C6

Modes d'écran MSX1 :

C6 à C13 = Ces bits codent les 8 bits de poids fort de l'adresse du début de la table des couleurs. L'adresse réelle en mémoire s'obtient donc en multipliant le contenu du registre 3 par 40h. Par exemple si l'on lit 0B6h dans le registre 3, l'adresse de début de la table des couleurs est  $0B6h \times 40h = 2D80h$ . L'adresse peut varier entre 0 et 3FC0h.

Attention : En SCREEN 2, le fonctionnement est différent, la table des couleurs ne peut se trouver qu'en 0 ou 2000h. Seul le bit de poids fort intervient. Les autres bits sont tous à 1. Le registre 3 ne devra donc contenir que 7Fh ou FFh.

Autres modes d'écran :

C6 à C13 = Ces bits s'utilisent avec C14, C15 et C16 du registre 10 du VDP. Tous ces 11 bits codent l'adresse du début de la table des couleurs comme sur MSX1 (voir ci-dessus). L'adresse peut varier entre 0 et 1FFC0h.



	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 4 :	0	0	F16	F15	F14	F13	F12	F11

Modes d'écran MSX1 :

F11 à F13 = Ces bits codent les 3 bits de poids fort de l'adresse du début de la table des formes en VRAM. L'adresse véritable s'obtient donc en multipliant la valeur de ces 3 bits par 800h. Par exemple, si les bits sont 100 (soit 4 en hexa), la table des formes se trouve en  $4 \times 800h = 2000h$ .

F14 à F16 = N'existent pas. L'adresse peut varier dans ces conditions entre 0 et 3800h.

Attention : En SCREEN 2, le fonctionnement est différent. La table des formes ne peut commencer qu'en 0 ou 2000h. Le registre ne peut avoir que 03h ou 07h comme valeur. De surcroît, F14 doit toujours être positionné à l'inverse du bit 8 du registre 3. (Vous me suivez bien ?)

Autres modes d'écran :

F11 à F16 = Ces bits codent les 6 bits de poids fort de l'adresse du début de la table des formes. Le fonctionnement est le même que sur MSX1 (voir ci-dessus). L'adresse varie entre 0 et 1F800h.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 5 :	S14	S13	S12	S11	S10	S9	S8	S7

Modes d'écran MSX1 :

S7 à S13 = Ces bits codent les 7 bits de poids fort de l'adresse du début de la table des attributs de Sprites. L'adresse réelle s'obtient donc en multipliant le contenu du registre 5 par 80h. Par exemple si l'on lit 037h dans le registre 5, l'adresse du début de la table des attributs de Sprites est  $037h \times 80h = 1B80h$ .

S14 = Ignoré. L'adresse peut varier entre 0 et 3F80h.

Autres modes d'écran :

S7 à S14 = S'utilisent avec S15 et S16 du registre 11 du VDP. Ces 10 bits codent l'adresse du début de la table des attributs de Sprites comme sur MSX1 (voir ci-dessus). L'adresse peut varier entre 0 et 1FF80h. Attention, S7 et S8 sont ignorées et S9 doit être toujours à 1.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 6 :	0	0	P16	P15	P14	P13	P12	P11

Modes d'écran MSX1 :

P11 à P13 = Codent les 3 bits de poids fort de l'adresse du début de la table des formes de Sprite en VRAM. L'adresse véritable s'obtient en multipliant la valeur de ces 3 bits par 800h. Par conséquent, pour mettre la table des formes de Sprite à l'adresse 2000h, il faut diviser 2000h par 800h et écrire le résultat dans ce registre.

P14 à P16 = Ignorés. L'adresse peut varier dans ces conditions qu'entre 0 et 3800h.

Autres modes d'écran :

P11 à P16 = Codent les 6 bits de poids fort de l'adresse du début de la table de génération des Sprites. L'adresse véritable s'obtient en multipliant la valeur de ce registre par 800h. L'adresse varie entre 0 et 1F800h.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 7 :	TC3	TC2	TC1	TC0	BD3	BD2	BD1	BD0

BD0 à BD3 = Donnent la couleur du fond dans tous les mode d'écran.

TC0 à TC3 = Donnent la couleur du texte dans les modes textes.

En SCREEN 8 à 12, tous les bits définissent la couleur de font.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 8 :	MS	LP	TP	CB	VR	0	SPD	BW	(V9938 à V9958)

BW = 1 pour régler l'affichage en noir et blanc. Sinon l'affichage se fait en couleur.

SPD = 1 pour désactiver l'affichage des Sprites. Les commandes du VDP travaillent un peu plus vite lorsque les Sprites sont désactivés.

VR = Définit le type de mémoire vidéo (« Video Ram ») :

1 = 64Ko × 1 bit ou bien 64Ko × 4 bits ;

0 = 16 Ko × 1 bit ou bien 16 Ko × 4 bits.

CB = Direction du bus de couleur (« Color Bus ») du VDP. 1 pour entrée ; 0 pour sortie.

TP = 1 pour couleur 0 redéfinissable, par exemple, avec l'instruction COLOR= ( 0 , R , G , B ) . 0 pour couleur 0 « transparente ». Ceci permet d'afficher la vidéo en entrée à la place de la couleur n°0 sur les MSX dotés d'une entrée vidéo.

En Basic, on modifie l'état de TP par :

VDP ( 9 ) = VDP ( 9 ) OR &H20 pour mettre TP à 1

VDP ( 9 ) = VDP ( 9 ) AND &HDF pour mettre TP à 0

LP = 1 pour activer de mode crayon optique. (Utilisé uniquement les MSX2 coréens Daewoo CPC- 300 et CPC-400/400S, mettre à 0 autrement.) (N'existe pas sur le V9958.)

MS = 1 pour activer de mode souris. (Inutilisé sur MSX, mettre à 0.) (N'existe pas sur le V9958.)

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 9 :	LN	0	S1	S0	IL	E0	$\overline{\text{NT}}$	DC	(V9938 à V9958)

DC = 1 met la broche  $\overline{\text{DTCLK}}$  (horloge de base résolution) en mode entrée. 0 met en mode sortie. (Utilisé sur les MSX avec entrée vidéo.)

$\overline{\text{NT}}$  = 1 met l'affichage en PAL (313 lignes, 50 Hertz) autrement, affichage en NTSC (256 lignes, 60 Hertz). (Sortie RGB)

E0 = Dans un mode graphique de SCREEN 5 à 12, ce bit sert à activer l'affichage de 2 pages en alternance, 1 frame sur deux, pour la deuxième période. (Pour plus de précisions, voir le registre 13.)

IL = 1 pour affichage entrelacé de 2 pages. 0 pour affichage non-entrelacé.

S0 et S1 = Affichage. 0 = Normal ; 1 = Numérisation ; incrustation, etc ; 2 = Vidéo externe.

LN = 1 pour régler la hauteur de l'écran à 212 points, sinon 192 points.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 10 :	0	0	0	0	0	C16	C15	C14	(V9938 à V9958)

C14 à C16 = Détermine les 3 bits de poids fort de l'adresse du début de la table des couleurs (qui en comporte 17). Voir le registre 3 pour plus de précisions.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 11 :	0	0	0	0	0	0	S16	S15	(V9938 à V9958)

S15 et S16 = Détermine les 2 bits de poids fort de l'adresse du début de la table des attributs de Sprite (qui en comporte 17). Voir le registre 5 pour plus de précisions.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 12 :	T23	T22	T21	T20	BC3	BC2	BC1	BC0	(V9938 à V9958)

Couleur du texte utilisée pendant la seconde période lors d'un affichage périodique en mode texte 80 colonnes (SCREEN 0). Voir le registre 13 pour plus de précisions.

BC0 à BC3 = Couleur de fond du texte de la deuxième période.

T20 à T23 = Couleur du texte de la deuxième période.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 13 :	ON3	ON2	ON1	ON0	OF3	OF2	OF1	OF0	(V9938 à V9958)

OF0 à OF3 = Règle le temps de la seconde période.

ON0 à ON3 = Règle le temps de la première période et active l'affichage périodique lorsque un de ces bits est à 1.

Longueur d'une période en secondes pour une fréquence de 50 Hz :

0000 = 0,0	0100 = 0,8	1000 = 1,6	1100 = 2,4
0001 = 0,2	0101 = 1,0	1001 = 1,8	1101 = 2,6
0010 = 0,4	0110 = 1,2	1010 = 2,0	1110 = 2,8
0011 = 0,6	0111 = 1,4	1011 = 2,2	1111 = 3,0

En mode texte 80 colonnes, ce registre sert à paramétrer les temps des périodes du clignotement des couleurs de texte. La couleur de texte définie par le registre 13 s'affiche pendant la première période. La couleur actuelle s'affiche pendant la deuxième période.

Marche à suivre en mode texte (80 colonnes seulement) :

1. Charger le registre 7 avec les couleurs du texte pour la seconde période.
2. Charger le registre 12 avec les couleurs du texte pour la première période.
3. Modifier la table des couleurs en VRAM. Cette table occupe 240 octets (0800h à 08EFh par défaut), chaque bit correspond à caractère (24 lignes × 80 colonnes = 1920 caractères d'où 1920 bits, soit 240 octets). Si le bit est à 1, le caractère clignote avec les couleurs définies par les registres 7 et 12 alors que s'il se trouve à 0, le caractère ne clignote pas. Il garde la couleur du registre 7.
4. Régler le registre 13 avec les temps d'affichage adéquats.

Voici un exemple en Basic :

```
10 SCREEN 0: WIDTH80
20 COLOR 10,0,0: FOR I=&H800+239: VPOKE I,0: NEXT
30 VDP(13)=&H10:VDP(14)=&H44 ' registre 12 = 010H et reg. 13 = 044H
40 PRINT"Ce petit programme permet de faire clignoter automatiquement un
texte !": PRINT: PRINT: VPOKE &H804,&H1F: VPOKE &H805,&HF0
```

Après avoir lancé le programme ci-dessus, vous pouvez continuer à travailler. Essayez de changer le &H44 de la ligne 30 en &H11 et refaites RUN. Vous avez changé la vitesse dans le registre 13. Mettez maintenant &HBB à la place de &H11, en même temps, changez le &H10 en &H67. Le mot apparaît à présent en jaune sur noir (comme le reste) puis en rouge sur fond cyan. Vous avez agi sur le registre 12. Enfin, transformez le VPOKE &H805,&HF0 en VPOKE &H805,&HFF. A l'exécution, vous observerez que vous avez modifié des indicateurs et que de nouveaux caractères clignotent.

En mode graphique, ce registre sert à paramétrer un cycle de deux périodes d'affichage d'une page paire ou impaire. La page paire inférieure s'affiche pendant première période du cycle. La page impaire ou, l'alternance des deux pages si le bit E0 du registre 9 à 1, s'affiche pendant la seconde période.

Marche à suivre en mode graphique (SCREEN 5 à 12) :

Les pages graphiques vont par deux. En SCREEN 5 et 6, on peut choisir de faire afficher en alternance les pages 0 et 1 ou alors 2 et 3 mais jamais 0 et 3, 1 et 3 ou 0 et 2.

1. Paramétrer les bits N10 à N14 du registre 2, afin d'afficher une page paire.
2. Régler les temps d'affichages respectifs et activer l'affichage périodique grâce au registre 13.
3. Mettre le bit E0 du registre 9 à 1 pour activer l'alternance de 2 page sur la seconde période.

Exemple d'utilisation en Basic :

```
10 COLOR 10,0,0:SCREEN 5
20 ON STOP GOSUB 190:STOP ON
30 PI=3.141592654#
40 CIRCLE(100,100),50,7,PI/2,2*PI
50 LINE(50,100)-(150,100),7:LINE(100,50)-(100,150),7:PAINT
(98,98),10,7:PAINT(102,102),6,7:PAINT(98,102),12,7
60 SET PAGE 1,1:CLS
70 CIRCLE(100,100),50,7,PI/4,2*PI-PI/4:LINE(65,65)-(135,135),7:LINE(65,135)-
(135,65),7:PAINT(100,98),10,7:PAINT(100,102),6,7:PAINT(98,100),12,7
80 'VDP(2)=&H3F
90 A$=INKEY$
100 IF A$="1" THEN VDP(14)=&H33:VDP(10)=VDP(10) OR 4
110 IF A$="2" THEN VDP(14)=&H3:VDP(10)=VDP(10) OR 4
120 IF A$="3" THEN VDP(14)=&H30:VDP(10)=VDP(10) OR 4
130 IF A$="4" THEN VDP(14)=&H0:VDP(10)=VDP(10) OR 4
140 IF A$="5" THEN VDP(14)=&H33:VDP(10)=VDP(10) AND 251
150 IF A$="6" THEN VDP(14)=&H3:VDP(10)=VDP(10) AND 251
160 IF A$="7" THEN VDP(14)=&H30:VDP(10)=VDP(10) AND 251
170 IF A$="8" THEN VDP(14)=&H0:VDP(10)=VDP(10) AND 251
180 GOTO 90
190 STOP OFF:VDP(14)=0:VDP(10)=VDP(10) AND 251:END
```

Les lignes 30 à 70 se chargent de faire un joli dessin sur la page 0 et la 1. Les lignes 20 et 190 désactive l'affichage périodique et alterné automatique.

Lorsqu'on presse une touche de 1 à 8, voici ce qui se passe :

- 1 = La page 0 s'affiche pendant la première période puis les pages 1 et 0 s'affichent une frame sur deux pendant la deuxième période.
- 2 = Les pages 1 et 0 s'affichent une frame sur deux continuellement. L'affichage périodique est désactivé.
- 3 = La page 0 s'affiche pendant la première période. Il n'y a pas de deuxième période donc la page 0 reste toujours affichée.
- 4 = Les pages 1 et 0 s'affichent une frame sur deux continuellement. L'affichage périodiques est désactivé.
- 5 = Affiche la page 0 pendant la première période puis affiche la page 1 pendant la deuxième période.
- 6 = Affiche la page 1. L'affichage périodique est désactivé.
- 7 = Affiche la page 0 pendant la première période. Il n'y a pas de deuxième période donc la page 0 reste toujours affichée.
- 8 = Affiche la page 1. L'affichage périodique est désactivé. (paramètres par défaut)

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 14 :	0	0	0	0	0	V16	V15	V14	(V9938 à V9958)

V14 à V16 = Ce registre permet d'accéder aux adresses supérieures à 03FFFh (16Ko) de la VRAM. Ces bits sont les 3 bits de poids fort de l'adresse de la VRAM à laquelle on désire accéder. Voir le paragraphe 7.5 « [Lecture et écriture dans la mémoire vidéo](#) », page 181.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 15 :	0	0	0	0	ST3	ST2	ST1	ST0	(V9938 à V9958)

ST0 à ST3 = Indiquent le numéro de registre de statut que l'on désire lire (0 ~ 9). Voir le paragraphe 7.4 « Comment accéder aux registres du VDP » rubrique « [Lire un registre de statut \(STATUS REGISTER\)](#) » page 180 pour plus de précisions.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 16 :	0	0	0	0	CC3	CC2	CC1	CC0	(V9938 à V9958)

CC0 à CC3 = Spécifie le numéro de la couleur (0 ~ 15) à laquelle la palette sera modifiée en envoyant des données au port d'entrée/sortie 09Ah. Les données sont envoyées par pair au format ci-dessous.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Port 09Ah :	0	R2	R1	R0	0	B2	B1	B0	Première écriture
Port 09Ah :	0	0	0	0	0	V2	V1	V0	Seconde écriture

Une fois les deux données du RVB envoyées, le numéro de la couleur est incrémenté.

Voir le paragraphe « Comment accéder au VDP » rubrique « [Ecrire dans un registre de la palette des couleurs](#) », page 179, pour plus de précisions.

Valeur de la palette de chaque couleur par défaut :

Numéro de couleur	R	V	B
0	0	0	0
1	0	0	0
2	1	6	1
3	3	7	3
4	1	1	7
5	2	3	7
6	5	1	1
7	2	6	7

Numéro de couleur	R	V	B
8	7	1	1
9	7	3	3
10	6	6	1
11	6	6	4
12	1	4	1
13	6	2	5
14	5	5	5
15	7	7	7

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 17 :	AII	0	RS5	RS4	RS3	RS2	RS1	RS0	(V9938 à V9958)

Ce registre est utilisé lors d'un accès indirect à un autre registre du VDP.

RS0 à RS5 = Numéro de registre désiré du VDP.

AII = Mode auto-incrémentation OFF.

La donnée du registre spécifié s'écrit sur le port d'entrée/sortie 9Bh. En mode auto-incrémentation (AII à 0), chaque donnée écrite sur le port d'entrée/sortie 9Bh provoque une incrémentation du numéro de registre contenu dans le registre 17. Cela permet d'envoyer des données dans une série de registre.

Voir le paragraphe 7.4 « Comment accéder au VDP » rubrique « [Ecrire dans un registre de contrôle](#) », page 177, pour plus de précisions.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 18 :	V3	V2	V1	V0	H3	H2	H1	H0	(V9938 à V9958)

H0 à H3 = Ajustement horizontal de l'affichage de l'écran.

7 - 6 - 5 -	...	- 1 - 0 - 15 - 14 -	...	- 10 - 9 - 8
gauche		centre		droite

V0 à V3 = Ajustement vertical de l'affichage de l'écran.

8 - 7 - 6 -	...	- 1 - 0 - 15 - 14 -	...	- 9 - 8 - 7
bas		centre		haut

Ce registre est équivalent au SET ADJUST du Basic mais sans sauvegarde du paramètre dans la SRAM.

Note : Veuillez attendre que le bit CE du registre 2 de lecture soit à 0 avant d'accéder à ce registre lorsque l'affichage de l'écran ou des Sprites est désactivé.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 19 :	IL7	IL6	IL5	IL4	IL3	IL2	IL1	IL0	(V9938 à V9958)

IL0 à IL7 = Défini le numéro de ligne où une interruption programmée doit se produire lorsque le bit 4 (IE1) du registre 0 est à 1.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 20 :	0	0	0	0	0	0	0	0	(V9938)
Registre 21 :	0	0	1	1	1	0	1	1	(V9938)
Registre 22 :	0	0	0	0	0	1	0	1	(V9938)

La valeur de ces 3 registres ne doivent pas être modifié. Elles indiquent la fréquence du signal « Colorburst » prévu pour une sortie vidéo composite. Exception à la règle, on peut les mettre à 0 pour couper la sortie composite. Il suffit de remettre les valeurs initiales pour rétablir la sortie composite.

Note : Beaucoup de MSX2 n'utilisent pas ce signal car ils utilisent un dispositif externe pour faire la sortie composite à partir du signal RGB. Le V9958 n'a pas de signal « Colorburst ».

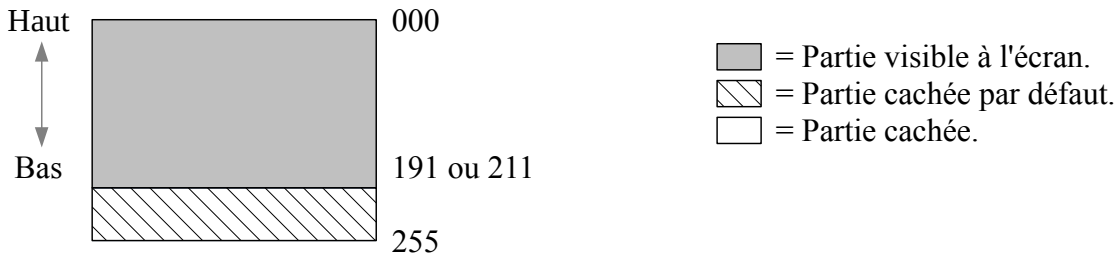
bit 7 bit 6 bit 5 bit 4 bit 3 bit 2 bit 1 bit 0  
 Registre 23 : 

DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
-----	-----	-----	-----	-----	-----	-----	-----

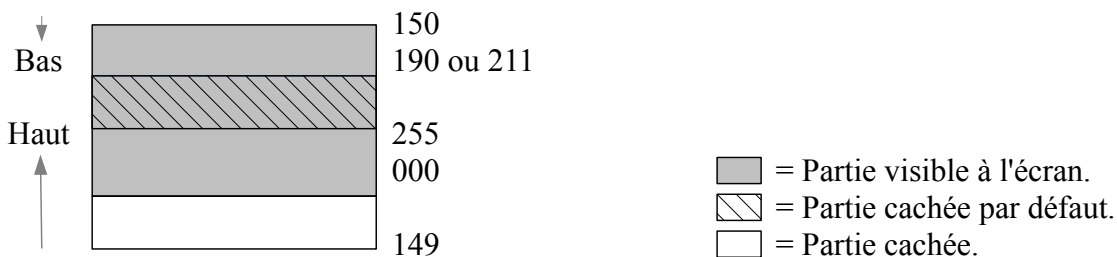
 (V9938 à V9958)

D0 à D7 = Défini le décalage vertical de l'avant-plan.

Affichage d'une page sans décalage :



Exemple avec 150 dans le registre 23 :



Note : Ce registre permet de réaliser très facilement des scrollings verticaux. Voici un petit programme Basic pour illustrer cette facilité :

```
10 SCREEN 8
20 CIRCLE(100,100),50,200
30 FOR I=0 TO 50
40 VDP(24)=I ' (équivalent à C=PEEK(7)+1: OUT C,I: OUT C,&H80+23)
50 NEXT
60 FOR I=50 TO 0 STEP -1
70 VDP(24)=I
80 NEXT
90 GOTO 30
```

L'équivalent en assembleur serait :

```
EXTROM equ 0015Fh
CHGMOD equ 000D1h
CHGCLR equ 00062h
CLS equ 000C3h
NVBXLN equ 000C9h
BREAKX equ 000B7h
TOTEXT equ 000D2h
;
FORCLR equ 0F3E9h
BAKCLR equ 0F3EAh
BDRCLR equ 0F3EBh
GXPOS equ 0FCB3h
GYPOS equ 0FCB5h
ATRBYT equ 0F3F2h
LOGOPR equ 0FB02h
```



```

        org    0c000h
DEBUT:
        ld     a,8
        ld     ix,CHGMOD
        call   EXTROM      ; SCREEN 8
;
        ld     a,0ah
        ld     (FORCLR),a
        ld     a,0
        ld     (BAKCLR),a
        ld     (BDRCLR),a
        call   CLS         ; Efface l'écran
;
        ld     bc,060h
        ld     de,060h
        ld     hl,090h
        ld     (GXPOS),hl
        ld     (GYPOS),hl
        ld     a,0fch
        ld     (ATRBYT),a
        ld     (LOGOPR),a
        ld     ix,NVBXLN
        call   EXTROM      ; Trace un rectangle
;
        ld     a,(7)       ; C = Numéro du port E/S correspondant
        ld     c,a         ; au port 1 d'écriture du VDP.
        inc    c           ; Port E/S du port 1 d'écriture
;
; Boucle Principale
;
LOOP:   call   UP
        call   BREAKX      ; Test du CTRL+STOP
        jr     c,EXIT
        call   DOWN
        call   BREAKX      ; Test du CTRL+STOP
        jr     c,EXIT      ; Sortie du programme si CTRL+STOP
        jp     LOOP
;
EXIT:   call   TOTEXT      ; Retour au mode texte
        ret
;
UP:     ld     b,032h
LOOP1:  ld     a,032h
        sub    b
        call   VDP
        call   WAIT
        djnz   LOOP1      ; saut à LOOP1: tant que B n'est pas à 0
        ret
;
DOWN:   ld     b,032h
LOOP2:  ld     a,b
        call   VDP
        call   WAIT
        djnz   LOOP2      ; saut à LOOP2: tant que B n'est pas à 0
        ret
;
VDP:    di                     ; Désactive les interruptions
        out    (c),a
        ld     a,080H+23
        out    (c),a

```

```

ei                ; Active les interruptions
ret
;
WAIT: ld  hl,002FFh
HEP:  dec hl
      ld  a,h
      or  l
      jr  nz,HEP    ; saut à HEP: tant que HL n'est pas à 0
      ret

```

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 25 :	0	CMD	VDS	YAE	YJK	WTE	MSK	SP2	(V9958)

SP2 = 0 pour scrolling horizontal de l'écran sur une page (valeur par défaut),

1 pour scrolling horizontal de l'écran sur deux page.

MSK = Active un masque de 8 ou 16 lignes sur la partie gauche de l'écran.

WTE = 0 désactive la fonction « Wait » (le VDP fonctionne comme un V9938),

1 active la fonction « Wait ». Met en attente le CPU lorsqu'un accès du CPU à la VRAM est effectué. (Inutilisé sur MSX.)

YJK = 1 pour configurer la table Bitmap du SCREEN 8 au format YJK au lieu de RGB. Cela permet d'afficher jusqu'à 19268 couleurs à l'écran au lieu de 256. Les signaux de sortie sont tout de même convertis en RGB (sur 5 bits) et affiché en analogique. La palette des couleurs d'affichage du Sprite reste au format RGB.

Format de la table des formes en mode YJK pour chaque ligne de 4 pixels :

	bit 7	bit 6	bit 5	bit 4	bit 2	bit 2	bit 1	bit 0
Premier octet :	Y <sub>1</sub>					bits de poids faible de K		
Second octet :	Y <sub>2</sub>					bits de poids fort de K		
Troisième octet :	Y <sub>3</sub>					bits de poids faible de J		
Quatrième octet :	Y <sub>4</sub>					bits de poids fort de J		

La couleur du premier octet est codée par Y<sub>1</sub>, J et K. Le second par Y<sub>2</sub>, J et K, le troisième par Y<sub>3</sub>, J et K puis le quatrième par Y<sub>4</sub>, J et K. Et ainsi de suite...

Formules de conversion :

$$R=Y+J$$

$$Y=(2R+G+4B)/8$$

$$G=Y+K$$

$$J=(6R-G+4B)/8$$

$$B=1,25Y-0,5J-0,25K$$

$$K=(-2R+7G-4B)/8$$

YAE = 1 configure la table Bitmap en mode YAE. (Mettre aussi YJK à 1.) Cela donne le mode SCREEN 10/11. C'est à dire qu'un bit attribue à chaque pixel le mode dans lequel il doit s'afficher, RVB ou YJK (aux mêmes caractéristiques que le SCREEN 5).

Format de la table des formes en mode YAE pour chaque ligne de 4 pixels :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Premier octet :	Y <sub>1</sub>				A <sub>1</sub>	bits de poids faible de K		
Second octet :	Y <sub>2</sub>				A <sub>2</sub>	bits de poids fort de K		
Troisième octet :	Y <sub>3</sub>				A <sub>3</sub>	bits de poids faible de J		
Quatrième octet :	Y <sub>4</sub>				A <sub>4</sub>	bits de poids fort de J		

La couleur du premier pixel est codée par Y<sub>1</sub>, J et K. Le second par Y<sub>2</sub>, J et K, le troisième par Y<sub>3</sub>, J et K puis le quatrième par Y<sub>4</sub>, J et K. Et ainsi de suite... Lorsque l'attribut d'un octet est à 1, J et K sont ignorés. Y<sub>1</sub> ~ Y<sub>4</sub> devient le numéro de couleur comme en SCREEN 5.

VDS = Ce bit détermine la fonction de la broche 8 du VDP. Le bit VDS est à zéro, il s'agit d'un signal d'horloge à 3,579545 MHz pour le Z80A. Sinon, il s'agit du signal  $\overline{\text{VDS}}$  en sortie. (Laisser ce bit à 0)

CMD = Ce bit permet d'utiliser les commandes du VDP dans tous les modes d'écran. Pour les modes SCREEN 0 à 4, les coordonnées X et Y fonctionnent comme en SCREEN 8. (Ce bit est à 0 par défaut.)

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 26 :	0	0	HO8	HO7	HO6	HO5	HO4	HO3	(V9958)
Registre 27 :	0	0	0	0	0	HO2	HO1	HO0	(V9958)

HO0 à HO8 = Numéro de ligne verticale qui se trouvera le plus à gauche (le bit HO8 n'est utile que si SP2 du registre 25 est à 1. En screen 6 et 7, le défilement se fera de 2 en 2 lignes)

Les quinze derniers registres du VDP sont utilisés pour l'exécution des commandes graphiques. Voir le paragraphe 7.8 « [Les commandes internes du VDP](#) », page 200, pour plus de précisions.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	(V9938 à V9958)
Registre 33 :	0	0	0	0	0	0	0	SX8	(V9938 à V9958)

SX0 à SX8 = Codent l'abscisse du point d'origine ( $0 < X < 511$ )

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	(V9938 à V9958)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	(V9938 à V9958)

SY0 à SY9 = Codent l'ordonnée du point d'origine ( $0 < Y < 1023$ )

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	(V9938 à V9958)
Registre 37 :	0	0	0	0	0	0	0	DX8	(V9938 à V9958)

DX0 à DX8 = Abscisse du premier pixel de destination ( $0 < X < 511$ )

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	(V9938 à V9958)
Registre 39 :	0	0	0	0	0	0	DY9	DY8	(V9938 à V9958)

DY0 à DY9 = Codent l'ordonnée du point de destination ( $0 < Y < 1023$ )

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	(V9938 à V9958)
Registre 41 :	0	0	0	0	0	0	0	NX8	(V9938 à V9958)

NX0 à NX8 = Déplacement horizontal ( $0 < NX < 511$ ).

Note : Ce registre se comporte différemment avec la commande LINE.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	(V9938 à V9958)
Registre 43 :	0	0	0	0	0	0	NY9	NY8	(V9938 à V9958)

NY0 à NY9 = Déplacement vertical ( $0 < NY < 1023$ ).

Note : Ce registre se comporte différemment avec la commande LINE.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 44 :	CL7	CL6	CL5	CL4	CL3	CL2	CL1	CL0	(V9938 à V9958)

CL0 à CL7 = Codent la couleur.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 45 :	0	<del>MXC</del>	MXD	MXS	DIY	DIX	EQ	MAJ	(V9938 à V9958)

Registre de données pour les commandes du VDP. Nous verrons ces bits en détail dans le chapitre 6.8 « Les commandes du VDP ».

MAJ = Déplacement le plus long. 0 pour horizontal ; 1 pour vertical. (Utilisé uniquement par la commande LINE.)

EQ = 0 pour stopper la recherche lorsque la couleur est trouvée ; 1 pour stopper la recherche lorsque la couleur est différente de celle paramétrée.

DIX = Lorsque ce bit est à 1, la valeur de déplacement horizontal (registre 40 et 41) est considérée négative. Le déplacement se fait dans l'autre sens.

DIY = Lorsque ce bit est à 1, la valeur de déplacement vertical (registre 42 et 43) est considérée négative. Le déplacement se fait dans l'autre sens.

MXS = Mémoire vidéo source. 0 pour VRAM principale ; 1 pour VRAM étendue (pour les MSX avec 192 Ko de VRAM).

MXD = Mémoire vidéo de destination. 0 pour VRAM principale ; 1 pour VRAM étendue (pour les MSX avec 192 Ko de VRAM).

MXC = Commutation RAM/VRAM. 1 pour extension de RAM ; 0 pour VRAM. Inutilisé sur MSX, mettre à 0.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 46 :	CM3	CM2	CM1	CM0	LO3	LO2	LO1	LO0	(V9938 à V9958)

LO0 à LO3 = Définissent le code de l'opérateur logique à utiliser.

CM0 à CM3 = Définissent la commande du VDP à exécuter.

## 7.7 Les registres de statut du VDP

Voici la liste complète de tous les registres de statut, qui ne sont accessibles qu'en lecture pour l'utilisateur, avec la signification des informations qu'ils contiennent.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 0 :	F	5S	C	5/9SN				

5/9SN = Ces bits donnent le numéro du 5ème Sprite (ou du 9ème dans les SCREEN 4 à 12) lorsque le bit 5S se trouve à 1.

C = Ce bit passe à 1 lorsque deux Sprites entrent en collision.

5S = Indicateur qui passe à 1 lorsque 5 Sprites (9 dans les SCREEN 4 à 12) se trouvent sur une même ligne.

F = Indicateur de l'interruption qui se produit à chaque fin d'affichage de l'écran (de l'avant-plan). Ce bit repasse à 0 chaque fois que le registre est lu ou que le VDP est initialisé (de façon externe).

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 1 :	FL	LPS	ID				FH	(V9938 à V9958)

FH = Indicateur de l'interruption qui se produit avant le balayage de la ligne horizontale indiquée au registre 19.

ID = Ces 5 bits donnent le numéro d'identification du processeur vidéo. 00000 pour V9938 ; 00010 pour V9958.

LPS = Indicateur du bouton du crayon optique ou du bouton gauche de la souris. Ce bit passe à 1 lorsque le bouton est pressé. (Utilisé uniquement sur les MSX2 coréens Daewoo CPC- 300 et CPC-400/400S .) (N'existe pas sur le V9958.)

FL = Indicateur du crayon optique ou du bouton gauche de la souris. Pour que ce dernier fonctionne, il faut que le bit IE2 soit à 1. Ce bit passe à 1 lorsque le crayon optique détecte de la lumière ou bien lorsque le bouton est pressé. Ce bit est automatiquement remis à 0 quand on lit le registre 1. (Utilisé uniquement sur les MSX2 coréens Daewoo CPC-400 et CPC400S.) (N'existe pas sur le V9958.)

Note : pour plus de renseignements sur le fonctionnement du crayon optique et de la souris, voir le paragraphe 7.11 « [A propos de la souris et du crayon optique](#) », page 255.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 2 :	TR	VR	HR	BD	1	1	EO	CE	(V9938 à V9958)

CE = Ce bit (Command Execution) est à 1, lorsque le VDP est en train d'exécuter une commande interne.

EO = Indicateur de période. 0 pour la première période; 1 pour la seconde période.

BD = Ce bit (Bordure Detect) passe à 1 lorsque la commande SRCH a trouvée la couleur de bordure.

HR = Ce bit (Horizontal Retrace) est à 1 pendant le balayage de la zone non visible de gauche et de droite de l'écran (HBLANK).

VR = Ce bit (Vertical Retrace) est à 1 pendant le balayage de la zone non visible supérieure et inférieure de l'écran (VBLANK).

TR = Ce bit (Transfer ready) est à 1 tant que le transfert entre le CPU et la VRAM n'est pas fini (commande HMMC, LMMC ou LMCM).

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 3 :	X7	X6	X5	X4	X3	X2	X1	X0	(V9938 à V9958)
Registre 4 :	1	1	1	1	1	1	1	X8	(V9938 à V9958)

X0 à X8 = Indiquent l'abscisse du point de collision des Sprites.

(Sur le V9938, ces bits peuvent indiquer aussi l'abscisse du point où pointe le crayon optique ou bien, le déplacement horizontal relatif de la souris mais ces modes ne sont pas utilisés sur MSX.)

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 5 :	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	(V9938 à V9958)
Registre 6 :	1	1	1	1	1	1	Y9	Y8	(V9938 à V9958)

Y0 à Y9 = Indiquent l'ordonnée du point de collision des Sprites.

(Sur le V9938, ces bits peuvent indiquer aussi l'ordonnée du point où pointe le crayon optique ou bien, le déplacement vertical relatif de la souris mais ces modes ne sont pas utilisés sur MSX.)

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 7 :	C7	C6	C5	C4	C3	C2	C1	C0	(V9938 à V9958)

C0 à C7 = Utilisés par les commandes POINT et LMCM. Contient l'octet lu en VRAM.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 8 :	BX7	BX6	BX5	BX4	BX3	BX2	BX1	BX0	(V9938 à V9958)
Registre 9 :	1	1	1	1	1	1	1	BX8	(V9938 à V9958)

BX0 à BX8 = Indiquent l'abscisse de la couleur de bordure trouvée lors d'une recherche avec la commande SRCH du VDP.

## 7.8 Les commandes internes du V9938 et V9958.

Ces VDP disposent d'un jeu de commandes qui permettent de copier une zone de mémoire vers une autre, de tracer quelques formes simples, etc dans les graphiques SCREEN 5 à 12. Dans cette partie, nous allons détailler ces commandes une à une, et voir comment on les met en œuvre.

Il y a en tout 12 commandes distinctes plus une commande d'arrêt (« STOP »).

Voici un tableau récapitulatif :

Mnémonique	Code	Source	Destination	Signification
HMMC	0F0h	Z80	VRAM	High speed Move to Memory from CPU
YMMM	0E0h	VRAM	VRAM	Y Move to Memory from Memory
HMMM	0D0h	VRAM	VRAM	High speed Move to memory from memory
HMMV	0C0h	VDP	VRAM	High speed Move to Memory from VDP
LMMC	0Bxh	Z80	VRAM	Logical Move to Memory from CPU
LMCM	0Axxh	VRAM	Z80	Logical Move to CPU from Memory
LMMM	09xxh	VRAM	VRAM	Logical Move to Memory from memory
LMMV	08xxh	VDP	VRAM	Logical Move to Memory from VDP
LINE	07xxh	VDP	VRAM	LINE
SRCH	060h	VDP	VRAM	SeaRCH a border color
PSET	05xxh	VDP	VRAM	Point SET
POINT	040h	VRAM	VDP	Is POINT set ?
STOP	00h	-	-	STOP

Note : Il est nécessaire de vérifier que le bit CE du registre de statut 2 soit à 0 avant d'exécuter une de ces commandes.



A partir du SCREEN 5, le VDP travaille uniquement en coordonnées X et Y. Les notions d'adresse mémoire et de page n'existent pas. Tout se passe comme si le VDP opérait sur un écran géant de 256x1024 (ou 512x1024). La partie visible ne serait alors qu'une fenêtre suivant le schéma :

SCREEN 5		VRAM	SCREEN 6	
0, 0	255, 0	00000H	0, 0	511, 0
page 0			page 0	
0, 255	255, 255	07FFFH	0, 255	511, 255
0, 256	255,256	08000H	0, 256	511, 256
page 1			page 1	
0, 511	255,511	0FFFFH	0, 511	511, 511
0, 512	255,512	10000H	0, 512	511, 512
page 2			page 2	
0, 767	255, 767	17FFFH	0, 767	511, 767
0, 768	255, 768	18000H	0, 768	511, 768
page 3			page 3	
0, 1023	255, 1023	1FFFFH	0, 1023	511, 1023

SCREEN 7		VRAM	SCREEN 8 à 12	
0, 0	511, 0	00000H	0, 0	255, 0
page 0			page 0	
0, 255	511, 255	0FFFFH	0, 255	255, 255
0, 256	511, 256	10000H	0, 256	255,256
page 1			page 1	
0, 511	511, 511	1FFFFH	0, 511	255,511

Note : La VRAM est configurée physiquement de façon à ce que les octets paires contiennent les pages paires et les octets impaires les pages impaires. Il faut en tenir compte si vous devez changer de mode sans initialiser la VRAM ou lors de l'utilisation de la VRAM étendue. C'est la raison qui fait qu'un MSX2 n'ayant que 64Ko de VRAM ne peut pas afficher d'écran en mode SCREEN 7 ni 8.

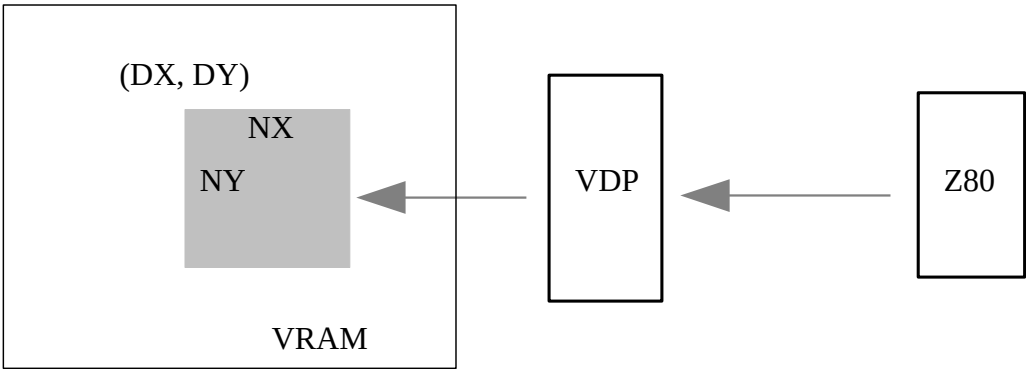
Il est possible d'exécuter certaines commandes en utilisant un opérateur logique qui s'effectue avec les pixels affichées sur la zone à tracer grâce aux bits LO0 à LO3 du registre 45.

Opérateur	Code	Fonction
IMP	0000	La couleur affichée sera remplacée par la nouvelle.
AND	0001	Couleur résultante = (couleur affichée) ET (nouvelle couleur).
OR	0010	Couleur résultante = (couleur affichée) OU (nouvelle couleur).
XOR	0011	Couleur résultante = (couleur affichée) XOR (nouvelle couleur).
NOT	0100	La couleur affichée sera remplacée par la nouvelle.

Opérateur	Code	Fonction
TIMP	1000	Idem à IMP mais si la couleur source est 0, celui-ci restera 0.
TAND	1001	Idem à AND mais si la couleur source est 0, celui-ci restera 0.
TOR	1010	Idem à OR mais si la couleur source est 0, celui-ci restera 0.
TXOR	1011	Idem à XOR mais si la couleur source est 0, celui-ci restera 0.
TNOT	1100	Idem à NOT mais si la couleur source est 0, celui-ci restera 0.

**HMMC (High speed Move to Memory from CPU)**

Schéma :



Fonction : HMMC permet de remplir une zone rectangulaire à l'écran avec des données provenant du Z80.

Source : Z80.

Destination : VRAM.

Voici la marche à suivre pour exécuter la commande HMMC :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination
Registre 37 :	0	0	0	0	0	0	0	DX8	(0 à 511)
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination
Registre 39 :	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DX0 à DX8 = Abscisse du premier pixel de destination.  
En SCREEN 5 et 7, le bit DX0 est ignoré.  
En SCREEN 6, les bits DX0 et DX1 sont ignorés.  
DY0 à DY9 = Ordonnée du premier pixel de destination.

Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
Registre 41 :	0	0	0	0	0	0	0	NX8	(0 à 511)
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
Registre 43 :	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NX0 à NX8 = Nombre de pixels à placer dans la direction horizontale (largeur).  
NY0 à NY9 = Nombre de pixels à placer dans la direction verticale. (hauteur)

Registre 44 :	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée
---------------	-----	-----	-----	-----	-----	-----	-----	-----	--------

SCREEN 5 et 7 :

CR0 à CR3 = Couleur du pixel d'abscisse impair.

CR4 à CR7 = Couleur du pixel d'abscisse pair.

SCREEN 6 :

CR0 et CR1 = Couleur du troisième pixel à droite de celui à l'abscisse multiple de 4.

CR2 et CR3 = Couleur du deuxième pixel à droite de celui à l'abscisse multiple de 4.

CR4 et CR5 = Couleur du pixel à droite de celui dont l'abscisse est multiple de 4.

CR6 et CR7 = Couleur du pixel dont l'abscisse est multiple de 4.

SCREEN 8 à 12 :

CR0 à CR7 = Couleur du pixel.

Registre 45 : 

0	-	MXD	-	DIY	DIX	-	-
---	---	-----	---	-----	-----	---	---

 paramètres

DIX = Direction horizontale pour NX. 0 pour droite ; 1 pour gauche.

DIY = Direction verticale pour NY. 0 pour bas ; 1 pour haut.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

2. Mettre le registre 46 à 11110000B (0F0h) pour exécuter la commande.
3. Lire le registre de statut 2, si le bit CE est à 0, alors le processus est terminé. Sinon, tester l'état du bit TR, si celui-ci est à 0, alors le processeur vidéo n'est pas prêt à recevoir l'octet suivant, recommencer l'étape 3. Si, par contre, ce bit est à 1, continuer en 4.
4. Envoyer l'octet suivant à mettre en VRAM dans le registre 45 et reprendre toute l'opération à l'étape 3.

Une fois la commande terminée, les registres de commande se trouveront dans l'état suivant :

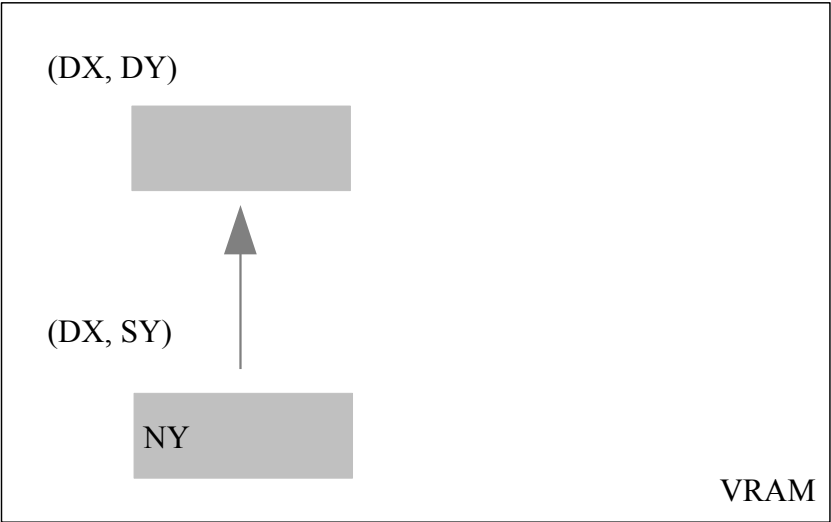
32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	*	idem	**	idem	idem

(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.

**YMMM (Y Move to Memory from Memory)**

Schéma :



Fonction : YMMM autorise la copie d'octets d'une zone de mémoire vidéo à une autre, le point d'origine de cette dernière ayant la même abscisse.

Source : VRAM

Destination : VRAM

Voici la marche à suivre pour exécuter la commande YMMM :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SY0 à SY9 = Ordonnée du pixel d'origine.  
En SCREEN 5 et 7, SX0 est ignoré.  
En SCREEN 6, SX0 et SX1 sont ignorés.

Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination (0 à 511)
Registre 37 :	0	0	0	0	0	0	0	DX8	
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination (0 à 1023)
Registre 39 :	0	0	0	0	0	0	DY9	DY8	

DX0 à DX8 = Abscisse du premier pixel de destination.  
En SCREEN 5 et 7, DX0 est ignoré.  
En SCREEN 6, DX0 et DX1 sont ignorés.  
DY0 à DY9 = Ordonnée du pixel de destination.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
Registre 43 :	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 = Nombre de pixels à copier dans la direction verticale.

Registre 45 :	0	-	MXD	-	DIY	DIX	-	-	paramètres
---------------	---	---	-----	---	-----	-----	---	---	------------

DIX = Direction horizontale pour NX. 0 pour droite ; 1 pour gauche.

DIY = Direction verticale pour NY. 0 pour bas ; 1 pour haut.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

2. Mettre 11100000B (0E0h) dans le registre 46 pour exécuter la commande.
3. Le bit CE du registre de statut 2 restera à 1 tant que le processus n'est pas terminé. Vérifiez que ce bit soit à 0 avant d'exécuter la commande suivante ou de changer de paramètre.

Une fois la commande terminée, les registres de commande se trouveront dans l'état suivant :

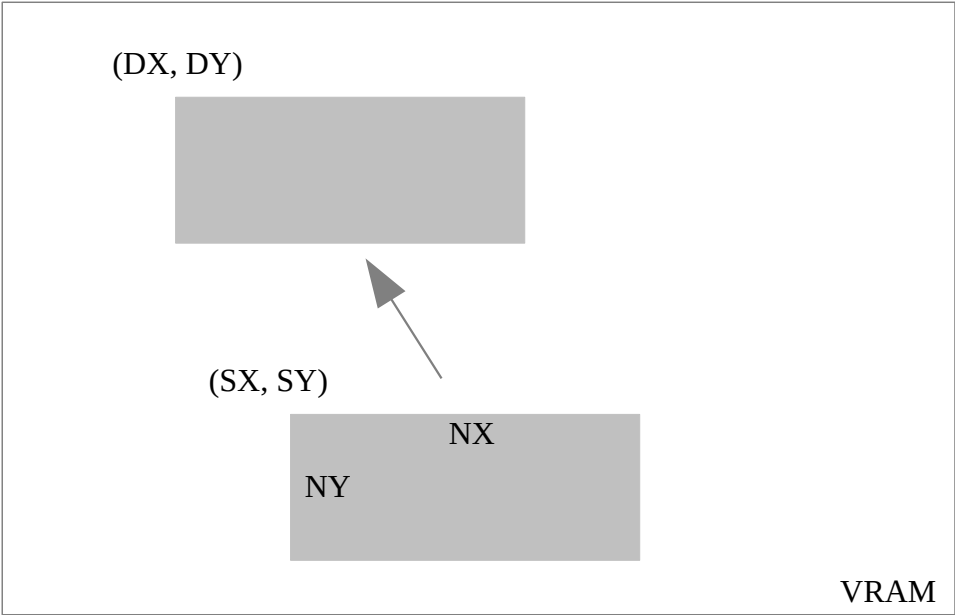
32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	*	idem	*	idem	**	idem	idem

(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.

**HMMM (High Speed Move to Memory from Memory)**

Schéma :



Fonction : HMMM autorise la copie rapide d'octets d'une zone de mémoire vidéo à une autre de taille équivalente.

Source : VRAM

Destination : VRAM

Voici la marche à suivre pour exécuter la commande HMMM :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source (0 à 256/511)
Registre 33 :	0	0	0	0	0	0	0	SX8	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SX0 à SX8 = Abscisse du pixel d'origine.

En SCREEN 5 et 7, SX0 est ignoré.

En SCREEN 6, SX0 et SX1 sont ignorés.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination (0 à 511)
Registre 37 :	0	0	0	0	0	0	0	DX8	
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination (0 à 1023)
Registre 39 :	0	0	0	0	0	0	DY9	DY8	

DX0 à DX8 = Abscisse du premier pixel de destination.

En SCREEN 5 et 7, DX0 est ignoré.

En SCREEN 6, NX0 et DX1 sont ignorés.

DY0 à DY9 = Ordonnée du pixel de destination.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal (0 à 511)
Registre 41 :	0	0	0	0	0	0	0	NX8	
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	ordonnée destination (0 à 1023)
Registre 43 :	0	0	0	0	0	0	NY9	NY8	

NX0 à NX8 = Nombre de pixels à copier dans la direction horizontale.

NY0 à NY9 = Nombre de pixels à copier dans la direction verticale.

Registre 45 :	0	-	MXD	MXS	DIY	DIX	-	-	paramètres
---------------	---	---	-----	-----	-----	-----	---	---	------------

DIX = Direction horizontale pour NX. 0 pour droite ; 1 pour gauche.

DIY = Direction verticale pour NY. 0 pour bas ; 1 pour haut.

MXS = Mémoire vidéo source. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

- Écrire 11010000B (0D0h) dans le registre 46 pour exécuter la commande.
- Le bit CE du registre de statut 2 restera à 1 tant que le processus n'est pas terminé. Vérifiez que ce bit soit à 0 avant d'exécuter la commande suivante ou de changer de paramètre.

Une fois la commande terminée, les registres de commande du VDP se trouveront dans l'état suivant :

32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	*	idem	*	idem	**	idem	idem

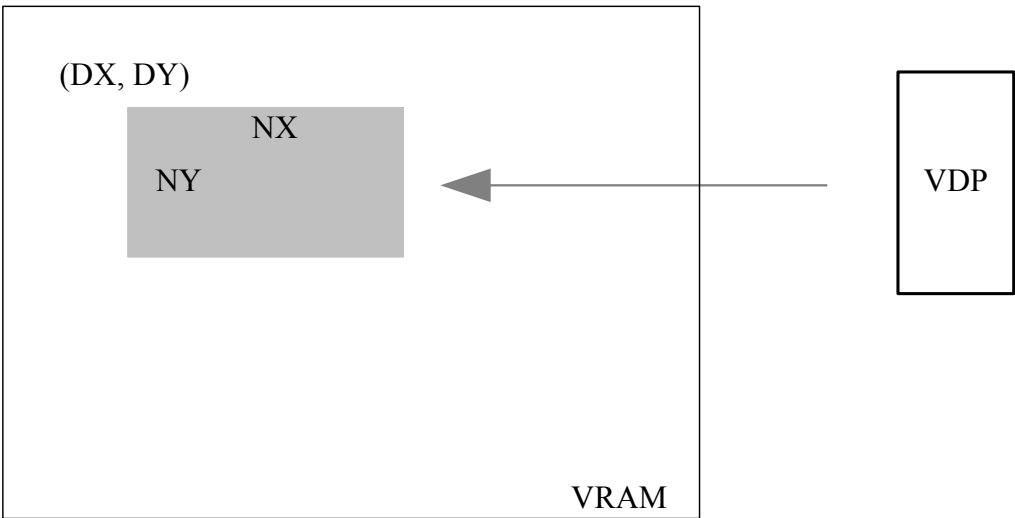
(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.



**HMMV (High speed Move to Memory from VDP)**

Schéma :



Fonction : HMMV permet de remplir la mémoire vidéo avec une seule donnée. A la différence des routines du Bios (BIGFIL ou FILVRM), la mémoire vidéo est définie par des coordonnées en X et Y.

Source : VDP.

Destination : VRAM.

Voici la marche à suivre pour exécuter l'instruction HMMV :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse
Registre 37 :	0	0	0	0	0	0	0	DX8	(0 à 511)
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée
Registre 39 :	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DX0 à DX8 = Abscisse du premier pixel de destination.

En SCREEN 5 et 7, DX0 est ignoré.

En SCREEN 6, DX0 et DX1 sont ignorés.

DY0 à DY9 = Ordonnée du pixel origine.

Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
Registre 41 :	0	0	0	0	0	0	0	NX8	(0 à 511)
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
Registre 43 :	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NX0 à NX8 = Nombre de pixels à placer dans la direction horizontale.

NY0 à NY9 = Nombre de pixels à placer dans la direction verticale.

Registre 44 :

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
-----	-----	-----	-----	-----	-----	-----	-----

donnée

SCREEN 5 et 7 :

CR0 à CR3 = Couleur du pixel d'abscisse impair.

CR4 à CR7 = Couleur du pixel d'abscisse pair.

SCREEN 6 :

CR0 et CR1 = Couleur du troisième pixel à droite de celui à l'abscisse multiple de 4.

CR2 et CR3 = Couleur du deuxième pixel à droite de celui à l'abscisse multiple de 4.

CR4 et CR5 = Couleur du pixel à droite de celui dont l'abscisse est multiple de 4.

CR6 et CR7 = Couleur du pixel dont l'abscisse est multiple de 4.

SCREEN 8 à 12 :

CR0 à CR7 = Couleur du pixel.

Registre 45 :

0	-	MXD	-	DIY	DIX	-	-
---	---	-----	---	-----	-----	---	---

paramètres

DIX = Direction horizontale pour NX. 0 pour droite ; 1 pour gauche.

DIY = Direction verticale pour NY. 0 pour bas ; 1 pour haut.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

2. Écrire 11000000B (0C0h) dans le registre 46 pour exécuter la commande.
3. Le bit CE du registre de statut 2 restera à 1 tant que le processus n'est pas terminé. Vérifier que ce bit soit à 0 avant d'exécuter la commande suivante ou de changer de paramètre.

Une fois la commande terminée, les registres de commande se trouveront dans l'état suivant :

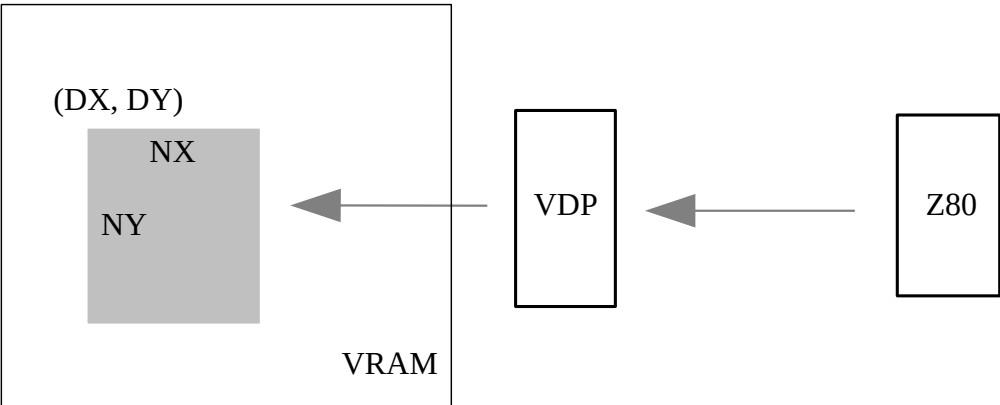
32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	*	idem	**	idem	idem

(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.

**LMMC (Logical Move to Memory from CPU)**

Schéma :



Fonction : LMMC permet de remplir la mémoire vidéo avec des données provenant du Z80. Le remplissage se fait pixel par pixel. Cette instruction s'exécute donc moins rapidement que HMMC. En contrepartie, il est possible de définir un opérateur logique. A la différence des routines du Bios (BIGFIL ou FILVRM), les données servant à remplir la mémoire vidéo peuvent être toutes différentes. De plus, la mémoire vidéo est définie par des coordonnées X et Y.

Source : Z80

Destination : VRAM

Voici la marche à suivre pour exécuter l'instruction LMMC :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source (0 à 256/511)
Registre 33 :	0	0	0	0	0	0	0	SX8	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SX0 à SX8 = Abscisse du pixel d'origine.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal (0 à 511)
Registre 41 :	0	0	0	0	0	0	0	NX8	
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical (0 à 1023)
Registre 43 :	0	0	0	0	0	0	NY9	NY8	

NX0 à NX8 = Nombre de pixels à placer dans la direction horizontale.

NY0 à NY9 = Nombre de pixels à placer dans la direction verticale.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 44 :	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée

CR0 à CR7 = Couleur en SCREEN 8 à 12.

En SCREEN 5 et 7, les CR4 à CR7 bits sont ignorés.

En SCREEN 6, les CR2 à CR7 bits sont ignorés.

Registre 45 :	0	0	MXD	0	DIY	DIX	0	0	paramètres
---------------	---	---	-----	---	-----	-----	---	---	------------

DIX = Direction horizontale pour NX. 0 pour droite ; 1 pour gauche.

DIY = Direction verticale pour NY. 0 pour bas ; 1 pour haut.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour VRAM étendue des MSX ayant 192 Ko de VRAM.

2. Écrire le code de la commande sur les 4 bits de poids fort ainsi que le code de l'opérateur logique désiré sur les 4 bits de poids faible du registre 46 pour exécuter la commande.

Registre 46 :	1	0	1	1	LO3	LO2	LO1	LO0	commande+opérateur
---------------	---	---	---	---	-----	-----	-----	-----	--------------------

LO0 à LO3 = Opérateur logique à appliquer.

0000 = IMP	1000 = TIMP
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

3. Lire le registre de statut 2.
4. Tester l'état du bit CE. Si celui-ci est à 1, alors le processus est terminé. Sinon, on passe à l'étape suivante.
5. Tester l'état du bit TR pour savoir si le transfert s'est effectué, si celui-ci se trouve à 0, alors le VDP n'est pas prêt à recevoir l'octet suivant, recommencer en 3. Si par contre ce bit est à 1, effectuer l'étape suivante.
6. Envoyer l'octet suivant à mettre en VRAM dans le registre 44 (le premier octet a été traité à l'étape 1) puis reprendre l'opération à l'étape 3.

Une fois la commande terminée, les registres de commande du VDP se trouveront dans l'état suivant :

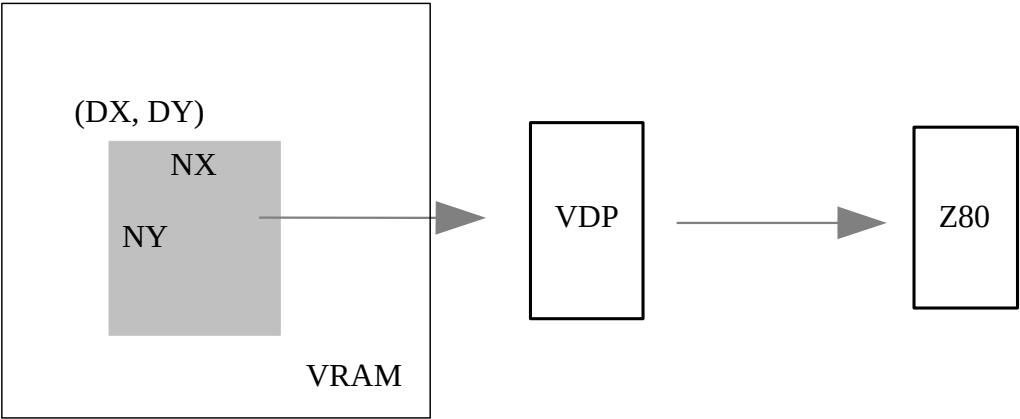
32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	*	idem	**	idem	idem

(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.

**LMCM (Logical Move to CPU from Memory)**

Schéma :



Fonction : LMCM permet de récupérer en mémoire central (via le Z80) le contenu d'une zone de la mémoire vidéo. Le transfert se fait pixel par pixel.

Source : VRAM

Destination : Z80

Voici la marche à suivre pour exécuter l'instruction LMCM :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source (0 à 511)
Registre 33 :	0	0	0	0	0	0	0	SX8	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SX0 à SX8 = Abscisse du pixel d'origine.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal (0 à 511)
Registre 41 :	0	0	0	0	0	0	0	NX8	
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical (0 à 1023)
Registre 43 :	0	0	0	0	0	0	NY9	NY8	

NX0 à NX8 = Nombre de pixels à transférer dans la direction horizontale.

NY0 à NY9 = Nombre de pixels à transférer dans la direction verticale.

Registre 45 :	0	-	-	MXS	DIY	DIX	-	-	paramètres
---------------	---	---	---	-----	-----	-----	---	---	------------

DIX = Direction horizontale pour NX. 0 = Droite ; 1 = Gauche.

DIY = Direction verticale pour NY. 0 = Bas ; 1 = Haut.

MXS = Mémoire vidéo source. 0 pour la VRAM principale ; 1 pour la VRAM étendue des

1. Écrivez la valeur comme suit dans le registre 46 pour exécuter la commande.

Registre 46 : 

1	0	1	0	LO3	LO2	LO1	LO0
---	---	---	---	-----	-----	-----	-----

 commande+opérateur

LO0 à LO3 = Opérateur logique à appliquer.

0000 = IMP	1000 = TIMP
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

2. Lire le bit TR du registre 2 de statut, si celui-ci se trouve à 1, c'est que le VDP n'a pas encore transféré toutes les données, vous pouvez donc lire le registre de statut 7 pour obtenir la donnée suivante issue de la mémoire vidéo. Si par contre, bit TR passera à 0 une fois que toutes les données auront été lues.

La couleur dans le registre de statut 7 prend la forme suivante.

Registre de statut 7 : 

bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0
CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0

 donnée

CR0 à CR7 = Couleur en SCREEN 8 à 12.

En SCREEN 5 et 7, les CR4 à CR7 bits sont ignorés.

En SCREEN 6, les CR2 à CR7 bits sont ignorés.

3. Tester l'état du bit CE, si celui-ci est à 0, alors le processus est terminée. Dans le cas contraire (CE à 1), on recommence à l'étape 4.

Notes :

- TR doit être à 0 pour exécuter cette commande. (Lire le registre 7 de statut si ce n'est pas le cas)
- Après la lecture de la dernière donnée, même si TR est à 1, la commande sera terminée lorsque le bit CE sera à 0.

Une fois la commande terminée, les registres de commande du VDP se trouveront dans l'état suivant :

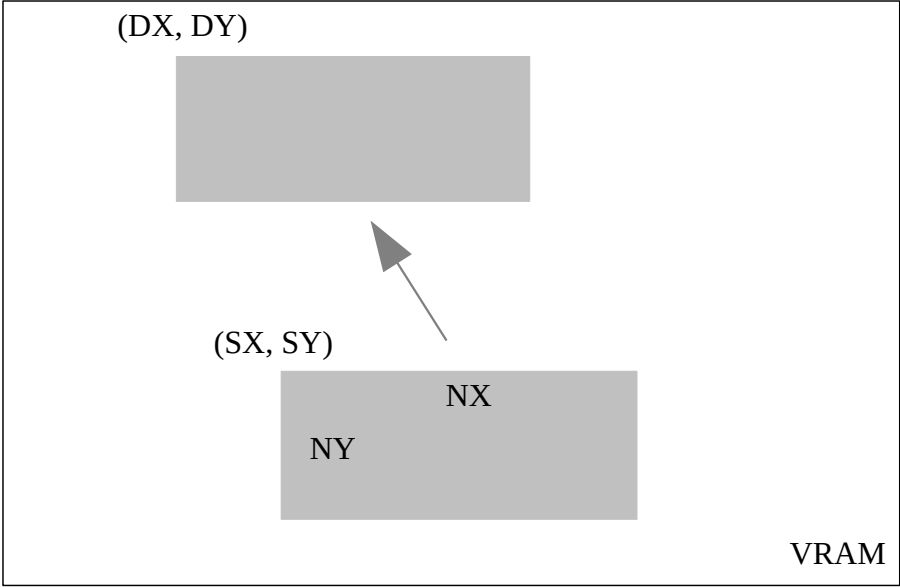
32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	*	idem	idem	idem	**	idem	idem

(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.

**LMMM (Logical Move to Memory from Memory)**

Schéma :



Fonction : LMMM fait la copie d'une zone de mémoire vidéo vers une autre une autre zone. La copie se fait pixel par pixel.

Source : VRAM.

Destination : VRAM.

Voici la marche à suivre pour exécuter l'instruction LMMM:

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source (0 à 511)
Registre 33 :	0	0	0	0	0	0	0	SX8	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SX0 à SX8 = Abscisse du pixel d'origine.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination (0 à 511)
Registre 37 :	0	0	0	0	0	0	0	DX8	
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination (0 à 1023)
Registre 39 :	0	0	0	0	0	0	DY9	DY8	

DX0 à DX8 = Abscisse du premier pixel de destination.

DY0 à DY9 = Ordonnée du pixel de destination.

Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal (0 à 511)
Registre 41 :	0	0	0	0	0	0	0	NX8	
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical (0 à 1023)
Registre 43 :	0	0	0	0	0	0	NY9	NY8	

NX0 à NX8 = Nombre de pixels à copier dans la direction horizontale.

NY0 à NY9 = Nombre de pixels à copier dans la direction verticale.

Registre 45 :	0	-	MXD	MXS	DIY	DIX	-	-	paramètres
---------------	---	---	-----	-----	-----	-----	---	---	------------

DIX = Direction horizontale pour NX. 0 pour droite ; 1 pour gauche.

DIY = Direction verticale pour NY. 0 pour bas ; 1 pour haut.

MXS = Mémoire vidéo source. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

- Écrire le code de la commande sur les 4 bits de poids fort ainsi que le code de l'opérateur logique désiré sur les 4 bits de poids faible du registre 46 pour exécuter la commande.

Registre 46 :	1	0	0	1	LO3	LO2	LO1	LO0	commande+opérateur
---------------	---	---	---	---	-----	-----	-----	-----	--------------------

LO0 à LO3 = Opérateur logique à appliquer.

0000 = IMP	1000 = TIMP
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

- Le bit CE du registre de statut 2 restera à 1 tant que le processus n'est pas terminé. Vérifiez que ce bit soit à 0 avant d'exécuter la commande suivante ou de changer de paramètre.

Une fois la commande terminée, les registres de commande se trouveront dans l'état suivant :

32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	*	idem	*	idem	**	idem	idem

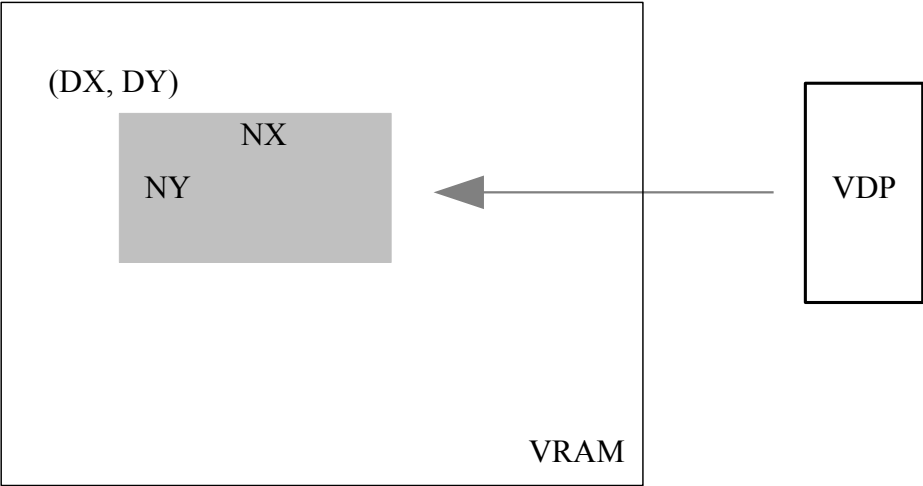
(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.



**LMMV (Logical Move to Memory from VDP)**

Schéma :



Fonction : LMMV permet de remplir la mémoire vidéo avec une seule donnée. A la différence des routines du Bios (BIGFIL ou FILVRM), la mémoire vidéo est définie par des coordonnées en X et Y. Le remplissage se fait pixel par pixel, ce qui autorise l'emploi d'un opérateur logique.

Source : VDP.

Destination : VRAM.

Voici la marche à suivre pour exécuter l'instruction LMMV:

1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source (0 à 511)
Registre 33 :	0	0	0	0	0	0	0	SX8	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SX0 à SX8 = Abscisse du pixel d'origine.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal (0 à 511)
Registre 41 :	0	0	0	0	0	0	0	NX8	
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical (0 à 1023)
Registre 43 :	0	0	0	0	0	0	NY9	NY8	

NX0 à NX8 = Nombre de pixels à placer dans la direction horizontale.

NY0 à NY9 = Nombre de pixels à placer dans la direction verticale.

Registre 44 :	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée
---------------	-----	-----	-----	-----	-----	-----	-----	-----	--------

CR0 à CR7 = Couleur en SCREEN 8 à 12.

En SCREEN 5 et 7, les bits CR4 à CR7 sont ignorés.

En SCREEN 6, les CR2 à CR7 bits sont ignorés.

Registre 45 : 

0	-	MXD	-	DIY	DIX	-	-
---	---	-----	---	-----	-----	---	---

 paramètres

DIX = Direction horizontale pour NX. 0 pour droite, 1 pour gauche.

DIY = Direction verticale pour NY. 0 pour bas, 1 pour haut.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

2. Écrire le code de la commande sur les 4 bits de poids fort ainsi que le code de l'opérateur logique désiré sur les 4 bits de poids faible du registre 46 pour exécuter la commande.

Registre 46 : 

1	0	0	0	LO3	LO2	LO1	LO0
---	---	---	---	-----	-----	-----	-----

 commande+opérateur

LO0 à LO3 = Opérateur logique à appliquer.

0000 = IMP	1000 = TIMP
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

3. Le bit CE du registre de statut 2 restera à 1 tant que le processus n'est pas terminé. Vérifiez que ce bit soit à 0 avant d'exécuter la commande suivante ou de changer de paramètre.

Une fois la commande terminée, les registres de commande se trouveront dans l'état suivant :

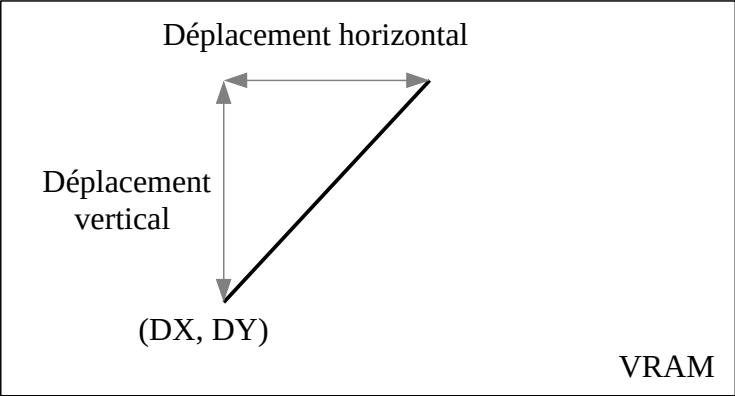
32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	*	idem	*	idem	idem

(\*) Prend la dernière valeur en fin de commande.

(\*\*) Prend la valeur de la longueur entre point départ et la fin de l'écran lorsque celui-ci est atteint.

**LINE (draw a LINE)**

Schéma :



Fonction : LINE trace une ligne à l'écran. L'utilisateur définit un triangle rectangle et le processeur vidéo en trace l'hypoténuse.

Source : VDP

Destination : VRAM

Voici la marche à suivre pour exécuter l'instruction LMMV :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination (0 à 511)
Registre 37 :	0	0	0	0	0	0	0	DX8	
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination (0 à 1023)
Registre 39 :	0	0	0	0	0	0	DY9	DY8	

DX0 à DX8 = Abscisse du premier pixel du tracé.

DY0 à DY9 = Ordonnée du premier pixel du tracé.

Registre 40 :	MJ7	MJ6	MJ5	MJ4	MJ3	MJ2	MJ1	MJ0	déplacement le plus long (0 à 1023)
Registre 41 :	0	0	0	0	0	0	MJ9	MJ8	
Registre 42 :	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0	déplacement le plus court (0 à 511)
Registre 43 :	0	0	0	0	0	0	MI9	MI8	

NX0 à NX8 = Déplacement le plus long.

NY0 à NY9 = Déplacement le plus court.

Registre 44 :	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée
---------------	-----	-----	-----	-----	-----	-----	-----	-----	--------

CR0 à CR7 = Couleur du tracé en SCREEN 8 à 12.

En SCREEN 5 et 7, les CR4 à CR7 bits sont ignorés.

En SCREEN 6, les CR2 à CR7 bits sont ignorés.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 45 :	0	-	MXD	-	DIY	DIX	-	MAJ	paramètres

MAJ = Mettre à 1 pour que le déplacement le plus long se fasse à la verticale.

DIX = Mettre à 1 pour tracer la ligne vers la gauche.

DIY = Mettre à 1 pour tracer la ligne vers le haut.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue sur les MSX ayant 192 Ko de VRAM.

- Écrire le code de la commande sur les 4 bits de poids fort ainsi que le code de l'opérateur logique désiré sur les 4 bits de poids faible du registre 46 pour exécuter la commande.

Registre 46 :	0	1	1	1	LO3	LO2	LO1	LO0	commande+opérateur
---------------	---	---	---	---	-----	-----	-----	-----	--------------------

LO0 à LO3 = Opérateur logique à appliquer.

0000 = IMP	1000 = TIMP
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

- Le bit CE du registre de statut 2 restera à 1 tant que le processus n'est pas terminé. Vérifier que ce bit soit à 0 avant d'exécuter la commande suivante ou de changer de paramètre.

Une fois la commande terminée, les registres de commande du VDP se trouveront dans l'état suivant :

32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	*	idem	idem	idem	idem

(\*) Prend la dernière valeur en fin de commande.

**SRCH (SeaRCH for color)**

Schéma :



Fonction : SRCH permet de rechercher une couleur de contour donnée sur une ligne horizontale. Cette instruction est particulièrement utile pour le remplissage d'une surface (PAINT).

Source : VDP.

Destination : VRAM.

Voici la marche à suivre pour exécuter l'instruction SRCH :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source (0 à 511)
Registre 33 :	0	0	0	0	0	0	0	SX8	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SX0 à SX8 = Abscisse du pixel d'origine.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 44 :	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée
---------------	-----	-----	-----	-----	-----	-----	-----	-----	--------

CR0 à CR7 = Couleur de contour (SCREEN 8 à 12).

En SCREEN 5 et 7, les CR4 à CR7 bits sont ignorés.

En SCREEN 6, les CR2 à CR7 bits sont ignorés.

Registre 45 :	0	-	MXD	-	-	DIX	EQ	-	paramètres
---------------	---	---	-----	---	---	-----	----	---	------------

EQ = 0 pour que la recherche s'arrête lorsqu'une couleur est différente de celle de contour,

1 pour que la recherche s'arrête lorsqu'une couleur est la même que celle de contour.

DIX = Direction horizontale de la recherche à partir du pixel origine.

0 pour chercher vers la droite ; 1 pour chercher vers la gauche.

MXD = Mémoire vidéo de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue sur les MSX ayant 192 Ko de VRAM.

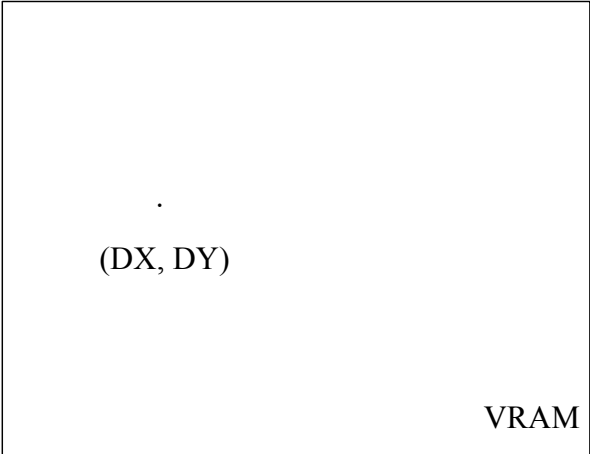
2. Écrire l'octet 01100000B (060h) dans le registre 46 exécuter la commande.
3. Lire le registre de statut 2.
4. Tester l'état du bit CE, si celui-ci est à 1, on retourne à l'étape 3. Si le bit CE est à 0, on continue.
5. Tester l'état du bit BD. Si celui-ci se trouve à 1, c'est que la commande s'est terminée sur la même couleur que celle de contour (bit EQ à 1) ou une couleur différente (bit EQ à 0). L'abscisse de ce pixel s'obtient alors en lisant le contenu des registres de statut 8 et 9. Le premier renferme les 8 bits de poids faible de cette abscisse, alors que le bit 0 du registre 9 donne le bit de poids fort. Si BD est à 0, c'est que la commande s'est terminée en fin de ligne.

Une fois la commande terminée, les registres de commande se trouveront dans l'état suivant :

32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	idem	idem	idem	idem	idem

**PSET (Point SET)**

Schéma :



Fonction : PSET affiche un point dans la mémoire vidéo

Source : VDP.

Destination : VRAM.

Voici la marche à suivre pour exécuter l'instruction PSET :

- 1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source (0 à 511)
Registre 33 :	0	0	0	0	0	0	0	SX8	
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source (0 à 1023)
Registre 35 :	0	0	0	0	0	0	SY9	SY8	

SX0 à SX8 = Abscisse du pixel d'origine.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 44 :	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	couleur
---------------	-----	-----	-----	-----	-----	-----	-----	-----	---------

CR0 à CR7 = Couleur du point à placer en SCREEN 8 à 12.

En SCREEN 5 et 7, les CR4 à CR7 bits sont ignorés.

En SCREEN 6, les CR2 à CR7 bits sont ignorés.

Registre 45 :	0	0	MXD	0	0	0	0	0	paramètres
---------------	---	---	-----	---	---	---	---	---	------------

MXD = Mémoire de destination. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

- 2. Écrire le code de la commande sur les 4 bits de poids fort ainsi que le code de l'opérateur logique

désiré sur les 4 bits de poids faible du registre 46 pour exécuter la commande.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 46 :	0	1	0	1	LO3	LO2	LO1	LO0	commande+opérateur

LO0 à LO3 = Opérateur logique à appliquer.

0000 = IMP	1000 = TIMP
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

3. Le bit CE du registre 2 de statut restera à 1 tant que le processus n'est pas terminé. Vérifier que ce bit soit à 0 avant d'exécuter la commande suivante ou de changer de paramètre.

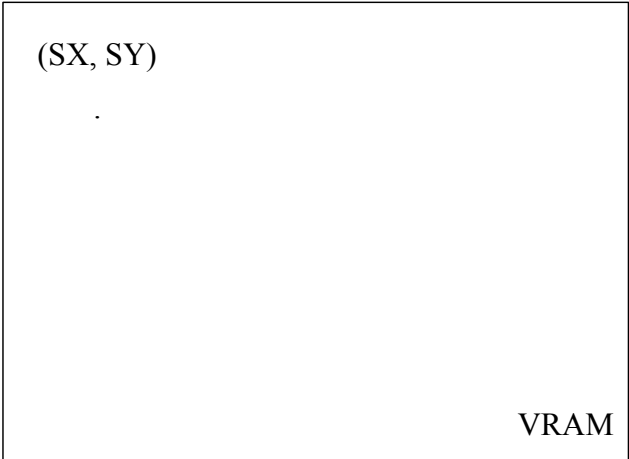
Une fois la commande terminée, les registres de commande du VDP se trouveront dans l'état suivant :

32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	idem	idem	idem	idem	idem



**POINT (is POINT set?)**

Schéma :



Fonction : POINT donne la couleur d'un pixel dans la mémoire vidéo.

Source : VRAM

Destination : VDP.

Voici la marche à suivre pour exécuter l'instruction POINT :

1. Écrire les paramètres dans les registres suivants.

	bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0	
Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source
Registre 33 :	0	0	0	0	0	0	0	SX8	(0 à 511)
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source
Registre 35 :	0	0	0	0	0	0	SY9	SY8	(0 à 1023)

SX0 à SX8 = Abscisse du pixel d'origine.

SY0 à SY9 = Ordonnée du pixel d'origine.

Registre 45 :	-	-	-	MXS	-	-	-	-	paramètres
---------------	---	---	---	-----	---	---	---	---	------------

MXS = Mémoire vidéo source. 0 pour la VRAM principale ; 1 pour la VRAM étendue des MSX ayant 192 Ko de VRAM.

2. Écrire le code de la commande (040h) dans le registre 46 pour exécuter la commande.
3. Lire le bit CE du registre 2 de statut. Si il est à 1, alors l'instruction n'est pas terminée, recommencer l'étape 3. Lorsque le bit CE passe à 0, on peut poursuivre à l'étape 4.
4. Lire le registre de statut 7, il contient le code de la couleur du pixel sous la forme suivante.

bit 7	bit 6	bit 5	bit 4	bit 4	bit 2	bit 1	bit 0
-------	-------	-------	-------	-------	-------	-------	-------

Registre de statut 7 :	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
------------------------	-----	-----	-----	-----	-----	-----	-----	-----

donnée

CR0 à CR7 = Couleur en SCREEN 8 à 12.

En SCREEN 5 et 7, les CR4 à CR7 bits sont ignorés.

En SCREEN 6, les CR2 à CR7 bits sont ignorés.

Une fois la commande terminée, les registres de commande du VDP se trouveront dans l'état suivant :

32~33(SX)	34~35 (SY)	36~37 (DX)	38~39 (DY)	40~41 (NX)	42~43 (NY)	44	45
idem	idem	idem	idem	idem	idem	Couleur lue	idem

## 7.9 Les différents modes d'affichage (SCREEN 0 à SCREEN 12)

Nous allons voir exactement comment fonctionne chaque mode d'affichage.

### SCREEN 0 en mode 40 colonnes (MSX1~)

Résolution :	256 × 192 pixels.	
Type :	Mode texte, 40 colonnes × 24 lignes, caractères 6 × 8.	
Couleurs :	2 couleurs parmi 15. (2 parmi 512 à partir de MSX2.)	
Sprites :	<i>Inutilisés.</i>	
Taille d'une page écran :	4 Ko.	
Nombre de pages maximum :	32 avec 128 Ko de VRAM.	
Table des positions de caractère :	00000h~003BFh	960 octets.
Table de la palette (MSX2~) :	00400h~0041FH	32 octets.
Table des formes de caractère :	00800h~00FFFh	2048 octets.
Table des couleurs de caractère :	<i>Inutilisée.</i>	
Table des attributs de Sprite :	<i>Inutilisée.</i>	
Table des formes de Sprite :	<i>Inutilisée.</i>	
Table des couleurs de Sprite :	<i>Inutilisée.</i>	

Description du fonctionnement du mode texte :

Exécutons le programme Basic suivant.

```
10 SCREEN0: WIDTH40: CLS
20 PRINT"HELLO!"
```

Le texte « HELLO! » s'affiche sur le coin tout en haut à gauche de l'écran. Ces caractères sont disposés en VRAM de la façon suivante dans la table des formes de caractère :

Adresse	+0	+1	+2	+3	+4	+5	+6	...
00000h	48h (H)	45h (E)	4Ch (L)	4Ch (L)	4Fh (O)	21h (!)	20h	...
00028h	20h	20h	20h	20h	20h	...		
00050h	20h	20h	20h	...				
00078h	20h	...	...					
:								

Chaque caractères ont une taille de 6x8 mais prennent une place de 8x8 dans la table des formes de caractère. À l'initialisation du SCREEN 0, la table des formes est importée depuis la Main-ROM et disposées dans l'ordre du code ASCII.

Table des formes de caractère :

Caractère 00h	Caractère 01h	Caractère 02h	Caractère 03h	... jusqu'à 0FFh
00800h	00808h	00810h	00818h	
00801h	00809h	00811h	00819h	
00802h	0080Ah	00812h	0081Ah	
00803h	0080Bh	00813h	0081Bh	
00804h	0080Ch	00814h	:	
00805h	0080Dh	00815h		
00806h	0080Eh	00816h		
00807h	0080Fh	00817h		

Écran :

Ligne 0	H	E	L	L	O	!		...
Ligne 1								...
Ligne 2				...				
:								

La position de chaque caractère à l'écran est codé sur un octet dans la table des formes de caractère. Il y a 40 colonnes multipliées par 24 lignes, soit 960 caractères donc 960 octets. Chaque octet renferme le code ASCII du caractère à afficher à l'écran. Ainsi, si à l'écran se trouvent uniquement les caractères « HELLO » dans le coin supérieur gauche, la table des formes commencera par :

00000h	72 (048h), code du « H »
00001h	69 (045h), code du « E »



7 6 5 4 3 2 1 0

Pour cela, il suffit d'exécuter le petit programme suivant :

Tous les « A » de l'écran se transforment en signes cabalistiques d'un ésotérisme certain. Un simple SCREEN 0 fera revenir les choses à la normale.

Résolution :	512 × 192 / 212 pixels.
Type :	Mode texte, 80 colonnes × 24 / 26,5 lignes, caractères 6 × 8.
Couleurs :	4 couleurs parmi 512.
Sprites :	<i>Inutilisés.</i>
Taille d'une page écran :	8 Ko.
Nombre de pages maximum :	16 avec 128 Ko de VRAM.

Le mode 80 colonnes fonctionne de manière identique au mode texte 40 colonnes (voir ci-dessus pour

plus de précisions). La table des formes de caractère contient toujours un octet par caractère à l'écran, soit cette fois 80 colonnes fois 24 lignes donc 1920 octets, ou 80 colonnes fois 26,5 lignes (allez, on arrondit à 27), soit 2160 octets. Chaque octet renferme le code ASCII du caractère à afficher à l'écran. Ainsi, si à l'écran se trouvent uniquement les caractères « A ELLE » dans le coin supérieur gauche, la table des formes commencera par :

00000h	65 (41h), code du « A »
00001h	32 (020h), code d'espacement
00002h	69 (045h), code du « E »
00003h	76 (04Ch), code du « L »
00004h	76 (04Ch), code du « L »
00005h	69 (045h), code du « E »
00006h	32 (020h), code d'espacement
⋮	⋮

Le reste de la table serait alors rempli de 32 (020h), code d'espacement (ou éventuellement de 0).

La table des formes de caractère contient, elle, 2048 octets regroupés par paquets de 8 octets. Ce qui donne 2048 divisé par 8, soit 256 paquets. Chaque paquet correspond à un caractère du code ASCII et en définit la forme. Par exemple le 66ème paquet code le caractère « B » et ressemble à ceci :

Paquet n°66, donc octet  $66 \times 8 = 528$  (210h). On trouvera donc l'information qui nous intéresse à l'adresse de début de la table des formes (01000h) plus 210h, soit 01210h.

							7	6	5	4	3	2	1	0	
01210h	contient	240	=	0F0h	=	11110000B	=>	X	X	X	X				
01211h	contient	72	=	048h	=	01001000B	=>		X			X			
01212h	contient	72	=	048h	=	01001000B	=>		X			X			
01213h	contient	112	=	070h	=	01110000B	=>		X	X	X				
01214h	contient	72	=	048h	=	01001000B	=>		X			X			
01215h	contient	72	=	048h	=	01001000B	=>		X			X			
01216h	contient	240	=	0F0h	=	11110000B	=>	X	X	X	X				
01217h	contient	0	=	000h	=	00000000B	=>								

Les deux bits de poids faible doivent toujours rester à 0 et ne sont de toute manière jamais pris en compte (car tout caractère se trouve défini dans une matrice 6 sur 8).

Redéfinissons à présent le caractère « B », faisons la démarche en sens inverse

7	6	5	4	3	2	1	0		en binaire	en hexadécimal
X	X	X	X	X				=>	11111000B	= 0F8h
X				X				=>	10001000B	= 088h
X		X	X	X				=>	10111000B	= 0B8h
X			X	X				=>	10011000B	= 098h
X		X	X	X				=>	10111000B	= 0B8h
X				X				=>	10001000B	= 088h
X				X				=>	10001000B	= 088h
X	X	X	X	X				=>	11111000B	= 0F8h

Il suffit à présent d'exécuter le petit programme suivant :

```
10 SCREEN 0: WIDTH 80: CLS: AD=&H1210
20 FOR I=0 TO 7:READ A$:VPOKE AD+I,VAL("&H"+A$):NEXT I
30 PRINT"B B B B"
50 DATA F8,88,B8,98,B8,88,88,F8,00
```

Tous les « B » de l'écran se transforment en petit signe « E » en vidéo inverse. L'instruction SCREEN 0 fera revenir les choses à la normale.

La table des couleurs de caractère permet de définir la couleur lors d'un clignotement le texte. Chaque bit correspond à un caractère à l'écran, voir le paragraphe 7.6 sur [les registres de contrôle du processeur vidéo](#) (Registres 12 et 13).

### **SCREEN 1** (MSX1~)

Résolution :	256 × 192 pixels.
Type :	Mode texte, 32 colonnes × 24 lignes, caractères 8 × 8.
Couleurs :	16 couleurs. (16 couleurs parmi 512 à partir du MSX2).
Sprites :	Type 1.
Taille d'une page écran :	4 Ko.
Nombre de pages maximum :	32 avec 128 Ko de VRAM.

Table des formes de caractère :	00000h~007FFh	2048 octets
Table des positions de caractère :	01800h~01AFFh	768 octets
Table des attributs de Sprite :	01B00h~01B7Fh	128 octets
Table des couleurs de caractère :	02000h~0201Fh	32 octets
Table de la palette des couleurs :	02020h~0203Fh	32 octets (MSX2~)
Table des formes de Sprite :	03800h~03FFFh	2048 octets
Table des couleurs de Sprite :	<i>Inutilisée.</i>	

Le SCREEN 1 fonctionne de manière proche du mode texte 40 colonnes (voir ci-dessus pour plus de précisions). La table des caractères est toujours constituée d'un octet par caractère à l'écran, soit cette fois 32 colonnes fois 24 lignes donc 768 octets. Chaque octet renferme le code ASCII du caractère à afficher à l'écran. Ainsi, si à l'écran se trouvent uniquement les caractères « A BEN » dans le coin supérieur gauche, la table des caractères commencera par :

01800H	65 (41h), code du « A »
01801H	32 (020h), code de l'espace
01802H	66 (042h), code du « B »
01803H	69 (045h), code du « E »
01804H	78 (04Eh), code du « N »
01805H	32 (020h), code de l'espace
:	:

Le reste de la table serait alors rempli de 32 (020h), code de l'espace (ou éventuellement de 0).

La table des formes de caractère contient aussi 2048 octets regroupés par paquets de 8 octets. Ce qui donne 2048 divisé par 8, soit 256 paquets. Chaque paquet correspond à un caractère (code ASCII) et en définit la forme. Par exemple, le 32ème paquet code le caractère d'espacement. Le paquet n°32, donc octet 32\*8 = 256 (100h). On trouvera donc l'information qui nous intéresse à l'adresse de début de la table des formes (00000h) plus 100h, soit 0100h. Les cases de 00100h à 00107h en VRAM sont toutes remplies de 0 (Espace = rien). Un simple VPOKE &H107, 255 devrait vous convaincre de l'utilité de bien connaître le processeur vidéo.

Notez que dans ce mode, tous les bits peuvent être utilisés pour définir un caractère (voir les modes 40 et 80 colonnes) car les caractères se trouvent cette fois définis dans une matrice 8 sur 8).

Essayez donc le programme Basic suivant (tout à fait désopilant) :

```
10 SCREEN 1:WIDTH 32
20 LOCATE 12,12:PRINT"COUCOU"
30 FOR J=0 TO 7
40 FOR I=0 TO 7
50 VPOKE &H100+I,2^J
60 NEXT I,J
70 GOTO 30
```

Ce que l'on fait avec 7 lignes Basic, c'est fou ! Après un BREAK, l'instruction SCREEN 1 fera revenir les choses à la (triste) normale.

Quant à la table des couleurs de caractère, rien de bien affriolant dans ce mode. Les 32 octets de la table des couleurs codent chacun la couleur d'une suite de huit caractères du code ASCII. Ce qui signifie par exemple que le contenu de 02009h donne la couleur des lettres H à O. Les 4 bits de poids fort codent la couleur du texte alors que les 4 bits de poids faible codent la couleur du fond. Le seul moyen d'utiliser un tel système est de redéfinir le jeu de caractères? A part cela, on peut toujours s'amuser à changer la couleur du curseur (code 255 ou 0FFh) :

```
10 SCREEN 1: WIDTH 32: COLOR 10,0
20 VPOKE &H201F,&HD7
```



L'exécution de ce programme rend le curseur mauve (0Dh) avec des caractères cyan (07h).

Après toutes ces émotions, et suivant l'adage « All work and no play makes Jack a sad boy », je vous propose un petit jeu. Profitez-en, amusez-vous bien, et hop (fonctionne sur MSX1 en retirant les instructions COLOR =) :

```
10 SCREEN 1: WIDTH 32: COLOR 10,1: Sprite$(1)="I"+CHR$(127)+"I"+CHR$(8)+
"I"+CHR$(127)+"I"+CHR$(8): PRINT: PRINT: FOR I=1 TO 4:PRINT"- RALLY -";:
NEXT: LOCATE 5,12: INPUT"NIVEAU (1 ou 2)";NN: NN=NN+1:IF NN=3 THEN N=4: DD=4
ELSE N=8: DD=2
20 FOR I=0 TO 7: VPOKE&H208+I,128: NEXT: FOR I=0 TO 7:VPOKE&H210+I,1: NEXT:
T=3:X=40:Y=88: PUT Sprite 1, (X,Y),9:COLOR 10,2: FOR I=0 TO 23: PRINT TAB(T-
1);"B"+STRING$(N+1,"O")+"A": NEXT: VPOKE&H2004,&HC2: VPOKE&H2009,17:
VPOKE&H2006,&HF2: COLOR=(2,1,1,2)
30 LOCATE 20,5:PRINT"Attention": FOR J=3 TO 1 STEP -1:LOCATE 23,7: PRINT J :
PLAY"L8D": FOR I=1 TO 800: NEXT I,J: LOCATE 20,5: PRINT SPACE$(9):LOCATE
24,7:PRINT" ": LOCATE 0,24:PLAY"L2O6A"
40 SC=SC+1: PUT Sprite 1, (X,Y),8:I=&H1960+X/8: IF VPEEK(I)<>79 OR VPEEK
(I+1)<>79 THEN 100 ELSE D=INT(RND(14)*3):D=-D*(T>2 AND T+N<29)-2*(T<=2)-
(T+N>=29): ON D GOTO 60,70
50 PRINT TAB (T-1);"B"+STRING$(N+1,"O")+"A":GOTO 80
60 T=T-1:PRINT TAB (T);"/"+STRING$(N,"O")+"/":GOTO 80
70 T=T+1:PRINT TAB (T-1);"\ "+STRING$(N,"O")+"\ ":GOTO 80
80 I=STICK(0): IF I=3 THEN X=X+DD else IF I=7 THEN X=X-DD
90 GOTO 40
100 VPOKE&H2007,&HF2: VPOKE&H200A,&H72: VPOKE&H200C,&H72: VPOKE&H200D,&H72:
VPOKE&H200E,&H72: LOCATE 1,23: PRINT"Score =";SC*10;: FOR I=1 TO 100000!:
NEXT
```

Voici l'exemple parfait de ce que l'on peut faire pour sa petite voisine par un après-midi pluvieux, gris.

## **SCREEN 2** (MSX1~)

Résolution :	256 × 192 pixels.
Type :	Caractères graphiques 8 × 8, 32 colonnes sur 3 × 8 lignes.
Couleurs :	15 couleurs dont 2 par ligne de caractère. (MSX1) (16 couleurs parmi 512 au lieu de 15 à partir du MSX2)
Sprites :	Type 1.
Taille d'une page écran :	16 Ko.
Nombre de pages maximum :	8 avec 128 Ko de VRAM.

Table des formes de caractère :	00000h~017FFh	6144 octets
Table des positions de caractère :	01800h~01AFFh	768 octets
Table des attributs de Sprite :	01B00h~01B7Fh	128 octets
Table des couleurs de caractère :	02000h~037FFh	6144 octets
Table de la palette des couleurs :	02020h~0203Fh	32 octets (MSX2~)
Table des formes de Sprite :	03800h~03FFFh	2048 octets
Table des couleurs de Sprite :	<i>Inutilisée</i>	

Le SCREEN 2 ressemble beaucoup à un mode texte dans son utilisation mais dans ce mode, tous les caractères sont graphiques. Ils ne sont pas définis selon le code ASCII. L'écran est divisé en 3 bandes horizontales. Chaque bande est composée d'un jeu de 256 caractères (de 0 à 255 par défaut).

Format de la table des positions de caractère :

1 <sup>er</sup> jeu de caractères	{	0	1	2	3	4	...						
		32	33	34	35	...							
		⋮											
		⋮											
		192	193	...									
		224	225	226	...								
2 <sup>e</sup> jeu de caractères	{	0	1	2	3	4	...						
		32	33	34	35	...							
		⋮											
		⋮											
		192	193	...									
		224	225	226	...								
3 <sup>e</sup> jeu de caractères	{	0	1	2	3	4	...						
		32	33	34	35	...							
		⋮											
		⋮											
		192	193	...									
		224	225	226	...								

...	27	28	29	30	31						
...		60	61	62	63						
⋮											
⋮											
...						126	127				
...						253	254	255			
...	27	28	29	30	31						
...		60	61	62	63						
⋮											
⋮											
...						126	127				
...						253	254	255			
...	27	28	29	30	31						
...		60	61	62	63						
⋮											
⋮											
...						126	127				
...						253	254	255			

La table des formes a le même format qu'en SCREEN1. Elle contient 2048 octets regroupés par paquets de 8 octets. Ce qui donne 2048 divisé par 8, soit 256 paquets. Chaque paquet correspond à un caractère. Sachant que dans ce mode d'écran, il y a 3 jeux de caractères, la table des formes fait donc 3 × 2048 octets. Les bits de ces octets à 1 représentent les points du tracé. Ceux à 0 représentent le fond du caractère graphique.

La table des couleurs défini la couleur pour chaque ligne de caractères. Les 4 bits de poids fort codent la couleur du tracé (0 ~ 15), alors que les 4 bits de poids faible codent la couleur de font (0 ~ 15). Par exemple si l'adresse 00009h contient 081h alors que 02009h renferme 0D1h, alors les pixels de coordonnées (8, 1) et (15, 1) seront allumés en magenta (0Dh) alors que les pixels (9, 1) à (14, 1)

resteront (ou deviendront) noirs.

Bien entendu, rien n'empêche le programmeur averti de modifier la table des formes de caractère. Il suffit de faire attention à quel tiers d'écran on désire accéder. Voyons un petit exemple en Basic :

```
10 SCREEN 2: COLOR 10,1,1: CLS
20 '
30 ' TABLE DES FORMES, FORME N°10
40 FOR I=0 TO 7: READ A$: VPOKE I,VAL("&H"+A$): NEXT
50 '
60 ' TABLE DES COULEURS, FORME N°0
70 FOR I=&H2000 TO &H2007: READ A$: VPOKE I,VAL("&H"+A$): NEXT
80 '
90 ' TABLE DES MOTIFS, JOLI CADRE EN BRIQUES
100 FOR I=&H1800 TO &H181F: VPOKE I,0: NEXT
110 FOR I=&H18E0 TO &H18FF: VPOKE I,0: NEXT
120 FOR I=&H1800 TO &H18E0 STEP 32 :VPOKE I,0: NEXT
130 FOR I=&H181F TO &H18FF STEP 32 :VPOKE I,0: NEXT
140 '
150 CIRCLE(127,98),50,14: PAINT(127,98),14
160 '
170 GOTO 170
180 '
190 ' DONNÉES POUR LA FORME
200 DATA 00,BB,00,DD,00,BB,00,DD
210 '
220 ' DONNÉES POUR LES COULEURS
230 DATA 00,60,00,50,00,A0,00,20
```

Nous avons redéfini la forme 0, puis nous l'avons affichée sur tout la première et la septième ligne, ainsi que sur les côtés. Essayez donc de faire la même chose avec le Basic classique.

Vous remarquerez, à la ligne 150, que tous les dessins s'effectuent derrière notre cadre (ce qui est logique mais agréable). Attention cependant à ne pas utiliser les pixels dans le cadre entre (0, 0) et (7, 7).

### **SCREEN 3** (MSX1~)

Résolution :	64 × 48.
Type :	Caractères graphiques 2 × 4 pixels.
Couleurs :	15 couleurs (MSX1) 16 couleurs parmi 512 (MSX2~)
Sprites :	type 1

Taille d'une page écran :	4 Ko
Nombre de pages maximum :	32 avec 128 Ko de VRAM

Table des formes de caractère :	00000h~007FFh	2048 octets
Table des positions de caractère :	00800h~00AFFh	768 octets
Table des attributs de Sprite :	01B00h~01B7Fh	128 octets

Table de la palette des couleurs :	02020h~0203Fh	32 octets (MSX2~)
Table des formes de Sprite :	03800h~03FFFh	2048 octets
Table des couleurs de caractère :	<i>Inutilisée</i>	
Table des couleurs de Sprite :	<i>Inutilisée</i>	

Le mode d'écran 3 fonctionne comme le mode 2 (voir le SCREEN 2 pour plus de précisions), mais avec un écran de 64 sur 48. En matière de taille, un pixel mode 3 équivaut à 4 pixels mode 2.

0, 0	4, 0	8, 0		4 « gros » pixels
0, 3	7, 3		15, 3	
0, 4	4, 4	8, 4		4 « gros » pixels
0, 7	7, 7		15, 7	

Dans la table des formes, chaque octet code 2 gros pixels. Les 4 bits de poids fort donnent la couleur (0 à 15) du pixel d'abscisse paire, alors que les 4 bits de poids faible indiquent la couleur (0 à 15) du pixel d'abscisse impaire.

Si l'on néglige la table Bitmap, on ne travaille que sur la table des formes qui ressemble à :

00000H	0, 0	63, 0
00100H	0, 7	63, 7
	0, 8	63, 8
00200H	0, 15	63, 15
	0, 16	63, 16
00300H	0, 23	63, 23
	0, 24	63, 24
00400H	0, 31	63, 31
	0, 32	63, 32
00500H	0, 39	63, 39
	0, 40	63, 40
	0, 47	63, 47

Il est alors facile d'adresser n'importe quel pixel, comme le montre l'exemple suivant en Basic :

```

10 SCREEN 3: COLOR 10,1,1: CLS
20 '
30 X=25 : Y=117 : C=6 : PSET (X,Y),7 : FOR I=1 TO 1000: NEXT
40 VPOKE (INT(Y/32)*256) + INT(X/8)*8 + (Y MOD 32)/4, -(C*16)*(X MOD 4<2) -
C*(X MOD 4>1)
140 GOTO 140

```

Pour ceux qui désirent profiter des possibilités offertes par la table des formes de caractère, sachez que chaque octet de cette table contient le numéro de la forme à utiliser. Seulement, on n'utilise que deux lignes de la forme suivant la règle :

lignes 0 et 1 --- VRAM de 00800h à 0081Fh  
lignes 2 et 3 --- VRAM de 00820h à 0083Fh  
lignes 4 et 5 --- VRAM de 00840h à 0085Fh  
lignes 6 et 7 --- VRAM de 00860h à 0087Fh  
lignes 0 et 1 --- VRAM de 00880h à 0089Fh  
lignes 2 et 3 --- VRAM de 008A0h à 008BFh  
et ainsi de suite ...

Voici un nouvel exemple Basic utilisant cette technique :

```
10 SCREEN 3: COLOR 10,1,1: CLS
20 '
30 ' TABLE DES FORMES, FORME N°0
40 VPOKE 0,&HA8: VPOKE 1,&H8A: VPOKE 2,&HA8: VPOKE 3,&H8A: VPOKE 4,&HA8:
VPOKE 5,&H8A: VPOKE 6,&HA8: VPOKE 7,&H8A
50 '
60 ' TABLE DES CARACTÈRES, JOLI CADRE EN BRIQUES
70 FOR I=0 TO 31 : VPOKE &H800+I,0: NEXT
80 FOR I=0 TO 31 : VPOKE &HA00+I,0: NEXT
90 FOR I=&H800 TO &HA00 STEP 32: VPOKE I,0: NEXT
100 FOR I=&H81F TO &HA1F STEP 32: VPOKE I,0: NEXT
110 '
120 CIRCLE(127,98),50,14: PAINT(127,98),14
130 '
140 GOTO 140
```

Notez qu'en SCREEN 3, lorsque vous utilisez les instructions Basic OPEN « GRP: » AS #1 puis PRINT #1, « Texte » le texte s'affiche en grandes lettres (4 fois la taille normale 40 colonnes).

#### **SCREEN 4 (MSX2~)**

Résolution :	256 × 192 avec contraintes de couleur
Type :	Caractères graphiques 8 × 8, 32 colonnes sur 3 × 8 lignes.
Couleurs :	16 couleurs parmi 512
Sprites :	Type 2
Taille d'une page écran :	16 Ko
Nombre de pages maximum :	8 avec 128 Ko de VRAM

Table des formes de caractère :	00000h~017FFh	6144 octets
Table des positions de caractère :	01800h~01AFFh	768 octets
Table des couleurs de Sprite :	01C00h~0D7FFh	512 octets
Table de la palette des couleurs :	01E80h~01E9Fh	32 octets

Table des attributs Sprite :	01E00h~01E7Fh	128 octets
Table des couleurs de caractère :	02000h~037FFh	6144 octets
Table des formes de Sprite :	03800h~03FFFh	2048 octets

Le SCREEN 4 est en tous points identique au SCREEN 2 à l'exception des Sprites qui sont de type 2. Voir le paragraphe concernant le SCREEN 2 pour plus d'information sur l'affichage. Pour de plus amples renseignements sur les Sprites, voir le paragraphe 7.10 « [Les Sprites et leur fonctionnement](#) », page 245.

### **SCREEN 5** (MSX2~)

Résolution :	256 × 212 / 192
Type :	Bitmap
Couleurs :	16 couleurs parmi 512
Sprites :	Type 2
Taille d'une page écran :	32 Ko
Nombre de pages maximum :	4 avec 128 Ko de VRAM

Table Bitmap :	00000h~05FFFh	24576 octets (affichage 192 lignes)
" " :	00000h~069FFh	27136 octets (affichage 212 lignes)
Table des attributs de Sprite :	07600h~0767Fh	128 octets
Table des formes de Sprite :	07800h~07FFFh	2048 octets
Table des couleurs de Sprite :	07400h~075FFh	512 octets
Table de la palette des couleurs :	07680h~0769Fh	32 octets
Table des couleurs de caractère :	<i>Inutilisée</i>	
Table des formes de caractère :	<i>Inutilisée</i>	

Avec le mode d'écran 5, on entre dans le domaine des graphismes en « bitmap ». Ce terme signifie que chaque pixel est codé directement par sa couleur, et ce, indépendamment des autres pixels. Il n'existe donc plus de contrainte de couleur, ni même de table de couleur ou de formes. Ce mode « bitmap » apparaît donc comme idéal puisqu'il allie puissance et simplicité.

Dans ce mode, chaque octet de la table Bitmap code deux pixels. Les 4 bits de poids fort donnent la couleur (0 ~ 15) du pixel d'abscisse paire, alors que les 4 bits de poids faible donnent la couleur (0 ~ 15) du pixel d'abscisse impaire.

Le registre 2 du VDP indique la page à afficher (0 à 3) :

Registre 2 :	0	A16	A15	1	1	1	1	1
--------------	---	-----	-----	---	---	---	---	---

A16 et A15 : page à afficher 0 à 3.

### **SCREEN 6** (MSX2~)

Résolution : 512 × 212 / 192  
 Type : Bitmap  
 Couleurs : 4 couleurs parmi 512  
 Sprites : type 2  
 Taille d'une page écran : 32 Ko  
 Nombre de pages maximum : 4 avec 128 Ko de VRAM

Table Bitmap : 00000h~05FFFh 24576 octets (affichage 192 lignes)  
 " : 00000h~069FFh 27136 octets (affichage 212 lignes)  
 Table des couleurs de Sprite : 07400h~075FFh 512 octets  
 Table de la palette des couleurs : 07680h~0769Fh 32 octets  
 Table des attributs Sprite : 07600h~0767Fh 128 octets  
 Table des formes de Sprite : 07800h~07FFFh 2048 octets  
 Table des couleurs de caractère : *inutilisée*  
 Table des formes de caractère : *inutilisée*

En SCREEN 6, chaque octet de la table Bitmap code 4 pixels. Les bits 6 et 7 (poids fort) définissent la couleur (0 ~ 3) du pixel dont l'abscisse est multiple de 4. Les bits 4 et 5 codent la couleur du pixel immédiatement à la droite de celui dont l'abscisse est multiple de 4, et ainsi de suite...

La table Bitmap ressemble donc à :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00000h	pixel (0,0)		pixel (1,0)		pixel (2,0)		pixel (3,0)	
00001h	pixel (4,0)		pixel (5,0)		pixel (6,0)		pixel (7,0)	
⋮								
0007Fh	pixel (508,0)		pixel (509,0)		pixel (510,0)		pixel (511,0)	
00080h	pixel (0,1)		pixel (1,1)		pixel (2,1)		pixel (3,1)	
⋮								

Le processeur vidéo ne peut afficher que 4 couleurs en SCREEN 6. C'est vrai, mais tout le monde sait qu'en juxtaposant deux couleurs, avec des pixels suffisamment petits, on obtient une troisième couleur. Les concepteurs du VDP ont conçu une instruction qui mélange automatiquement deux couleurs (« hardware tiling »). Cette fonction peut s'appliquer à la marge (ainsi, 7 couleurs de marge sont possibles) et surtout aux Sprites de la manière suivante :

– pour la marge, préciser la couleur comme suit :

0	0	0	1	X	X	Y	Y	
				!	!	!	!	
				!	!	+-----+--		Couleur des pixels abscisse impaire
				+-----+	-----			Couleur des pixels abscisse paire

- pour les Sprites, préciser la couleur comme suit :

0	0	0	0	X	X	Y	Y	
				!	!	!	!	
				!	!	+-----+	+-----	Couleur partie gauche du pixel
				+-----+	+-----	Couleur partie droite du pixel		

Voici un exemple en Basic :

```

10 VDP(9)=VDP(9)OR &H20: ' Pour utiliser la couleur 0
20 SCREEN 6: COLOR 10,0,&B11001: CLS
30 COLOR=(1,7,0,0) : COLOR=(2,0,7,0) : COLOR=(3,0,0,7)
40 FOR I=10 TO 310 STEP 100: LINE(I,100)-(I+75,200),(I-10)/100,BF: NEXT
50 Sprite$(1)=STRING$(8,255)
60 OPEN"GRP:" AS #1: PRESET(110,80):PRINT#1,"SCREEN 6 : QUATRE COULEURS MAXI
!"
70 FOR I=0 TO 248: PUT Sprite 1,(I,25),&B0111: NEXT
80 FOR I=248 TO 0 STEP -1: PUT Sprite 1,(I,25),&B0111: NEXT
90 GOTO 70

```

Le registre 2 du V9938 indique la page à afficher (0 à 3) :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 2 :	0	A16	A15	1	1	1	1	1

A16 et A15 = Page à afficher 0 à 3.

Les modes d'écran 7 et 8 ne sont disponibles que sur les MSX2 équipés de 128 Ko de mémoire vive vidéo (VRAM).

### **SCREEN 7** (MSX2 avec 128Ko de VRAM ou plus)

Résolution : 512 × 192 / 212

Type : Bitmap

Couleurs : 16 couleurs parmi 512

Sprites : Type 2

Taille d'une page écran : 64 Ko

Nombre de pages maximum : 2

Table Bitmap :	00000h~0BFFFh	49152 octets (affichage 192 lignes)
" :	00000h~0D3FFh	54272 octets (affichage 212 lignes)
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs de Sprite :	0F800h~0F9FFh	512 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets
Table des couleurs de caractère :	<i>inutilisée</i>	
Table des formes de caractère :	<i>inutilisée</i>	



Dans ce mode, chaque octet de la table Bitmap code deux pixels. Les 4 bits de poids fort donnent la couleur (0 ~ 15) du pixel d'abscisse paire, alors que les 4 bits de poids faible donnent la couleur (0 ~ 15) du pixel d'abscisse impaire.

Le registre 2 du VDP indique la page à afficher (0 à 3) :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 2 :	0	0	A16	1	1	1	1	1

A16 et A15 = Page à afficher 0 à 1.

### **SCREEN 8** (MSX2 avec 128Ko de VRAM ou plus)

Résolution :	256 × 192 / 212 bitmap
Type :	Bitmap
Couleurs :	256 couleurs
Sprites :	Type 2
Taille d'une page écran :	64 Ko
Nombre de pages maximum :	2

Table Bitmap :	00000h~0BFFFh	49152 octets (affichage 192 lignes)
" :	00000h~0D3FFh	54271 octets (affichage 212 lignes)
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs de Sprite :	0F800h~0F9FFh	512 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets
Table des couleurs de caractère :	<i>inutilisée</i>	
Table des formes de caractère :	<i>inutilisée</i>	

Dans ce mode, chaque octet de la table des Bitmap définit la couleur d'un pixel (0-255).

La couleur est déterminée de la manière suivante :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Octet :	G2	G1	G0	R2	R1	R0	B1	B0

G0 à G2 = Intensité de vert (0 ~ 7). R0 à R2 = Intensité de rouge (0 ~ 7).

B0 et B1 = Intensité de bleu (0 ~ 3).

En Basic, on peut calculer le numéro de la couleur désirée avec la formule :

$$100\ C = 32 * G + 4 * R + B$$

Le registre 2 du VDP indique la page à afficher (0 à 3) :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 2 :	0	0	A16	A15	1	1	1	1

A16 et A15 = Page à afficher 0 à 1.

### **SCREEN 9** (MSX2 coréen)

Ce mode d'écran est utilisé par le mode « Hangul » qui permet d'afficher les caractères coréens. Ce mode utilise en fait le SCREEN 6.

### **SCREEN 10 et 11** (MSX2+~)

Résolution :	256 × 192 / 212
Type :	Semi-bitmap
Couleurs :	12499 couleurs codées en YJK sur 4 octets ou 16 couleurs en RGB
Sprites :	Type 2
Taille d'une page écran :	64 Ko
Nombre de pages maximum :	2

Table des pixels :	00000h~0BFFFh	49152 octets (affichage 192 lignes)
" :	00000h~0D3FFh	54271 octets (affichage 212 lignes)
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs de Sprite :	0F800h~0F9FFh	512 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets
Table des couleurs de caractère :	<i>inutilisée</i>	
Table des formes de caractère :	<i>inutilisée</i>	

Ces modes d'écran sont en fait les mêmes. La seule différence sont les instructions graphiques du basic qui travaillent en 16 couleurs RGB pour le SCREEN 10 et en 12499 couleurs YJK pour le SCREEN 11.

C'est le bit YAE du registre 25 qui différencie le mode SCREEN 10 du 11.

La couleur de chacun des 4 pixels alignés horizontalement est déterminée de la manière suivante :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Premier octet :	Y <sub>1</sub>				A <sub>1</sub>	bits de poids faible de K		
Second octet :	Y <sub>2</sub>				A <sub>2</sub>	bits de poids fort de K		
Troisième octet :	Y <sub>3</sub>				A <sub>3</sub>	bits de poids faible de J		
Quatrième octet :	Y <sub>4</sub>				A <sub>4</sub>	bits de poids fort de J		

La couleur du premier pixel est codée par Y<sub>1</sub>, J et K. Le second par Y<sub>2</sub>, J et K, le troisième par Y<sub>3</sub>, J et K puis le quatrième par Y<sub>4</sub>, J et K. Et ainsi de suite... Lorsque l'attribut d'un octet est à 1, J et K sont ignorés. Y<sub>1</sub> ~ Y<sub>4</sub> devient le numéro de couleur du pixel correspondant comme en SCREEN 5.

Le registre 2 du VDP indique la page à afficher (0 à 3) :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 2 :	0	0	A16	A15	1	1	1	1

A16 et A15 = Page à afficher 0 à 1.

## **SCREEN 12** (MSX2+~)

Résolution : 256 × 192 / 212

Type : YJK

Couleurs : 19268 couleurs codées en YJK sur 4 octets

Sprites : Type 2

Taille d'une page écran : 64 Ko

Nombre de pages maximum : 2 avec 128 Ko de VRAM

Table des pixels :	00000h~0BFFFh	49152 octets (affichage 192 lignes)
" :	00000h~0D3FFh	54271 octets (affichage 212 lignes)
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs de Sprite :	0F800h~0F9FFh	512 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets
Table des couleurs de caractère :	<i>inutilisée</i>	
Table des formes de caractère :	<i>inutilisée</i>	

Dans ce mode d'écran les couleurs sont codées en YJK. Ce qui permet d'afficher en 19268 couleurs simultanément au prix de contraintes sur chaque 4 pixels alignés horizontalement.

La couleur de chacun des 4 pixels alignés horizontalement est déterminée de la manière suivante :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Premier octet :	Y <sub>1</sub>					bits de poids faible de K		
Second octet :	Y <sub>2</sub>					bits de poids fort de K		
Troisième octet :	Y <sub>3</sub>					bits de poids faible de J		
Quatrième octet :	Y <sub>4</sub>					bits de poids fort de J		

La couleur du premier pixel est codée par Y<sub>1</sub>, J et K. Le second par Y<sub>2</sub>, J et K, le troisième par Y<sub>3</sub>, J et K puis le quatrième par Y<sub>4</sub>, J et K. Et ainsi de suite...

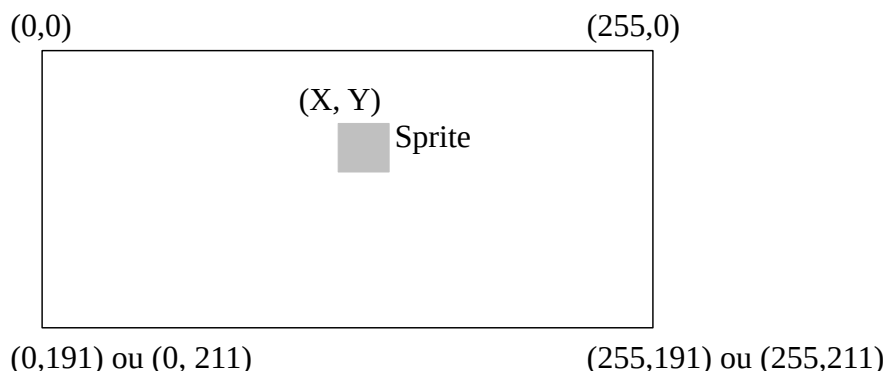
Le registre 2 du VDP indique la page à afficher (0 à 3) :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 2 :	0	0	A16	A15	1	1	1	1

A16 et A15 = Page à afficher 0 à 1.

## 7.10 Les Sprites et leur fonctionnement

Les Sprites, ou lutins graphiques, sont des motifs graphiques pouvant être affichés automatiquement à n'importe quel pixel dans n'importe quelle couleur, ceci en se superposant à l'avant-plan de l'écran, sans l'effacer. Ces facilités les prédestinent au mouvement puisqu'il suffit de modifier deux octets (les coordonnées du Sprite) pour faire déplacer tout un motif sans toucher au reste de l'écran. Le VDP peut mémoriser jusqu'à 256 formes de Sprite, il peut en afficher simultanément au maximum 32 à l'écran. Les Sprites se trouvent définis dans une matrice 8 sur 8 ou 16 sur 16. Ils peuvent être de 2 tailles, normale ou double.



Attention : En SCREEN 6 et 7, les coordonnées du côté droit de l'écran n'est pas (511,Y) mais bien (255,Y). Les Sprites se déplacent de 2 pixel par 2. Pour vous en persuader, essayez donc ce petit programme...

```
10 SCREEN 7: COLOR 10,0,0
20 LINE(0,0)-(511,0),15
30 Sprite$(0)=CHR$(&HFF)
40 FOR I=0 TO 255
50 PUT SPRITE 0,(I,0),8
60 NEXT
70 GOTO 40
```

... puis celui-ci :

```
10 SCREEN 7: COLOR 10,0,0
20 LINE(0,0)-(511,0),15
30 Sprite$(0)=CHR$(&HFF)
40 FOR I=0 TO 255
50 PUT Sprite 0,(I,255),8: ' ici Y=255
60 NEXT
70 GOTO 40
```

Il existe deux modes de fonctionnement des Sprites bien distincts. Le VDP choisit automatiquement le mode adéquat suivant le mode écran. En SCREEN 1, 2 et 3, c'est le mode Sprites type 1 appelé aussi mode restreint (qui correspond au MSX1) alors qu'en SCREEN 4 à 8, c'est le mode Sprites type 2 ou mode étendu (MSX2).

Fonctionnement :

Dans ce mode, il peut y avoir 32 Sprites à l'écran numérotés de 0 à 31. Plus un Sprite a un numéro faible, plus il est prioritaire. Ceci signifie que si le Sprite n°3 croise le Sprite n°16, on verra le Sprite n°3 passer dessus (lorsque des pixels des deux Sprites se trouveront aux mêmes coordonnées, c'est les pixels du Sprite n°3 qui seront affichés).

4 Sprites peuvent, au maximum, se trouver sur la même ligne. A partir du 5ème Sprite, toutes les portions communes aux 5 Sprites disparaissent sur le (ou les) Sprites de plus faible priorité.

Paramètres :

Matrice des Sprites    Le bit 1 (SI) du registre 1 du VDP définit la taille de la matrice.

SI à 0 pour Sprites de 8 × 8 pixels.

SI à 1 pour Sprites de 16 × 16 pixels.

Taille des Sprites    Le bit 0 (MAG) du registre 1 du VDP permet de doubler la taille.

MAG à 0 pour taille des pixels normale.

MAG à 1 pour taille des pixels double.

Pour bien comprendre le mécanisme de la taille, voici un exemple en Basic :

```
10 SCREEN 2: COLOR 10,0,0: CLS
20 OPEN"GRP:" AS #1
30 Sprite$(0)="ABABABAB"
40 PRESET(16,5): PRINT #1,"Un Sprite Quelconque.": COLOR
8,1: PRESET(16,155): PRINT #1,"MAG ="
50 LINE(63,155)-(72,164),1,BF: PRESET(64,155): PRINT #1,"0"
60 VDP(1)=VDP(1) AND &HFE
70 FOR I=0 TO 248
80 PUT Sprite 0,(I,35),7
90 NEXT
100 LINE(63,155)-(72,164),1,BF: PRESET(64,155): PRINT #1,"1"
110 VDP(1)=VDP(1) OR 1
120 FOR I=0 TO 248
130 PUT Sprite 0,(I,35),7
140 NEXT
150 GOTO 50
```

Collisions :

Lorsque deux Sprites entrent en collision (deux pixels ou plus avec les mêmes coordonnées), le bit 5 du registre de statut 0 passe à 1.

5 Sprites :

Lorsque plus de 4 Sprites se trouvent sur la même ligne, le bit 6 (5S) du registre de statut 0 passe à 1. De plus les 5 bits de poids faible du registre de statut 0 donnent le numéro du 5ème Sprite.

Utilisation :

Pour afficher un Sprite, il faut d'abord le définir dans la table des formes de Sprites.

Pour les Sprites 8 sur 8, il suffit de 8 octets pour définir un Sprite, sachant qu'un bit représente un pixel,

par exemple, en SCREEN 2, la table des formes de Sprites pourrait être :

		huit octets pour le Sprite n°0 qui aura cette forme							
	7 ~ 0								
03800h	11111111	X	X	X	X	X	X	X	X
03801h	10100001	X		X					X
03802h	10010001	X			X				X
03803h	10001001	X				X			X
03804h	10000101	X					X		X
03805h	10000101	X					X		X
03806h	10001001	X				X			X
03807h	11111111	X	X	X	X	X	X	X	X

Dans le cas de Sprites 16 sur 16, les choses deviennent un tout petit peu plus compliquées. Le Sprite se décompose en 4 parties :

0	2
1	3

	7	0	7	6	5	4	3	2	1	0	
03800h	11111110		X	X	X	X	X	X	X		
03801h	10000010		X							X	
03802h	10001110		X				X	X	X		
03803h	10001000		X				X				
03804h	10001000		X				X				
03805h	10001000		X				X				
03806h	10001000		X				X				
03807h	10001000		X				X				
03808h	10001000		X				X				
03809h	10001000		X				X				
0380Ah	10001000		X				X				
0380Bh	10001000		X				X				
0380Ch	10001000		X				X				
0380Dh	10001111		X				X	X	X	X	
0380Eh	10000000		X								
0380Fh	11111111		X	X	X	X	X	X	X	X	
03810h	01111111			X	X	X	X	X	X	X	
03811h	01000001			X						X	
03812h	01110001			X	X	X				X	
03813h	00010001					X				X	
03814h	00010001					X				X	
03815h	00010001					X				X	
03816h	00010001					X				X	
03817h	00010001					X				X	
03818h	00010001					X				X	
03819h	00010001					X				X	
0381Ah	00010001					X				X	
0381Bh	00010001					X				X	
0381Ch	00010001					X				X	
0381Dh	11110001		X	X	X	X				X	
0381Eh	00000001									X	
0381Fh	11111111		X	X	X	X	X	X	X	X	

huit octets pour le premier quart du Sprite n°0  
 huit octets pour le second quart du Sprite n°0  
 huit octets pour le troisième quart du Sprite n°0  
 huit octets pour le dernier quart du Sprite n°0



La réunion des quatre parties donne le résultat ci-dessous :

### Première forme de Sprite (forme 0)

Une fois le Sprite défini, on peut le manipuler à souhait grâce à la table des attributs de Sprites.

Cette table fonctionne avec des plans graphiques, et compte 4 octets par plan. On appelle « plan graphique » un espace comprenant un élément graphique. Il y a donc 32 plans (numérotés de 0 à 31) avec chacun un Sprite, puis un plan avec les graphismes ordinaires, enfin parfois un plan en entrée vidéo.

La table des attributs de Sprites code les caractéristiques de chaque plan sur 4 octets :

	7	6	5	4	3	2	1	0	
01B00H	Ordonnée du Sprite 0 (0-255)								} Attributs du Sprite 0
01B01H	Abscisse du Sprite 0 (0-255)								
01B02H	Numéro de forme de Sprite (0-255)								
01B03H	EC	0	0	0	Couleur (0~15)				
01B04H	Ordonnée du Sprite 1 (0-255)								
⋮	⋮								

L'ordonnée peut varier entre 0 et 255. Si le Sprite sort de l'écran, le VDP n'affichera que la partie visible du Sprite. En réalité, on utilise les coordonnées de 225 (-31 en complément à 2) à 255, et 0 à 191. Ceci permet de faire entrer un Sprite dans l'écran (-31 à -1 ou 225 à 255) ou au contraire de le faire sortir (183 à 191 pour un 8 sur 8) petit à petit.

Notez que si l'ordonnée de n'importe quel Sprite est à 208 (0D0h), alors tous les Sprites dans les plans supérieurs deviennent invisibles. Par exemple si l'on met l'ordonnée du Sprite contenu dans 12 à 208, alors tous les Sprites dans les plans 13 à 31 disparaissent.

L'abscisse prend une valeur entre 0 et 255. Pour faire sortir le Sprite, pas de problème, on utilise les abscisses entre 247 et 255 (pour un Sprite 8 sur 8). Par contre, pour le faire entrer, il faut utiliser le bit EC (voir ci-dessous).

Le numéro de Sprite donne la forme à utiliser à l'affichage. On peut définir 256 Sprites 8 sur 8, ou 64 Sprites 16 sur 16. pour les Sprites 8 sur 8, on choisit simplement le numéro de celui que l'on désire afficher. Pour les Sprites 16 sur 16, on donne le numéro de Sprite multiplié par quatre.

La couleur est toujours codée sur 4 bits.

Le bit EC (« Early Clock ») sert à décaler le Sprite de 32 pixels vers la gauche. Ceci permet de faire « entrer » un Sprite à l'écran comme dans l'exemple suivant :

```
10 SCREEN 2: COLOR 10,0,0: CLS
20 Sprite$(1)=STRING$(8,255)
30 FOR I=24 TO 40: FOR J=0 TO 100: NEXT J
40 PUT Sprite 0,(I,35): VPOKE &H1B03,&H87
50 NEXT I
55 IF STRIG(0)=0 THEN 55
```

## Le mode 2 (MSX2~)

### Fonctionnement :

Dans ce mode, il peut y avoir 32 Sprites à l'écran numérotés de 0 à 31. Plus un Sprite a un numéro faible, plus il est prioritaire. Ceci signifie que si le Sprite n°3 croise le Sprite n°16, on verra le Sprite n°3 passer dessus (lorsque des pixels des deux Sprites se trouveront aux mêmes coordonnées, c'est les pixels du Sprite n°3 qui seront affichés).

8 Sprites peuvent, au maximum, se trouver sur la même ligne. A partir du 9ème Sprite, toutes les portions communes aux 9 Sprites disparaissent sur le (ou les) Sprites de plus faible priorité.

### Paramètres :

Matrice des Sprites	Le bit 1 (SI) du registre 1 du VDP définit la taille de la matrice. SI à 0 pour Sprites de 8 × 8 pixels. SI à 1 pour Sprites de 16 × 16 pixels.
Taille des Sprites	Le bit 0 (MAG) du registre 1 du VDP permet de doubler la taille. MAG à 0 pour taille des pixels normale. MAG à 1 pour taille des pixels double.
État des Sprites	Le bit 1 (SPD) du registre 8 du VDP permet de désactiver des Sprites. SPD à 0 = Les Sprites s'affichent. (Sprites désactivés.) SPD à 1 = Les Sprites ne s'affichent pas. (Sprites activés.)

### Collisions :

Lorsque deux Sprites entrent en collision (deux pixels ou plus ont les mêmes coordonnées), le bit 5 du registre de statut passe à 1 (si le bit CC est à 0). Le bit 5 est automatiquement remis à 0 lors d'une lecture du registre de statut 2. Les registres de statut n°3, 4, 5 et 6 indiquent alors les coordonnées du pixel où la collision s'est produite (à condition que les bits MS et LP du registre n°8 du processeur soient tous les deux à 0).

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre de statut 3 :	X7	X6	X5	X4	X3	X2	X1	X0
Registre de statut 4 :	1	1	1	1	1	1	1	X8

Registre de statut 5 :	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Registre de statut 6 :	1	1	1	1	1	1	Y9	Y8

Lorsque le registre n°5 est lu, les registres sont tous les 4 remis à 0.

Une possibilité supplémentaire est offerte au programmeur dans le mode 2. En effet, si le bit CC est mis à 1 dans la table des couleurs de Sprite, alors un « ou » logique (OR) est effectué avec la même ligne du Sprite suivant dans la table des formes lorsqu'il est superposées.

### 9 Sprites :

Lorsque plus de 8 Sprites se trouvent sur la même ligne, le bit 6 (5S) du registre de statut 0 passe à 1. De plus les 5 bits de poids faible du registre de statut 0 donnent le numéro du 9ème Sprite (0 à 31).

### Utilisation :

Pour afficher un Sprite, il faut d'abord le définir dans la table des formes de Sprites.

Pour les Sprites 8 sur 8, il suffit de 8 octets pour définir la forme d'un Sprite, sachant qu'un bit représente un pixel, par exemple, en SCREEN 7, la table des formes de Sprites pourrait commencer par :

		Huit octets pour définir la forme de sprite n°0 qui aura celle-ci :							
		7	~	0					
0F000H	11111111	X	X	X	X	X	X	X	X
0F001H	11000011	X	X					X	X
0F002H	10100101	X		X			X		X
0F003H	10011001	X			X	X			X
0F004H	10011001	X			X	X			X
0F005H	10100101	X		X			X		X
0F006H	11000011	X	X					X	X
0F007H	11111111	X	X	X	X	X	X	X	X

Dans le cas de Sprites 16 sur 16, les choses deviennent un tout petit peu plus compliquées. Le Sprite se décompose en 4 parties :

0	2
1	3

	7 ~ 0	7 6 5 4 3 2 1 0	
0F000H	11111110	X X X X X X X	Huit octets pour le premier quart du Sprite n°0
0F001H	10000010	X X X X X	
0F002H	10001110	X X X X X	
0F003H	10001000	X X X X X	
0F004H	10001000	X X X X X	
0F005H	10001000	X X X X X	
0F006H	10001000	X X X X X	
0F007H	10001000	X X X X X	
0F008H	10001000	X X X X X	Huit octets pour le second quart du Sprite n°0
0F009H	10001000	X X X X X	
0F00AH	10001000	X X X X X	
0F00BH	10001000	X X X X X	
0F00CH	10001000	X X X X X	
0F00DH	10001111	X X X X X X X	
0F00EH	10000000	X X X X X X X	
0F00FH	11111111	X X X X X X X X	Huit octets pour le troisième quart du Sprite n°0
0F010H	01111111	X X X X X X X	
0F011H	01000001	X X X X X X X	
0F012H	01110001	X X X X X X X	
0F013H	00010001	X X X X X X X	
0F014H	00010001	X X X X X X X	
0F015H	00010001	X X X X X X X	
0F016H	00010001	X X X X X X X	
0F017H	00010001	X X X X X X X	Huit octets pour le dernier quart du Sprite n°0
0F018H	00010001	X X X X X X X	
0F019H	00010001	X X X X X X X	
0F01AH	00010001	X X X X X X X	
0F01BH	00010001	X X X X X X X	
0F01CH	00010001	X X X X X X X	
0F01DH	11110001	X X X X X X X	
0F01EH	00000001	X X X X X X X	
0F01FH	11111111	X X X X X X X X	

La réunion des quatre parties donne le résultat ci-dessous :

Première forme  
de Sprite (forme 0)

```

X X X X X X X      X X X X X X X
X                    X      X      X
X          X X X      X X X      X
X          X          X      X
X          X          X      X
X          X          X      X
X          X          X      X
X          X          X      X
X          X          X      X
X          X          X      X
X          X          X      X
X          X X X X X X X X X      X
X                                  X
X X X X X X X X X X X X X X X

```

Il faut ensuite remplir la table des couleurs de Sprites :

	7	6	5	4	3	2	1	0	
0F800H	EC	CC	IC	0	Couleur ligne 0				Couleur de chaque ligne d'un Sprite 8 sur 8 ou 16 sur 16 numéro 0
0F801H	EC	CC	IC	0	Couleur ligne 1				
0F802H	EC	CC	IC	0	Couleur ligne 2				
0F803H	EC	CC	IC	0	Couleur ligne 3				
⋮									
0F80EH	EC	CC	IC	0	Couleur ligne 14				
0F80FH	EC	CC	IC	0	Couleur ligne 15				
0F810H	EC	CC	IC	0	Couleur ligne 0				
0F811H	EC	CC	IC	0	Couleur ligne 1				Idem pour les Sprites suivants
⋮					⋮				

Le bit EC (« Early Clock ») sert à décaler le Sprite de 32 pixels vers la gauche. Ceci permet de faire « entrer » un Sprite à l'écran.

Le bit CC permet de choisir le mode d'affichage des Sprites lors d'une collision :

CC à 0 pour afficher le Sprite prioritaire.

CC à 1 pour faire un « ou » logique (OR) est effectué sur les portions de Sprite superposés.

Le bit IC sert à activer la détection de collision :

IC à 0, active la détection.

IC à 1, désactive la détection.

La couleur est toujours codée sur 4 bits, même en mode d'écran 6 (SCREEN 6) bien que l'on ne dispose que de 4 couleurs (voir le mode d'écran 6 pour plus de précisions, fonction « hardware tiling »).

En SCREEN 8, les valeurs sont fixes et non re-définissables. En voici le détail :

couleur	vert			rouge			bleu			
	R2	R1	R0	G2	G1	G0	B2	B1	B0	
0000B	0	0	0	0	0	0	0	0	0	(Noir)
0001B	0	0	0	0	0	0	0	1	0	(Bleu foncé)
0010B	0	0	0	0	1	1	0	0	0	(Rouge foncé)
0011B	0	0	0	0	1	1	0	1	0	(Violet foncé)
0100B	0	1	1	0	0	0	0	0	0	(Vert foncé)
0101B	0	1	1	0	0	0	0	1	0	(Turquoise)
0110B	0	1	1	0	1	1	0	0	0	(Olive)
0111B	0	1	1	0	1	1	0	1	0	(Gris)
1000B	1	0	0	1	1	1	0	1	0	(Orange clair)
1001B	0	0	0	0	0	0	1	1	1	(Bleu)
1010B	0	0	0	1	1	1	0	0	0	(Rouge)
1011B	0	0	0	1	1	1	1	1	1	(Violet)
1100B	1	1	1	0	0	0	0	0	0	(Vert)
1101B	1	1	1	0	0	0	1	1	1	(Bleu clair)
1110B	1	1	1	1	1	1	0	0	0	(Jaune)
1111B	1	1	1	1	1	1	1	1	1	(Blanc)

Notez que vous pouvez régler la couleur, aussi que les autres bits, pour CHAQUE ligne du Sprite. Notez également qu'on utilise toujours 16 octets, même pour les Sprites 8 sur 8.

Une fois la forme d'un Sprite défini, on peut le manipuler à souhait grâce à la table des attributs de Sprites.

Cette table est composée de 4 octets par Sprite. Soit au total 128 octets pour 32 Sprites affichables (numérotés de 0 à 31) par dessus les plans graphiques, puisqu'il est parfois possible d'afficher un plan avec une entrée vidéo.

La table des attributs de Sprites code les caractéristiques de chaque Sprite sur 4 octets :

	7	6	5	4	3	2	1	0	
01B00H	Ordonnée du Sprite 0 (0-255)								} Attributs du Sprite 0
01B01H	Abscisse du Sprite 0 (0-255)								
01B02H	Numéro de forme de Sprite (0-255)								
01B03H	---- réservé au système ----								
01B04H	Ordonnée du Sprite 1 (0-255)								
⋮	⋮								

L'ordonnée peut varier entre 0 et 255. Si le Sprite sort de l'écran vers la droite ou le bas, le VDP n'affichera que la partie visible du Sprite. En réalité, ce sont les coordonnées de 225 (-31 en complément à 2) à 255, et 0 à 191 sont utilisées. Ceci permet de faire entrer un Sprite dans l'écran (-31 à -1 ou 225 à 255) ou au contraire de le faire sortir (183 à 191 pour un 8 sur 8) petit à petit.

Notez que si l'ordonnée d'un Sprite est à 216 (0D8h), alors tous les Sprites supérieurs deviennent invisibles. Par exemple si l'on place le Sprite 8 sur l'ordonnée 216, alors tous les Sprites de 9 à 31 disparaissent. Ceci est un point important lorsque vous utilisez le registre 23 pour décaler la zone visible de l'écran.

L'abscisse prend une valeur entre 0 et 255. Dans les modes à 512 pixels de large (SCREEN 6 et 7), un pixel Sprite équivaut à deux pixels graphiques. Pour faire sortir le Sprite on utilise les abscisses entre 247 et 255 (pour un Sprite 8 sur 8). Par contre, pour le faire entrer, il faut utiliser le bit EC dans la table des couleurs.

Le numéro de Sprite donne la forme à utiliser à l'affichage. On peut définir 256 Sprites 8 sur 8, ou 64 Sprites 16 sur 16. pour les Sprites 8 sur 8, on choisit simplement le numéro de celui que l'on désire afficher. Pour les Sprites 16 sur 16, on donne le numéro de Sprite multiplié par quatre.

### ***7.11 A propos de la souris et du crayon optique***

Bien que le V9938 soit capable de gérer la souris et le crayon optique, ces deux périphériques de pointage ne sont pas gérés par le processeur vidéo sur MSX (excepté les MSX2 coréens Daewoo CPC-300, CPC-400 et CPC400S pour le crayon optique). Ces fonctions ont d'ailleurs été retirées du V9958. Pour gérer la souris et le crayon optique, il est fortement conseillé d'utiliser la routine du Bios **GTPAD** (000DBh en Main-ROM) ou **NEWPAD** (001ADh en Sub-ROM) ([voir le chapitre 3](#)).

## 8 Le processeur sonore PSG

Le PSG est un générateur de son programmable fabriqué par la firme General Instrument (AY-3-8910) ou par Yamaha (YM2149). L'un ou l'autre équipe tous les MSX. Le PSG est souvent intégré dans le « MSX-Engine » sur les MSX récents.

### 8.1 Caractéristiques

Sur MSX, le PSG n'est pas seulement utilisé pour faire de la musique et des sons. Il sert aussi à contrôler les ports généraux (manette de jeu, souris, Trackball, molettes, tablette graphique, etc). Il donne aussi l'état de la lecture sur cassette et celui du mode Kana ou JIS des MSX japonais.

Pour la musique et les sons, le PSG dispose de trois canaux indépendants dotés chacun d'un générateur de son et d'un générateur de bruit « blanc » mixable et dont l'enveloppe est paramétrable. Notez que le PSG ne produit pas le cliquetis des touches du clavier ni le son Beep.

Le PSG est cadencé à une fréquence de 1,7897725Mhz en interne.

### 8.2 Descriptions du PSG et accès aux registres

Le PSG possède 16 registres dans lesquels on peut lire et écrire le contenu excepté le registre 14 qu'on ne peut que lire. Pour accéder au registre du PSG, le Bios en Main-ROM dispose de deux routines. La première est **WRTPSG** (00093h) qui permet de paramétrer le PSG. La seconde, **RDPSG** (00096h), sert à lire contenu d'un registre.

Les registres du PSG sont aussi accessible directement par les ports E/S : 0A0h, 0A1h et 0A2h.

Pour écrire dans un registre directement par les ports E/S, il faut écrire le numéro de registre au port 0A0h puis la valeur à écrire au port 0A1h. Prenez soin de couper les interruptions pendant l'écriture du numéro de registre et de la valeur.

Pour lire un registre par les ports E/S, il faut écrire le numéro de registre à lire au port 0A0h puis lire la valeur au port 0A1h.

Le Bios en Main-ROM dispose aussi d'une routine, appelée **GICINI** (00090h), qui sert à initialiser le PSG et les données de l'instruction PLAY.

Une fois initialisé, tous les registres de 0 à 13 seront mis à zéro sauf le registre 0 qui aura la valeur 01010101b (55h), le registre 7 qui aura 10111000b (B8h) et le registre 11 qui aura 1011b (0BH) pour la période de l'enveloppe.



### 8.3 Registres de contrôle de fréquence

Les six premiers registres servent à paramétrer la fréquence à générer pour produire un son.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 0 :	8 bits de poids faible de la fréquence de la voix 1							
Registre 1 :	-	-	-	-	4 bits forts de la fréquence voix 1			
Registre 2 :	8 bits de poids faible de la fréquence de la voix 2							
Registre 3 :	-	-	-	-	4 bits forts de la fréquence voix 2			
Registre 4 :	8 bits de poids faible de la fréquence de la voix 3							
Registre 5 :	-	-	-	-	4 bits forts de la fréquence voix 3			

La valeur indiquant la fréquence tient sur 12 bits. Elle est donc répartie sur deux registres. La valeur à écrire s'obtient à l'aide de la formule suivante.

$$\text{Valeur} = \frac{F_i}{16 \times F_n}$$

$F_i$  = Fréquence interne du PSG (1789772,5 Hz)

$F_n$  = Fréquence du son à produire (varie entre 27 et 111960 Hz)

Pour simplifier, voici un tableau de notes de musique obtenues en fonction de la valeur indiquée aux registres 0~1, 2~3 ou 4~5.

Octave Note	1	2	3	4	5	6	7	8
Do (C)	D5Dh	6AFh	357h	1ACh	0D6h	06Bh	035h	01Bh
Do# (C#)	C9Ch	64Eh	327h	194h	0CAh	065h	032h	019h
Ré (D)	BE7h	5F4h	2FAh	17Dh	0BEh	05Fh	030h	018h
Ré# (D#)	B3Ch	59Eh	2CFh	168h	0B4h	05Ah	02Dh	016h
Mi (E)	A9Bh	54Eh	2A7h	153h	0AAh	055h	02Ah	015h
Fa (F)	A02h	501h	281h	140h	0A0h	050h	028h	014h
Fa# (F#)	973h	4BAh	25Dh	12Eh	097h	04Ch	026h	013h
Sol (G)	8EBh	476h	23Bh	11Dh	08Fh	047h	024h	012h
Sol# (G#)	86Bh	436h	21Bh	10Dh	087h	043h	022h	011h
La (A)	7F2h	3F9h	1FDh	0FEh*	07Fh	040h	020h	010h
La# (A#)	780h	3C0h	1E0h	0F0h	078h	03Ch	01Eh	00Fh
Si (B)	714h	38Ah	1C5h	0E3h	071h	039h	01Ch	00Eh

(\*) 0FEh est la note produite par les diapasons.

Par exemple, pour produire la note Do sur l'octave 4 par la voix 1, nous devons écrire 1h dans le registre 1 et ACh dans le registre 0. En pratique, ça donne ceci :

En assembleur :

```
WRTPSG equ 00093h

; --- Entête du fichier
    db 0feh ; Fichier binaire code
    dw START ; Adresse de destination du programme
    dw END ; Adresse de fin du programme
    dw START ; Adresse d'exécution du programme
; ---
    org 0c000h
START:
    ld b,13
PSGini: ld a,b ;
        ld e,0 ; 8 bits de poids faible
        cp 7
        jr nz,NoR7 ; Saut si registre différent de 7
        ld e,10111111b ; Le bit 7 à 1 et le bit 6 à 0
NoR7: call WRTPSG
        djnz PSGini ; boucle pour initialiser les registres

        ld a,0 ; Registre 0
        ld e,0ach ; 8 bits de poids faible
        call WRTPSG
        ld a,1 ; Registre 1
        ld e,1 ; 4 bits de poids fort
        call WRTPSG
        ld a,8 ; Registre 8
        ld e,1100b ; Volume de la voix 1 à 12
        call WRTPSG
        ld a,7 ; Registre 7
        ld e,10111110b ; Active la voix 1
        call WRTPSG
        ret
END:
```

Une fois assemblé et sauvegardé sous le nom « V1501C.BIN », exécuter la routine avec l'instruction suivante.

```
BLOAD"V1501C.BIN",R
```

En Basic :

```
5 ' Initialise les registres sonores du PSG
10 FOR R=0 TO 13
20 IF R=7 THEN SOUND R,&B10111111 ELSE SOUND R,0
30 NEXT
40 ' Joue la note Do sur la voix 1 avec un volume à 12
50 SOUND 0,&hAC ' 8 bits de poids faible dans registre 0
60 SOUND 1,1 ' 4 bits de poids fort dans registre 1
70 SOUND 8,&b1100 ' Réglage du volume de la voix 1 à 12
80 SOUND 7,&b10111110 ' Active le générateur sonore sur la voix 1
```

## 8.4 Registre de contrôle de la fréquence de bruit blanc

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 6 :	-	-	-	Fréquence du générateur de bruit blanc				

La valeur indiquant la fréquence du générateur de bruit blanc tient sur 5 bits. La valeur à écrire s'obtient à l'aide de la formule suivante. Seuls les 5 bits de la valeur sont à écrire dans le registre 6.

$$\text{Valeur} = \frac{F_i}{16 \times F_b}$$

$F_i$  = Fréquence interne du PSG (1789772,5 Hz)

$F_b$  = Fréquence de base du bruit à produire (varie entre 27 et 3608 Hz)

## 8.5 Registre de contrôle des voix et des ports d'E/S du PSG

Le registre sert à activer ou désactiver le générateur de son ainsi que le générateur de bruit. Il sert aussi à régler le sens des ports d'entrées / sorties du PSG.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 7 :	Ports d'E/S du PSG		Désactivation du bruit			Désactivation du son		
	B = 1	A = 0	voix 3	voix 2	voix 1	voix 3	voix 2	voix 1

Notes :

- Pour couper le son sur une voix, on peut mettre le volume de la voix à 0 (registres 8 à 10) ou désactiver le générateur sonore et de bruit de cette voix en mettant les bits correspondants du registre 7 à 1.
- Afin de garantir le bon fonctionnement des ports d'E/S du PSG, le bit 7 du registre 7 doit toujours rester à 1 (port B en mode sortie) et le bit 6 à 0 (port A en mode entrée).
- Il est donc possible d'activer le générateur sonore et le générateur de bruit en même temps sur chaque voix. C'est à dire mixer les deux.

## 8.6 Registres de contrôle de l'amplitude et du volume

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 8 :	-	-	-	V/A	Volume / Amplitude de la voix 1			
Registre 9 :	-	-	-	V/A	Volume / Amplitude de la voix 2			
Registre 10 :	-	-	-	V/A	Volume / Amplitude de la voix 3			

Mettre le bit 4 (V/A) à 0 pour régler le volume sonore de la voix correspondante.

Lorsque le bit 4 (V/A) est mis à 1, le volume variera proportionnellement à la forme de l'enveloppe définie par le registre 13.

## 8.7 Registres de contrôle la forme et de la période de l'enveloppe

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 11 :	8 bits de poids faible de la période de l'enveloppe (T)							
Registre 12 :	8 bits de poids fort de la période de l'enveloppe (T)							
Registre 13 :	-	-	-	-	Forme de l'enveloppe			

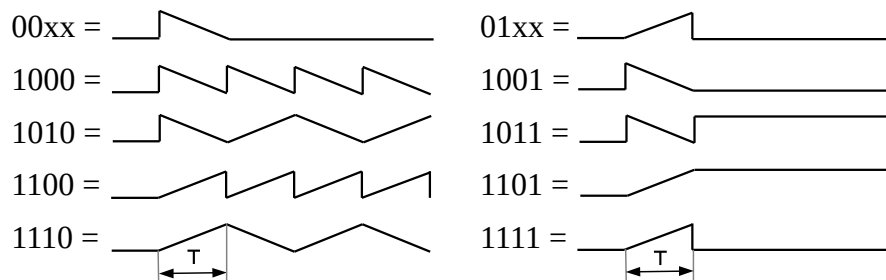
Les registres 11 et 12 déterminent la période de l'enveloppe (T). La période se calcule à l'aide de la formule suivante.

$$\text{Valeur} = \frac{F_i}{256 \times T}$$

$F_i$  = Fréquence interne du PSG (1789772,5 Hz)

$T$  = Période de l'enveloppe (en  $\mu\text{s}$ )

Le registre 13 définit la forme de l'enveloppe. Voici les formes possibles :



Pour les détails :

- Le bit 0 (Hold) spécifie le maintien ou pas le volume en fin de période.
- Le bit 1 (Alternate) spécifie l'inversion ou pas de la forme de l'enveloppe à chaque période paire.
- Le bit 2 (Attack) spécifie l'inversion ou pas de la forme de l'enveloppe.
- Le bit 3 (Continue) spécifie si la forme de l'enveloppe doit être appliquée ou non à chaque période.

## 8.8 Registres des ports d'entrées / sorties du PSG

Le registre 14 se comporte différemment des autres car c'est un registre de statut. On ne peut que le lire afin connaître l'état des broches des ports généraux (aka : Ports Joystick), pour connaître le type de clavier japonais utilisé et pour savoir si une lecture est en cours sur cassette. Le registre 15 sert à contrôler les ports généraux.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 14 :	Port A d'E/S du PSG (à régler toujours en entrée avec le bit 6 du R#7)							
Registre 15 :	Port B d'E/S du PSG (à régler toujours en sortie avec le bit 7 du R#7)							

Détails:

- Registres 14 :

Ce registre permet de lire l'état des broches de 1 à 6 d'un port général du MSX (pour manettes de jeu, souris, tablette graphique, etc), l'état de la touche JIS/Kana des claviers japonais et l'état du signal de lecture sur cassette.

- Bit 0 = État de la broche 1 du port général sélectionné (Haut si joystick)
- Bit 1 = État de la broche 2 du port général sélectionné (Bas si joystick)
- Bit 2 = État de la broche 3 du port général sélectionné (Gauche si joystick)
- Bit 3 = État de la broche 4 du port général sélectionné (Droite si joystick)
- Bit 4 = État de la broche 6 du port général sélectionné (Bouton 1 si joystick)
- Bit 5 = État de la broche 7 du port général sélectionné (Bouton 2 si joystick)
- Bit 6 = 1 pour JIS, 0 pour Kana (valable seulement pour les MSX japonais )
- Bit 7 = CASRD (Signal de lecture sur cassette)

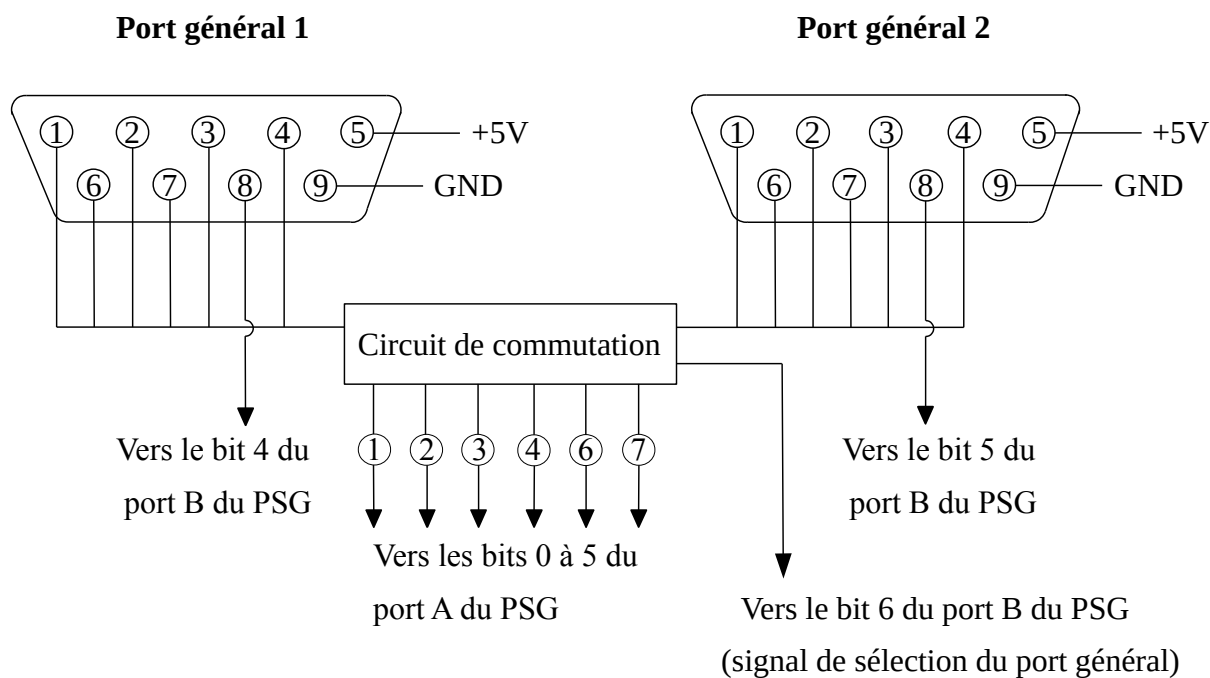
- Registre 15 :

Ce registre permet de contrôler les broches des ports généraux à lire via le registre 14 ainsi que l'état de la LED « code » ou « Kana » selon le type de clavier.

- Bit 0 = Contrôle de la broche 6 du port général 1\*
- Bit 1 = Contrôle de la broche 7 du port général 1\*
- Bit 2 = Contrôle de la broche 6 du port général 2\*
- Bit 3 = Contrôle de la broche 7 du port général 2\*
- Bit 4 = Contrôle de la broche 8 du port général 1 (0 pour mode joystick standard)
- Bit 5 = Contrôle de la broche 8 du port général 2 (0 pour mode joystick standard)
- Bit 6 = Sélection du port général lisible via le registre 14 (1 pour port 2)
- Bit 7 = Contrôle de la LED de la touche « code » ou « Kana ». (1 pour éteindre)

\* Mettre à 1 si le port général est utilisé comme entrée (lecture).

Les broches des ports généraux sont connectés de la façon suivante.



Exemples de fonctionnement :

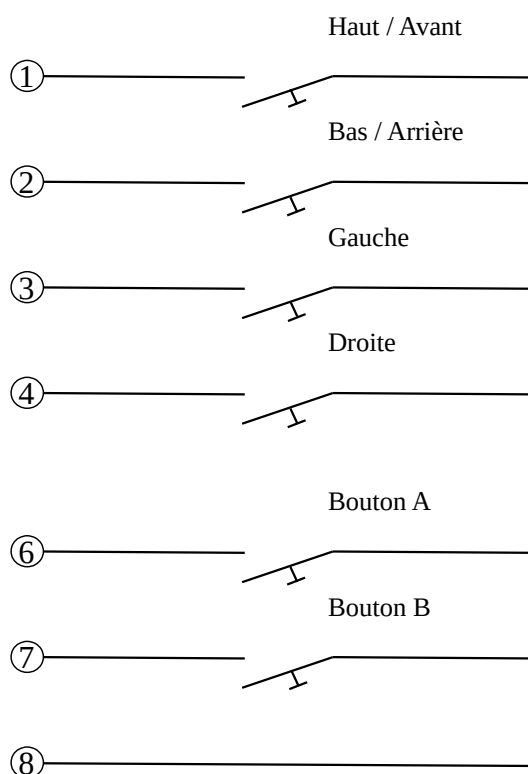
Les exemples suivants ont pour but de montrer le fonctionnement des registres 14 et 15 du PSG. Il est préférable de se servir des routines du Bios autant que possible pour gérer les manettes de jeu, une souris, une tablette graphique ou les molettes ASCII.

## Manettes de jeu

Pour lire une manette de jeu, il faut d'abord lire le registre 15 pour connaître l'état de la LED du clavier afin d'écrire la même valeur au bit 7, mettre le bit 6 à 0 ou 1 selon la manette à lire, les bit 5 et 4 peuvent être tout les deux à 0 et mettre les autres bits (0 à 3) à 1.

Le schéma d'une manette standard (Type B) est le suivant.

Broches

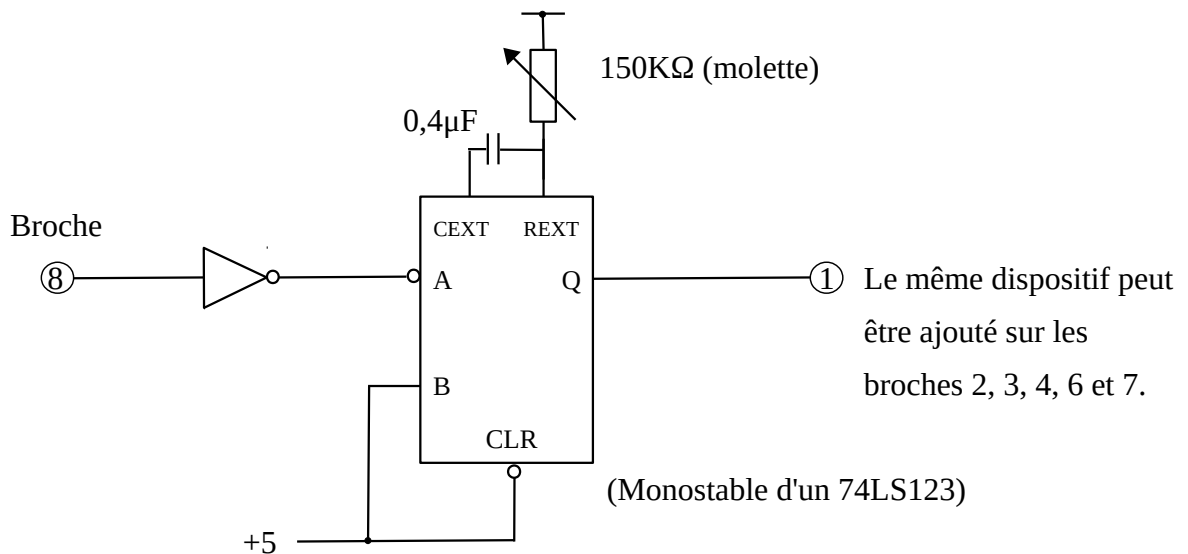


Lorsque l'on presse un bouton, la broche correspondante reçoit le signal zéro de la broche 8. Ainsi, le bit correspondant du registre 14 sera ainsi mis à 0. Notez que, dans ce cas, la broche 8 peut être remplacée par la masse. Il est possible d'ajouter des boutons à la manette en reliant la broche 8 à, par exemple, une puce 74LS157 pour commuter entre la manette standard et les boutons supplémentaires (non standard).

Les routines du Bios pour gérer les manettes de jeu sont **GTSTCK** (000D5h) et **GTTRIG** (000D8h) en Main-ROM. Voir aussi la variable système **TRGFLG** (0F3E8h).

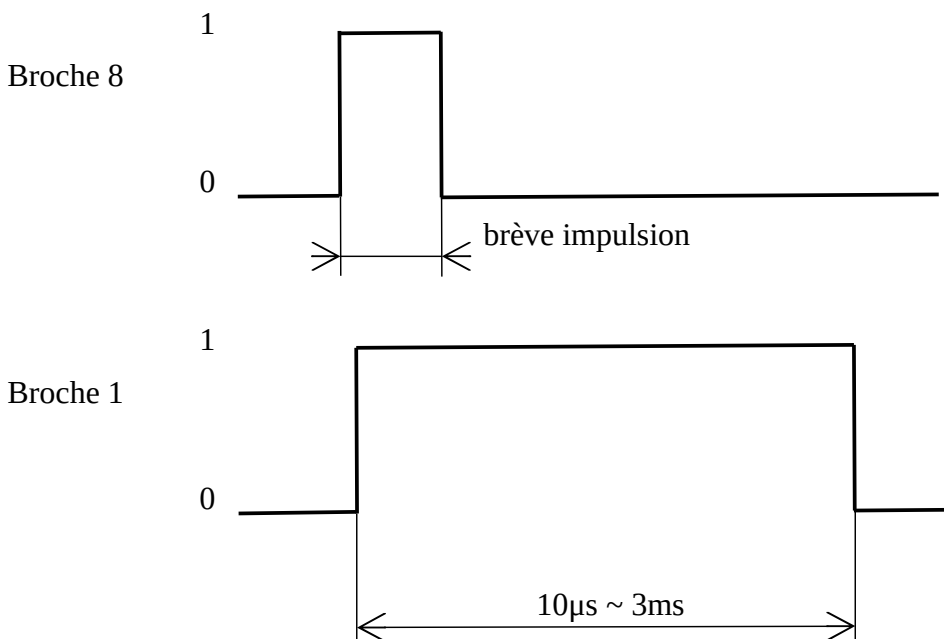
## Molettes de jeu ASCII

Les molettes ASCII n'ont été utilisées que par quelques jeux dont Breack Out édité par ASCII au Japon. Le support de ses molettes a même été retiré du Bios des MSX turbo R. Le schéma est le suivant.



Le but de ce circuit est d'envoyer un bref signal sur la broche 8 pour obtenir un signal sur la broche 1 dont la longueur varie en fonction de la résistance variable contrôlée par la molette. La valeur de la position est ensuite à déduire en fonction de la longueur du signal. Il est possible de connecter jusqu'à 6 molettes par port général donc dans ce cas un bref signal sur la broche 8 déclenchera un signal de longueur différente sur chacune des broches 1, 2, 3, 4, 6 et 7.

Nature du signal :



La longueur du signal permet de déterminer la position.



La routine du Bios pour gérer les molettes ASCII est **GTPDL** (000DEh en Main-ROM). Cette routine n'étant pas implémentée dans les MSX Turbo R, voici une routine de remplacement :

```
; Entrée : A = Numéro de molette (0~12)
; Sortie : A = Position de la molette (0~255)

GTPDL:
    inc     a
    and     a
    rra                     ; Carry=1 pour Port 2
    push    af
    ld      b,a
    xor     a               ; A=0 (et Carry=0)
    scf                     ; Carry=1
PDL_LOOP:
    rla
    djnz    PDL_LOOP
    ld      b,a             ; B=bit correspondant à la broche à 1
    pop     af
    ld      c,10h           ; Broche 8 à 1 sur port général 1
    ld      de,03afh        ; D=11b (broche 6 & 7 port 1)
                                ; E=10101111b (broche 8 port 1 à 0)
    jr      nc,PDL_PORT1
    ld      c,20h           ; Broche 8 à 1 sur port général 2
    ld      de,4c9fh        ; D=1001100b (broche 6 & 7 port 2)
                                ; E=10011111 (broche 8 port 2 à 0)
PORT1:
    ld      a,0fh
    di
    out     (0a0h),a        ; Register 15
    in      a,(0a2h)
    and     e               ; Garde le bit de sélection du port général
    or      d
    or      c
    out     (0a1h),a        ; Envoie de l'impulsion (broche 8 = 1)
    xor     c
    out     (0a1h),a        ; Fin de l'impulsion (broche 8 = 0)

    ld      a,0eh
    out     (0a0h),a        ; Register 14
    ld      c,0
WAIT:
    in      a,(0a2h)
    and     b
    jr      z,DONE          ; Saut si broche de la molette est à 0
    inc     c               ; Incrémente C
    jp      nz,WAIT         ; Saut tant que C n'est pas à 0
    dec     c
DONE:
    ei
    ld      a,c             ; A = 0~255
    ret
```

Note : Cette routine ne fonctionne qu'en mode Z80 (cadencé à 3,579545 MHz).

La souris

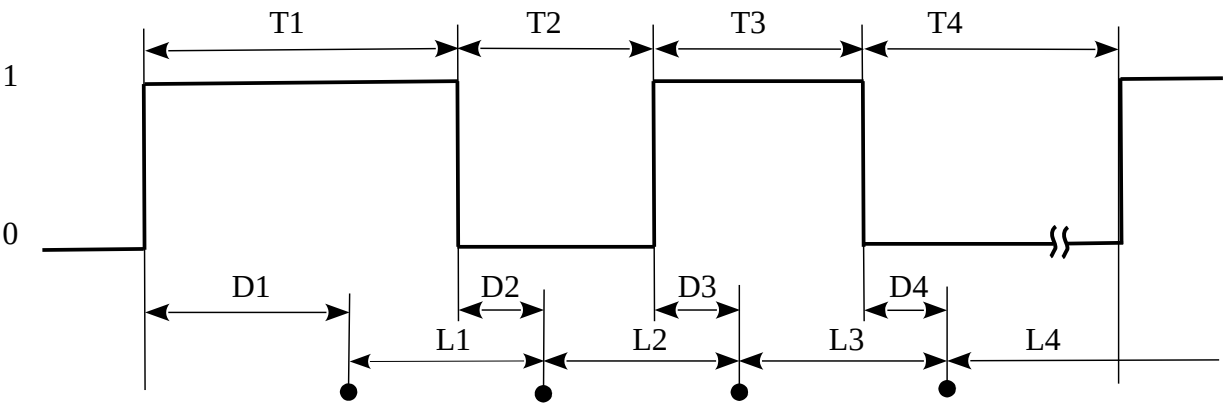
Les boutons de la souris fonctionnent comme ceux d'une manette standard. Pour indiquer le déplacement, la souris dispose de 4 registres de 4 bits. La lecture des registres est commandée par la broche 8. Une première mise à 1 de cette broche commande la lecture du premier registre. La remise à 0 commande la lecture du second registre. Ensuite, la remise à 1 commande la lecture du troisième registre. Puis, la remise à 0 commande la lecture du quatrième registre. Et ainsi de suite. Le tout, en respectant les timings indiqué plus bas.

Contenu des registres :

Broches correspondantes :

Registre	Coordonnée relative	4	3	2	1
1	4 bits de poids fort de l'abscisse	X4	X6	X5	X4
2	4 bits de poids faible de l'abscisse	X3	X2	X1	X0
3	4 bits de poids fort de l'ordonnée	Y4	Y6	Y5	Y4
4	4 bits de poids faible de l'ordonnée	Y3	Y2	Y1	Y0

Timings de lecture des coordonnées en fonction de l'état la broche 8 :



- T1 = Intervalle entre 100 et 150  $\mu$ s
- T2 = Intervalle entre 50 et 150  $\mu$ s (si T2 excède les 150  $\mu$ s, la souris initialisera ses registres.)
- T4 = Intervalle de 300  $\mu$ s minimum durant lequel la souris scrute les coordonnées.
- D1 = 100  $\mu$ s. Temps d'attente avant de pouvoir lire le premier registre.
- D2~D4 = 50  $\mu$ s. Temps d'attente avant de pouvoir lire le registre suivant.
- L1~L4 = Intervalle durant lequel il est possible de lire le registre correspondant.

Chaque lecture du registre 14 donnera les informations de la façon suivantes.

Bit 0 à 3 = 4 bits de poids fort/faible de l'abscisse/ordonnée relative.

Bit 4 = État du bouton gauche

Bit 5 = État du bouton droit

Bit 6 = 1 pour JIS, 0 pour Kana (valable seulement pour les MSX japonais )

Bit 7 = CASRD (Signal de lecture sur cassette)

La routine du Bios pour gérer la souris est **GTPAD** (000DBh en Main-ROM) ou **NEWPAD** (001ADh en Sub-ROM) et **GTTRIG** (000D8h) en Main-ROM pour les boutons mais voici une routine de remplacement pour le principe (page suivante) :

Note : Les souris MSX (compatibles PC-8801 et FM-Towns) possèdent un mode « Joystick » que l'on peut activer en pressant le bouton gauche lors de la mise sous tension de la souris. Ce mode simule une manette de jeu. Il reste activé jusqu'à l'extinction de la souris.

```

; Routine pour lire la souris par accès direct au PSG
;
; Entrée :
;
; Mettre D à 10010011b et E à 00010000b (DE=9310h) si la souris est dans le port 1
; Mettre D à 11101100b et E à 00100000b (DE=EC20h) si la souris est dans le port 2
;
; Sortie :
;
; H = abscisse relative (-127~+128), L = ordonnée relative (-127~+128)
; H et L = 255 si la souris n'est pas trouvée
;
GTMOUS:
    ld    b,30          ; Long délai d'attente pour la première lecture
    call  GTOFS2        ; Lecture des 4 bits de poids fort de l'abscisse relative
    and   0fh
    rlca
    rlca
    rlca
    rlca
    ld    c,a           ; Garde les bits de poids fort dans C
    call  GTOFST        ; Lecture des 4 bits de poids faible de l'abscisse relative
    and   0fh
    or    c             ; Place les bits de poids fort
    ld    h,a           ; Recombine l'abscisse relative dans H

    call  GTOFST        ; Lecture des 4 bits de poids fort de l'ordonnée relative
    and   0fh
    rlca
    rlca
    rlca
    rlca
    ld    c,a           ; Garde les bits de poids fort dans C
    call  GTOFST        ; Lecture des 4 bits de poids faible de l'ordonnée relative
    and   0fh
    or    c             ; Place les bits de poids fort
    ld    l,a           ; Recombine l'ordonnée relative dans L
    ret

GTOFST:
    ld    b,10          ; Court délai d'attente
GTOFS2:
    ld    a,15
    out   (0a0h),a      ; Registre 15
    in    a,(0a1h)
    and   80h           ; Garde le bit de l'état de la LED Code/Kana
    or    d
    out   (0a1h),a
    xor   e             ; Inverse le bit de la broche 8
    ld    d,a

WAIT:   djnz  WAIT      ; Délai d'attente pour lire la valeur au bon timing.

    ld    a,14
    out   (0a0h),a      ; Registre 14
    in    a,(0a2h)
    ret

```

Note : Cette routine ne fonctionne qu'en mode Z80 (cadencé à 3,579545 MHz).

## Track-Ball

Le Track-Ball fonctionnent à peu près de la même manière que la souris. Les boutons fonctionnent comme ceux d'une manette standard. Quant à la broche 8, une première mise à 1 commande l'envoi l'abscisse relatif sur 4 bits. Ensuite, la mise à 0 de la broche 8 commandera l'envoi de l'ordonnée relatif sur 4 bits.

Chaque lecture du registre 14 donnera les informations de la façon suivantes.

Bit 0 à 3 = Abscisse/ordonnée relative sur 4 bits. La valeur lu doit être interprétée comme suit.

Valeur lue	Coordonnée correspondante
0000b	-8
⋮	⋮
0111b	-1
1000b	0
⋮	⋮
1111b	7

Bit 4 = État du bouton 1

Bit 5 = État du bouton 2

Bit 6 = 1 pour JIS, 0 pour Kana (valable seulement pour les MSX japonais )

Bit 7 = CASRD (Signal de lecture sur cassette)

Note : Certains modèles ont trois boutons. Ce bouton à le même effet que les boutons 1 et 2 pressés en même temps.

La routine du Bios pour gérer le Track-Ball est **GTPAD** (000DBh en Main-ROM) ou **NEWPAD** (001ADh en Sub-ROM) et **GTTRIG** (000D8h) en Main-ROM

## 9 Le MSX-Music

Le MSX-Music est une option du standard pour jouer les musiques au moyen d'un un générateur de son programmable à synthèse FM conçu et fabriqué par Yamaha dont la référence est YM2413 . Cette puce est appelée « OPLL » car c'est une version très simplifiée de l'OPL2 (YM3812). La cartouche Sound Blaster 1.0 utilise l'OPL2. Un matériel ou un logiciel compatible MSX-Music doit comporter le logo suivant.



### 9.1 Caractéristiques

L'OPLL du MSX-Music peut produire jusqu'à 9 voix FM ou 6 voix + boîte à rythme. Les sons sont produit par deux opérateurs. Les 24 registres de l'OPLL sont accessible par écritures aux ports d'entrée/sortie 07Ch et 07Dh. L'MSX-Music comprend aussi une ROM de 16Ko qui contient un Bios et des instructions Basic étendues. L'option MSX-Music peut être ajoutée au MSX en interne ou en externe. Les registres d'une version externe sont accessible aussi par écritures aux adresses 07FF4h et 7FF5h dans le Slot de la cartouche. Le but de ces adresses est de contrôler un OPPL externe indépendamment d'un autre en interne.

La sélection du registre à modifier se fait via le port 07Ch (ou l'adresse 07FF4h) et l'écriture des données via le port 07Dh (ou l'adresse 07FF5h). Il faut respecter un délais de 3,36 µs entre chaque sélection de registre et un délais de 23,52 µs entre chaque écriture de donnée.

La norme du MSX-Music stipule que les ports d'accès à un OPLL externe doivent être désactivés par défaut. Pour les activer, il faut écrire une valeur avec le bit 0 à 1 à l'adresse 7FF6h (dans le Slot correspondant). Attention, assurez-vous qu'il s'agit bien d'un MSX-Music externe avant d'écrire cette valeur. En effet, car si l'on écrit cette valeur entre 7FF0h et 7FFFh sur un MSX2+ Panasonic FS-A1WX ou FS-A1WSX, cela aura pour effet de désactiver la ROM.

Note : Selon la norme, l'accès aux registres par adressage mémoire devrait être effectif sur tous les MSX-Music externes mais, dans les faits, la plupart des MSX-Music qui ont été vendus hors du Japon sont des versions non officielles (FM-PAQ, FM-PAK, etc) qui n'ont pas cette fonctionnalité.

## 9.2 Registres de l'OPLL

Les informations sur les registres sont là à titre informatif. Normalement, il faut utiliser le [FM Bios](#) pour accéder à l'OPLL.

### Registres de contrôle sonore de l'utilisateur :

Ces huit registres permettent de créer son propre instrument.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 0 :	AM	VIB	EGTYP	KSR	MULTIPLE				(opérateur 0)
Registre 1 :	AM	VIB	EGTYP	KSR	MULTIPLE				(opérateur 1)
Registre 2 :	KSL (op. 0)		Total Level Modulator						
Registre 3 :	KSL (op. 1)		-	DC	DM	Feedback			
Registre 4 :	AR				DR				(opérateur 0)
Registre 5 :	AR				DR				(opérateur 1)
Registre 6 :	SL				RR				(opérateur 0)
Registre 7 :	SL				RR				(opérateur 1)

**MULTIPLE** = Contrôle de l'échantillon de l'onde - Relation harmonique.

**KSR** (Key Scale Rate) = Taux de .

**EGTYP** = 0 pour une tonalité de percussion, 1 pour une tonalité soutenue.

**VIB** (Vibrato) = Active / Désactive le vibrato.

**AM** (Amplitude Modulation) = Active / Désactive la modulation d'amplitude.

**Total Level Modulator** = Niveau total de l'onde modulée. Contrôle de l'index de modulation.

**KSL** (Key Scale Level) = Niveau de « Key scale ».

**FR** (Feedback Rate) = Constante de retour FM.

**DM** (Distortion Module) = Distorsion de l'onde modulée de l'enveloppe. (Rectification par ondes planes.)

**DC** (Distortion Carrier) = Distorsion de l'onde porteuse de l'enveloppe. (Rectification par ondes planes.)

**DR** (Decay Rate) = Contrôle de la fréquence de changement d'enveloppe de décomposition.

**AR** (Attack Rate) = Contrôle du changement du taux d'enveloppe d'attaque.

**RR** (Release Rate) = Contrôle du niveau de changement d'enveloppe de sortie.

**SL** (Sustain Level) = Indication de décroissance - niveau de maintien.

L'opérateur 0 est un opérateur modulateur et l'opérateur 1 est un opérateur de type porteur.

### Registre de contrôle de la boîte à rythme:

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 14 :	-		R	BD	SD	TOM	T-CT	HH

HH = Active / Désactive la charleston (High-Hat) de la boîte à rythme.

T-CT = Active / Désactive la cymbale du haut (Top Cymbale) de la boîte à rythme.

TOM = Active / Désactive le Tom de la boîte à rythme.

SD = Active / Désactive la caisse claire de la boîte à rythme.

BD = Active / Désactive la grosse caisse de la boîte à rythme.

R = Active / Désactive la boîte à rythme. Désactive les voix 6 à 8 pour la boîte à rythme.

### Registre de test de l'OPLL :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Registre 15 :	TEST							

TEST = Donnée de test de l'OPLL.

### Registre de réglages :

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
Registre 16 à 24 :	Bits 0 ~ 7 de F-NUM								(voix 0 à 8)
Registre 32 à 40 :	-		SUS	KEY	BLOCK			Bit 8 de F-NUM	(voix 0 à 8)
Registre 48 à 56 :	INST				VOL				(voix 0 à 8)

F-NUM = Indication de la fréquence.

BLOCK = Réglage de l'octave.

KEY = Touche maintenue ou pas.

SUS = Active / Désactive le Sustain.

VOL = Réglage du volume sonore.

INST = Sélection de l'instrument.

0 = Original*	4 = Flute	8 = Orgue	12 = Vibraphone
1 = Violon	5 = Clarinette	9 = Corne	13 = Basse de synthétiseur
2 = Guitare	6 = Hautbois	10 = Synthétiseur	14 = Basse acoustique
3 = Piano	7 = Trompette	11 = Clavecin	15 = Guitare électrique

(\*) Instrument créé par l'utilisateur via les registres 0 à 7.



## 10 MSX-JE

MSX-JE est le dispositif standard de saisie de texte japonais qui sert entre autre à convertir les Kana-Kanji sur MSX. Ce qui n'était pas normalisée avant l'apparition de MSX-JE. Les traitements de texte japonais, etc, possédaient leur propre système de saisie.

MSX-JE est apparu avec le traitement de texte "Japanese MSX WRITE" d'ASCII sortie en 1986 et depuis lors, la plupart des logiciels nécessitant une entrée japonaise sur MSX sont compatibles avec MSX-JE et sont conformes à cette norme.

Par exemple, le traitement de texte japonais "Bunshosaku Zaemon" de Sony (HBS-B012D) est vendu sans aucun système de saisie additionnel. Pour l'utiliser, il est donc nécessaire d'avoir une machine MSX avec le MSX-JE installé.

MSX-JE est installé dans la plupart des MSX2+. Le Kanji BASIC de la version 3 du MSX-BASIC est compatible avec MSX-JE donc, la saisie de texte japonais peut être faite sous BASIC.

## 11 Le système des disques et MSX-DOS

Dans ce livre, je vais vous présenter uniquement les routines et la structure du MSX-DOS en mémoire. Je ne parlerai pas de l'utilisation du système en tant que tel. Pour cela, veuillez vous référer à un manuel d'utilisation du MSX-DOS.

Le MSX-DOS nécessite un minimum de 64Ko de RAM. Dans l'environnement du MSX-DOS, la mémoire est répartie de la façon suivante.

Base du MSX-DOS	00000h
Zone libre (TPA)	00100h
Pile (SP)	Adresse indiquée à 00006h
MSX-DOS.SYS	
Cache des FAT	
Cache des fichiers	
E/S du secteur	
Cache du secteur à lire/écrire	
Zone de travail dynamique des disques	0F1C9h
Zone de communication fixe des disques	0F380h
Variables et zone de travail du système	0FFFFh

La zone de travail des contrôleurs de disques est composée d'une zone de variables de quelques octets suivi du DPB (Disk Parameter Block) de chacun des disques du contrôleur et ce pour chaque contrôleur. Un DPB est une zone de 21 octets comprenant les paramètres d'un disque.

La zone du cache de secteur prendra la taille du plus grand secteur présent sur un des disques installés.

## 11.1 Base du MSX-DOS (System Scratch Area)

Cette zone en mémoire, qui s'étend de 0 à FFh, se met en place lorsqu'on lance une commande du MSX-DOS (un fichier .COM). Elle contient des données utilisées pour communiquer avec les programmes utilisateurs par l'intermédiaire de paramètres entrées par la ligne de commande ainsi que des adresses de saut pour les appels système ou dans le programme de l'utilisateur. Voici cette zone en détail.

Adresse	Nom	Longueur	Fonction
0000h		3	Routine de réinitialisation du MSX-DOS. Permet de quitter votre programme
0003h	IOBYTE	1	IOByte. (À prendre en compte que pour les programmes et DOS compatibles CP/M.)
0004h		1	Lecteur / Utilisateur par défaut. (À prendre en compte que pour les programmes et DOS compatibles CP/M.)
0005h	BDOS	3	Appel à une routine BDOS qui se trouve au tout début du MSX-DOS.SYS résident en mémoire juste après la zone réservée à l'utilisateur pour la commande (TPA). Par conséquent, l'adresse de fin de la TPA sera indiquée par le contenu de l'adresse 0006h-1 (voir ci-dessous).
0006h		2	Adresse du premier octet derrière la TPA. Lorsqu'une commande du MSX-DOS (fichier COM) est lancée, elle se charge à l'adresse 0100h et sa taille peut aller jusqu'à l'adresse qui précède celle indiquée ici.
0008h	RST08	4	Appel à une routine crée par l'utilisateur. (RST 8)
000Ch	RDSLT	4	Appel à la routine RDSLT (Main-ROM) qui sert à lire un octet dans un Slot.
0010h	RST10	4	Appel à une routine crée par l'utilisateur. (RST 10)
0014h	WRSLT		Appel à la routine WRSLT (Main-ROM) qui sert à écrire un octet dans un Slot.
0018h	RST18	4	Appel à une routine crée par l'utilisateur. (RST 18)
001Ch	CALSLT	4	Même routine que CALSLT (Main-ROM) qui sert à faire un appel inter-Slot.
0020h	RST20	4	Appel à une routine crée par l'utilisateur. (RST 20)
0024h	ENASLT	4	Appel à la routine ENASLT (Main-ROM) qui sert à sélectionner un Slot.
0028h	RST28	4	Appel à une routine crée par l'utilisateur. (RST 28)
002Ch	-	4	Réservé.
0030h	CALLF	4	Appel à la routine CALLF (Main-ROM) qui sert à faire un appel inter-Slot via RST 30.
0038h		3	Routine d'interruption.
003Bh		33	Routine de changement de Slot utilisée par le système.
005Ch	FCB1	16	Cette zone contient une copie des 16 premiers octets du FCB correspondant au premier nom de fichier indiqué comme

			paramètre derrière la commande entrée. Le premier octet indique le numéro de lecteur, les 11 suivants le nom du fichier, etc...
006Ch	FCB2	16	Cette zone contient une copie des 16 premiers octets du FCB correspondant au deuxième nom de fichier indiqué comme paramètre derrière la commande entrée. Le premier octet indique le numéro de lecteur, les 11 suivants le nom du fichier, etc...
007Ch		4	Contient des zéros.
0080h	DTA/DMA	127	Zone DTA par défaut. Cette zone contient tous paramètres spécifiés derrière la commande entrée. Le premier octet indique le nombre de caractères utilisés pour les paramètres. L'octet en fin de chaîne est 00h souvent précédé de 0dh.

---

## 11.2 Zone fixe de travail, Hooks et des variables du MSX-DOS1 et du Disk-BASIC

Cette zone contient aussi des Hooks spécifiques au MSX-DOS.

Adresse	Nom	Long.	Fonction
0F1C9h		24	Routine pour afficher une chaîne de caractères précédée du caractère « \$ ». Entrée : DE = Adresse du début de la chaîne.
0F1E2h		6	Routine pour interrompre un programme en cas d'erreur.
0F1E8h		12	Routine d'appel à une routine en Main-RAM sur la page 1 lorsque la Disk-ROM est sélectionnée. Entrée : Contenu de l'adresse pointée par HL = Adresse de la routine à appeler.
0F1F4h		3	Routine de changement de nom de fichier. Entrée : HL = Adresse du premier caractère du nom de fichier.
0F1F7h		4	Nom de périphérique « PRN »
0F1FBh		4	Nom de périphérique « LST »
0F1FFh		4	Nom de périphérique « NUL »
0F203h		4	Nom de périphérique « AUX »
0F207h		4	Nom de périphérique « CON ».
0F20Bh		8	Nom trouvé au périphérique. (PRN, LST, NUL, etc)
0F213h		3	Nom d'extension trouvé au périphérique. (PRN, LST, NUL, etc)
0F216h		1	Périphérique actuelle. PRN = -5, LST = -4, ..., CON = -1.
0F221h		2	Date du fichier trouvé.
0F223h		2	Heure du fichier trouvé.

0F22Bh		12	Tableau contenant le nombre de jours de chaque mois de l'année sans tenir compte des années bissextiles.
0F237h	BUFINP	4	Zone de travail de la fonction 10 du BDOS.
0F23Bh		1	Indicateur pour envoyer les caractères vers l'imprimante. 0 = Ne pas envoyer ; Autre valeur = Caractères vers l'imprimante.
0F23Dh		2	Adresse actuelle du FCB. (MSX-DOS)
0F23Fh		4	Numéro du secteur actuel du disque.
0F243h		2	Pointeur à l'adresse du DPB du lecteur actuel.
0F245h		1	Numéro relatif du secteur dans le dossier. Utilisé par les routines SEARCH FIRST et NEXT. (Ajouter le numéro du premier secteur du répertoire pour connaître le numéro de secteur réel.)
0F246h		1	Numéro du lecteur en cours. (0 = « A: », 1 = « B: », etc, 0FFh = Pas de lecteur)
0F247h		1	Numéro du lecteur par défaut. ( 0 = « A: », 1 = « B: », etc, 0FFh = Pas de lecteur)
0F248h		3	Jour, Mois et Année. 0 = 1980, 1 = 1981, 2 = 1982, etc. pour l'année. Valeurs stockées lorsque la commande DATE ou TIME est utilisée. (MSX-DOS)
0F24Ch		2	Heure et minute. Valeurs stockées lorsque la commande DATE ou TIME est utilisée. (MSX-DOS)
0F24Eh		1	Jour de la semaine. 0 = Dimanche, 1 = Lundi, 2 = Mardi, etc. Valeur stockées lorsque la commande DATE ou TIME est utilisée. (MSX-DOS)
0F24Fh	H.PROMPT	3	Hook appelé avant l'affichage du message « Insert diskette for drive » de l'émulation d'un deuxième lecteur. Lors de l'appel à ce Hook, le registre A contient la lettre du nom de lecteur en code ASCII.
0F252h	H.BDOS	3	Hook appelé avant l'exécution d'une routine BDOS. (évaluation de la fonction appelée.)
0F255h		3	Hook appelé au début de la routine qui vérifie le nom de périphérique dans le nom de fichier.
0F258h	H.SEARCHDIR	3	Hook appelé au début de la routine de recherche du fichier suivant. (Utile pour changer 0F2DCh afin de trouver aussi les fichiers cachés.)
0F25Bh		3	Hook appelé au début de la routine d'obtention du fichier suivant.
0F25Eh	H.NEXTDIRSECT	3	Hook appelé au début de la routine de passage au fichier suivant.
0F261h		3	Hook appelé au début de la routine de vérification du FCB du disque et du nom de fichier.

0F264h	H.OPEN	3	Hook appelé avant de vérifier la taille du fichier que l'on ouvre.
0F267h		3	Hook appelé au début de la routine d'obtention de la dernière FAT pour vérifier si le disque a changé ou pas.
0F26Ah	H.GETDPB	3	Hook appelé au début de la routine d'obtention du DPB du disque actuel.
0F26Dh	H.CLOSE	3	Hook appelé avant la fermeture un fichier.
0F270h	H.RDABS	3	Hook appelé au début de la routine de lecture de secteur(s). Utilisé au début de la routine RDABS du BDOS (2Fh).
0F273h	H.DSKERR	3	Hook appelé avant la routine traitement d'erreur lors d'accès au disque. (Lors d'un changement de disque par exemple)
0F276h		3	Hook appelé au début de la routine d'écriture d'un fichier.
0F279h	H.WRABS	3	Hook appelé au début de la routine d'écriture de secteur(s). Utilisé au début de la routine WRABS du BDOS (30h).
0F27Ch		3	Hook appelé au début de la routine interne de multiplication. (HL et BC = DE*BC)
0F27Fh		3	Hook appelé au début de la routine interne de division. (BC = BC/DE, HL = Reste)
0F282h		3	Hook appelé au début de la routine d'obtention du cluster absolu.
0F285h		3	Hook appelé au début de la routine d'obtention du cluster absolu suivant.
0F288h		3	Hook appelé au début de la routine de lecture partielle de secteur.
0F28Bh		3	Hook appelé au début de la routine d'écriture partielle de secteur.
0F28Eh		3	Hook appelé au début de la routine de lecture d'un enregistrement sur un disque.
0F291h		3	Hook appelé en fin de la routine de lecture d'un enregistrement sur un disque.
0F294h		3	Hook appelé après la lecture d'un enregistrement sur un disque.
0F297h		3	Hook appelé au début de la routine d'un enregistrement
0F29Ah		3	Hook appelé au début de la routine de l'écriture d'un enregistrement sur un disque.
0F29Dh		3	Hook appelé en fin de la routine de l'écriture d'un enregistrement sur un disque.
0F2A0h		3	Hook appelé au début de la routine de calcul séquentiels de secteurs.
0F2A3h		3	Hook appelé au début de la routine pour obtenir le numéro de secteur du cluster.
0F2A6h		3	Hook appelé au début de la routine d'allocation de chaîne dans la FAT.
0F2A9h		3	Hook appelé au début de la routine de réactualisation de chaîne dans la FAT.

0F2ACh	H.BUFINP	3	Hook appelé au début de la routine BUFINP qui ajoute une donnée au cache du clavier.
0F2AFh	H.CONTOUT	3	Hook appelé au début de la routine CONTOUT qui vérifie les codes CTRL.
0F2B2h		3	Hook appelé au début de la routine de lecture de la date et de l'heure d'un fichier.
0F2B5h		3	Hook appelé au début de la routine de calcul des années bissextiles.
0F2B8h	DIRENT	1	Position du fichier ouvert sur le disque. (0FFh = Aucune)
0F2B9h		11	Nom du fichier ouvert avec son extension (à 0F2C1h).
0F2C4h		1	Attributs du fichier ouvert. Si le bit 7 est 1 alors les fichiers avec un attribut NOT à 0 pourront être ouverts.
0F2D3h		2	
0F2D5h		4	Premier cluster.
F2DCh		1	Si différent de zéro, les fichiers avec un attribut NOT à 0 ne seront pas ignorés. (le bit 7 à F2C4h annule ceci!)
0F2DEh		1	Résultat de l'enregistrement.
0F2DFh		1	Indicateur du secteur relatif actuel dans le cluster (0 = Aucun)
0F2E0h		1	Indicateur de lecture. (0 = lu)
0F2E1h	DRIVE	1	Lecteur en cours de lecture / écriture absolue (0 ~ 7). 0 = Lecteur « A », 1 = Lecteur « B », etc.
0F2E2h		2	Compteur auxiliaire pour lecteur de disque.
0F2E4h		4	Emplacement du premier enregistrement.
0F2E8h		2	Nombre d'enregistrement.
0F2EAh		2	Cluster relatif du fichier actuel.
0F2ECh		2	Cluster du fichier actuel.
0F2EEh		2	Secteur relatif de l'enregistrement.
0F2F0h		2	Cluster relatif en fin de fichier après un enregistrement.
0F2F2h		2	
0F2F4h		2	
0F2F8h		2	Premier secteur transmettre.
0F2FAh		2	Dernier secteur à transmettre.
0F2FCh		2	Nombre de secteur à transmettre.
0F2FEh		1	Première entrée libre trouvée. (0FFh = Aucune)
0F2FFh		1	Indicateur de lecture (0) / écriture (1).

0F300h		2	Adresse de la routine de gestion des erreurs de disque.
0F302h		2	Adresse de la routine gestionnaire d'annulation.
0F304h		2	Stockage temporaire du registre pointeur de la pile.
0F306h		1	Lecteur utilisé par défaut (0 ~ 7). 0 = « A: », 1 = « B: », etc.
0F30Dh	RAWFLG	1	Indicateur de vérification. 0 = Vérification terminée ; Autre valeur = Vérification en cours.
0F30Eh		1	Indicateur du format de la date. 0 = AAMMJJ, 1= MMJJAA, 2= JJMMAA.
0F30Fh	KANJTABLE	4	Deux paires de limites pour les premiers octets des caractères Shift-JIS.
0F313h	DISKVE	1	Version de la Disk-ROM. 0 = Disk-ROM v.1.x, 2xH = Disk-ROM v.2.x.
0F323h	ERRADR	2	Adresse de la routine de gestion des erreurs de disque.
0F325h	BREAKV	2	Adresse de la routine de gestion de CTRL+C.
0F327h		5	Hook appelé par la routine AUXINP.
0F32Ch		5	Hook appelé par la routine AUXOUT.
0F331h		5	Hook appelé par les fonctions BDOS.
0F336h		1	Indicateur de touche pressée. Contient 0FFh lorsqu'une touche est pressée. Contient 03h si CTRL+STOP sont pressées.
0F337h		1	Indicateur de touche pressée. Contient le code ASCII de la touche pressée. Contient 03h si CTRL+STOP sont pressées.
0F338h		1	Drapeau pour indiquer la présence de l'horloge interne. 0 = horloge interne, Autre valeur = Pas d'horloge interne.
0F339h		6	Zone de travail de la puce de l'horloge interne.
0F33Fh		1	Nom du second lecteur virtuel (0 ~ 7). 0 = « A: », 1 = « B: », etc.
0F340h	REBOOT	1	0 si le DOS doit être initialisé au démarrage.
0F341h	RAMAD0	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 0.
0F342h	RAMAD1	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 1.
0F343h	RAMAD2	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 2. (Pour le Disk-Basic aussi)
0F344h	RAMAD3	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 3. (Pour le Disk-Basic aussi)
0F345h		1	Utilisé lors du calcul du nombre de caches disponibles.
0F346h		1	Indicateur pour savoir si le système a démarré sous DOS ou pas (0). Par conséquent, CALL SYSTEM est possible si différent de 0.
0F347h	NMBDRV	1	Nombre de disque logique connectés (0 ~ 7).



0F348h	MASTERS	1	Numéro du Slot qui contient la Disk-ROM maitre.
0F349h	HIMSAV	2	Adresse de début de la zone de travail des disques.
0F34Bh		2	Adresse de fin du noyau du DOS. (0000h par défaut sous Basic.) Note : L'adresse de début du noyau est stockée à l'adresse 0006h. Le fichier MSX-DOS.SYS est chargé dans cette zone.
0F34Dh	SECBUF	2	Adresse du cache temporaire utilisé lors de lecture / écriture dans la FAT.
0F34Fh	BUFFER	2	Adresse du cache temporaire utilisé lors de lecture / écriture dans les secteurs de données.
0F351h	DIRBUF	2	Adresse du cache temporaire utilisé lors de lecture / écriture d'un secteur. Utilisé par les instructions DSKI\$ & DSKO\$ du Disk-Basic.
0F353h	FCBBASE	2	Adresse du FCB du fichier actuel.
0F355h	DPBLIST	2	Adresses du DPB de chaque disque. DPBLST=Adresse du DPB des disques « A: » DPBLST+2=Adresse du DPB du disque « B: » DPBLST+4=Adresse du DPB du disque « C: » DPBLST+6=Adresse du DPB du disque « D: » DPBLST+8=Adresse du DPB du disque « E: » DPBLST+10=Adresse du DPB du disque « F: » DPBLST+12=Adresse du DPB du disque « G: » DPBLST+14=Adresse du DPB du disque « H: ».
0F365h		3	Routine de lecture de l'état des Slot primaires. Sortie : A = État des Slot primaires.
0F368h	SETROM	3	Routine pour sélectionner la Disk-ROM sur la plage 04000h~7FFFh. (MSX-DOS seulement.)
0F36Bh	SETRAM	3	Routine pour sélectionner la Main-RAM sur la plage 04000h~7FFFh. (MSX-DOS seulement.)
0F36Eh	SLTMOV	3	Routine de copie de la Disk-ROM vers la Main-RAM. (Actif sous MSX-DOS seulement.) Entrée : HL = Source ; DE = Destination ; BC = Longueur.
0F371h	AUXINP	3	Routine de lecture du périphérique auxiliaire. Sortie : A = Valeur lue. 01Ah si CTRL+Z.
0F374h	AUXOUT	3	Routine d'écriture au périphérique auxiliaire. Entrée : A = Octet à envoyer.
0F377h	BLDCHK	3	Routine BLOAD du Disk-Basic. (Contient JP 0000h sous MSX-DOS)
0F37Ah	BSVCHK	3	Routine BSAVE du Disk-Basic. (Contient JP 0000h sous MSX-DOS)
0F37Dh	ROMBDOS	3	Routine BDOS de la Disk-ROM. (Peut-être appelée sous environnement Basic.) Entrée : C = Numéro de la Routine à appeler.
0F459h		100	Contient la commande entrée sous MSX-DOS.

0F85Fh	MAXFIL	1	Nombre maximum de fichiers autorisés. Modifié par l'instruction MAXFILES du Disk-Basic.
0F860h	FILTAB	2	Pointeur sur l'adresse des données du fichier.
0F862h	NULBUF	2	Pointeur du cache des instructions SAVE et LOAD du Disk-Basic.
0F864h	PTRFIL	2	Pointeur sur les données du fichier sélectionné.
0F866h	RUNFLG	1	Indicateur d'exécution pour l'instruction LOAD. 0 si le programme a été exécuté par l'option R.
0F866h	FILNAM	11	Nom du fichier d'une instruction du Disk-Basic.
0F871h	FILNM2	11	Nom du second fichier des instructions du Disk-Basic (NAME, COPY, MOVE, etc).
0F87Ch	NLONLY	1	Indicateur différent de 0 lors d'un chargement de programme. Utilisé par les instructions BSAVE et CREATE.
0F87Dh	SAVEND	2	Adresse de fin de l'instruction BSAVE du Disk-Basic.
0FB21h	DRVINF	1	Nombre de lecteur physique contrôlé par la première Disk-ROM. (8 maximum)
0FB22h	DRVINF+1	1	Numéro de Slot de la Disk-ROM des premiers disques installés.
0FB23h	DRVINF+2	1	Nombre de lecteur physique contrôlé par la seconde Disk-ROM. (8 disques maximum au total)
0FB24h	DRVINF+3	1	Numéro de Slot de la Disk-ROM des disques installés en second.
0FB25h	DRVINF+4	1	Nombre de lecteur physique contrôlé par la troisième Disk-ROM. (8 disques maximum au total)
0FB26h	DRVINF+5	1	Numéro de Slot de la Disk-ROM des disques installés en troisième.
0FB27h	DRVINF+6	1	Nombre de lecteur physique contrôlé par la quatrième Disk-ROM. (8 disques maximum au total)
0FB28h	DRVINF+7	1	Numéro de Slot de la Disk-ROM des disques installés en quatrième.
0FB29h	DRVINT	12	Numéro de Slot et adresse de chaque gestionnaire d'interruption des interfaces de disque.
0FCBFh	SAVENT	2	Adresse de début spécifiée par l'instruction BSAVE du Disk-Basic.
0FD99h	DEVICE	1	Cet octet passe à 255 si la touche SHIFT a été pressée au démarrage pour empêcher l'installation des disques. Sinon, cet octet reste à 0.

## 12 Le système des disques et MSX-DOS 2

Dans ce livre, je vais vous présenter uniquement les routines et la structure du MSX-DOS 2 en mémoire. Je ne parlerai pas de l'utilisation du système en tant que tel. Pour cela, veuillez vous référer à un manuel d'utilisation du MSX-DOS 2.

Les principales nouveautés du MSX-DOS 2 sont la gestion des dossiers, des variables, du Memory Mapper, de fichiers temporaires « Pipe » et la redirection. La redirection peut-être désactivée avec la commande SET REDIR= OFF. Dès lors, le COMMAND2.COM fonctionnera comme un MSX-DOS1 et CP/M standard.

Il y a trois versions officielle du MSX-DOS 2 :

1. MSX-DOS 2 v2.20 : Cette version a été développée par ASCII. Il existe des versions pirates vendues en Europe et ailleurs. Les différences principales sont le Mapper de la ROM et les Kanji qui ne sont pas présents.
2. MSX-DOS 2 v2.30 : Cette version était vendue avec le MSX turbo R FS-A1ST.
3. MSX-DOS 2 v2.31 : Cette version était vendue avec le MSX turbo R FS-A1GT.

Il y a aussi Nextor qui est un MSX-DOS 2 amélioré par [Konamiman](#) qui possède le code source du MSX-DOS 2. Veuillez vous référer à sa documentation pour toutes informations.

Le MSX-DOS 2 (v2.20) nécessite un MSX2 ou plus récent avec un minimum de 128Ko de RAM. Dans l’environnement du MSX-DOS 2, la mémoire est répartie de la façon suivante.

Base du MSX-DOS	00000h
Zone libre (TPA)	00100h
Pile (SP)	Adresse indiquée à 00006h
MSX-DOS2.SYS	
Cache des FAT	
Cache des fichiers	
E/S du secteur	
Cache du secteur à lire/écrire	
Zone de travail dynamique des disques	0F1C9h
Zone de communication fixe des disques	0F380h
Variables et zone de travail du système	0FFFFh

La zone de travail des contrôleurs de disques est composée d'une zone de variables de quelques octets suivi du DPB (Disk Parameter Block) de chacun des disques du contrôleur et ce pour chaque contrôleur. Un DPB est une zone de 21 octets comprenant les paramètre d'un disque.

La zone du cache de secteur prendra taille du plus grand secteur présent sur un des disque installé.

Le MSX-DOS 2 possède une routine pour manipuler le Memory Mapper. On y accès par la routine du Bios étendu. (voir au chapitre 2.4 sur le [Memory Mapper](#) page 17).

**12.1 Base du MSX-DOS 2 (System Scratch Area)**

Cette zone en mémoire, qui s’étend de 0 à FFh, se met en place lorsqu'on lance une commande du MSX-DOS (un fichier .COM). Elle contient des données utilisées pour communiquer avec les programmes utilisateurs par l’intermédiaire de paramètres entrées par la ligne de commande ainsi que des adresses de saut pour les appels système ou dans le programme de l'utilisateur. Voici cette zone en détail.

Adresse	Nom	Longueur	Fonction
0000h		3	Routine de réinitialisation du MSX-DOS2. Permet de quitter votre programme

0003h	-	2	Réservée.
0005h	BDOS	3	Appel à une routine BDOS qui se trouve au tout début du MSX-DOS.SYS résident en mémoire juste après la zone réservée à l'utilisateur pour la commande (TPA). Par conséquent, l'adresse de fin de la TPA sera indiquée par le contenu de l'adresse 0006h -1 (voir ci-dessous).
0006h		2	Adresse du premier octet derrière la TPA. Lorsqu'une commande du MSX-DOS (fichier COM) est lancée, elle se charge à l'adresse 0100h et sa taille peut aller jusqu'à l'adresse qui précède celle indiquée ici.
0008h	RST08	4	Appel à une routine créée par l'utilisateur. (RST 8)
000Ch	RDSLT	4	Appel à la routine RDSLT (Main-ROM) qui sert à lire un octet dans un Slot.
0010h	RST10	4	Appel à une routine créée par l'utilisateur. (RST 10)
0014h	WRSLT		Appel à la routine WRSLT (Main-ROM) qui sert à écrire un octet dans un Slot.
0018h	RST18	4	Appel à une routine créée par l'utilisateur. (RST 18)
001Ch	CALSLT	4	Même routine que CALSLT (Main-ROM) qui sert à faire un appel inter-Slot.
0020h	RST20	4	Appel à une routine créée par l'utilisateur. (RST 20)
0024h	ENASLT	4	Appel à la routine ENASLT (Main-ROM) qui sert à sélectionner un Slot.
0028h	RST28	4	Appel à une routine créée par l'utilisateur. (RST 28)
002Ch	-	4	Réservée.
0030h	CALLF	4	Appel à la routine CALLF (Main-ROM) qui sert à faire un appel inter-Slot via RST 30.
0038h		3	Routine d'interruption.
003Bh		33	Routine de changement de Slot utilisée par le système.
005Ch	FCB1	16	Cette zone contient une copie des 16 premiers octets du FCB correspondant au premier nom de fichier indiqué comme paramètre derrière la commande entrée. Le premier octet indique le numéro de lecteur, les 11 suivants le nom du fichier, etc...
006Ch	FCB2	16	Cette zone contient une copie des 16 premiers octets du FCB correspondant au deuxième nom de fichier indiqué comme paramètre derrière la commande entrée. Le premier octet indique le numéro de lecteur, les 11 suivants le nom du fichier, etc...
007Ch		4	Contient des zéros.
0080h	DTA/DMA	127	Zone DTA par défaut. Cette zone contient tous paramètres spécifiés derrière la commande entrée. Le premier octet indique le nombre de caractères utilisés pour les paramètres. L'octet en fin de chaine est 00h souvent précédé de 0dh.

---

## 12.2 Zone fixe de travail, Hooks et des variables du MSX-DOS2 et du Disk-BASIC 2

Cette zone est assez différente de celle du MSX-DOS1. Seules quelques parties sont communes dont les Hooks.

Adresse	Nom	Long.	Fonction
0F1D3h		3	Saut à la routine ?
0F1D6h		3	Saut à la routine ?
0F1D9h		3	Saut à la routine ?
0F1DCh		3	Saut à la routine ?
0F1DFh		3	Saut à la routine ?
0F1E2h		3	Saut à la routine ?
0F1E5h		3	Saut à la routine de gestion d'interruption. (appelé seulement lors du traitement des fonctions BDOS).
0F1E8h		3	Saut à la routine RDSLT du Bios (000Ch). (appelé seulement lors du traitement des fonctions BDOS).
0F1EBh		3	Saut à la routine WRSLT du Bios (0014h). (appelé seulement lors du traitement des fonctions BDOS).
0F1EEh		3	Saut à la routine CALSLT du Bios (001Ch). (appelé seulement lors du traitement des fonctions BDOS).
0F1F1h		3	Saut à la routine ENASLT du Bios (0024h). (appelé seulement lors du traitement des fonctions BDOS).
0F1F4h		3	Saut à la routine CALLF du Bios (0030h). (appelé seulement lors du traitement des fonctions BDOS).
0F1F7h		3	Saut à la routine pour mettre en mode DOS. (pages 0 et 2 avec le système)
0F1FAh		3	Saut à la routine pour mettre en mode utilisateur (« User Mode »).
0F1FDh		3	Routine de sélection de la ROM du MSX-DOS 2 sur la plage 00000h~03FFFh.
0F200h		3	Saut à la routine qui alloue une page de Memory Mapper.
0F203h		3	Saut à la routine qui libère une page de Memory Mapper.
0F206h		3	Saut à la routine de lecture d'un octet sur une page de Memory Mapper. Entrée : HL = Adresse, A = Numéro de page.
0F209h		3	Saut à la routine d'écriture d'un octet sur une page de Memory Mapper. Entrée : HL = Adresse, A = Numéro de page. Sortie : E = Octet lu.
0F20Ch		3	Saut à la routine d'appel de routine se trouvant sur une page de Memory Mapper. Entrée : IX = adresse de la routine à appeler, IYh = page.
0F20Fh		3	Saut à la routine d'appel de routine se trouvant sur une page

de Memory Mapper.

Entrée : Octet après l'adresse de l'instruction CALL IX = numéro de page. Les deux octets suivants = adresse de la routine à appeler.

0F212h	3	Saut à la routine de sélection d'une page de Memory Mapper. Entrée : HL = adresse quelconque dans la plage mémoire sur laquelle la page sera sélectionnée, A = numéro de page à sélectionner.
0F215h	3	Saut à la routine indiquant la page de Memory Mapper actuelle. Entrée : HL = adresse quelconque dans la plage mémoire sur laquelle on cherche à connaître la page actuellement sélectionnée. Sortie : A = numéro de page.
0F218h	3	Routine de sélection d'une page de Memory Mapper sur la plage mémoire 0000h~03FFFh. Entrée : A = numéro de page.
0F21Bh	3	Routine d'écriture dans la variable système du numéro page du Memory Mapper actuellement sélectionnée sur la plage mémoire 00000h~03FFFh. Entrée : A = numéro de page.
0F21Eh	3	Routine de sélection d'une page de Memory Mapper sur la plage mémoire 04000h~07FFFh. Entrée : A = numéro de page.
0F221h	3	Routine d'écriture dans la variable système du numéro page du Memory Mapper actuellement sélectionnée sur la plage mémoire 04000h~07FFFh. Entrée : A = numéro de page.
0F224h	3	Routine de sélection d'une page de Memory Mapper sur la plage mémoire 8000h~0BFFFh. Entrée : A = numéro de page.
0F227h	3	Routine d'écriture dans la variable système du numéro page du Memory Mapper actuellement sélectionnée sur la plage mémoire 08000h~0BFFFh. Entrée : A = numéro de page.
0F22Ah	3	Pas de routine pour la plage mémoire 3. (saut à un RET.)
0F22Dh	3	Saut à la routine indiquant la page de Memory Mapper se trouvant dans la plage mémoire 3. Entrée : A = numéro de page.
0F23Ch	1	Numéro du lecteur logique actuelle.
0F23Dh	2	Adresse actuelle de la PTA. (0080h par défaut)
0F23Fh	4	Numéro du secteur actuel du disque.
0F243h	2	Pointeur à l'adresse du DPB du lecteur actuel.
0F245h	1	Numéro relatif du secteur dans le dossier. Utilisé par les routines SEARCH FIRST et NEXT. (Ajouter le numéro du

			premier secteur du répertoire pour connaître le numéro de secteur réel.)
0F246h		1	Numéro du lecteur dont le dossier a été lu. ( 0 = « A: », 1 = « B: », etc, 0FFh = Pas de lecteur)
0F247h		1	Numéro du lecteur par défaut. ( 0 = « A: », 1 = « B: », etc, 0FFh = Pas de lecteur)
0F24Fh		3	Hook appelé avant ?
0F252h	H.BDOS	3	Hook appelé avant l'exécution d'une routine BDOS. Page 0 - map bloc (F2D0h) ; Page 2 - map block (F2CFh)
0F255h		3	Hook appelé avant ?
0F258h		3	Hook appelé avant ?
0F261h	H.	3	Hook appelé au début de la fonction 02h du BDOS.
0F283h		2	Adresse de la TPA. (MSX-DOS 2.33)
0F2B6h		1	Indicateur disponibles sous MSX-DOS 2.33 bit 0~2 = Réservés, bit 3 = VDP rapide OFF, bit 4 = TPA propre OFF (0F2B3h = adresse de la TPA), bit 5 = RESET ON, bit 6 = BUSRESET OFF, bit 7 = REBOOT ON.
0F2B9h		1	Compteur ?
0F2C0h		5	Second Hook de la routine d'interruption (utilisé par la Disk-ROM).
0F2C5h		2	Adresse de la table de correspondance.
0F2C7h		1	Page du Memory Mapper sélectionnée sur la plage 0000h~3FFFh.
0F2C8h		1	Page du Memory Mapper sélectionnée sur la plage 4000h~7FFFh.
0F2C9h		1	Page du Memory Mapper sélectionnée sur la plage 8000h~BFFFh.
0F2CAh		1	Page du Memory Mapper sélectionnée sur la plage C000h~FFFFh.
0F2CBh		1	Le contenu de F2C7h est copié ici lors de l'exécution de la routine BDOS.
0F2CCh		1	Le contenu de F2C8h est copié ici lors de l'exécution de la routine BDOS.
0F2CDh		1	Le contenu de F2C9h est copié ici lors de l'exécution de la routine BDOS.
0F2CEh		1	Le contenu de F2CAh est copié ici lors de l'exécution de la routine BDOS.
0F2CFh		1	Numéro de la dernière page disponible dans le Memory



			Mapper primaire. Cache utilisé lors d'un appel à une routines BDOS sur la page 2. (zone de travail temporaire.)
0F2D0h		1	Numéro de la dernière page disponible dans le Memory Mapper primaire. Cache utilisé lors d'un appel à une routines BDOS sur la page 0. (zone de travail temporaire.)
0F2D1h		2	Adresse
0F2D3h		2	Adresse
0F2D5h		5	Deuxième Hook de la routine FCALL (FFCAh) du Bios étendu.
0F2DAh		4	Adresse de la routine d'appel des fonctions BDOS de la deuxième ROM.
0F2DEh		4	Adresse de la routine d'appel des fonctions BDOS de la ROM du DOS2. actuellement sélectionnée
0F2E0h		1	Numéro des Slot primaires sélectionnés sur chaque plage.
0F2E6h		2	Cache utilisé pour registre de stockage temporaire IX
0F2E8h		2	Cache utilisé pour le stockage temporaire du registre SP.
0F2EAh		1	Statut des Slot primaires après l'exécution d'une fonction de BDOS.
0F2EBh		1	Statut des Slot secondaires après l'exécution d'une fonction de BDOS.
0F2ECh		1	Indicateur pour vérifier l'état du disque. (0 = OFF, autre valeur = ON.)
0F2FBh		2	Pointeur vers un cache temporaire utilisé pendant l'interprétation d'un code d'erreur.
0F2FDh		1	Adresse du cache temporaire de la fonction « Explain Error Code ».
0F2FEh		2	Pointeur du haut la pile du DOS2.
0F300h		13	?
0F30Dh		1	Vérification du disque (00h = OFF, 01h = ON.)
0F30Fh	KANJTABLE	4	Deux paires de limites pour les premiers octets des caractères Shift-JIS.
0F313h		1	Version de la Disk-ROM. 0 = Disk-ROM v.1.x, 2xh = Disk-ROM v.2.x.
0F314h		1	Page du Memory Mapper sur la plage 0000h~3FFFh.
0F315h		1	Page du Memory Mapper sur la plage 4000h~7FFFh.
0F316h		1	Page du Memory Mapper sur la plage 8000h~BFFFh.
0F317h		1	Page du Memory Mapper sur la plage C000h~FFFFh.
0F323h	DISKVE	2	Adresse du pointeur de la routine de gestion des erreurs de disque.

0F325h	BREAKV	2	Adresse du pointeur de la routine de gestion de CTRL+C
0F339h		6	Zone de travail de la puce de l'horloge interne.
0F33Fh		1	Nom du second lecteur virtuel (0 ~ 7). 0 = « A: », 1 = « B: », etc.
0F340h	REBOOT	1	0 si le DOS doit être initialisé au démarrage.
0F341h	RAMAD0	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 0.
0F342h	RAMAD1	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 1.
0F343h	RAMAD2	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 2.
0F344h	RAMAD3	1	Numéro du Slot qui contient la RAM principale du DOS de la plage 3.
0F345h		1	Nombre de caches disponible trouvé lors du calcul.
0F346h		1	Indicateur pour savoir si le système a démarré sous DOS ou pas (0). Par conséquent, CALL SYSTEM est possible si différent de 0.
0F347h	NMBDRV	1	Nombre de disque logique connectés (0 ~ 7).
0F348h	MASTERS	1	Numéro du Slot qui contient la Disk-ROM maitre.
0F349h		2	Adresse de début de la zone de travail de la Disk-ROM.
0F34Bh		2	Adresse de début de la zone disponible en RAM principale du DOS. Le fichier COMMAND.COM se charge à cette adresse. (0000h par défaut.)
0F34Dh	SECBUF	2	Adresse du cache temporaire utilisé lors de lecture / écriture dans la FAT.
0F34Fh	BUFFER	2	Adresse du cache temporaire utilisé lors de lecture / écriture dans les secteurs de données.
0F351h	DIRBUF	2	Adresse du cache temporaire utilisé lors de lecture / écriture d'un secteur. Utilisé par les instructions DSKI\$ & DSKO\$ du Disk-Basic.
0F353h	FCBBASE	2	Adresse du FCB du fichier actuel.
0F355h	DPBLST	2	Adresses du DPB de chaque disque. DPBLST=Adresse du DPB des disques « A: » DPBLST+2=Adresse du DPB du disque « B: » DPBLST+4=Adresse du DPB du disque « C: » DPBLST+6=Adresse du DPB du disque « D: » DPBLST+8=Adresse du DPB du disque « E: » DPBLST+10=Adresse du DPB du disque « F: » DPBLST+12=Adresse du DPB du disque « G: » DPBLST+14=Adresse du DPB du disque « H: ».
0F365h		3	Routine de lecture de l'état des Slot primaires. Sortie : A = État des Slot primaires.

0F377h		3	Routine d'appel des fonctions BDOS en page 0. (Peut-être appelée sous environnement Basic.)
0F37Ah		3	Deuxième routine d'appel des fonctions BDOS en page 0.
0F37Dh		3	Routine BDOS de la Disk-ROM (du MSX-DOS2). Entrée : C = Numéro de la Routine à appeler.
0F459h		100	Contient la dernière commande entrée sous MSX-DOS.
0F85Fh	MAXFIL	1	Nombre maximum de fichiers autorisé. Modifié par l'instruction MAXFILES du Disk-Basic.
0F860h	FILTAB	2	Pointeur sur l'adresse des données du fichier.
0F862h	NULBUF	2	Pointeur du cache des instructions SAVE et LOAD du Disk-Basic.
0F864h	PTRFIL	2	Pointeur sur les données du fichier sélectionné.
0F866h	RUNFLG	1	Indicateur d'exécution pour l'instruction LOAD. 0 si le programme a été exécuté par l'option R.
0F866h	FILNAM	11	Nom du fichier d'une instruction du Disk-Basic.
0F871h	FILNM2	11	Nom du second fichier des instructions du Disk-Basic (NAME, COPY, MOVE, etc).
0F87Ch	NLONLY	1	Indicateur différent de 0 lors d'un chargement de programme. Utilisé par les instructions BSAVE et CREATE.
0F87Dh	SAVEND	2	Adresse de fin de l'instruction BSAVE du Disk-Basic.
0FB21h	DRVINF	1	Nombre de lecteur physique contrôlé par la première Disk-ROM. (8 maximum)
0FB22h	DRVINF+1	1	Numéro de Slot de la Disk-ROM des premiers disques installés.
0FB23h	DRVINF+2	1	Nombre de lecteur physique contrôlé par la seconde Disk-ROM. (8 disques maximum au total)
0FB24h	DRVINF+3	1	Numéro de Slot de la Disk-ROM des disques installés en second.
0FB25h	DRVINF+4	1	Nombre de lecteur physique contrôlé par la troisième Disk-ROM. (8 disques maximum au total)
0FB26h	DRVINF+5	1	Numéro de Slot de la Disk-ROM des disques installés en troisième.
0FB27h	DRVINF+6	1	Nombre de lecteur physique contrôlé par la quatrième Disk-ROM. (8 disques maximum au total)
0FB28h	DRVINF+7	1	Numéro de Slot de la Disk-ROM des disques installés en quatrième.
0FB29h	DRVINT	12	Numéro de Slot et adresse de chaque gestionnaire d'interruption des interfaces de disque.
0FCBFh	SAVENT	2	Adresse de début spécifiée par l'instruction BSAVE du

## Disk-Basic.

0FD99h	DEVICE	1	Cet octet passe à 255 si la touche SHIFT a été pressée au démarrage pour empêcher l'installation des disques. Sinon, cet octet reste à 0.
0FFCAh	EXTBIO	5	<p>Appel à une routine d'un Bios du étendu.  Entrée : D = Identifiant de l'extension.  0 = Tous,  4 = Memory Mapper,  8 = MSX-Modem ou RS-232C,  10 = MSX-Audio,  16 = MSX-JE,  17 = Kanji driver,  34 = UNAPI (Konamiman),  77 = Memman,  132 = <math>\mu</math>-driver (Yoshikazu Yamamoto),  241 = MultiMente (Mogu),  255 = Système.</p> <p>E = Numéro de la routine à appeler.  Sortie : Dépend de la routine appelée.  Note : Cette routine est présente si le bit 0 de HOKVLD (0FB20h) est à 1. (Voir le paragraphe 3.6 sur <a href="#">le Bios étendu</a> pour plus de précision)</p>
0FFCFh	DISINT	5	<p>Appel à une routine qui permet de désactiver les interruptions pour un long moment. Il s'agit en fait de la routine suivante.</p> <pre> disint:     ld    d,0                ;broad-cast command     ld    e,2                ;'disable interrupt'     call  extbio     ret </pre>
0FFD4h	ENAINIT	5	<p>Appel à une routine qui permet de ré-activer les interruptions coupées avec DISINT. Il s'agit en fait de la routine suivante.</p> <pre> enaint:     ld    d,0                ;broad-cast command     ld    e,3                ;'enable interrupt'     call  extbio     ret </pre>
0FFD9h		14	Zone de travail pour DISINT / ENAINIT.

---

## 13 Applications types

Ce chapitre est destiné à soulager le programmeur en lui proposant des solutions « prêtes à l'emploi » aux problèmes qu'on rencontre dans la majorité des applications. Muni de cette « bibliothèque de base », le programmeur ne perdra plus de temps en recherches inutiles et pourra se consacrer entièrement à la programmation de son application.

### 13.1 Revenir à l'interpréteur Basic

Il est souvent utile de pouvoir retourner à l'interpréteur Basic depuis un programme en langage machine, autrement que par le RET du Z80. Pour cela il faut :

1. Sélectionner la Main-ROM en pages 0 et 1.
2. Effectuer un saut en 0409Bh.

Ce qui donnerait, en assembleur, les lignes de programme suivantes :

```
MNROM    equ    0fcc1h
ENASLT    equ    00024h
RETURN    equ    0409bh
;
BACK2BASIC:
    ld      a, (MNROM)
    ld      hl, 0          ; Bios en page 0
    call    ENASLT
    ld      hl, 04000h      ; Basic en page 1
    call    ENASLT
    jp      RETURN
```

Lorsque l'on travaille sous Basic, ou que l'on ne touche pas aux Slot, il suffit d'effectuer le JP 0409Bh.

### 13.2 Quel type de MSX ?

Pour savoir si l'ordinateur utilisé est un MSX1, MSX2 ou autre, il suffit d'examiner le contenu de la case mémoire 0002Dh en Main-ROM.

contenu de 02DH	ordinateur
0	MSX1
1	MSX2
2	MSX2+
3	MSX turbo R
4 à 255	non défini

Attention : La Main-ROM ne se trouve pas toujours dans un Slot primaire. Il faut lire le contenu de la variable système MNROM (0FCC1h) pour savoir si le Slot de la Main-ROM se trouve sur un Slot secondaire ou non.

### 13.3 Le « PRINT » en assembleur

Je vous propose une petite routine très simple et fort utile en mode texte. Elle permet d'afficher une chaîne de caractères à l'écran en 32, 40 ou 80 colonnes :

```
CHPUT:    equ    00A2h
PRINT:    ld      hl, DATA
          call   ECRIRE
          ret
ECRIRE:    ld      a, (hl)
          cp     0
          ret    z
          call   CHPUT
          inc    hl
          jr     ECRIRE
DATA:     db      'COUCOU', 0
```

Dans cet exemple, l'adresse du premier caractère de la chaîne à afficher est chargée dans HL. Puis la routine ECRIRE est appelée pour afficher le texte. Le 0 en fin de la chaîne sert à indiquer la fin. Cette routine comme le PRINT du Basic gère les codes CTRL et ESC.

### 13.4 Système à disquettes ou pas ?

Pour savoir si l'ordinateur possède une interface de disque ou non, il faut vérifier l'état du premier octet du Hook « H.PHYD » à l'adresse 0FFA7h. Celui-ci contient 0C9h (code du RET en Z80) lorsqu'il n'est pas modifié par une Disk-ROM. Donc si 0FFA7h contient autre chose que 0C9h (qui devrait être 0F7h, code du RST 30), alors l'ordinateur possède un lecteur de disquette (intégré ou pas) ou une interface disque.

De plus, lorsqu'un lecteur de disquette est présent, certaines variables système sont modifiées :

Adresse	Contenu
0FB21h	Nombre de lecteurs connectés au premier contrôleur (1 ou 2)
0FB22h	Numéro de Slot du premier contrôleur
0FB23h	Nombre de lecteurs connectés au second contrôleur (1 ou 2)
0FB24h	Numéro de Slot du second contrôleur
0FB25h	Nombre de lecteurs connectés au troisième contrôleur (1 ou 2)
0FB26h	Numéro de Slot du troisième contrôleur
0FB27h	Nombre de lecteurs connectés au quatrième contrôleur (1 ou 2)
0FB28h	Numéro de Slot du quatrième contrôleur

Un MSX peut donc gérer jusqu'à 8 disques simultanément (4 contrôleurs de lecteurs de disquettes, chacun d'eux ne pouvant gérer que deux lecteurs). S'il y a moins de quatre contrôleurs, les variables système correspondantes au contrôleur inexistant contiennent 0.

En cas d'absence de lecteur de disquette, cette zone mémoire n'est pas initialisée et peut contenir donc n'importe quoi.

### 13.5 Traiter les erreurs liées aux disquettes

Il est fort désagréable de voir s'afficher, au milieu d'un programme en français, un message du genre « Disk write protected » ou « Disk offline ». Il est heureusement possible de remédier à ce genre de désagrément. Voici la marche à suivre :

1. Modifier l'adresse contenue en 0F323h et 0F324h pour détourner le traitement des erreurs disque vers votre propre routine.
2. Votre routine pourra envoyer les messages adéquats sachant que l'accumulateur contient le numéro du lecteur, alors que le bit 0 du registre C indique si l'erreur s'est produite lors d'une lecture (bit à 0) ou d'une écriture (bit à 1). Les bits 1, 2 et 3 indiquent le type d'erreur selon la table suivante :

3 2 1	Type d'erreur
0 0 0	Protégée contre l'écriture (« Write protected »)
0 0 1	Disque pas prêt (« Disk offline »)
0 1 0	Erreur de CRC (« CRC error »)
0 1 1	Erreur de lecture (« Seek error »)
1 0 0	Fichier non trouvé (« File not found »)
1 0 1	Erreur d'écriture (« Write error »)
1 1 0	Autre erreur

3. Votre routine doit rendre le contrôle au DOS par un RET après avoir chargé le registre C avec l'action à exécuter (0 = Ignorer ; 1 = Recommencer ; 2 = Annuler).

### 13.6 Faire de la musique en assembleur

La méthode classique pour faire de la musique en assembleur consiste à utiliser une interruption pour charger les registres du PSG 8910 à intervalle régulier.

Le programmeur peut aussi utiliser le système des queues musicales. Malheureusement, cette méthode est difficile à mettre en œuvre.

Il existe une méthode très simple pour exécuter en langage machine l'équivalent de l'instruction PLAY du Basic. Mais cette méthode N'EST PAS GARANTIE officiellement par Microsoft et ASCII. Quelque peu empirique, elle est cependant officieusement garantie, puisque ASCII a affirmé qu'elle ne serait pas modifiée. Cela se vérifie pour l'instant puisque tous les MSX vendus en France à ce jour exécutent parfaitement cette routine. Vous utilisez cette routine à vos risques et périls quant à une éventuelle compatibilité avec les MSX à venir.

Pour exécuter cette routine, il faut charger le registre double HL avec l'adresse du premier octet de la chaîne de caractères à jouer (la chaîne est la même qui pour un PLAY en Basic), puis appeler la routine située à l'adresse 073E5h en Main-ROM. Voici un exemple en assembleur :

```
PLAY    equ    073e5h
DB      0feh          ; Entête pour les
```

```

        DW      START,END,START  ; fichiers binaires

        org     0c000h

START:   ld      hl,DATA
        call   PLAY
        ret

;
DATA:   db      22h,'O5L6DABDACF',22h
        db      22h,'O3L4DEDE',22h
        db      22h,'O7L8ADEFGE',22h,0
END:

```

Attention à ne pas oublier le 0 en fin de chaîne. Dans l'exemple ci-dessus, on présume que la Main-ROM est sélectionnée sur la plage 4000h~7FFFh.

### 13.7 Passage de paramètres du Basic au langage machine

Dans la plupart des applications, il n'est pas indispensable que l'intégralité du programme soit écrit en langage machine. Il suffit bien souvent de quelques routines en Z80 pour donner un look professionnel à un programme en Basic. Dans ces cas, le problème de l'échange d'informations entre Basic et langage machine se pose.

La méthode classique consiste à définir une zone de communication à laquelle on accède sous Basic par les instructions POKE et PEEK. Cette méthode présente plusieurs inconvénients : l'exécution est plutôt lente, les instructions sont longues et occupent un précieux espace mémoire, toutes les données sont enregistrées deux fois (donc perte très importante de place mémoire).

La fonction USR du Basic (que la majorité des utilisateurs, y compris moi, emploient comme une simple instruction CALL) permet l'échange automatique de données dans les 2 sens. Voyons dans le détail le fonctionnement de la fonction USR. Plusieurs cas se présentent suivant la nature du paramètre à passer :

Le paramètre entre parenthèse est un entier :

C'est le cas le plus simple. La fonction USR met 2 dans la variable système VALTYP (0F663h) pour indiquer qu'il s'agit d'un entier. Le registre A contient aussi cette valeur (2) à l'entrée de votre routine. La valeur du paramètre se trouve sur 2 octets dans la variable système DAC+2 (0F7F6H+2). Le registre HL donne l'adresse de la variable DAC.

Prenons un exemple : Nous désirons réaliser une routine en langage machine qui multipliera automatiquement un entier positif par 2.

```

VALTYP   equ     0f663h

        org     0c000h

DEBUT:   cp      2                ; Avons-nous bien affaire à un entier ?
        ret     nz                ; Retour au Basic si ce n'est pas le cas

;

        inc     hl
        inc     hl                ; HL = position de l'entier
        xor     a                ; carry à 0

```



```

        rl      (hl)          ; décalage 8 bits de poids faible(*2)
        inc     hl
        rl      (hl)          ; décalage 8 bits de poids faible(*2)
        ret     nc            ; Retour si il n'y a pas de dépassement
;
ERROR:   ld      hl, DATA     ; HL = Adresse message
        call    ECRIRE        ; à l'écran
        ret

NONENT:  ld      hl, DATA     ; HL = Adresse message
        call    ECRIRE        ; à l'écran
        ret

DATA:    db      'Overflow', 0
;
;
ECRIRE:  call    00A2h
        inc     hl
        ld      a, (hl)
        cp      0
        ret     z
        jr      ECRIRE

```

Une fois la routine assemblée, lancer la par le programme Basic suivant :

```

10 CLEAR, &HC000:BLOAD"Routine.bin"
20 A%=2:DEFUSR=&HC000:PRINT USR(A%)

```

#### Le paramètre est en simple précision :

C'est un peu plus compliqué. La fonction USR met 4 dans la variable système VALTYP (0F663h) pour indiquer qu'il s'agit d'une valeur en simple précision. Le registre A contient aussi cette valeur (4) à l'entrée de votre routine. La valeur du paramètre se trouve sur 4 octets dans la variable système DAC (0F7F6h). Le registre HL donne l'adresse de la variable DAC.

Format d'une valeur en simple précision :

Octet 1 :

SM	SE	EX5	EX4	EX3	EX2	EX1	EX0
----	----	-----	-----	-----	-----	-----	-----

EX0 à EX5 = Valeur de l'exposant (0 ~ 31).

SE = Signe de l'exposant. 0 pour négatif ; 1 pour positif.

SM = Signe de la mantisse. 0 pour positif ; 1 pour négatif.

Octet 2 :

PC3	PC2	PC1	PC0	SC3	SC2	SC1	SC0
-----	-----	-----	-----	-----	-----	-----	-----

SC0 à SC3 = Second chiffre en BCD.

PC0 à PC3 = Premier chiffre en BCD.

Octet 3 :

TC3	TC2	TC1	TC0	QC3	QC2	QC1	QC0
-----	-----	-----	-----	-----	-----	-----	-----

QC0 à QC3 = Quatrième chiffre en BCD.

TC0 à TC3 = Troisième chiffre en BCD.

Octet 4 : 

CC3	CC2	CC1	CC0	SC3	SC2	SC1	SC0
-----	-----	-----	-----	-----	-----	-----	-----

SC0 à SC3 = Sixième chiffre en BCD.

CC0 à CC3 = Cinquième chiffre en BCD.

Par exemple - 3483200 serait codé :

$-3483200 = -0,348320 * 10^7$

donc

le premier octet contient 11000111B, soit 0C7h

le second octet code les 2 premiers chiffres, soit 034h

le troisième octet code les 2 chiffres suivants, soit 083h

le quatrième octet code les deux derniers chiffres, soit 020h

Le paramètre est en double précision :

la fonction USR met 8 dans la variable système VALTYP (0F663h) pour indiquer qu'il s'agit d'une valeur en double précision. Le registre A contient aussi cette valeur (8) à l'entrée de votre routine. La valeur du paramètre se trouve sur 8 octets dans la variable système DAC (0F7F6h). Le registre HL donne l'adresse de la variable DAC.

Les 8 octets codent le nombre en double précision de la même manière qu'en simple précision (voir ci-dessus pour plus de précisions), si ce n'est que la mantisse est sur 7 octets au lieu de 3.

Le paramètre est une chaîne de caractères :

La fonction USR met 3 dans la variable système VALTYP (0F663h) pour indiquer qu'il s'agit d'une chaîne de caractères. Le registre A contient aussi cette valeur (3) à l'entrée de votre routine. Le premier octet de la variable système DAC (0F7F6h) donne la longueur de la chaîne. Les deux octets suivants indiquent l'adresse où se trouve la chaîne. Le registre DE donne l'adresse de la variable DAC.

Voici un exemple qui transforme tous les caractères majuscules en minuscules dans une chaîne :

```

org 0c000h

cp 3
ret nz          ; Retour au Basic si ce n'est pas une chaîne
;
ld a,(de)       ; Longueur de la chaîne
or a
ret z           ; Retour au Basic si longueur = 0

CONVC:
ld b,a          ; Nombre de caractères dans B pour DJNZ
inc de
ld a,(de)       ; Prend un caractère dans la chaîne
cp 041h
jr c,NOCHAR     ; Si A < 041H, ce n'est pas une majuscule
cp 05ah
```

```

jr    nc,NOCHAR    ; Si A > 064H, ce n'est pas une majuscule
or    020h         ; Convertie la majuscule en minuscule
ld    (de),a       ; Place le caractère dans la chaîne
NOCHAR: djnz CONVC
ret                                ; retour au Basic

```

Après avoir tapé et assemblé cette routine au format binaire, entrez le programme Basic suivant et faites RUN :

```

10 CLS
20 CLEAR 200,&HC000
30 BLOAD".bin":DEFUSR=&HC000
40 A$="TRANSFORMATION":A$=USR(A$)
50 PRINT A$

```

### 13.8 Ajouter une instruction au Basic avec CMD ou IPL

A partir de la version 2.0, le Basic possède deux instructions CMD et IPL qui ne font rien excepté un appel à leur propre Hook. IPL appelle H.IPL (0FE03h) et CMD appelle H.CMD (0FE0Dh). Ceci permet de créer nos propres instructions.

Lors de l'appel au Hook, le registre HL du CPU contient le pointeur de l'interpréteur Basic dans le cache de l'instruction en cours d'exécution. La dernière valeur mise dans la pile contient les valeurs des registres AF (F contenant les indicateurs d'erreurs de l'interpréteur).

Il est possible d'ajouter des paramètres derrière ces deux instructions mais cela nécessite une connaissance avancée de l'interpréteur du Basic car le pointeur ne pointe pas un simple texte ASCII mais un texte codé par l'interpréteur du Basic. Par exemple, si vous entrez « CMD PRINT », le registre HL pointera le code de l'instruction PRINT (091h) et non pas le texte 070h, 072h, 069h 06Eh, 074h (« PRINT »).

Pour créer votre instruction, vous disposez des trois routines suivantes en ROM :

#### 00010h ou 04666h (Main-ROM)      **CHRGTR**      CHaracter GeTteR

Rôle :            Récupération d'un caractère ou d'un chiffre dans un programme Basic.

Entrée :        HL = Adresse actuelle. (Pointeur)

Sortie :        HL = Adresse de caractère récupéré.  
                  A = Caractère ou chiffre récupéré.  
                  F = Indicateur Z à 1 si il s'agit d'un code de fin de ligne (00h ou « : »).  
                  Indicateur C à 1 si il s'agit d'un chiffre de 0 à 9.

Modifie :       AF, HL.

Notes :        - Cette routine passe les codes d'espacement (020h).  
                  - Cette routine est utilisée par l'interpréteur Basic. (RST 10)

#### 04C64h (Main-ROM)      **FRMEVL**

Rôle : Envoie la valeur au pointeur dans la variable système DAC au format approprié.  
 Entrée : HL = Pointeur actuel.  
 Sortie : HL = Pointeur placé sur l'octet suivant la valeur.  
 VALTYP (0F663h) = 2, 3, 4 ou 8 selon le type de la valeur.  
 DAC (0F7F6h) = Valeur convertie au format approprié.  
 Modifie : AF, HL.

## 0542Fh (Main-ROM) **FRMQNT**

Rôle : Convertie la valeur au pointeur au format simple précision (sur 2 octets).  
 Entrée : HL = Pointeur actuel.  
 Sortie : HL = Pointeur placé sur l'octet suivant la valeur.  
 DE = Valeur codée sur 2 octets.  
 Modifie : AF, HL et DE.  
 Note: Si la valeur dépasse, il y aura retour au Basic avec l'erreur "Overflow".

Voici un exemple avec CMD :

< Donner la fonction CAPS ON/OFF à l'instruction CMD du Basic. >

```
;
; Instruction CMD CAPS routine
;
CHGCAP equ 0132h      ; LED CAPS ON/OFF
CAPST  equ 0fcabh     ; Statut de CAPS
HCMD   equ 0fe0dh     ; CMD Hook

        org 0d000h-7   ; Adresse de la routine - Taille Header
;
; Header
;
        db 0feh
        dw DEBUT
        dw FIN
        dw DEBUT
;
; Détournement du Hook CMD
;
DEBUT:  ld  bc,5        ; place les données du nouveau Hook
        ld  de,HCMD
        ld  hl,HDAT
        ldir
        ret
;
; Donnée de la routine de détournement (5 octets)
;
HDAT:   jp  CAPKEY
        nop
        nop
;
; Routine exécutée par l'instruction CMD
;
```

```

CAPKEY:
    cp    043h          ; Teste si le premier caractère est "C"
    ret   nz
    rst   10            ; INC HL possible
    ld    a,(hl)
    cp    041h          ; Teste si le second caractère est "A"
    ret   nz
    rst   10            ; INC HL possible
    ld    a,(hl)
    cp    050h          ; Teste si le second caractère est "P"
    ret   nz
    rst   10            ; INC HL possible
    ld    a,(hl)
    cp    053h          ; Teste si le second caractère est "S"
    ret   nz

    ld    a,(CAPST)
    cpl
    ld    (CAPST),A
    and   1
    call  CHGCAP

RETBASIC:
    pop   af            ; Pour ne pas avoir d'erreur
    rst   10            ; Pointeur à la fin de l'instruction
    ret

FIN:

```

Une fois la routine assemblée et sauvegardé au format binaire sous le nom "CMDCAPS.BIN", entrer la ligne suivante pour activer l'instruction.

```
CLEAR300,&HD000: BLOAD"CMDCAPS.BIN",R
```

Ensuite, vous pourrez entrer l'instruction CMD CAPS pour allumer ou éteindre la LED de la touche CAPS. Cette exemple marche parce que le Mot « CAPS » ne contient aucun mot clé du Basic.

Autre exemple :

< Créer une instruction pour changer le CPU du MSX turbo R. >

```

;
; Utilisation : CMD Z80 ou CMD R800 sous MSX-Basic
;
CHGCPUR equ    0180h          ; Change le CPU Z80/R800
HCMD     equ    0fe0dh        ; CMD Hook

                org    0d000h-7    ; Adresse de la routine - 7
;
; Header (taille = 7 octets)
;
                db     0feh
                dw     Debut
                dw     Fin
                dw     Debut

;
; Détournement de l'instruction CMD via le Hook
;
Debut:        ld     bc,5
                ld     de,HCMD
                ld     hl,HDAT

```

```

        ldir                ; place les données du nouveau Hook
        ret

;
; Nouvelles données du HOOK (5 octets)
;
HDAT:    jp      R800ROM
        nop
        nop

;
; CMD Z80 ou CMD R800? (Routine exécutée par CMD)
;
R800ROM: cp      5Ah          ; Teste si le premier caractère
        jr      z,Z80MODE    ; du paramètre est "Z"
        cp      052h         ; Teste si c'est "R"
        ret      nz
        rst      10          ; INC HL possible
        ld      a,(hl)
        cp      038h         ; Teste si le second caractère est "8"
        ret      nz
        rst      10          ; INC HL possible
        ld      a,(hl)
        cp      030h         ; Teste si le troisième est "0"
        rst      10          ; INC HL possible
        ld      a,(hl)
        cp      030h         ; Teste si le quatrième est "0"
        ret      nz
        ld      a,081h        ; R800 mode ROM
        jr      RETBASIC
Z80MODE: rst      10          ; INC HL possible
        ld      a,(hl)
        cp      038h         ; Teste si le second caractère est "8"
        ret      nz
        rst      10          ; INC HL possible
        ld      a,(hl)
        cp      030h         ; Teste si le troisième est "0"
        ret      nz
        ld      a,080h        ; Z80
RETBASIC: call    CHGCPU      ; Change le CPU
        pop     af           ; Pour ne pas avoir d'erreur
        rst      10          ; Pointeur à la fin de l'instruction
        ret

Fin:

```

Une fois la routine assemblée et sauvegardée au format binaire sous le nom "CMDR800.bin", entrer la ligne suivante pour initialiser la nouvelle instruction.

```
CLEAR300,&HD000:BLOAD"CMDR800.bin",R
```

Ensuite, vous pourrez entrer l'instruction CMD R800 pour activer le mode R800 ROM ou CMD Z80 pour activer le mode Z80.

### 13.9 Les codes de contrôle [CTRL]

Chacun sait que sous Basic, en enfonçant simultanément les touches « CTRL » et « L », on efface l'écran. Le MSX dispose de plusieurs autres codes du même type qui peuvent être utilisés en mode direct ou dans un programme. En voici la liste :

code	touches	effet
2	CTRL+B	place le curseur au début du mot précédent
3	CTRL+C	interrompt le programme (BREAK)
5	CTRL+E	efface toute la ligne à droite du curseur
6	CTRL+F	place le curseur au début du mot suivant
7	CTRL+G	émission d'un bref bip sonore
8	CTRL+H	back space (BS)
9	CTRL+I	tabulation / idem touche TAB
10 (0Ah)	CTRL+J	descend le curseur d'une ligne
11 (0Bh)	CTRL+K	idem touche EFE ou HOME
12 (0Ch)	CTRL+L	effacement de l'écran
13 (0Dh)	CTRL+M	effectue une entrée / idem touche RETURN
14 (0Eh)	CTRL+N	place le curseur en fin de ligne
18 (12h)	CTRL+R	idem touche INS
21 (15h)	CTRL+U	efface la ligne où se trouve le curseur
24 (18h)	CTRL+X	idem touche SELECT
27 (1Bh)	CTRL+[	idem touche ESC
28 (1Ch)	CTRL+\	idem flèche droite
29 (1Dh)	CTRL+]	idem flèche gauche
30 (1Eh)	CTRL+^	idem flèche haut
31 (1Fh)	CTRL+_	idem flèche bas

Essayez donc cet exemple (MSX2) :

```
10 SCREEN 0: WIDTH 80: COLOR 10,0,0: CLS
20 A$="Ce texte s'affiche bizarrement !!!"
25 LOCATE 25,10
30 FOR I=1 TO LEN (A$)
40 PRINT CHR$(30)+MID$(A$,I,1);
50 NEXT I
```

### 13.10 Les codes escape [ESC]

Le MSX peut utiliser toute la série de code ESC compatible avec les terminaux VT-52 ou HEATH-19 dont la liste suit :

code	touches	effet
27+65	ESC A	monte le curseur d'une ligne
27+66	ESC B	descend le curseur d'une ligne
27+67	ESC C	déplace le curseur vers la droite
27+68	ESC D	déplace le curseur vers la gauche
27+72	ESC H	curseur en haut à droite (HOME)
	ESC Y col ligne	curseur en X, Y (LOCATE) voir ci-dessous
27+106	ESC j	efface l'écran (CLS)
27+69	ESC E	efface l'écran (CLS)
27+75	ESC K	efface jusqu'à la fin de la ligne
27+74	ESC J	efface jusqu'à la fin de l'écran
27+108	ESC l	efface toute la ligne
27+76	ESC L	insère une ligne
27+77	ESC M	détruit une ligne
27+120+52	ESC x4	définit un curseur de type entier
27+120+53	ESC x5	éteint le curseur
27+121+52	ESC y4	définit un curseur de type barre
27+121+53	ESC y5	allume le curseur

La séquence ESC Y est légèrement plus compliquée à utiliser. Il faut en effet envoyer les deux octets de ESC Y (27 et 89), puis les faire suivre par deux octets qui définissent la colonne (n° de colonne + 020h) ainsi que la ligne (n° de la ligne + 020h). Affichons par exemple la chaîne « ICI » à la position (10, 33) :

```
10 PRINT CHR$(27)+"Y*AI CI"
```

On envoie un CHR\$(27) puis « Y » suivi de « \* » (code ASCII = 42, soit 10 + 020h), et « A » code ASCII = 65 soit 33 + 020h).

Très peu de gens connaissent l'existence de ces codes ESC qui peuvent pourtant parfois rendre de grands services comme le démontre l'exemple suivant :

```
10 SCREEN 0: WIDTH 40: COLOR 10,0,0: CLS: KEYOFF
20 E$=CHR$(27)
30 FOR I=0 TO 17: PRINT"LIGNE";I: NEXT
40 PRINT: PRINT"On peut effacer les dernières lignes.": PRINT
50 PRINT"Comme dans une fenêtre...": PRINT
60 PRINT TAB(5);"... aussi rapidement qu'avec un CLS";
70 FOR I=1 TO 1500: NEXT
80 LOCATE 0,19: PRINT"Sans toucher aux autres !" + E$ + "J"
90 FOR I=1 TO 1500: NEXT I: LOCATE 0,18: GOTO 40
```

Essayez d'enlever +E\$+"J" à la ligne 80.



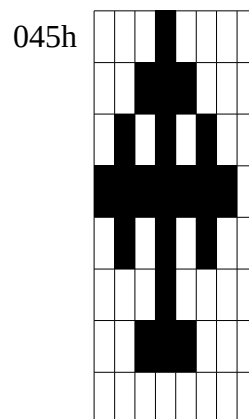
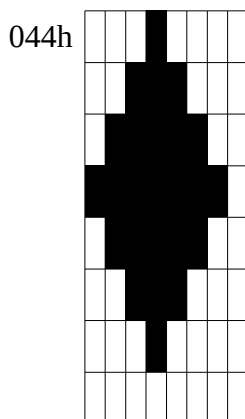
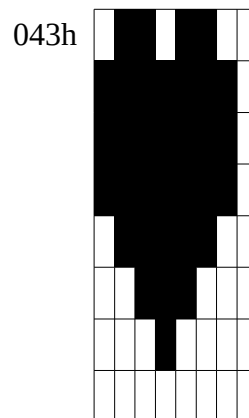
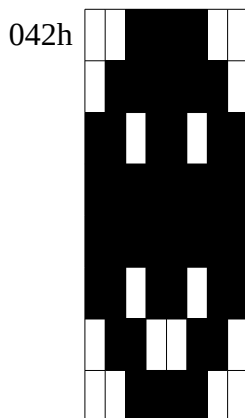
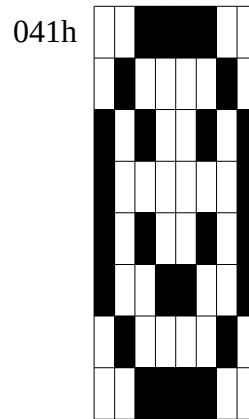
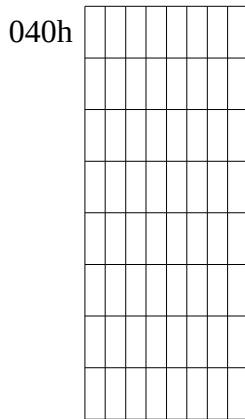
### 13.11 Utiliser le second jeu de caractères

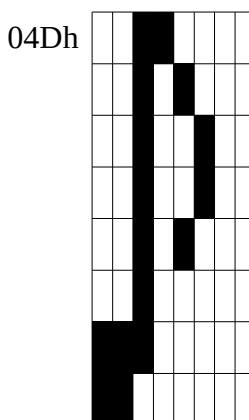
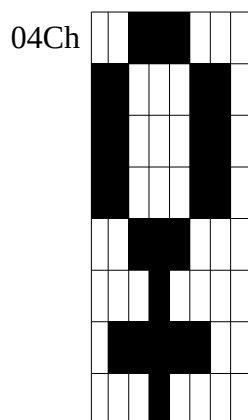
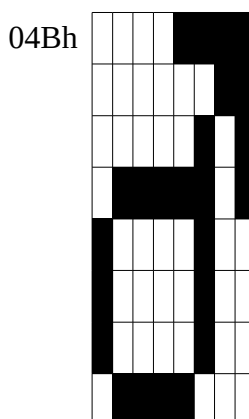
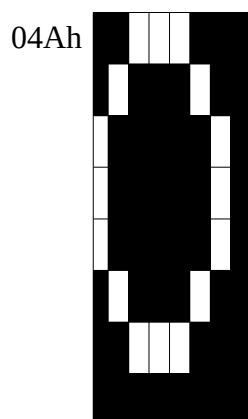
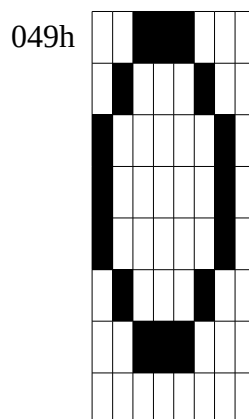
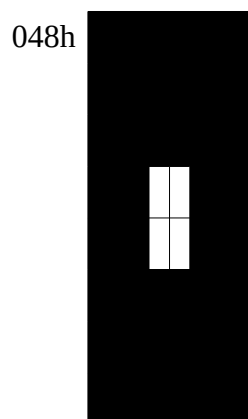
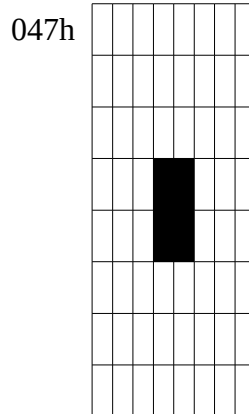
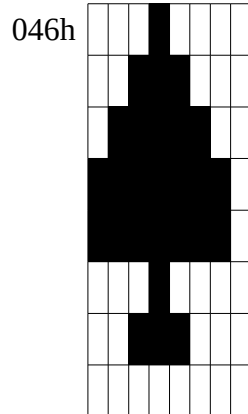
Les MSX possèdent un second jeu de caractères. Ce dernier est réduit (32 caractères) et on y accède par un CHR\$(1) suivi d'un code ASCII entre 64 (040h) et 95 (05Fh). Par exemple, en SCREEN 1, la ligne :

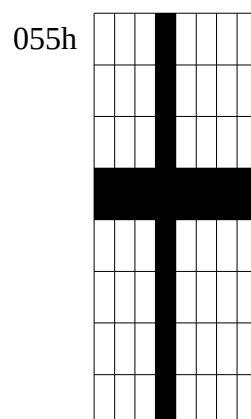
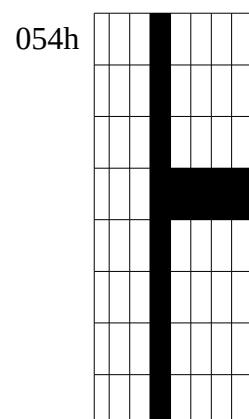
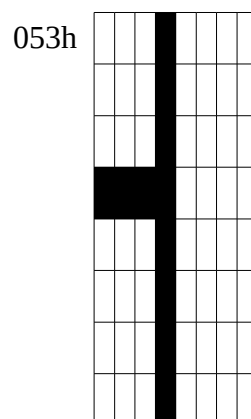
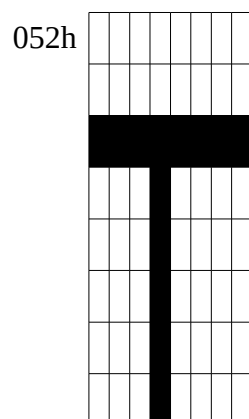
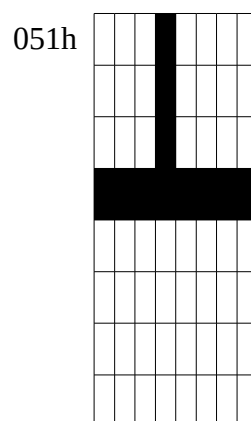
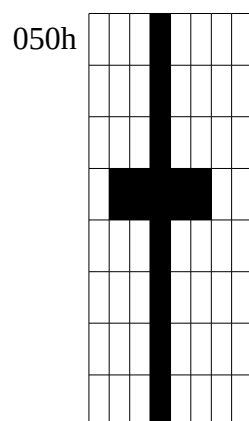
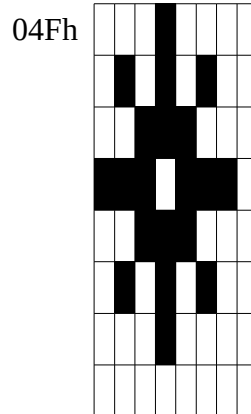
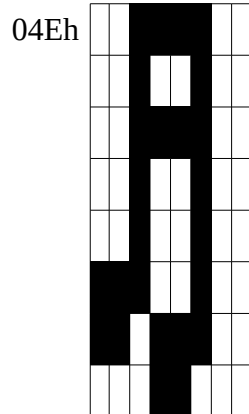
```
PRINT CHR$(1) + "A"
```

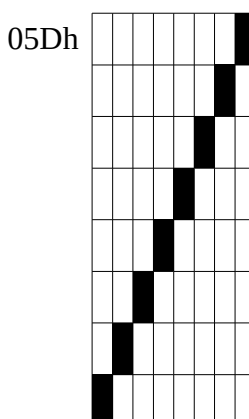
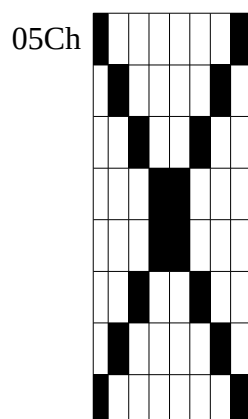
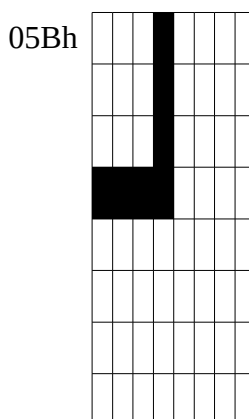
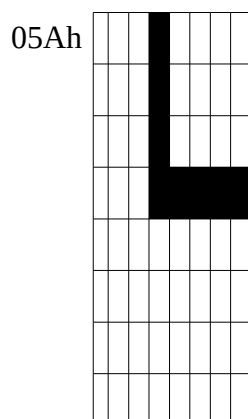
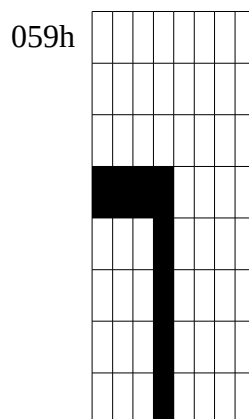
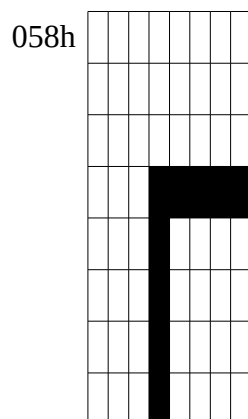
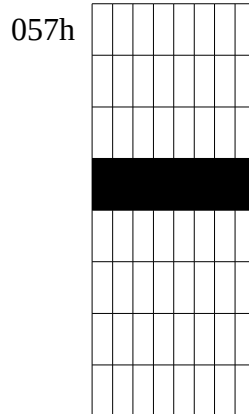
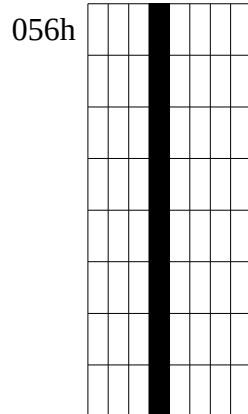
fera s'afficher à l'écran une petite tête sympathique. Notez que tous ces caractères se trouvent définis dans des matrices 8 sur 8. On perdra donc, en SCREEN 0, les 2 bits de poids faible, et le caractère sera amputé de ses 2 colonnes les plus à droite.

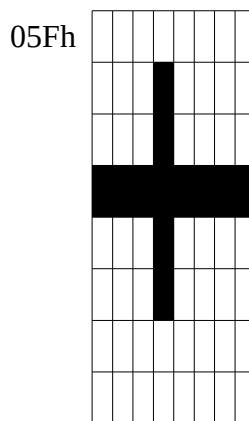
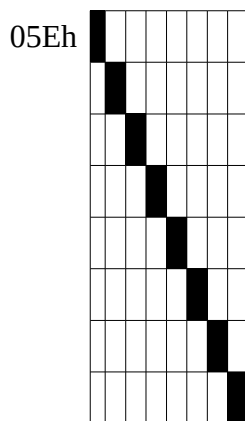
Voici les matrices des caractères de ce second jeu :











### 13.12 Détourner le Reset

Presque tous les MSX vendus en France possèdent un bouton RESET (le V20 de Canon et le YC-64 Yashica sont les seuls MSX sans RESET ayant connu une grande diffusion). Le programmeur peut interdire l'accès à son application à l'utilisateur final en détournant le RESET.

Le principe est assez simple. On installe en mémoire vive des codes qui font croire au MSX qu'il ne s'agit pas de RAM mais d'une cartouche. En effet, dans ce dernier cas, le MSX passe toujours la main à la cartouche (sinon le programme en cartouche ne démarrerait pas automatiquement). Etant trompé (comme nous sommes diaboliques), le MSX va donc rendre la main au programme que nous aurons installé auparavant.

On utilise le programme du paragraphe précédent pour trouver la RAM en plage 1 (04000h à 07FFFh). Le reste du programme est simple à comprendre :

```

; org 0c000h
;
ENASLT equ 00024h
FIND: ld b,0fh
      ld hl,04000h ; Plage 1
LOOP: ld a,b
      or 080h
      push bc
      push af
      push hl
      call ENASLT ; Sélection de Slot
      pop hl
      pop af
      ld (hl),a
      ld b,(hl)
      cp b
      pop bc
      jr z,RAM
      djnz LOOP ; Saute à LOOP si pas de RAM
;
RIEN: jr RETOUR
;
RAM: ld ix,04000h
      ld (ix+0),041h ; Code "A"
      ld (ix+1),042h ; Code "B"
      ld (ix+2),000h ; Adresse de
      ld (ix+3),0c1h ; départ
      jr RETOUR

```

```

;
MNROM equ 0fcc1h
;
RETOUR: ld a, (MNROM)
        ld hl, 04000h ; Replace la Main-ROM
        call ENASLT ; en plage 1.
        ret

;
;
        org 0c100h
;
BREAKX equ 000b7h
INITXT equ 0006ch
;
        call INITXT ; SCREEN 0, 40 colonne
        ld hl, MESS
        call ECRIRE
WAIT:   call BREAKX ; Teste CTRL+STOP
        ret c
        jr WAIT
ECRIRE: ld a, (hl)
        cp 0
        ret z
        call 0a2h
        inc hl
        jr ECRIRE
MESS:   db 01bh, 'Y*&'
        db 'LE RESET NE REND'
        db ' PAS LA MAIN !', 0

```

Dans l'exemple ci-dessus, tapez et assemblez le programme, puis lancez l'exécution en 0C000h. A partir de ce moment, chaque fois que vous appuierez sur le bouton RESET, le programme affichera le message « Le reset ne rend pas la main ! ». Il suffit d'enfoncer les touches CTRL et STOP pour récupérer le contrôle.

Notez que l'on utilise un code ESC pour positionner le curseur (première ligne de « MESS: »).

Une cartouche se distingue des autres supports. En effet, toute cartouche contient « AB » (codes 041H et 042h) comme deux premiers octets (en général c'est aux adresses 04000H et 04001h). Elle renferme ensuite (en 04002H et 04003h) l'adresse de démarrage du programme en cartouche.

### 13.13 Ajouter des mots clés au Basic

Tant qu'on y est, continuons à simuler le fonctionnement d'une cartouche. Vous savez sans doute qu'une cartouche peut parfois contenir des extensions au MSX-Basic. On accède à ces nouvelles instructions par l'instruction CALL (ou « \_ ») suivi d'un nom et éventuellement de paramètres.

L'adresse de la routine de traitement des mots clefs est donnée par le 5ème et le 6ème octet (adresse 04004H et 04005h) de la cartouche. A chaque fois que le Basic trouve un CALL, il appelle la routine à cette adresse. Le mot clef se trouve alors sur 16 octets dans la zone des variables système (variable PROCNM, adresses 0FD89h à 0FD98h). Le registre double HL pointe sur le premier caractère non blanc (code 020h) après le mot clef, ce qui permet de récupérer les paramètres. Il faut réactualiser HL de manière à poursuivre l'exécution du programme Basic après traitement du CALL. Il suffit pour cela d'incrémenter HL jusqu'au moment où HL pointe sur un 0 (code de fin de ligne Basic) ou 03Ah (code des deux points « : », séparant deux instructions Basic). De plus, votre routine doit toujours rendre la main

avec l'indicateur Carry à 0. Dans la routine de traitement, tous les registres peuvent être utilisés (sauf SP, bien sûr). Si le mot clef est inconnu, il faut mettre l'indicateur Carry à 1, puis rendre la main à l'interpréteur par un RET, sans avoir modifié HL.

Dans l'exemple suivant, nous créons un mot clef « FILLSCREEN » qui remplit l'écran (en SCREEN 0 uniquement, 40 ou 80 colonnes) avec le caractère qui suit :

```
10 _FILLSCREEN ("Z") :BEEP
```

Cet exemple fonctionne parfaitement sur tous MSX.

```
ENASLT: equ 00024h
MNROM: equ 0fcc1h

;
org 0c000h
FIND:
    ld b,0fh
    ld hl,04000h
LOOP:  ld a,b
        or 080h
        push bc
        push af
        push hl
        call ENASLT
        pop hl
        pop af
        ld (hl),a
        ld b,(hl)
        cp b
        pop bc
        jr z,RAM
        djnz LOOP
RIEN:
    jr RETOUR
RAM:
    ld ix,04000h
    ld (ix+0),041h
    ld (ix+1),042h
    ld (ix+2),000h
    ld (ix+3),000h
    ld (ix+4),000h
    ld (ix+5),0c1h
    jp 0c100h
;
RETOUR: ld a,(MNROM)
        ld hl,04000h
        jp ENASLT

FILVRM: equ 00056h
;
org 0c100h

push hl
ld hl,WORD
ld de,0fd89h
LOOP2: ld a,(DE)
        ld b,(HL)
        cp b
        jr nz,SYNTAXERR
```

```

        cp      0
        inc     hl
        inc     de
        jr      z,OUT
        jr      LOOP2
SYNTAXERR:
        pop     hl
        xor     a
        ccf
        ret
OUT:
        pop     hl
        push    hl
        ld      a,(hl)
        cp      028h
        jr      nz,SYNTAXERR
        inc     hl
        ld      a,(hl)
        cp      022h
        jr      nz,SYNTAXERR
        inc     hl
        ld      b,(hl)
        inc     hl
        ld      a,(hl)
        cp      022h
        jr      nz,SYNTAXERR
        inc     hl
        ld      a,(hl)
        cp      029h
        jr      nz,SYNTAXERR
LOOP3:  inc     hl
        ld      a,(hl)
        cp      020h
        jr      z,LOOP3
        cp      0
        jr      Z,SUITE
        cp      03Ah
        jr      nz,SYNTAXERR
SUITE:
        push    hl
        ld      hl,0
        ld      a,b
        ld      bc,00800h
        call    FILVRM
        pop     hl
        pop     bc
        xor     a
        ret
WORD:
        db      'FILLSCREEN',0

```

Après avoir tapé et assemblé le programme ci-dessus, lancez l'exécution en 0C000h. Le MSX fera automatiquement un RESET (pour que la recherche de la cartouche se produise). Lors du retour sur Basic, tapez CLEAR 200, &HC100

A partir de là, vous pouvez à tout moment appeler la nouvelle fonction par un CALL FILLSCREEN (“caractère”).

Pour créer une autre fonction, il suffit de modifier le programme à partir du label « SUITE : »



### 13.14 Manipuler la souris

Je vous propose un sous-programme en assembleur qui permet d'utiliser la souris depuis le Basic.

```
EXTROM      equ      0015fh
NEWPAD      equ      001Adh
GTTRIG      equ      000d8h
NWRVRM      equ      00177h

Souris:      org      0c000h

            ld        a,0ch
            ld        ix,NEWPAD
            call       EXTROM
            ld        a,0dh
            ld        ix,NEWPAD
            call       EXTROM
            ld        b,a
            ld        a,(X)
            add        a,b
            ld        (X),a
;
            ld        a,0ch
            ld        ix,NEWPAD
            call       EXTROM
            ld        a,0eh
            ld        ix,NEWPAD
            call       EXTROM
            ld        b,a
            ld        a,(Y)
            add        a,b
            ld        (Y),a
;
            call       Sprite
            ld        a,1
            call       GTTRIG
            cp         0ffh
            ret        z
            jr         Souris
;
Sprite:      ld        a,(X)
            ld        hl,0fa01h
            call       NWRVRM
            ret
X:          db        00h
Y:          db        00h
```

Voici un exemple d'exploitation de la routine ci-dessus en Basic :

```
10 SCREEN 7: SET PAGE 0,0: COLOR 10,0,0: CLS: X=&HC045: Y=X+1
20 Sprite$(0)=CHR$(&HF8)+CHR$(&HC0)+CHR$(&HA0)+CHR$(&H90)+CHR$
(&H88)+CHR$(4)+CHR$(2)
30 PUT Sprite 0,(100,100),7: DEFUSR=&HC000: I=USR(0)
110 PSET(PEEK(&HC054)*2,PEEK(&HC055)+1),14
120 I=USR(0):GOTO 110
```

### 13.15 Développer un programme en cartouche

Après avoir trouvé la RAM et initialisé les variables système, le MSX cherche l'entête d'une ROM dans tous les Slot sur les plages mémoire 4000h~7FFFh et 8000h~BFFFh. La recherche s'effectue dans l'ordre croissant. Lorsqu'un Slot primaire est étendu, la recherche se fait dans les Slot secondaire correspondants avant de passer au slot primaire suivant.

Lorsque le système trouve une entête, il sélectionne le Slot de la ROM mais uniquement sur la plage mémoire correspondante à l'adresse valide indiquée puis, exécute le programme en ROM à cette même adresse.

Une entête se compose de 16 octets et doit être placés à l'adresse 4000h ou 8000h.

Format de l'entête d'une ROM :

Entête	Nom	Rôle
+0	ID	Les deux premiers octets doivent être 041H et 042H (« AB ») pour indiquer qu'il s'agit d'un ROM additionnelle.
+2	INIT	Ces deux octets sont prévus pour contenir l'adresse de la routine à appeler pour initialiser une zone de travail ou des ports d'E/S. Indiquer 0000h pour ignorer l'exécution.
+4	STATEMENT	Ces deux octets contiennent l'adresse d'exécution d'un programme dont le but est d'ajouter des instructions au MSX-Basic au moyen de l'instruction CALL. Indiquer 0000h si la ROM ne contient pas d'instruction Basic à ajouter.
+6	DEVICE	Adresse d'exécution d'un programme servant à contrôler un périphérique intégré à la cartouche. Par exemple, une interface pour disque. Indiquer 0000h si la cartouche ne contient pas de matériel à installer.
+8	TEXT	Pointeur du programme Basic contenu en ROM. Indiquer 0000h si la ROM ne contient pas de programme Basic.
+10~15	Reserved	Réservé. Ces octets doivent être mis à zéro.

Description :

#### **INIT**

C'est la première adresse prise en compte. Lorsque cette adresse est supérieure 0000h, le système sélectionne le Slot de la ROM sur la plage mémoire correspondante à l'adresse puis, exécute le programme en ROM à cette même adresse.

Au moment de l'exécution du programme, le registre C contient le numéro de Slot de la ROM sous la forme F000SSPP. La routine doit se terminer par un RET. Tous les registres peuvent être modifiés par la routine sauf SP. À la place d'une routine d'initialisation, la ROM peut très bien contenir un jeu.

Astuces :

Si votre ROM a une taille de 32Ko et que INIT a une valeur entre 4000h~7FFFh, vous pourrez sélectionner la deuxième partie (8000h~BFFFh) de la façon suivante.

```
ld    h,080h ; La routine ENASLT ne tient pas compte du registre L
call  ENASLT ; Sélection du Slot de la ROM
```

Si votre ROM a une taille de 32Ko et que INIT a une valeur entre 8000h~BFFFh, vous pourrez sélectionner la première partie (4000h~7FFFh) de la façon suivante.

```
ld    a,c
ld    h,040h ; La routine ENASLT ne tient pas compte du registre L
call  ENASLT ; Sélection du Slot de la ROM
```

## **STATEMENT**

Le programme de traitement de l'instruction doit résider sur la plage 4000h~7FFFh.

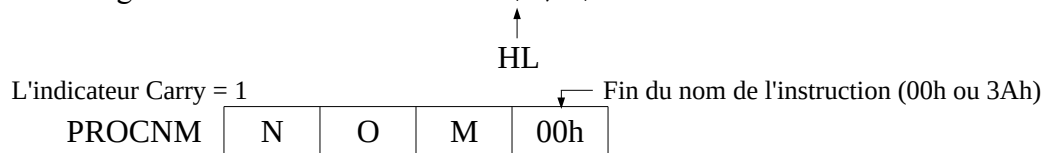
Une instruction appelée par CALL doit avoir le format suivant :

CALL <Nom de l'instruction> [(variable[, variable][,...])]

Le nom de l'instruction peut faire jusqu'à 15 caractères. Lorsque l'interpréteur du Basic trouve l'instruction CALL, il copie le nom de l'instruction dans la zone de travail PROCNM (0FD89h) puis cherche dans les Slots par ordre croissant une adresse STATEMENT supérieure à 0000h afin de transmettre le contrôle pour cette instruction. À ce moment là, le registre double HL contient l'adresse du paramètre qui suit le nom de l'instruction dans le listing. L'instruction peut être traitée. En sortie, HL doit indiquer l'instruction suivante à traiter et l'indicateur Carry doit indiquer si il y a eu une erreur.

Voici un exemple de procédure avec l'instruction CALL NOM (0,0) suivit de A=0 :

1. Le listing contient donc CALL NOM (0,0) : A=0



2. Traitement de l'instruction par la routine à l'adresse indiquée à STATEMENT.

Si le nom ne correspond pas alors laissez HL tel quel et mettez Carry à 1 avant de rendre la main à l'interpréteur (par un RET).

Si nom ne correspond, exécutez la routine de l'instruction et ses paramètres puis, pointez l'instruction suivante avec HL et mettez Carry à 0 si il n'y a pas erreur dans les paramètres.

3. Fin du traitement.

CALL NOM (0,0) : A=0

$\uparrow$   
 HL

Carry = 0 si il n'y a pas erreur

Rendez la main à l'interpréteur (par un RET).

Note : Evitez de donner un nom déjà existant à votre instruction car selon la position de la ROM dans les Slot, elle ne pourrait ne pas être prise en compte ou même provoquer une erreur à cause des

paramètres.

## **DEVICE**

Cette adresse doit être comprise entre 04000h et 07FFFh. Le système peut contrôler jusqu'à 4 périphériques par cartouche. Le nom du périphérique doit faire 15 caractères maximum. Lorsque l'interpréteur Basic rencontre un nom de périphérique, il le copie dans la zone de travail PROCNM (0FD89h) puis met le registre A à 255 et cherche dans les Slots par ordre croissant une adresse DEVICE supérieure à 0000h afin de transmettre le contrôle du périphérique correspondant.

Voici un exemple de procédure avec l'instruction OPEN "NOM:" :

4. Le listing contient donc OPEN "NOM:"

A = Numéro de l'instruction (voir le tableau plus bas)

L'indicateur Carry = 1

↖ Fin du nom du périphérique (00h ou 3Ah)

PROCNM	N	O	M	00h
--------	---	---	---	-----

5. Contrôle du périphérique par la routine à l'adresse indiquée à DEVICE.

Si le nom ne correspond pas alors mettez le registre A à 255 et Carry à 1 avant de rendre la main à l'interpréteur (par un RET).

Si nom ne correspond, exécutez la routine de contrôle puis, pointez l'instruction suivante avec HL et mettez Carry à 0 si il n'y a pas erreur dans les paramètres.

6. Fin du traitement.

A = Identifiant du périphérique (0~3)

Carry = 0 si il n'y a pas erreur

Rendez la main à l'interpréteur (par un RET).

Numéro de l'instruction

Registre A	Instruction d'appel
0	OPEN
2	CLOSE
4	Accès aléatoire
6	Sortie séquentielle
8	Entrée séquentielle
10	Fonction LOC
12	Fonction LOF
14	Fonction EOF
16	Fonction FPOS
18	Caractère de sauvegarde

## TEXT

Ce pointeur TEXT indique le début du programme Basic à exécuter automatiquement au démarrage du MSX. Le premier octet du programme est toujours zéro. Le programme ne peut avoir une taille de 16Ko maximum et doit se situer entre 08000h et 0BFFFh. Il doit aussi être format codé (« tokenized ») et non pas au format texte ASCII. De plus, les adresses correspondantes aux numéros de lignes du programme doivent indiquer les adresses de destination réelles dans le programme.

Méthode pour mettre un programme Basic en ROM :

1. Un programme Basic commence généralement à l'adresse 08000h. Il faut donc le décaler au minimum vers l'adresse 08012h pour pouvoir insérer l'entête de la ROM. Pour cela, entrer la ligne suivante sous Basic :

```
POKE &HF676,&H13: POKE &HF677,&H80: POKE &H8012,0: NEW
```

2. Charger le programme Basic à mettre en ROM en entrant l'instruction suivante.

```
LOAD"Name.BAS"
```

3. Puis sauvegarder le programme en entrant l'instruction suivante.

```
SAVE"Name2.BAS"
```

4. Ensuite, créer l'entête de la ROM avec un éditeur hexadécimal et ajouter 00 00.

```
000000h: 41 42 00 00 00 00 00 00 12 80 00 00 00 00 00
000010h: 00 00
```

5. Ajouter les données du fichier "Name2.BAS" à la suite par un copier/coller sans oublier de remplacer le premier octet 0FFh du programme par 000H. Ensuite, remplissez de 00h ou autres derrière le programme afin de faire un fichier 16384 octets précisément. Pour finir, sauvegardez par exemple sous le nom "Name.rom". Vous aurez juste à graver la ROM avec ce fichier pour créer votre cartouche.

Le système dispose aussi de variables pour les programmes en cartouche. Voici un descriptif celles-ci :

– **SLTATR** (0FCC9h~0FD08h)

Les 64 octets de SLTATR informent sur le contenu des ROM de chaque Slot. Quatre plages de mémoire de 16 Ko réparties sur 16 Slot secondaires possibles, ce qui fait  $4 \times 16$  combinaisons.

Format de ces variables :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
BT	DE	SE	-	-	-	-	-

Les 3 bits de poids fort sont définis en fonction de l'entête de la ROM logée au Slot correspondant.

SE (« Statement Expander ») = ROM contenant des instructions Basic étendus CALL si à 1.

DE (« Device Expander ») = ROM contenant un programme pour gérer un dispositif si à 1.

BT (« Basic Text ») = ROM contenant un programme Basic si à 1.

Les autres bits de poids faible sont inutilisés. Ils sont en général à 0 mais pas toujours.

Voici un petit programme Basic qui donne un descriptif des cartouches insérées :

```
10 SCREEN 0:WIDTH80:COLOR 10,0,0:CLS
20 FOR I=&HFCC9 TO &HFD08
30 IF PEEK(I)=0 THEN NEXT:END
40 A=I-&HFCC9
50 PRINT"Page"+STR$(A MOD4)+SPACE$(2)+"Slot"+STR$(INT(A/16))+"-"+STR$(A/4)
MOD 4)+": ";
60 IF (PEEK(I)AND &H80)=128 THEN PRINT TAB(20)+"Extension Basic."
70 IF (PEEK(I)AND &H40)=64 THEN PRINT TAB(20)+"Extension Matériel."
80 IF (PEEK(I)AND &H20)=32 THEN PRINT TAB(20)+"Extension Logiciel."
90 PRINT:NEXT
```

– **SLTWRK** (0FD09h~0FD88h)

SLTWRK est un tableau de variables de 128 octets qui sert à réserver une zone travail en RAM pour les applications en ROM. Ce tableau est composé de 8 octets par Slot (2 par plage mémoire). Chacun de ces 2 octets sont prévus pour y placer un indicateur sur un octet (sur la partie basse) ou sur deux octets de la façon suivante.

	Partie basse	Partie Haute	
SLTWRK			Indicateur de la zone de travail de l'application en Slot 0-0.
SLTWRK+2			Indicateur de la zone de travail de l'application en Slot 0-1.
.	.	.	.
.	.	.	.
.	.	.	.
SLTWRK+124			Indicateur de la zone de travail de l'application en Slot 3-2.
SLTWRK+126			Indicateur de la zone de travail de l'application en Slot 3-3.

Si votre programme en ROM a besoin de réserver plusieurs zones de travail, vous pouvez y mettre un pointeur indiquant d'autres pointeurs.

L'indicateur sur un octet permet de réserver toute la RAM sur la plage 0000h~7FFFh. Il doit avoir le format suivant. La partie haute doit contenir 00h.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
F	RMD	APP	RES	SS1	SS0	PP1	PP0

PP0 et PP1 = Numéro de Slot primaire de la RAM à réserver.

SS0 et SS1 = Numéro de Slot secondaire de la RAM à réserver.

RES = Bit réservé.

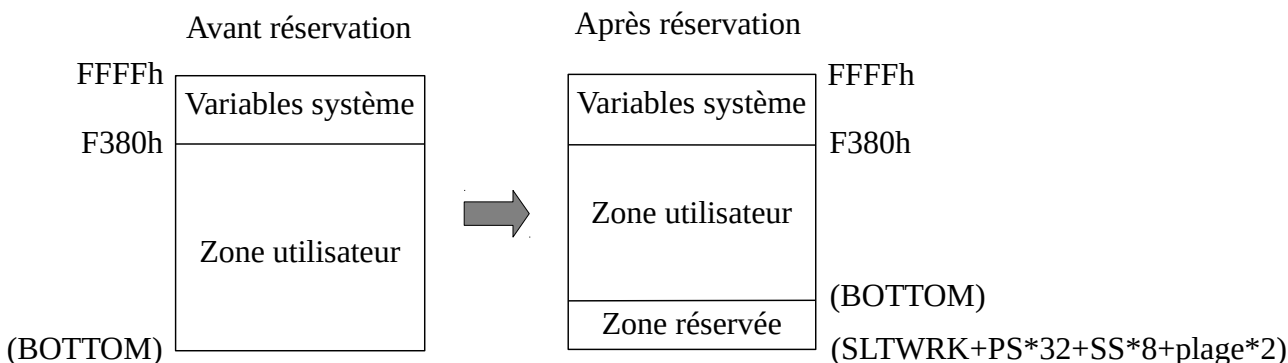
APP = 1 si la RAM utilisée par une application ; 0 dans le cas contraire.

RMD = 1 si la RAM est utilisée par l'instruction CALL MEMINI ; 0 si ce n'est pas le cas.

F = 0 pour slot primaire ; 1 pour Slot secondaire.

L'indicateur sur 2 octets est pointeur permettant de réserver la RAM entre 8000h et F37Fh.

Voici un exemple de réservation d'une zone de travail juste avant un programme Basic :



Dans cet exemple, la variable correspondante à l'application en ROM doit contenir l'adresse 08000h comme pointeur.

### 13.16 Trouver le Slot de la RAM depuis une ROM

Pour connaître le Slot de la Main-RAM lorsque le MSX a un disque installé, il suffit de lire les variables RAMAD0 à RAMAD3 (0F341h~ 0F344h) qui sont installées par la Disk-ROM. Celle-ci indique le Slot pour chacune des 4 plages mémoire. Par contre, lorsque votre programme est lancée depuis une ROM, étant donnée que la Disk-ROM n'est probablement pas encore installée, il faut donc créer une routine soit-même pour chercher la RAM.

Pour trouver la RAM sur la plage mémoire 3 (0C000h~0FFFFh), c'est relativement simple puisque le système a déjà sélectionné le Slot sur cette plage quelque soit le MSX. Il suffit donc de lire l'état des registre des Slot primaires et secondaire et ne garder que les bits utiles.

Voici un programme d'exemple qui renvoie le numéro de Slot (au format F000SSPP) de la RAM sur la plage mémoire 3 (0C000h~0FFFFh) dans le registre A :

```
RSLREG equ 00138h ; Lecture du registre des Slot primaires
EXPTBL equ 0FCC1h

SEEK_RAMAD3:
    call RSLREG
    and 0c0h ; Garde les bits 7 et 6 correspondant à la plage 3
    rlca
    rlca ; Décalage des bits 7 et 6 vers les bits 1 et 0
    ld e,a ; Garde la valeur du Slot primaire dans le reg. E
    ld a,(0ffffh) ; Lecture du registre des Slot secondaire du
    cpl ; slot primaire sélectionné puis inversion des bits
    and 0c0h ; Garde les bits 7 et 6 correspondant à la plage 3
    rrca
    rrca
    rrca
    rrca ; Décalage des bits 7 et 6 vers les bits 3 et 2
    or e ; Récupère la valeur du Slot primaire
    ld hl,EXPTBL+3
    ld d,0
    add hl,de
    or (hl) ; Bit7=1 si le slot est étendu
    ret
```

Ce programme est valable pour tous les MSX à partir de 16Ko et à condition que votre ROM ne démarre pas sur la plage mémoire 3 (0C000h~0FFFFh).

Pour trouver la RAM de la plage mémoire 2 (08000h~0BFFFh), nous pouvons employer la même méthode à condition que la ROM ne démarre pas sur la plage 2.

Voici un programme d'exemple qui renvoie le numéro de Slot de la RAM sur la plage mémoire 2 (08000h~0BFFFh) dans le registre A :

```
RSLREG equ 00138h ; Lecture du registre des Slot primaires
EXPTBL equ 0FCC1h

SEEK_RAMAD2:
    call RSLREG
    and 030h ; Garde les bits 5 et 4 correspondant à la plage 2
    rlca
    rlca
    rlca
    rlca ; Décalage des bits 5 et 4 vers les bits 1 et 0
    ld e,a ; Garde la valeur du Slot primaire dans le registre E
    ld a,(0ffffh) ; Lecture du registre des Slot secondaire du
    cpl ; slot primaire sélectionné puis inversion des bits
    and 030h ; Garde les bits 5 et 4 correspondant à la plage 3
    rrca
    rrca ; Décalage des bits 5 et 4 vers les bits 3 et 2
    or e ; Récupère la valeur du Slot primaire
    ld hl,EXPTBL+2
    ld d,0
```



```

add    hl,de
or     (hl)      ; Bit7=1 si le slot est étendu
ret

```

Ce programme est valable pour tous les MSX à partir de 32Ko.

Pour trouver la RAM sur la plage 0 (0000h~3FFFh) ou 1 (4000h~7FFFh), c'est plus compliqué, il faut faire une recherche dans tous les Slot et de préférence dans l'ordre croissant.

Voici un exemple :

```

; Routine de recherche de la RAM sur la plage mémoire 0 ou 1
; Sortie: A=Numéro de slot et Carry = 0, Carry = 1 si Ram non trouvée

EXPTBL equ    0fcc1h          ; Indicateurs des Slot secondaires
RDSLT:  equ    0000ch
WRSLT:  equ    00014h
RAMSLT: equ    0c000h          ; Slot de la RAM trouvée
KBUF:   equ    0c001h          ; Longueur 4 octets (0C001h~0C004h)

        org    04010h          ; Adresse qui peut être 0000h~0BEFFh selon
                                ; la plage sur laquelle la ROM se trouve

        call   Ram_srch
        ld     (RAMSLT),a
        ret

Ram_srch:
        ld     b,4              ; Slot primaire
Ram_srch_loop:
        ld     a,b
        dec    a
        xor    3
        ld     (RAMSLT),a      ; Numéro du Slot secondaire en cours
        ld     e,a

        ld     hl,EXPTBL
        ld     d,0
        add    hl,de
        ld     a,(hl)
        ld     (KBUF),a        ; Stocke l'indicateur de slot secondaire
        exx

        ld     a,(KBUF)        ; Restitue l'indicateur de slot secondaire
        rlca
        ld     b,1
        ld     a,(RAMSLT)
        jr     nc,PrimSLT

        ld     b,4              ; Slot secondaire
Ram_srch_loop2:
        ld     a,b
        dec    a
        xor    3
        rlca
        rlca

```

```

        ld      c,a
        ld      a,(RAMSLT)
        or      c
        or      080h          ; Met le bit 7 à 1
PrimSLT:
        ld      (KBUF+1),a
        ld      hl,0000h      ; Plage 0 (mettre 4000h chercher la plage 1)
        push    bc
        call    RDSLT
        ld      (KBUF+2),a
        pop     bc
        cp      041h
        jr      nz,no_header   ; Saute si le premier octet = "A" (Rom?)

        inc     hl
        ld      a,(KBUF+1)
        push    bc
        call    RDSLT
        pop     bc
        dec     hl
        cp      042h
        jr      z,no_ram       ; Saute si le second octet <> "B"
no_header:
        ld      a,(KBUF+1)
        push    bc
        call    RDSLT          ; Lit le premier octet
        pop     bc
        ld      e,041h
        ld      a,(KBUF+1)
        push    bc
        call    WRSLT          ; Ecrit "A" au premier octet
        pop     bc
        ld      a,(KBUF+1)
        push    bc
        call    RDSLT          ; Lit le premier octet
        pop     bc
        cp      041h
        jr      z,Ram_found    ; Saute si le premier octet = "A"

no_ram:
        djnz    Ram_srch_loop2 ; Vers le Slot suivant si pas de RAM
        exx
        djnz    Ram_srch_loop  ; Vers le Slot suivant si pas de RAM
        scf
        ret

Ram_found:
        ld      a,(KBUF+2)
        ld      e,a
        ld      a,(KBUF+1)
        push    af
        or      080h
        call    WRSLT          ; Restitue le premier octet de la RAM
        pop     af             ; A=Slot de la Ram trouvée (sans le bit 7)
        or      a              ; Met Carry à 0
        ret

```

Ce programme est valable pour tous les MSX à partir de 64Ko. Cependant, il faut tenir compte qu'il est préférable, voir même nécessaire, de choisir la RAM interne pour le MSX Turbo R en mode R800. Si

une extension de mémoire est insérée dans un port cartouche, elle sera sélectionnée car elle se trouvera dans un Slot inférieur.

Voilà vous avez trouvé de la mémoire vive. Maintenant, il faut faire attention au deux cas suivants :

1. Sur les MSX1 Toshiba HX-20, HX-21, HX-22 (Japanese) et les MSX2, un disque virtuel peut être implanté par l'instruction CALL MEMINI du Basic dans la mémoire vive que vous venez de trouver. Si tel est le cas, la variable système SLTWRK (0FD09h~) contient le numéro du Slot dans lequel se trouve un RAM e disque virtuel sous la forme FXXXSSPP en binaire.

F        indique le type de Slot. (0 pour Slot Primaire ; 1 pour Slot étendu.)

SS       indique le numéro de Slot Secondaire (0 ~ 3).

PP       indique le numéro de Slot Primaire (0 ~ 3).

Nous verrons la signification des bits x plus bas, masquez ces bits pour pouvoir comparer avec le numéro de Slot retourné par la routine qui est donnée plus haut. Cependant, il faut faire attention car si SLTWRK donne 0xxx0011 (Slot primaire 3), ma routine renverra 1xxx1111 (Slot secondaire 3, primaire 3). Le résultat est équivalent lors d'un appel inter-Slot ou d'un ENASLT. Pour simplifier la programmation, ma routine donne TOUJOURS le résultat sous forme de Slot secondaire même s'il s'agit d'un Slot primaire. Il va de soi que s'il n'y a pas de disque virtuel, ces 5 bits seront tous à 0.

Le bit 6 de SLTWRK est un indicateur binaire. S'il est à 0, le disque virtuel n'est pas initialisé; au contraire, le disque virtuel est en place si ce bit est à 1.

Si le disque virtuel est installé, vous pouvez encore disposer de la mémoire vive non utilisée par celui-ci. Il suffit de savoir où s'arrête le disque virtuel. Les cases mémoires 00000H et 00001H contiennent l'adresse de début de la zone mémoire laissée libre par le disque virtuel. Les adresses 00002H à 0007FH ne sont pas utilisées sur MSX2, elles sont réservées aux futures versions du système MSX. Ne pas y toucher. Le disque virtuel commence en 00080h.

2. Une application utilise peut-être déjà la mémoire trouvée. Si tel est le cas, le bit 5 de SLTWRK sera à 1. Si le bit est à 0, si vous décidez vous même d'utiliser la mémoire vive trouvée pour votre application, vous devez mettre le bit 5 de SLTWRK à 1.

Voici le format des variables du tableau SLTWRK :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
F	RMD	APP	RES	SS1	SS0	PP1	PP0

PP0 et PP1 = Numéro de Slot primaire.

SS0 et SS1 = Numéro de Slot secondaire.

RES = Bit réservé.

APP = 1 si la RAM utilisée par une application ; 0 dans le cas contraire.

RMD = 1 si la RAM est utilisée par l'instruction CALL MEMINI ; 0 dans le cas contraire.

F = 0 pour slot primaire ; 1 pour Slot secondaire.

Note : Voir l'explication sur SLTWRK à « [Développer un programme en cartouche](#) », chapitre 13.15 page 314 pour plus de précision.

## Annexes

### *A – Liste des labels par ordre alphabétique*

Les listes suivantes sont sous forme de label que vous pourrez récupérer pour les insérer directement dans vos programmes en assembleur.

Liste des labels pour les routines du Bios de la Main-ROM :

<u>Nom</u>		<u>Adresse</u>	<u>Nom</u>		<u>Adresse</u>
BASRVN	equ	0002Bh	DOWNC	equ	00108h
BEEP	equ	000C0h	DSPFNK	equ	000CFh
BIGFIL	equ	0016Bh	ENASCR	equ	00044h
BREAKX	equ	000B7h	ENASLT	equ	00024h
CALATR	equ	00087h	EOL	equ	00168h
CALBAS	equ	00159h	ERAFNK	equ	000CCh
CALLF	equ	00030h	EXTROM	equ	0015Fh
CALPAT	equ	00084h	FETCHC	equ	00114h
CALSLT	equ	0001Ch	FILVRM	equ	00056h
CGTABL	equ	00004h	FNKSB	equ	000C9h
CHGCAP	equ	00132h	FORMAT	equ	00147h
CHGCPU	equ	00180h	GETCPU	equ	00183h
CHGCLR	equ	00062h	GETVCP	equ	00150h
CHGET	equ	0009Fh	GETVC2	equ	00153h
CHGMOD	equ	0005Fh	GETYPR	equ	00028h
CHGSND	equ	00135h	GICINI	equ	00090h
CHKNEW	equ	00165h	GRPPRT	equ	0008Dh
CHKSLZ	equ	00162h	GSPSIZ	equ	0008Ah
CHPUT	equ	000A2h	GTASPC	equ	00126h
CHRGTR	equ	00010h	GTPAD	equ	000DBh
CHSNS	equ	0009Ch	GTPDL	equ	000DEh
CKCNTC	equ	000BDh	GTSTCK	equ	000D5h
CLRSPR	equ	00069h	GTTRIG	equ	000D8h
CLS	equ	000C3h	INIFNK	equ	0003Eh
CNVCHR	equ	000ABh	INIGRP	equ	00072h
DCOMPR	equ	00020h	INIMLT	equ	00075h
DISSCR	equ	00041h	INITIO	equ	0003Bh

INITXT	equ	0006Ch	RSLREG	equ	00138h
INIT32	equ	0006Fh	SCALXY	equ	0010Eh
INLIN	equ	000B1h	SCANL	equ	0012Fh
ISCNTC	equ	000BAh	SCANR	equ	0012Ch
ISFLIO	equ	0014Ah	SETATR	equ	0011Ah
KEYINT	equ	00038h	SETC	equ	00120h
KILBUF	equ	00156h	SETGRP	equ	0007Eh
LDIRMV	equ	00059h	SETMLT	equ	00081h
LDIRVM	equ	0005Ch	SETRD	equ	00050h
LEFTC	equ	000FFh	SETTXT	equ	00078h
LFTQ	equ	000F6h	SETT32	equ	0007Bh
LPTOUT	equ	000A5h	SETWRT	equ	00053h
LPTSTT	equ	000A8h	SNSMAT	equ	00141h
MAPXYC	equ	00111h	STARUP	equ	00000h
MSXVER	equ	0002Dh	STMOTR	equ	000F3h
NMI	equ	00066h	STOREC	equ	00117h
NRDVRM	equ	00174h	STRTMS	equ	00099h
NSETCX	equ	00123h	SUBROM	equ	0015Ch
NSETRD	equ	0016Eh	SYNCHR	equ	00008h
NSTWRT	equ	00171h	TAPIN	equ	000E4h
NWRVRM	equ	00177h	TAPIOF	equ	000E7h
OUTLDP	equ	0014Dh	TAPION	equ	000E1h
OUTDO	equ	00018h	TAPOOF	equ	000F0h
PCMPLY	equ	00186h	TAPOON	equ	000EAh
PCMREC	equ	00189h	TAPOUT	equ	000EDh
PNTINI	equ	00129h	TDOWNC	equ	0010Bh
PINLIN	equ	000AEh	TOTEXT	equ	000D2h
POSIT	equ	000C6h	TUPC	equ	00105h
PUTQ	equ	000F9h	UPC	equ	00102h
PYSDIO	equ	00144h	VDP.DR	equ	00006h
QINLIN	equ	000B4h	VDP.DW	equ	00007h
RDPSG	equ	00096h	WRRES	equ	0017Dh
RDRES	equ	0017Ah	WRTPSG	equ	00093h
RDSLT	equ	0000Ch	WRTSLT	equ	00014h
RDVDP	equ	0013Eh	WRTVDP	equ	00047h
RDVRM	equ	0004Ah	WRTVRM	equ	0004Dh
READC	equ	0011Dh	WRTVRM	equ	00109h
RIGHTC	equ	000FCh	WSLREG	equ	0013Bh

Liste des labels pour les routines du Bios de la Sub-ROM :

<u>Nom</u>		<u>Adresse</u>	<u>Nom</u>		<u>Adresse</u>
ATRSCN	equ	00071h	INIPLT	equ	00141h
BASE	equ	00169h	INITXT	equ	000D5h
BASEF	equ	0016Dh	INIT32	equ	000D9h
BEEP	equ	0017Dh	INSLN0	equ	00125h
BLTDM	equ	001A9h	KNJPRT	equ	001BDh
BLTDV	equ	001A1h	KYKLOK	equ	00135h
BLTMD	equ	001A5h	LEFTC	equ	000ADh
BLTMV	equ	00199h	MAPXYC	equ	00091h
BLTVM	equ	00195h	NEWPAD	equ	001ADh
BLTVD	equ	0019Dh	NVBXFL	equ	000CDh
BLTVV	equ	00191h	NVBXLN	equ	000C9h
BOXLIN	equ	00081h	PAINT	equ	00069h
CALATR	equ	000FDh	PROMPT	equ	00181h
CALPAT	equ	000F9h	PSET	equ	0006Dh
CHGCLR	equ	00111h	PUTCHR	equ	00139h
CHGMDP	equ	001B5h	PUTSPR	equ	00151h
CHGMOD	equ	000D1h	RDVRM	equ	0010Dh
CLRSPP	equ	000F5h	READC	equ	00095h
CLRTXT	equ	00119h	REDCLK	equ	001F5h
CLS	equ	00115h	RIGHTC	equ	000A5h
COLOR	equ	00155h	RSTPLT	equ	00145h
DELLNO	equ	00121h	SCALXY	equ	0008Dh
DOBOXF	equ	00079h	SCANL	equ	000C5h
DOGRPH	equ	00085h	SCANR	equ	000C1h
DOLINE	equ	0007Dh	SCOPY	equ	0018Dh
DOWNC	equ	000B5h	SCREEN	equ	00159h
DSPFNK	equ	0011Dh	SDFSCR	equ	00185h
GETPAT	equ	00105h	SETATR	equ	00099h
GETPLT	equ	00149h	SETC	equ	0009Dh
GETPUT	equ	001B1h	SETGRP	equ	000EDh
GLINE	equ	00075h	SETMLT	equ	000F1h
GRPPRT	equ	00089h	SETPAG	equ	0013Dh
GSPSIZ	equ	00101h	SETPLT	equ	0014Dh
INIGRP	equ	000DDh	SETS	equ	00179h
INIMLT	equ	000E1h	SETSCR	equ	00189h

SETTXT	equ	000E5h	VDPF	equ	00165h
SETT32	equ	000E9h	VDPSTA	equ	00131h
TDOWNC	equ	000B1h	VPEEK	equ	00175h
TLEFTC	equ	000A9h	VPOKE	equ	00171h
TRIGHT	equ	000A1h	WIDTHS	equ	0015Dh
TUPC	equ	000B9h	WRTCLK	equ	001F9h
UPC	equ	000BDh	WRTVDP	equ	0012Dh
VDP	equ	00161h			

Liste des labels pour les variables système :

Nom		Adresse	Nom		Adresse
ACPAGE	equ	0FAF6h	CHRCNT	equ	0FAF9h
ARG	equ	0F847h	CLIKFL	equ	0FBD9h
ARYTAB	equ	0F6C4h	CLIKSW	equ	0F3DBh
ARYTA2	equ	0F7B5h	CLINEF	equ	0F935h
ASPCT1	equ	0F40Bh	CLMSLT	equ	0F3B2h
ASPCT2	equ	0F40Dh	CLOC	equ	0F92Ah
ASPECT	equ	0F931h	CMASK	equ	0F92Ch
ATRBAS	equ	0F928h	CNPNTS	equ	0F936h
ATRBYT	equ	0F3F2h	CNSDFG	equ	0F3DEh
AUTFLG	equ	0F6AAh	CODSAV	equ	0FBCCCh
AUTINC	equ	0F6ADh	CONLO	equ	0F66Ah
AUTLIN	equ	0F6ABh	CONSAV	equ	0F668h
AVCSAV	equ	0FAF7h	CONTXT	equ	0F666h
BAKCLR	equ	0F3EAh	CONTYP	equ	0F669h
BASROM	equ	0FBB1h	CPCNT	equ	0F939h
BDRCLR	equ	0F3EBh	CPCNT8	equ	0F93Bh
BOTTOM	equ	0FC48h	CPLOTF	equ	0F938h
BRDATR	equ	0FCB2h	CRCSUM	equ	0h93Dh
BUF	equ	0F55Eh	CRTCNT	equ	0F3B1h
BUFEND	equ	0FC18h	CS120	equ	0F3FCh
BUFMIN	equ	0F55Dh	CSAVEA	equ	0F942h
CAPST	equ	0FCABh	CSAVEM	equ	0F944h
CASPRV	equ	0FCB1h	CSCLXY	equ	0F941h
CENCNT	equ	0F933h	CSRSW	equ	0FCA9h
CGPBAS	equ	0F924h	CSRX	equ	0F3DDh
CGPNT	equ	0F91Fh	XSRY	equ	0F3DCh

CSTCNT	equ	0F93Fh	FLBMEM	equ	0FCAEh
CSTYLE	equ	0FCAAh	FLGINP	equ	0F6A6h
CURLIN	equ	0F41Ch	FKNFLG	equ	0FBCEh
CXOFF	equ	0F945h	FNKSTR	equ	0F87Fh
CYOFF	equ	0F947h	FNKSWI	equ	0FBCDh
DAC	equ	0F7F6h	FORCLR	equ	0F3E9h
DATLIN	equ	0F6A3h	FRCNEW	equ	0F3F5h
DATPTR	equ	0F6C8h	FRETOP	equ	0F69Bh
DECCNT	equ	0F7F4h	FSTPOS	equ	0FBCAh
DECTMP	equ	0F7F0h	FUNACT	equ	0F7BAh
DECTM2	equ	0F7F2h	GETPNT	equ	0F3FAh
DEFTBL	equ	0F6CAh	GRPACX	equ	0FCB7h
DEVICE	equ	0FD99h	GRPACY	equ	0FCB9h
DIMFLG	equ	0F662h	GRPATR	equ	0F3CDh
DORES	equ	0F664h	GRPCGP	equ	0F3CBh
DONUM	equ	0F665h	GRPCOL	equ	0F3C9h
DOT	equ	0F6B5h	GRPHED	equ	0FCA6h
DPPAGE	equ	0FAF5h	GRPNAM	equ	0F3C7h
DRWANG	equ	0FCBDh	GRPPAT	equ	0F3CFh
DRWFLG	equ	0FCBBh	GXPOS	equ	0FCB3h
DRWSCL	equ	0FCBCh	GYPOS	equ	0FCB5h
DSCPTR	equ	0F699h	HEADER	equ	0F40Ah
DSCTMP	equ	0F698h	HIGH	equ	0F408h
ENDBUF	equ	0F660h	HIMEM	equ	0FC4Ah
ENDFOR	equ	0F6A1h	HOLD	equ	0F83Eh
ENDPRG	equ	0F40Fh	HOLD2	equ	0F836h
ENSTOP	equ	0FBB0h	HIGH	equ	0F408h
ERRFLG	equ	0F414h	HOLD8	equ	0F806h
ERRLIN	equ	0F6B3h	INSFLG	equ	0FCA8h
ERRTXT	equ	0F6B7h	INTCNT	equ	0FCA2h
ESCCNT	equ	0FCA7h	INTFLG	equ	0FC9Bh
EXBRSA	equ	0FAF8h	INTVAL	equ	0FCA0h
EXPTBL	equ	0FCC1h	JIFFY	equ	0FC9Eh
FACLO	equ	0F7F8h	KANAMD	equ	0FCADh
FBuffer	equ	0F7C5h	KANAST	equ	0FCACH
FILNAM	equ	0F866h	KBFMIN	equ	0F41Eh
FILNM2	equ	0F871h	KBUF	equ	0F41Fh
FILTAB	equ	0F860h	KEYBUF	equ	0FBF0h



LFPROG	equ	0F954h	NULBUF	equ	0F862h
LINL32	equ	0F3AFh	OLDKEY	equ	0FBDAh
LINL40	equ	0F3AEh	OLDLIN	equ	0F6BEh
LINLEN	equ	0F3B0h	OLDSCR	equ	0FCB0h
LINTTB	equ	0FBB2h	OLDTXT	equ	0F6C0h
LINWRK	equ	0FC18h	ONEFLG	equ	0F6BBh
LOGOPR	equ	0FB02h	ONELIN	equ	0F6B9h
LOHADR	equ	0F94Bh	ONGSBF	equ	0FBD8h
LOHCNT	equ	0F94Dh	OPRTYP	equ	0F664h
LOHDIR	equ	0F94Ah	PADX	equ	0FC9Dh
LOHMSK	equ	0F949h	PADY	equ	0FC9Ch
LOW	equ	0F406h	PARM1	equ	0F6E8h
LOWLIM	equ	0FCA4h	PARM2	equ	0F750h
LPTPOS	equ	0F415h	PATBAS	equ	0F926h
MAXDEL	equ	0F92Fh	PATWRK	equ	0FC40h
MAXFIL	equ	0F85Fh	PDIREC	equ	0F953h
MAXUPD	equ	0F3ECh	PLYCNT	equ	0FB40h
MCLFLG	equ	0F958h	PRMFLG	equ	0F7B4h
MCLLEN	equ	0FB3Bh	PRMLEN	equ	0F6E6h
MCLPTR	equ	0FB3Ch	PRMLN2	equ	0F74Eh
MCLTAB	equ	0F956h	PRMPRV	equ	0F74Ch
MEMSIZ	equ	0F672h	PRMSTK	equ	0F6E4h
MINDEL	equ	0F92Dh	PROCNM	equ	0FD89h
MINUPD	equ	0F3EFh	PRSCNT	equ	0FB35h
MLTATR	equ	0F3D7h	PTRFIL	equ	0F864h
MLTCGP	equ	0F3D5h	PTRFLG	equ	0F6A9h
MLTCOL	equ	0F3D3h	PUTPNT	equ	0F3F8h
MLTNAM	equ	0F3D1h	QUEBAK	equ	0F971h
MLTPAT	equ	0F3D9h	QUETAB	equ	0F959h
MNROM	equ	0FCC1h	QUEUEN	equ	0FB3Eh
MODE	equ	0FAFCh	QUEUEES	equ	0F3F3h
MOVCNT	equ	0F951h	RAWPRT	equ	0F418h
MUSICF	equ	0FB3Fh	REPCNT	equ	0F3F7h
NAMBAS	equ	0F922h	REQSTP	equ	0FC6Ah
NEWKEY	equ	0FBE5h	RG0SAV	equ	0F3DFh
NLONLY	equ	0F87Ch	RG1SAV	equ	0F3E0h
NOFUNS	equ	0F7B7h	RG2SAV	equ	0F3E1h
NTMSXP	equ	0F417h	RG3SAV	equ	0F3E2h

RG4SAV	equ	0F3E3h	TEMP9	equ	0F7B9h
RG5SAV	equ	0F3E4h	TEMPPT	equ	0F678h
RG6SAV	equ	0F3E5h	TEMPST	equ	0F67Ah
RG7SAV	equ	0F3E6h	TOCNT	equ	0FB03h
ROMA	equ	0FAFAh	TRCFLG	equ	0F7C4h
RNDX	equ	0F857h	TRGFLG	equ	0F3E8h
RSFCB	equ	0FB04h	TRPTBL	equ	0FC4Ch
RSIQLN	equ	0FB06h	TTYPOS	equ	0F661h
RS2IQ	equ	0FAF5h	TXTATR	equ	0F3B9h
RTPROG	equ	0F955h	TXTCGP	equ	0F3B7h
RTYCNT	equ	0FC9Ah	TXTCOL	equ	0F3B5h
RUNBNF	equ	0FCBEh	TXTNAM	equ	0F3B3h
RUNFLG	equ	0F866h	TXTPAT	equ	0F3BBh
SAVEND	equ	0F87Dh	TXTTAB	equ	0F676h
SAVENT	equ	0FCBFh	T32ATR	equ	0F3C3h
SAVSP	equ	0FB36h	T32CGP	equ	0F3C1h
SAVSTK	equ	0F6B1h	T32COL	equ	0F3BFh
SAVTXT	equ	0F6AFh	T32NAM	equ	0F3BDh
SAVVOL	equ	0FB39h	T32PAT	equ	0F3C5h
SCNCNT	equ	0F3F6h	USRTAB	equ	0F39Ah
SCRMOD	equ	0FCAFh	VALTYP	equ	0F663h
SFTKEY	equ	0FBEBh	VARTAB	equ	0F6C2h
SKPCNT	equ	0F94Fh	VCBA	equ	0FB41h
SLTATR	equ	0FCC9h	VCBB	equ	0FB66h
SLTTBL	equ	0FCC5h	VCBC	equ	0FB8Bh
SLTWRK	equ	0FD09h	VLZADR	equ	0F419h
STATFL	equ	0F3E7h	VLZDAT	equ	0F41Bh
STKTOP	equ	0F674h	VOICAQ	equ	0F975h
STREND	equ	0F6C6h	VOICBQ	equ	0F9F5h
SUBFLG	equ	0F6A5h	VOICBC	equ	0FA75h
SWPTMP	equ	0F7BCh	VOICEN	equ	0FB38h
TEMP	equ	0F6A7h	WINWID	equ	0FCA5h
TEMP2	equ	0F6BCh	XSAVE	equ	0FAFEh
TEMP3	equ	0F69Dh	YSAVE	equ	0FB00h
TEMP8	equ	0F69Fh			

Liste des labels pour les Hooks :

<u>Nom</u>		<u>Adresse</u>	<u>Nom</u>		<u>Adresse</u>
H.ATTR	equ	0FE1Ch	H.ERRF	equ	0FF02h
H.BAKU	equ	0FEADh	H.ERRO	equ	0FFB1h
H.BINL	equ	0FE76h	H.ERRP	equ	0FEFDh
H.BINS	equ	0FE71h	H.EVAL	equ	0FF70h
H.BUFL	equ	0FF8Eh	H.FIEL	equ	0FE2Bh
H.CHGE	equ	0FDC2h	H.FILE	equ	0FE7Bh
H.CHPU	equ	0FDA4h	H.FILO	equ	0FE85h
H.CHRG	equ	0FF48h	H.FINE	equ	0FF1Bh
H.CLEA	equ	0FED0h	H.FING	equ	0FF7Ah
H.CMD	equ	0FE0Dh	H.FINI	equ	0FF16h
H.COMP	equ	0FF57h	H.FINP	equ	0FF5Ch
H.COPY	equ	0FE08h	H.FORM	equ	0FFACh
H.CRDO	equ	0FEE9h	H.FPOS	equ	0FEA8h
H.CRUN	equ	0FF20h	H.FRET	equ	0FF9Dh
H.CRUS	equ	0FF25h	H.FRME	equ	0FF66h
H.CVD	equ	0FE49h	H.FRQI	equ	0FF93h
H.CVI	equ	0FE3Fh	H.GEND	equ	0FEC6h
H.CVS	equ	0FE44h	H.SAVD	equ	0FE94h
H.DEVN	equ	0FEC1h	H.SAVE	equ	0FE6Ch
H.DGET	equ	0FFE8h	H.SCNE	equ	0FF98h
H.DIRD	equ	0FF11h	H.SCRE	equ	0FFC0h
H.DOGR	equ	0FEF3h	H.GETP	equ	0FE4Eh
H.DSKC	equ	0FEEeh	H.GONE	equ	0FF43h
H.DSKF	equ	0FE12h	H.INDS	equ	0FEA8h
H.RETU	equ	0FF4Dh	H.INIP	equ	0FDC7h
H.RSET	equ	0FE26h	H.INLI	equ	0FDE5h
H.RSLF	equ	0FE8Fh	H.IPL	equ	0FE03h
H.RUNC	equ	0FECBh	H.ISFL	equ	0FEDFh
H.DSKI	equ	0FE17h	H.ISMI	equ	0FF7Fh
H.DSKO	equ	0FDEFh	H.ISRE	equ	0FF2Ah
H.DSPC	equ	0FDA9h	H.KEYC	equ	0FDCCh
H.DSPF	equ	0FDB3h	H.KEYI	equ	0FD9Ah
H.EOF	equ	0FEA3h	H.KILL	equ	0FDFEh
H.ERAC	equ	0FDAEh	H.KYEA	equ	0FDD1h
H.ERAF	equ	0FDB8h	H.LIST	equ	0FF89h

H.LOC	equ	0FE99h	H.NTFL	equ	0FE62h
H.LOF	equ	0FE9Eh	H.NTFN	equ	0FF2Fh
H.LOPD	equ	0FED5h	H.NTPL	equ	0FF6Bh
H.LPTO	equ	0FFB6h	H.NULO	equ	0FE5Dh
H.LPTS	equ	0FFBBh	H.OKNO	equ	0FF75h
H.LSET	equ	0FE21h	H.ONGO	equ	0FDEAh
H.MAIN	equ	0FF0Ch	H.OUTD	equ	0FEE4h
H.MERG	equ	0FE67h	H.PARD	equ	0FEB2h
H.MKD\$	equ	0FE3Ah	H.PHYD	equ	0FFA7h
H.MKI\$	equ	0FE30h	H.PINL	equ	0FDDBh
H.SETF	equ	0FE53h	H.PLAY	equ	0FFC5h
H.SETS	equ	0FDF4h	H.POSD	equ	0FEBCh
H.SNGF	equ	0FF39h	H.PRGE	equ	0FEF8h
H.STKE	equ	0FEDAh	H.PRTF	equ	0FF52h
H.MKS\$	equ	0FE35h	H.PTRG	equ	0FFA2h
H.NAME	equ	0FDF9h	H.QINL	equ	0FDE0h
H.NEWS	equ	0FF3Eh	H.READ	equ	0FF07h
H.NMI	equ	0FDD6h	H.TIMI	equ	0FD9Fh
H.NODE	equ	0FEB7h	H.TOTE	equ	0FDBDh
H.NOFO	equ	0FE58h	H.TRMN	equ	0FF61h
H.NOTR	equ	0FF34h	H.WIDT	equ	0FF84h

## B – Les registres du VDP

Contenu des registres des VDP du MSX :

Registre 0 :	0	DG	IE2	IE1	M5	M4	M3	EV
Registre 1 :	0	BL	IE0	M1	M2	0	SI	MAG
Registre 2 :	0	N16	N15	N14	N13	N12	N11	N10
Registre 3 :	C13	C12	C11	C10	C9	C8	C7	C6
Registre 4 :	0	0	F16	F15	F14	F13	F12	F11
Registre 5 :	S14	S13	S12	S11	S10	S9	S8	S7
Registre 6 :	0	0	P16	P15	P14	P13	P12	P11
Registre 7 :	TC3	TC2	TC1	TC0	BD3	BD2	BD1	BD0
Registre 8 :	MS	LP	TP	CB	VR	0	SPD	BW
Registre 9 :	LN	0	S1	S0	IL	E0	NT	DC
Registre 10 :	0	0	0	0	0	C16	C15	C14
Registre 11 :	0	0	0	0	0	0	S16	S15
Registre 12 :	T23	T22	T21	T20	BC3	BC2	BC1	BC0
Registre 13 :	ON3	ON2	ON1	ON0	OF3	OF2	OF1	OF0
Registre 14 :	0	0	0	0	0	V16	V15	V14
Registre 15 :	0	0	0	0	ST3	ST2	ST1	ST0
Registre 16 :	0	0	0	0	CC3	CC2	CC1	CC0
Registre 17 :	AII	0	RS5	RS4	RS3	RS2	RS1	RS0
Registre 18 :	V3	V2	V1	V0	H3	H2	H1	H0
Registre 19 :	IL7	IL6	IL5	IL4	IL3	IL2	IL1	IL0
Registre 20 :	0	0	0	0	0	0	0	0
Registre 21 :	0	0	1	1	1	0	1	1
Registre 22 :	0	0	0	0	0	1	0	1
Registre 23 :	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Registre 25 :	0	CMD	VDS	YAE	YJK	WTE	MSK	SP2
Registre 26 :	0	0	HO8	HO7	HO6	HO5	HO4	HO3
Registre 27 :	0	0	0	0	0	HO2	HO1	HO0

} V9958

Registre 32 :	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0
Registre 33 :	0	0	0	0	0	0	0	SX8
Registre 34 :	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0
Registre 35 :	0	0	0	0	0	0	SY9	SY8

Registre 36 :	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0
Registre 37 :	0	0	0	0	0	0	0	DX8
Registre 38 :	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0
Registre 39 :	0	0	0	0	0	0	DY9	DY8
Registre 40 :	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0
Registre 41 :	0	0	0	0	0	0	0	NX8
Registre 42 :	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0
Registre 43 :	0	0	0	0	0	0	NY9	NY8
Registre 44 :	CL7	CL6	CL5	CL4	CL3	CL2	CL1	CL0
Registre 45 :	0	MXC	MXD	MXS	DIY	DIX	EQ	MAJ
Registre 46 :	CM3	CM2	CM1	CM0	LO3	LO2	LO1	LO0

Liste des bits :

Nom	Registre	Fonction
AII	17	0 = Auto incrémentation ; 1 = Normal.
BC3 ~ BC0	12	En cas de clignotement ; couleur 2nde partie.
BD3 ~ BD0	7	« back drop color ».
BL	1	1 = Affichage autorisé ; 0 = Affichage interdit.
BW	8	1 = Noir et blanc (32 teintes) ; 0 = Couleur.
CB	8	1 = Bus de couleur en entrée ; 0 = En sortie.
CC3 ~ CC0	16	N° de couleur lors d'un accès palette.
CL3 ~ CL0	44	Couleur lors d'une commande.
CM3 ~ CM0	46	Code de commande à exécuter.
C13 ~ C6	3	Bits de poids faible qui définissent l'adresse de la table des couleurs.
C16 ~ C14	10	Bits de poids fort qui définissent l'adresse de la table des couleurs.
DC	9	1 = DTCLK en entrée ; 0 = DTCLK en sortie
DG	0	1 = Bus de couleur en entrée ; 0 = En sortie
DIX	45	Direction horizontale 1 = Gauche ; 0 = Droite
DIY	45	Direction verticale 1 = Haut ; 0 = bas
DO7 ~ DO0	23	Décalage vertical de l'écran (0-255)
DX7 ~ DX0	36	Abscisse destination (8 bits poids faible)
DX8	37	Abscisse destination (bit poids fort)
DY7 ~ DY0	38	Ordonnée destination (8 bits poids faible)
DY9 et DY8	39	Ordonnée destination (2 bits poids fort)
EQ	45	Srch : 1 = Couleur de bordure trouvée ; 0 = Couleur autre que bordure
E0	9	1 = Affichage alterné de 2 pages ; 0 = Normal
F16 ~ F11	4	Adresse table des formes (6 bits poids forts sur 17)

H3 ~ H0	18	Ajustement horizontal de l'écran (7 = Gauche ; 0 = Centre ; 8 = Droite)
IE0	1	1 = Interruption scan horizontal ok ; 0 = Interdit
IE1	0	1 = Interruption scan horizontal ok ; 0 = Interdit
IE2	0	1 = Interruption stylo optique ok ; 0 = Interdit
IL	9	1 = Affichage entrelacé ; 0 = Non entrelacé
IL7 ~ IL0	19	N° de ligne où l'interruption doit se déclencher lors d'un scan
LN	9	Résolution verticale de l'écran. 1 = 212 points ; 0 = 192 points
LO3 ~ LO0	46	Opérateur logique lors d'une commande
LP	8	1 = Stylo optique autorisé ; 0 = Stylo interdit
MAG	1	1 = Sprites agrandis ; 0 = Sprites normaux
MAJ	45	Côté le plus long 1 = Vertical ; 0 = Horizontal
MS	8	1 = Souris autorisée ; 0 = Souris interdite
MXC	45	Inutilisé sur msx2
MXD	45	Destination : 1 = Vram étendue/ 0 = Vram
MXS	45	Source : 1 = Vram étendue/ 0 = Vram
M2 ~ M1	1	2 bits de poids faible du mode d'écran
M5 ~ M3	0	3 bits de poids fort du mode d'écran
NT	9	1 = Pal (313 lignes) ; 0 = Ntsc (262 lignes)
NX7 ~ NX0	40	Longueur (8 bits poids faible)
NX8	41	Longueur (bit poids fort)
NY7 ~ NY0	42	Hauteur (8 bits poids faible)
NY9 et NY8	43	Hauteur (2 bits poids fort)
N16 ~ N10	2	Adresse table des caractères ou Bitmap (7 bits de poids fort sur 17)
OF3 ~ OF0	13	En cas de clignotement ; temps éteint
ON3 ~ ON0	13	En cas de clignotement ; temps allumé
P16 ~ P11	6	Adresse table génératrice des Sprites (6 bits de poids fort sur 17)
RS5 ~ RS0	17	N° de registre lors d'un adressage indirect
SI	1	1 = Sprites 16*16 ; 0 = Sprites 8*8
SPD	8	1 = Sprites interdits ; 0 = Sprites autorisés
ST3 ~ ST0	15	N° registre statut lors d'une lecture
SX7 ~ SX0	32	Abscisse source (8 bits poids faible)
SX8	33	Abscisse source (bit poids fort)
SY7 ~ SY0	34	Ordonnée source (8 bits poids faible)
SY9 et SY8	35	Ordonnée source (2 bits poids fort)
S1 et S0	9	Choix du mode simultané ou pas
S14 ~ S7	6	Adresse table des attributs des Sprites (10 bits...
S16 et S15	11	... de poids fort sur 17)
TC3 ~ TC0	7	Couleur du texte en screen 0 et 1

TP	8	Couleur 0 égale à la couleur de la palette
T23 ~ T20	12	En cas de clignotement, couleur 1ère partie
V3 ~ V0	18	Ajustement vertical de l'écran (8 = Bas ; 0 = milieu ; 7 = Haut)
V16 ~ V14	14	3 bits de poids fort de l'adresse VRAM
VR	8	Type de VRAM 1 = 64x1 bit ou 64x4 bits ; 0 = 16x1 bit ou 16x4 bits



## ***C – Les cartes de la mémoire vidéo***

La carte mémoire vidéo par défaut pour chaque mode graphique :

### **SCREEN 0    40 colonnes                      MSX1 à MSX turbo R**

Table des caractères :	00000h~003BFh	960 octets
Table des formes :	00800h~00FFFh	2048 octets
Table de la palette (MSX2~) :	00400h~0041Fh	32 octets

### **SCREEN 0    80 colonnes                      MSX2 à MSX turbo R**

Table des caractères 24 / 26,5 lignes :	00000h~0077Fh / 0086Fh	1920 / 2160 octets
Table des couleurs 24 / 26,5 lignes :	00800h~008EFh / 0090Dh	240 / 270 octets
Table des formes :	01000h~017FFh	2048 octets
Table de la palette (MSX2~) :	00F00h~00F1Fh	32 octets

### **SCREEN 1    32 colonnes                      MSX1 à MSX turbo R**

Table des caractères :	01800h~01AFFh	768 octets
Table des couleurs :	02000h~0201Fh	32 octets
Table des formes :	00000h~007FFh	2048 octets
Table des attributs Sprite :	01B00h~01B7Fh	128 octets
Table des formes de Sprite :	03800h~03FFFh	2048 octets
Table de la palette (MSX2~) :	02020h~0203Fh	32 octets

### **SCREEN 2    MSX1 à MSX turbo R**

table des motifs :	01800h~01AFFh	768 octets
Table des couleurs :	02000h~037FFh	6144 octets
Table des formes :	00000h~017FFh	6144 octets
Table des attributs Sprite :	01B00h~01B7Fh	128 octets
Table des formes de Sprite :	03800h~03FFFh	2048 octets
Table de la palette (MSX2~) :	02020h~0203Fh	32 octets

### SCREEN 3

MSX1 à MSX turbo R

Table des caractères :	00800h~00AFFh	768 octets
Table des formes :	00000h~007FFh	2048 octets
Table des attributs Sprite :	01B00h~01B7Fh	128 octets
Table des formes de Sprite :	03800h~03FFFh	2048 octets
Table de la palette (MSX2~) :	02020h~0203Fh	32 octets

### SCREEN 4

MSX2 à MSX turbo R

table des motifs :	01800h~01AFFh	768 octets
Table des couleurs :	02000h~037FFh	6144 octets
Table des formes :	00000h~017FFh	6144 octets
Table des attributs Sprite :	01E00h~01E7Fh	128 octets
Table des formes de Sprite :	03800h~03FFFh	2048 octets
Table des couleurs de Sprite :	01C00h~0D7FFh	512 octets
Table de la palette des couleurs :	01E80h~01E9Fh	32 octets

### SCREEN 5

MSX2 à MSX turbo R

Table Bitmap 192 / 212 lignes :	00000h~05FFFh / 069FFh	24576 / 27136 octets
Table des attributs Sprite :	07600h~0767Fh	128 octets
Table des formes de Sprite :	07800h~07FFFh	2048 octets
Table des couleurs de Sprite :	07400h~075FFh	512 octets
Table de la palette des couleurs :	07680h~0769Fh	32 octets

### SCREEN 6

MSX2 à MSX turbo R

Table Bitmap 192 / 212 lignes :	00000h~05FFFh / 069FFh	24576 / 27136 octets
Table des attributs Sprite :	07600h~0767Fh	128 octets
Table des formes de Sprite :	07800h~07FFFh	2048 octets
Table des couleurs de Sprite :	07400h~075FFh	512 octets
Table de la palette des couleurs :	07680h~0769Fh	32 octets

## SCREEN 7

### MSX2 à MSX turbo R

Table Bitmap 192 / 212 lignes :	00000h~0BFFFh / 0D3FFh	49152 / 54272 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs de Sprite :	0F800h~0F9FFh	512 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets

## SCREEN 8

### MSX2 à MSX turbo R

Table Bitmap 192 / 212 lignes :	00000h~0BFFFh / 0D3FFh	49152 / 54272 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs de Sprite :	0F800h~0F9FFh	512 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets

## SCREEN 9

### MSX2 coréen

Table Bitmap 192 / 212 lignes :	00000h~05FFFh / 069FFh	24576 / 27136 octets
Table des attributs Sprite :	00800h~0767Fh ( <i>inutilisé par le Basic</i> )	128 octets
Table des formes de Sprite :	07800h~07FFFh ( <i>inutilisé par le Basic</i> )	2048 octets
Table des couleur des Sprite :	07400h~075FFh ( <i>inutilisé par le Basic</i> )	512 octets
Table de la palette des couleurs :	07680h~0769Fh	32 octets

## SCREEN 10 / 11

### MSX2+

Table Bitmap 192 / 212 lignes :	00000h~0BFFFh / 0D3FFh	49152 / 54272 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs de Sprite :	0F800h~0F9FFh	512 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets

## SCREEN 12

MSX2+

Table Bitmap 192 / 212 lignes :	00000h~0BFFFh / 0D3FFh	49152 / 54272 octets
Table des attributs Sprite :	0FA00h~0FA7Fh	128 octets
Table des formes de Sprite :	0F000h~0F7FFh	2048 octets
Table des couleurs des Sprites :	0F800h~0F9FFh	512 octets
Table de la palette des couleurs :	0FA80h~0FA9Fh	32 octets

## D – Jeux de caractères MSX

Les jeux de caractères MSX sont tous basés sur le code ASCII 7 bits et étendu de la façon suivante sur les MSX occidentaux / internationaux :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NULL	GRAPH	CTRL B	CTRL C	CTRL D	CTRL E	CTRL F	BEEP	BS	TAB	LF	HOME	CLS	RET	CTRL N	CTRL O
1	CTRL P	CTRL Q	INS	CTRL S	CTRL T	CTRL U	CTRL V	CTRL W	SEL	CTRL Y	CTRL Z	ESC	→	←	↑	↓
2	SPC	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8	ç	ü	é	à	ä	å	ä	ç	ê	ë	è	ï	î	ï	ä	Å
9	é	æ	Æ	ø	ö	ò	û	ù	ý	ö	Ü	¢	£	¥	₣	₧
A	á	í	ó	ú	ñ	ñ	ä	ö	¿	¬	¬	½	¼	¾	¼	¾
B	Ä	ä	ï	ï	ö	ö	Ü	ü	¼	¾	¾	¾	¾	¾	¾	¾
C	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
D	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
E	α	β	Γ	Π	Σ	σ	μ	τ	ϕ	θ	Ω	δ	ω	ω	€	Π
F	≡	±	≥	≤	↑	↓	÷	×	÷	÷	÷	÷	÷	÷	÷	÷
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4		☺	☹	♥	♣	♠	♣	♠	♣	♠	♣	♠	♣	♠	♣	♠
5	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

- Les 8 premières lignes sont les mêmes caractères que ceux du code ASCII 7 bits.
- Les lignes 0 et 1 ainsi que la case 7Fh sont des [codes de contrôle](#) qui ne s'affichent pas à l'écran mais ont pour effet suivant.

GRAPH = Permet d'afficher un caractère graphique étendu.

CTRL B = Place le curseur sur le mot précédent.

CTRL E = Efface la ligne à droite du curseur.

CTRL F = Place le curseur sur le mot suivant.

LF = Descend le curseur d'une ligne.

HOME = Positionnement du curseur tout en haut à gauche.

CTRL N = Place le curseur en fin de ligne.

INS = Passage en mode insertion / écraser.

CTRL U = Efface la ligne ou se trouve le curseur.

SEL = Touche de sélection nom.

↑ = Curseur vers le haut.

→ = Curseur vers la droite.

← = Curseur vers la gauche.

↓ = Curseur vers le bas.

- Les caractères verts sont des caractères spécifiques aux MSX occidentaux.
- Le dernier caractère (FFh) correspond au curseur.
- La table du bas, à deux lignes (orange), sont des caractères graphiques étendus. Pour les afficher, chacun doit être précédé du caractère 01h. Ils sont aussi spécifiques au MSX occidentaux.

Le jeu de caractères des MSX japonais :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	GRPH	CTRL E	CTRL C	CTRL D	CTRL E	CTRL F	BEEP	BS	TAB	LF	HOME	CLS	RET	CTRL M	CTRL O
1	CTRL P	CTRL R	INS	CTRL S	CTRL T	CTRL U	CTRL V	CTRL W	SEL	CTRL Y	CTRL Z	ESC	→	←	↑	↓
2	SPC	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8	☸	☸	☸	☸	☸	☸	を	あ	い	う	え	お	や	ゆ	よ	っ
9		あ	い	う	え	お	か	き	く	け	こ	さ	し	す	せ	ぞ
A		。」「	、	・	ヲ	ア	イ	ウ	エ	オ	カ	ユ	ヨ	ツ		
B	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
C	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
D	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	ゝ	。ゝ
E	た	ち	つ	て	と	な	に	ぬ	ね	の	は	ひ	ふ	へ	ほ	ま
F	み	む	め	も	や	ゆ	よ	ら	り	る	れ	ろ	わ	ん		■

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4		月	火	水	木	金	土	日	年	円	時	分	秒	百	千	万
5	元	十	十	十	十	十	一	一	一	一	一	一	一	一	一	一

- Les 8 premières lignes sont les mêmes caractères que ceux du code ASCII 7 bits. Excepté le caractère « \ » (5Ch) qui a été remplacé par le caractère « ¥ » du code JIS 8 bits de Microsoft. (Le code JIS 8 bits est basé sur le code ASCII 7 bits.)
- Les caractères violets proviennent du code JIS 8 bits.
- Les caractères verts, bien que non définis par la norme JIS et laissé au libre arbitre des , sont JIS 8

bits les mêmes Hiragana.

- Le dernier caractère (FFh) correspond au curseur.
- La table du bas, à deux lignes, sont des caractères étendus propres au MSX. Pour les afficher, chacun doit être précédé du caractère 01h.

Notes :

Les MSX2, ou plus récents, japonais possèdent en option ou en interne une Kanji ROM comprenant un police de caractères au format JIS 16 bits niveau 1 et niveau 2 pour les MSX plus récents.

Ces MSX ou cartouches ont en général aussi un « Kanji BASIC » et un « MSX-JE » pour les MSX plus récents. Le « Kanji BASIC » ajoute au MSX la possibilité d'afficher les caractères japonais, les Kanji et même des modes d'écran de texte adaptés aux Kanji. Le « MSX-JE » sert à faciliter la saisie des Kanji.

Le jeu de caractères des MSX arabes :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NULL	GRAPH	CTRL B	CTRL C	CTRL D	CTRL E	CTRL F	BEEP	BS	TAB	LF	HOME	CLS	RET	CTRL N	CTRL O
1	CTRL P	CTRL Q	INS	CTRL S	CTRL T	CTRL U	CTRL V	CTRL W	SEL	CTRL Y	CTRL Z	ESC	→	←	↑	↓
2	SPC	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8	é	à	á	ç	ê	ë	è	ï	î	ó	ô	ù	û	ö	°	no
9	p	q	r	s	t	u	v	w	x	y	z	{		}		
A		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
B	+	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
C	@	أ	ب	ت	ث	ج	د	هـ	و	ز	ح	ط	ظ	ع	ف	ق
D	ك	ر	س	ش	ص	ض	ط	ظ	ع	ف	ق	ك	ر	س	ش	ص
E	ض	ط	ظ	ع	ف	ق	ك	ر	س	ش	ص	ض	ط	ظ	ع	ف
F	ق	ك	ر	س	ش	ص	ض	ط	ظ	ع	ف	ق	ك	ر	س	ش

- Les 8 premières lignes sont les mêmes caractères que ceux du code ASCII 7 bits.
- Les caractères verts sont des caractères propres aux MSX arabes.
- Les MSX arabes n'ont pas de caractères étendus.

Note :

Les MSX arabes ont deux Main-ROM, lorsqu'on presse la touche CTRL pendant le démarrage du MSX jusqu'au son « bip », le MSX démarre sur la Main-ROM ayant un jeu de caractères international.

Le jeu de caractères des MSX coréens :

Les MSX coréens utilisent une police de caractères basée sur le code ASCII 7 bit avec les caractères coréens en plus.

Les 128 premiers caractères sont les mêmes caractères que ceux du code ASCII. Excepté le caractère « \ » (5Ch) qui a été remplacé par le caractère « \ ».

Les caractères suivants s'affichent d'une façon assez spécifique. Un caractère coréen (Hangul) est composé de 2 ou 3 caractères mais prend la place de quatre caractères 8x8 (deux au dessus deux en dessous) à l'écran.

Les MSX2 coréens, ont un mode d'écran spécifique (le SCREEN 9). Ce mode est un mode texte spécifique adapté au coréen basé sur le SCREEN 6. Ces MSX2 sont équipés d'une ROM supplémentaire de 32Ko pour afficher les caractères Hangul.



## *E – Exemples de matrices de clavier*

Le caractère à droite correspond à celui obtenu avec la touche SHIFT maintenue.

Clavier « AZERTY » des Philips NMS-8250/55/80 et Sony HB-F500/700/900 :

Ligne	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	è 7	§ 6	( 5	' 4	" 3	é 2	& 1	à 0
01	m M	\$ *	^ "	< >	- _	) °	ç 9	! 8
02	b B	q Q		= +	: /	; .	# £	ù %
03	j J	i I	h H	g G	f F	e E	d D	c C
04	r R	a A	p P	o O	n N	, ?	l L	k K
05	w W	x X	y Y	z Z	v V	u U	t T	s S
06	F3 F8	F2 F7	F1 F6	CODE	CAPS	GRAPH	CTRL	SHIFT
07	RET	SELECT	BS	STOP	TAB	ESC	F5 F10	F4 F9
08	→	↓	↑	←	SUP	INS	Home EFF	SPACE
09	4	3	2	1	0	/	+	*
10	.	,	-	9	8	7	6	5

Les deux dernières lignes sont celles du pavé numérique.

Clavier « AZERTY » du Canon V-20, Sanyo PHC-28 et Yeno MX-64 :

Ligne	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	7 è	6 §	5 (	4 '	3 "	2 é	1 &	0 à
01	m M	\$ *	^ "	< >	- _	) °	9 ç	8 !
02	b B	q Q	DEAD	= +	: /	; .	# £	ù %
03	j J	i I	h H	g G	f F	e E	d D	c C
04	r R	a A	p P	o O	n N	, ?	l L	k K
05	w W	x X	y Y	z Z	v V	u U	t T	s S
06	F3 F8	F2 F7	F1 F6	CODE	CAPS	GRAPH	CTRL	SHIFT
07	RET	SELECT	BS	STOP	TAB	ESC	F5 F10	F4 F9
08	→	↓	↑	←	SUP	INS	Home EFF	SPACE

Touche « DEAD » : Sanyo PHC-28.

Clavier « AZERTY » du Philips VG-8020, Sanyo PHC-28L, Schneider MC-810 et Yeno DPC-64 :

Ligne	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	è 7	§ 6	( 5	' 4	" 3	é 2	& 1	à 0
01	m M	\$ *	^ "	< >	- _	) °	ç 9	! 8
02	b B	q Q	DEAD	= +	: /	; .	# £	ù %
03	j J	i I	h H	g G	f F	e E	d D	c C
04	r R	a A	p P	o O	n N	, ?	l L	k K
05	w W	x X	y Y	z Z	v V	u U	t T	s S
06	F3 F8	F2 F7	F1 F6	CODE	CAPS	GRAPH	CTRL	SHIFT
07	RET	SELECT	BS	STOP	TAB	ESC	F5 F10	F4 F9
08	→	↓	↑	←	SUP	INS	Home EFF	SPACE

Touche « DEAD » : Sanyo PHC-28L.

Clavier « QWERTY » (Japonais) des Panasonic MSX2+ et MSX turbo R :

Ligne	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	7 '	6 &	5 %	4 \$	3 #	2 "	1 !	0
01	; +	[ {	@ `	¥	^ ~	- =	9 )	8 (
02	b B	a A	_	/ ?	. >	, <	] }	: *
03	j J	i I	h H	g G	f F	e E	d D	c C
04	r R	q Q	p P	o O	n N	m M	l L	k K
05	z Z	y Y	x X	w W	v V	u U	t T	s S
06	F3 F8	F2 F7	F1 F6	かな	CAPS	GRAPH	CTRL	SHIFT
07	RET	SELECT	BS	STOP	TAB	ESC	F5 F10	F4 F9
08	→	↓	↑	←	SUP	INS	Home CLS	SPACE
09	4	3	2	1	0	/	+	*
10	.	,	-	9	8	7	6	5
11					取消		実行	

Les deux dernières lignes sont celles du pavé numérique.

Clavier russe du Yamaha KYBT1 YIS-503II et KYBT2 YIS-503IIIR :

Ligne	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	& 6	% 5	⌘ 4	# 3	" 2	! 1	+ ;	) 9
01	V Ж	* :	H X	- ^ Ъ	= _	\$ 0	( 8	' 7
02	I И	F Ф	? /	< ,	@ Ю	B Б	> .	\ Э
03	o O	[ { Ш	R P	P П	A A	U y	W B	S C
04	K К	J Ё	Z 3	] } Щ	T T	X Ь	D Д	L Л
05	Q Я	N H	! ~ Ч	C Ц	M M	G Г	E E	Y Ы
06	F3 F8	F2 F7	F1 F6	РУС	CAPS	GRAPH	CTRL	SHIFT
07	RET	SELECT	BS	STOP	TAB	ESC	F5 F10	F4 F9
08	→	↓	↑	←	SUP	INS	Home CLS	SPACE

Clavier russe du Yamaha KYBT2 YIS-805 :

Ligne	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	& 6	% 5	␣ 4	# 3	" 2	! 1	+ ;	) 9
01	V Ж	* :	H X	- ^ Ъ	= _	\$ 0	( 8	' 7
02	I И	F Ф	? /	< ,	@ Ю	B Б	> .	\ Э
03	o O	[ { Ш	R P	P П	A A	U y	W B	S C
04	K К	J Ё	Z 3	] } Щ	T T	X Ъ	D Д	L Л
05	Q Я	N H	~ Ч	C Ц	M M	G Г	E E	Y Ы
06	F3 F8	F2 F7	F1 F6	РУС	CAPS	GRAPH	CTRL	SHIFT
07	RET	SELECT	BS	STOP	TAB	ESC	F5 F10	F4 F9
08	→	↓	↑	←	SUP	INS	Home CLS	SPACE
09	4	3	2	1	0	RET	+	*
10	.	,	-	9	8	7	6	5

Clavier russe du Sony HB-G9P :

Ligne	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	& 6	% 5	␣ 4	# 3	" 2	! 1	+ ;	) 9
01	V Ж	* :	H X	- ^ Ъ	= _	\$ 0	( 8	' 7
02	I И	F Ф	? /	< ,	@ Ю	B Б	> .	\ Э
03	o O	[ Ш	R P	P П	A A	U y	W B	S C
04	K К	J Ё	Z 3	] Щ	T T	X Ъ	D Д	L Л
05	Q Я	N H	Ч	C Ц	M M	G Г	E E	Y Ы
06	F3 F8	F2 F7	F1 F6	Code	CAPS	GRAPH	CTRL	SHIFT
07	RET	SELECT	BS	STOP	TAB	ESC	F5 F10	F4 F9
08	→	↓	↑	←	SUP	INS	Home CLS	SPACE
09	4	3	2	1	0	/	+	*
10	.	,	-	9	8	7	6	5

## ***F – Codes d'erreur du Basic et du disque Basic***

Code	Message	Description
1	NEXT without FOR	L'interpréteur a rencontré une instruction NEXT sans le FOR au préalable.
2	Syntax error	Une instruction a une mauvaise syntaxe.
3	RETURN without GOSUB	L'interpréteur a rencontré une instruction RETURN sans le GOSUB au préalable.
4	Out of DATA	READ a été exécuté alors qu'il n'y a plus de donnée à lire dans DATA.
5	Illegal function call	Une valeur dépasse la limite possible dans la fonction.
6	Overflow	La valeur d'une variable dépasse la limite possible.
7	Out of memory	La mémoire allouée au Basic est pleine.
8	Undefined line number	Une instruction indique une ligne inexistante.
9	Subscript out of range	Les paramètres d'une variable dépassent la taille du tableau.
10	Redimensioned array	Le tableau a déjà été créé.
11	Division by zero	On ne peut pas diviser par zéro.
12	Illegal direct	L'instruction entrée en mode direct ne peut être exécutée qu'en mode programme.
13	Type mismatch	Une valeur numérique a été affectée en tant que chaîne alpha-numérique ou vice-versa.
14	Out of string space	
15	String too long	La chaîne alpha-numérique dépasse les 256 caractères.
16	String formula too complex	La chaîne alpha-numérique est composée de trop de fonctions.
17	Can't CONTINUE	Le programme ne peut continuer son exécution.
18	Undefined user function	
19	Device I/O error	
20	Verify error	
21	No RESUME	La routine de traitement d'erreur doit se terminer par RESUME.
22	RESUME without error	RESUME a été exécutée alors qu'il n'y a pas eu d'erreur.
23	Unprintable error	Cette erreur n'a pas de message.
24	Missing operand	L'opérateur d'une expression n'a pas d'opérande.
25	Line buffer overflow	La ligne de programme entrée est trop longue. (256 caractères max.)
26~49	Indéfini	
50	FIELD overflow	Le nombre de caractères de FIELD dépassent les 256.
51	Internal error	Une erreur s'est produite dans le système.
52	Bad file number	Mauvais numéro de fichier.
53	File not found	Le fichier n'a pas été trouvé.
54	File already open	Le fichier que l'on veut ouvrir (ou effacer) est déjà ouvert.
55	Input past end	INPUT# a tenté de d'entrer une donnée hors du fichier.
56	Bad file name	Mauvais nom de fichier. Certains caractères de sont pas utilisables.
57	Direct statement in file	

58	Sequential I/O only	On essaie de lire un fichier séquentiel par accès direct.
59	File not OPEN	Le fichier n'a pas été ouvert.
60	Bad Allocation Table	La table allouée aux fichiers (FAT) est endommagée.
61	Bad file mode	Mauvaise utilisation de fichier. PUT ou GET a été utilisé sur un fichier séquentiel ou l'ouverture d'un fichier d'un format qui ne correspond pas a été tenté.
62	Bad drive name	Le nom de lecteur employé est différent de « A: », « B: », ... ou « H: ».
63	Bad sector number	
64	File still open	Un fichier est encore ouvert.
65	File already exists	Il y a déjà un fichier ayant le même nom au même endroit sur le disque.
66	Disk full	Il n'y a plus d'espace libre sur le disque.
67	Too many files	Le nombre de fichiers excède celui défini par MAXFILES.
68	Disk write protected	Une tentative d'écriture a été faite sur un disque protégé contre l'écriture.
69	Disk I/O error	Le système a trouvé une erreur lors de la lecture ou l'écriture sur le disque.
70	Disk offline	
71	Rename across disk	
72~255	Indéfini	

Voir un guide du MSX-BASIC pour plus de précision.

## G – Les ports d'entrée/sortie

Les ports d'entrée/sortie permettent de faire des accès directs au matériel qui compose le MSX. Sur MSX1, il était interdit de faire des accès directs pour garder la compatibilité. Puis la norme s'est assouplie à la sortie des MSX2 en tolérant les accès au PPI et au PSG. Ensuite la norme des MSX2+ conseillait même de faire des accès au VDP pour gagner en rapidité.

Dans les faits, les accès au PPI, au PSG et au VDP ne causent aucun problème sur toutes les générations de MSX, à condition de faire attention à certains timings qui sont un peu différents selon la génération du VDP par exemple. Cela est d'autant plus vrai lorsque le CPU a un « mode turbo ».

Le Z80 ne peut gérer que 256 ports d'entrée/sortie maximum. La norme MSX a réservé tous les ports de 0 à 63 (03Fh) pour les utilisateurs et ceux au dessus pour les constructeurs de matériel officiel.

Voici une liste non-exhaustive des matériels utilisant les ports d'entrée/sortie.

Ports	Matériel																		
00h~01h	Prise Midi (sortie) du Music Module.																		
00h~01h	<div>Cartouche « Sensor Kid ».</div> <div>Écriture :</div> <div><table><tr><td></td><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr><tr><td>Port 00h</td><td>OUT1</td><td>OUT0</td><td>-</td><td>CS</td><td>AI</td><td>RS</td><td>CLK</td><td></td></tr></table></div> <div>CLK = 1 pour spécifier la lecture du bit suivant de l'entrée analogique sélectionnée ;</div> <div>RS (Range Selection) = Signal vers la broche RS de la puce MB4052 ;</div> <div>AI = Numéro de l'entrée analogique à sélectionner. (0 = Capteur de luminosité, 1 = Capteur sonore, 2 = Thermomètre, 3 = Inutilisé) ;</div> <div>CS (Chip Select) = Signal vers la broche CS de la puce MB4052 ;</div> <div>OUT0 et OUT1 = Port 0 et port 1 de la prise Sortie à 0V ou 5V.</div> <div>Lecture :</div> <div>Port 00h = Le bit 0 donne l'état du bit actuel de l'entrée analogique sélectionnée.</div> <div>Port 01h = Le bit 0 et 1 indique l'état du port 0 et port 1 de la prise Sortie</div>		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Port 00h	OUT1	OUT0	-	CS	AI	RS	CLK	
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0											
Port 00h	OUT1	OUT0	-	CS	AI	RS	CLK												
02h~03h	Interface FAC MIDI. (Ces ports se reflètent jusqu'à 07h)																		
04h~05h	Prise Midi (entrée) du Music Module.																		
0Ah	DAC du Music Module.																		
0Fh	MegaRam Zemina. (Cartouches Black Box, Deluxe Box et Golden Box) Bit 4 et 5 = 0 et 1 pour mode RAM, 1 et 0 pour mode ROM (par défaut). Bit 6 et 7 = 0 et 1 pour Mapper 8k, 1 et 0 pour Mapper 16k.																		
10h~11h	PSG émulé des MegaFlashROM basées sur un FPGA.																		
14h~17h																			
18h~19h	Lecteur de code-barre Philips NMS-1170/20																		
20h~28h	Modem Philips NMS1251 (Ports paramétrables soit sur la plage 20h~28h, soit sur 30h~38h à l'aide de cavaliers)																		
20h~28h	Modem Miniware M4000 (Ports paramétrables soit sur la plage 20h~28h, soit sur 30h~38h à l'aide de cavaliers)																		

21h~27h	MP3 Player Sunrise
27h~2Fh	Serial interface Philips NMS 1210/1211/1212 (Ports paramétrables soit sur la plage 27h~2Fh, soit sur 37h~3Fh à l'aide de cavaliers)
28h~29h	Interface Ethernet DenYoNet
2Ah~2Bh	Registres de paramétrage de la cartouche PlaySoniq
30h~38h	Modem Philips NMS1251 (Ports paramétrables soit sur la plage 20h~28h, soit sur 30h~38h à l'aide de cavaliers)
30h~38h	Modem Miniware M4000 (Ports paramétrables soit sur la plage 20h~28h, soit sur 30h~38h à l'aide de cavaliers)
30h~38h	Interface SCSI GREEN/MAK
30h~38h	Interface CD-ROM Philips NMS 0210
37h~3Fh	MSX interface Modem Philips NMS 1250/1255
37h~3Fh	Serial interface Philips NMS 1210/1211/1212 (Ports paramétrables soit sur la plage 27h~2Fh, soit sur 37h~3Fh à l'aide de cavaliers)
3Ch	Accès au registre de contrôle du Musical Memory Mapper.
3Fh	Accès au registre de contrôle du SN76489 du Musical Memory Mapper.
40h~4Fh	Accès aux ports d'E/S étendus.
48h~49h	<i>Cartouche Franky. (SN76489 et VDP)</i>
50h~6Fh	<i>Ports réservés pour le système.</i>
5Eh~5Fh	Interface Ethernet GR8NET.
60h~6Fh	Graphics9000 / V9990.
64h~65h	Le port 064h sert d'identifiant et le port 065h sert à activer ou désactiver le mode turbo du Z80 des MSX2+ Panasonic. Si la lecture du port 064h donne la valeur 247 après y avoir écrit 8, c'est qu'il s'agit d'un MSX2+ Panasonic.
70h~73h	Midi Saurus. (Bit <sup>2</sup> )
74h~7Bh	<i>Ports réservés pour le système.</i>
7Ch~7Dh	MSX-Music (OPLL).
7Eh~7Fh	MoonSound (OPL4).
80h~83h	Interface RS-232C. (option) Ports 80h~83h : Émetteur-récepteur asynchrone universel (8251). Ports 84h~87h : Timer programmable (8253).
88h~8Bh	Accès aux registres de contrôle, de statut, de la palette de couleur et de transfert de données d'un V9938 externe. (ex : Extension MSX2 ou 80 colonnes pour MSX1.)
8Ch~8Dh	MSX Modem.
8Eh~8Fh	Accès aux registres de contrôle de la MegaRam / MegaRam Disk.
8Eh~8Fh	<i>Ports réservés pour le système.</i>
90h~91h	Accès aux registres de contrôle de l'imprimante.
94h	Accès au registre de contrôle de direction du port de l'imprimante. (Option non standardisée et peu répandus.)
92h~97h	<i>Ports réservés pour le système.</i>

98h~99h	Accès aux registres de contrôle et au(x) registre(s) de statut du VDP. (Voir le chapitre 7.4 « <a href="#">Comment accéder au VDP</a> »)																																																													
9Ah~9Bh	Accès aux registres de la palette de couleur et de transfert de données du VDP (v9938 / v9958 seulement) (Voir le chapitre 7.4 « <a href="#">Comment accéder au VDP</a> »)																																																													
9Ch~9Fh	<i>Ports réservés pour le système.</i>																																																													
A0h~A2h	Accès aux registres du PSG des MSX. (AY-3-8910 / YM2149)																																																													
A4h~A5h	<p>Accès aux registres du convertisseur A/D du MSX turbo R.</p> <p>En lecture :</p> <table> <tr> <th></th><th>bit7</th><th>bit6</th><th>bit5</th><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th></tr> <tr> <td>Port A4h</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="2">CT</td></tr> <tr> <td>Port A5h</td><td>COMP</td><td>0</td><td>0</td><td>SMPL</td><td>SEL</td><td>FILT</td><td>MUTE</td><td>BUFF</td></tr> </table> <p>CT = CT est incrémenté toutes les 63,5 µs.  Si ADDA est à 1, la donnée écrite au port A4h est envoyée à répétition à chaque incrémentation du compteur. L'écriture au port A4h initialise le compteur.  Si ADDA est à 0, la donnée écrite au port A4h est envoyée directement. Les écritures au port A4h n'initialise pas le compteur.</p> <p>BUFF = ADDA (voir plus bas)  COMP = 0 lorsque la sortie du comparateur D/A est supérieur à la donnée du port A4h ;  1 lorsque la sortie du comparateur D/A est inférieur à la donnée du port A4h.</p> <p>En écriture :</p> <table> <tr> <th></th><th>bit7</th><th>bit6</th><th>bit5</th><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th></tr> <tr> <td>Port A4h</td><td>DA7</td><td>DA6</td><td>DA5</td><td>DA4</td><td>DA3</td><td>DA2</td><td>DA1</td><td>DA0</td></tr> <tr> <td>Port A5h</td><td>0</td><td>0</td><td>0</td><td>SMPL</td><td>SEL</td><td>FILT</td><td>MUTE</td><td>ADDA</td></tr> </table> <p>DA0~DA7 = Donnée à envoyer au comparateur D/A.  ADDA = 0 pour cache unique* (conversion A/D) ;  1 pour double cache (conversion D/A).  MUTE = 1 pour couper le son* ; 0 pour activer le son.  FILT = 1 pour activer le filtre ; 0 pour signal normal*.  SEL = Sélection du signal à filtrer. 0 pour comparateur D/A* ; 1 pour l'ampli du micro.  SMPL = Signal d'échantillonnage. 0 pour numériser* ; 1 pour attendre.  (*) valeur par défaut.</p> <p>Exemples d'utilisation :</p> <pre> ; Routine PCM Play ; ; Entrée: HL = Adresse des données à lire ;         BC = Longueur des données ;         E  = Fréquence de l'échantillonnage ;         1  (15,75KHz), 2 (7,875KHz) ou 3 (5,25KHz)  PCMdac      equ    0A4h  ; Convertisseur D/A (sortie) PCMcnt      equ    0A4h  ; Compteur (entrée) PCMcntl     equ    0A5h  ; Contrôle du PCM (sortie)  PCMplay:     ld        a,00000011b     out       (PCMcntl),a ; Mode conversion D/A     di                          ; Pour garder de bons timings PCMplayLoop: </pre>									bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Port A4h	0	0	0	0	0	0	CT		Port A5h	COMP	0	0	SMPL	SEL	FILT	MUTE	BUFF		bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Port A4h	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	Port A5h	0	0	0	SMPL	SEL	FILT	MUTE	ADDA
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0																																																						
Port A4h	0	0	0	0	0	0	CT																																																							
Port A5h	COMP	0	0	SMPL	SEL	FILT	MUTE	BUFF																																																						
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0																																																						
Port A4h	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0																																																						
Port A5h	0	0	0	SMPL	SEL	FILT	MUTE	ADDA																																																						



```

in      a,(PCMcnt)
sub     e
jr      c,PCMplayLoop
ld      a,(hl)
out     (PCMdac),a
inc     hl
dec     bc
ld      a,c
or      b
jr      nz,PCMplayLoop    ; Saute si le compteur = 0
ei
ret

; Routine PCM Rec (assembleur M80)
;
; Entrée: HL = Adresse des données à lire
;         BC = Longueur des données
;         E = Fréquence de l'échantillonnage
;           1 (15,75KHz), 2 (7,875KHz) ou 3 (5,25KHz)

PCMdac      equ    0A4h  ; Convertisseur D/A
PCMcnt      equ    0A4h  ; Compteur
PCMcntl     equ    0A5h  ; Contrôle du PCM
PCMstat     equ    0A5h  ; Statut du PCM

adconv      macro next_bit,strip    ; Macro de conversion A/D 1 bit
    local adconv_not_change
    out      (PCMdac),a    ; Sortie de la donnée
    db       0EDh,70h      ; Instruction du Z80 non-documentée IN (C)
                        ; qui change les indicateurs du registre F
                        ; en fonction de ce qui est lu au port C.

    jp       m,adconv_not_change
    and      strip         ; Mets le bit à 0 car plus petit que le
                        ; signal d'entrée analogique
adconv_not_change:
    or       next_bit      ; Bit suivant à 1
    endm          ; Fin de création de la macro

PCMrec:
    ld       a,00001100b
    out      (PCMcntl),a    ; Mode conversion D/A
    ld       d,0            ; Pour mettre le compteur à 0
    di              ; Pour garder de bons timings
PCMrecLoop:
    in       a,(PCMcnt)    ; Lecture du compteur
    cp       d
    jr      nz,PCMrecLoop    ; Saut si la valeur est adéquate
    add      a,e            ; Création de la valeur du
    and      11b           ; compteur suivant
    ld       d,a
; Plus rapide de 7µs que ci-dessus
    exx
    ld       a,00011100b
    out      (PCMcntl),a    ; Maintient la donnée
    ld       c,PCMstat
    ld       a,80h          ; Bit COMP à 1
    adconv   01000000b,01111111b    ; récupération de chaque bit
    adconv   00100000b,10111111b
    adconv   00010000b,11011111b
    adconv   00001000b,11101111b
    adconv   00000100b,11110111b
    adconv   00000010b,11111011b
    adconv   00000001b,11111101b
    adconv   00000000b,11111110b

```

	<pre>exx ld    (hl),a           ; Stocke la donnée (l'octet) ld    a,00001100b out   (PCMCnt1),a ; Supprime la donnée maintenue inc   hl               ; Prochaine donnée dec   bc ld    a,c or    b jr    nz,PCMrecLoop    ; Saute si le compteur = 0 ld    a,00000011b out   (PCMCnt1),a ; Mode conversion D/A et active le son ei ret</pre>
A6h	Ports réservés pour le système.
A7h	Accès au registre de contrôle des LED Pause et R800 du MSX turbo R. bit 0 = 1 pour allumer la LED Pause ; (écriture) bit 7 = 1 pour allumer la LED Mode R800. (écriture) Le contenu de ce registre est sauvegardé à l'adresse 0FCB1h par le système.
A8h~ABh	Accès au registre de l'interface de port programmable (PPI : 8255 / ULA9RA041).  Port A8h = Accès au registre de sélection des Slot primaires (via le port A du PPI). <div><div>bit7   bit6   bit5   bit4   bit3   bit2   bit1   bit0</div><div>Port A8h   Slot primaire 3   Slot primaire 2   Slot primaire 1   Slot primaire 0</div></div> Port A9h = Lecture d'une ligne de la matrice du clavier (via le port B du PPI). Port AAh = Accès au registre de contrôle du clavier et du lecteur à cassette (via le port C du PPI). bits 0~3 = Numéro de la matrice du clavier à lire via le port B ; bit 4 = 0 pour faire tourner le moteur du lecteur cassette ; bit 5 = 1 commander l'écriture sur cassette ; bit 6 = 0 pour allumer la LED CAPS du clavier ; bit 7 = 1, puis 0 peu après pour émettre un cliquetis (pour les touches). Port ABh = Accès au registre de contrôle des ports du PPI. La fonction des bits 0~6 dépendent de l'état du bit 7. Lorsque celui-ci est à 0, ils ont la fonction suivante : bit 0 = Etat du bit à changer ; bits 1~3 = Numéro du bit à changer au port C du PPI. bits 4~6 = Inutilisés. Lorsque le bit 7 est à 1, ils ont en théorie la fonction suivante mais en fait, ils ne doivent/peuvent pas être modifiés : bit 0 = Direction des bits de point faible du port C. 0 pour sortie (MSX) ; bit 1 = Direction du port B. 0 pour sortie, 1 pour entrée (MSX) ; bit 2 = Mode des ports B et des bits de point faible du port C. 0 pour mode normal (MSX), 1 pour mode bidirectionnel ; bit 3 = Direction des bits de point fort du port C. 0 pour sortie (MSX) bit 4 = Direction du port A. 0 pour sortie (MSX) ; bits 5~6 = Mode des ports A et des bits de point fort du port C. 00 pour mode normal (MSX), 01 pour mode à impulsion et 10 pour mode bidirectionnel ;
ACh~AFh	1chipMSX I/O ports.
B0h~B3h	Accès aux registres de la cartouche Sony HBI-55 et Yamaha UDC-01. Ces cartouches contiennent 4Ko de SRAM. (External PPI 8255)
B4h~B5h	Accès aux registres de l'horloge en temps réel. (RP-5C01) (Voir les routines REDCLK et WRTCLK de la Sub-ROM.)

	Port B4h = Numéro de registre sur 4 bits (écriture seule) ; Port B5h = Donnée sur 4 bits à lire ou à écrire.
B6h~B7h	<i>Ports réservés pour le système.</i>
B8h~BBh	Interface pour crayon optique Sanyo.
BCh~BFh	Contrôleur VHD.
C0h~C1h	MSX Audio / Music Module.
C0h~C3h	Mounsound / OPL4 (ports alternatifs).
C2h~C3h	MSX Audio / Music Module (ports pour deuxième cartouche)
C4h~C7h	Mounsound / OPL4.
C8h~CCh	Accès aux registres de la « MSX Interface ». (Interface Série Asynchrone)
D0h~D7h	Contrôleur de lecteur de disquette. (WD2793 Microsol)
D8h~D9h	Kanji Rom niveau 1.
DAh~DBh	Kanji Rom niveau 2.
DCh	Kanji Rom étendue. (MSX-View)
DCh~DDh	<i>Support des manettes pour les jeux Sega avec la cartouche PlaySoniq.</i>
E4h~E5h	Accès aux registres de réglage du S1990. (MSX turbo R)
E6h~E7h	<p>Accès au registre du Timer du S1990. Ce Timer est incrémenté toutes les 3,911 µs.  E6h = 8 bits de point faible de la valeur du Timer. (lecture)  Une écriture à ce port remet le Timer à zéro.  E7h = 8 bits de point fort de la valeur du Timer. (lecture)</p> <p>Exemple d'utilisation :</p> <pre> ; Entrée: B = temps d'attente (B x 3,911 µs). ; Sortie: Les registres A, C et F seront modifiés ; Notes: B doit être supérieur à 0. ;       Les interruptions doivent être désactivées.  CountLSB    equ    0E6h CountMSB    equ    0E7h  Wait:     in      a, (CountLSB)     ld      c, a WaitLoop:     in      a, (CountMSB)     sub     c     cp      b     jr      c, WaitLoop ; Saute si le temps n'est pas écoulé     ret </pre>
F3h	<p>Accès au registre servant à connaître le mode d'affichage actuel. (MSX2+ seulement)</p> <p>bit 0 = M3 ;  bit 1 = M4 ;  bit 2 = M5 ;  bit 3 = M2 ;  bit 4 = M1 ;  bit 5 = TP ;  bit 6 = YUV ;  bit 7 = YAE.</p>

F4h	Accès au registre « Reset status » qui sert d'indicateur pour que le MSX2+ ou plus récent s'initialise de façon adaptée. bit 5 = 1 lorsque l'initialisation est matériel (provoquée par le bouton Reset) ; bit 7 = 1 lorsque le CPU actif est le R800.
F5h	Accès au registre de contrôle des ports du système. (MSX2 seulement) Ce registre n'est accessible qu'en écriture seule. Il sert à désactiver (bit à 0) certains matériels intégrés au MSX. Notez que le Bios désactive certains matériels pendant l'initialisation du MSX. bit 0 = Kanji Rom niveau 1 ; bit 1 = Kanji Rom niveau 2 (MSX2+ seulement) ; bit 2 = MSX-Audio. Si la valeur lu au port 0C0H est 255, c'est qu'il n'y en a pas en interne. ; bit 3 = Super impose. Effectuez un ET logique de la valeur des bits 3 lue aux ports 77h et F7h, savoir si le MSX possède la super impose en interne. ; bit 4 = MSX Interface. Si la valeur lu au port 0C0H est 255, c'est qu'il n'y en a pas en interne. MSX Interface est un LSI de communication à usage général qui n'est pas encore utilisé actuellement. ; bit 5 = RS-232C aux ports d'E/S mappé ; Si le registre de statut du i8351 vaut xx000101b après une initialisation interne, c'est qu'il y a une RS-232C en interne. ; bit 6 = Crayon optique. Si la valeur lu au port 0BBH est 255, c'est qu'il n'y en a pas en interne. ; bit 7 = RTC. Ce bit est mis à 0 pendant l'initialisation et reste ainsi lorsqu'une cartouche contient une RTC. Vous pouvez ignorer ce bit dans vos programmes.
F6h	Bus du port entrée/sortie des couleurs.
F7h	Accès au registre « Contrôle A/V ». Ce registre n'est accessible qu'en écriture seule sauf le bit 4 qui n'est accessible qu'en lecture. bit 0 = Mettre à 1 pour mixer des sorties audio droite et gauche ; bit 1 = Sortie audio gauche ; bit 2 = Mettre à 1 pour régler l'entrée vidéo sur la prise RGB21 ; bit 3 = Indicateur pour savoir si l'entrée vidéo est désactiver ou non ; bit 4 = Contrôle AV, 1 pour TV ; bit 5 = Contrôle Ym, 1 pour TV ; bit 6 = Inverse l'état du bit 4 du registre 9 du VDP ; bit 7 = Inverse l'état du bit 5 du registre 9 du VDP.
F8h	Contrôle A/V des MSX PAL. (Optionnel)
F8h~FBh	<i>Ports réservés pour le système.</i>
F8h~FBh	Accès aux 8 bits de point fort des registres 16 bits du Memory Mapper de la cartouche PlaySoniq. (Désactivé par défaut)
FCh~FFh	Accès aux registres du <a href="#">Memory Mapper</a> .

# Lexique

## **- *Bruit blanc***

Le bruit blanc du PSG est un son produit par des fréquences sonores pseudo-aléatoires.

## **- *Main-RAM***

Main-RAM désigne la RAM dans le Slot que le système a sélectionné au démarrage.

## **- *Mapper***

Mapper désigne un dispositif qui permet d'étendre la mémoire grâce à un système de changement de page. « Memory Mapper » désigne le système de changement de page de la RAM au standard MSX2. « Rom Mapper » désigne le système de changement de page utilisé dans les Megarom par exemple. Il en existe plusieurs et aucun sont standardisés.

## **- *Megarom***

Megarom désigne les cartouches ayant une Rom d'au moins 1 méga-bits (128Ko~). Celles-ci utilisent un Mapper destiné étendre la mémoire à plus de 64Ko. Il peut exister des cartouches de 64Ko, voir moins, munies d'un Mapper.

## **- *MSX-Engine***

« MSX-Engine » est le nom donné à une puce, fabriqué par Toshiba, qui intègre différents composants couramment utilisés dans les MSX. Les plus récentes intègrent même le PSG et le CPU. ASCII a fabriqué le même genre de puce. Bien que le nom de ces dernières soit « MSX-System », en Europe, on dit souvent « MSX-Engine ».

## **- *PPI (Programmable Port Interface)***

Sur MSX, le PPI est une puce utilisée pour commuter les Slot, pour gérer le clavier (cliquetis inclus) et le lecteur cassette.

## **- *PSG (Programmable Sound Generator)***

PSG signifie : Générateur de Son Programmable. Programmable Sound Generator est une appellation de General Instrument. Yamaha nomme ce genre de puce SSG (Software-controlled Sound Generator) et Texas Instrument DCSG (Digital Complex Sound Generator). Bref, tout ceci désigne une puce dédiée au effet sonore et à la musique. Le PSG des MSX gère aussi les manettes de jeu, la souris, les molettes, la tablette graphique, etc.

## **- *RAM (Random Access Memory)***

La RAM est la mémoire vive de l'ordinateur. Elle sert à y mettre les programmes et les données le temps que l'ordinateur est sous tension.

## **- *ROM (Read Only Memory)***

Une ROM est une « mémoire morte ». Elle sert à y mettre des programmes ou des données que l'ordinateur pourra exécuter ou lire même après une extinction ou un redémarrage.

## **- *Sprite***

Sprite est un terme anglais de l'époque que l'on traduit par lutin. Aujourd'hui, on appellerait ça plutôt un calque. Dans le cas du MSX, c'est une sorte petit calque d'une taille de 8 x 8 ou 16 x 16 pixels.

### **- *Slot***

Slot, que l'on traduit par fente, est un port cartouche physique ou non. Les ports cartouches d'un MSX sont des Slot primaires. Ce n'est pas toujours le cas pour les bus d'extension.

### **- *VRAM***

La VRAM est la mémoire vive du processeur vidéo. Celle-ci contient toutes les données graphiques.

### **- *VDP (Video Display processor)***

VDP veut dire : processeur de l'affichage vidéo. C'est un processeur dédiée à l'affichage à l'écran.