

TUTORIEL D'INITIATION A LA PROGRAMMATION EN LANGAGE ASSEMBLEUR POUR MSX

PARTIE 3 – LE MONDE DES GRAPHIQUES 1

Dans cette partie du tutoriel nous allons créer un [Hola Mundo Grafico](#) en utilisant un jeu de caractères de texte créé par nous et le tout réalisé en mode graphique SCREEN 2, ainsi que les outils nécessaires pour réaliser cette tâche.

Avant de commencer avec le code de [Hola Mundo Grafico](#), nous allons commencer avec la partie qui plait le moins, la théorie...mais bien sûr sans elle on ne pourrait pas comprendre les choses.

A la différence du 1er exemple dans le lequel nous avons seulement sélectionné la ligne, la colonne et le texte que nous voulions imprimer, et l'afficher à l'écran.

On se souvient que le mode d'écran sélectionné était [Screen 0](#) avec le code suivant.

```
call INITXT ; BIOS set SCREEN 0
```

Ceci est le mode d'écran qu'utilise le BASIC MSX, le set de caractères ou lettres qui sera imprimé à l'écran est intégré au BIOS, et c'est celui que nous voyons à l'écran quand on démarre l'ordinateur.

Maintenant que nous allons utiliser le [Screen 2](#) qui est un mode graphique, et que nous pourrions créer des caractères comme on le désire et leurs donner une couleur spécifique, il faut que notre ROM prenne un aspect plus professionnel. (Il faut préciser que tout ce qui est expliqué ici ne sert pas seulement à créer des caractères, cela sert aussi pour créer des graphiques que nous utiliserons dans nos ROM, le procédé est le même.)

MODES EN MSX1

SCREEN 0: texte de 40 x 24 sur 2 couleurs

SCREEN 1: texte de 32 x 24 sur 16 couleurs

SCREEN 2: graphiques de 256 x 192 pixels sur 16 couleurs

SCREEN 3: graphiques de 64 x 48 pixels sur 16 couleurs



Voici différents sets de caractères de divers jeux.

L'écran en [Screen 2](#) est composé de [256 Pixels en horizontal](#) sur [192 pixels en vertical](#) mais si l'on regarde en caractères de [8x8 pixels](#) on parle de [32 caractères en horizontal](#) pour [24 en vertical](#).

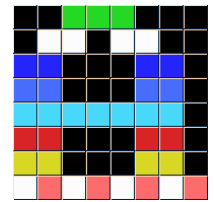
Si on multiplie [32x24x8](#) octets qui composent chaque caractère, cela nous donne un total de [6144 Octets - 1800h](#) soit [6Kb](#). Ces données graphiques sont sauvegardées dans la Video Ram ou VRAM de la position [0000h](#) à la position [17FFh](#) dans le monde du MSX cette zone s'appelle [Character Pattern Table](#).

Pour mieux comprendre voici une image qui vaut mieux que mille mots. Ceci est un caractère de [8x8 Pixels](#) qui fait 8 octets (et un octet se compose de [8 bits](#)) de large sur 8 octets de haut. C'est pourquoi on multiplie les 32 caractères de large par les 24 caractères de haut et par les 8 octets que font chaque caractères. Au total 6144 Octets 1800h en Hexa. (On peut voir le caractère en mode graphique et en mode binaire [00111000 – Byte 0](#))

Caractère A Graphique

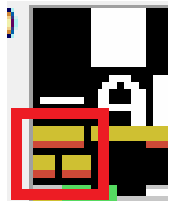


La couleur des caractères suit le standard du MSX1, on doit avoir un maximum de **2 couleurs par octet**, une pour le fond et une pour le caractère pour un total de **16 couleurs disponibles**. Voici la même image du caractère mais en couleur cette fois, dans le premier octet, le fond est noir et le caractère est vert. Le second octet, le fond est noir et le caractère blanc, mais si on regarde le dernier octet, le fond est blanc et le caractère rose.



Ces données graphiques se sauvegardent dans la VRAM de la position **2000h** à la position **37FFh**, dans le monde MSX cette zone s'appelle **Colour Table** et occupe le même espace que la **Character Pattern Table** donc $32 \times 24 \times 8 = 6144$ Octets - 1800h en Hexadécimal (Il vaut mieux travailler avec cette nomenclature)

Nous allons clarifier ce thème qui est très important, dans le MSX tout le thème graphique fonctionne avec des **caractères**, abréviation **CHR** de l'anglais **ChaRacter**. Si chaque page graphique occupe 6Kb sans couleur et que nous avons une ROM de 32Kb, cela nous donne 5 pages sans espace pour la couleur, les sprites, la musique et la programmation. Mais toutes les ROM ou les jeux utilisent un jeu de CHR ou Tuiles et c'est avec eux que l'on construit une mosaïque à base de répétition de Tuiles pour différentes zones des écrans que nous allons créer. (Passons par des images pour comprendre)



Comme on peut le voir sur l'image de **King's Valley de Konami** nous avons un caractère de brique jaune qui se répète dans différentes zones de l'écran, il en est de même pour les escaliers, cette technique d'utilisation de la répétition des caractères pour créer une page s'appelle le mappage. La page de droite c'est la **Character Pattern Table** additionnée de la **Colour Table** qui sont les graphiques que les dessinateurs de Konami ont créés, ainsi le mappage ou tuilage sur MSX se traduit par la **Name Table**. L'image de la gauche.

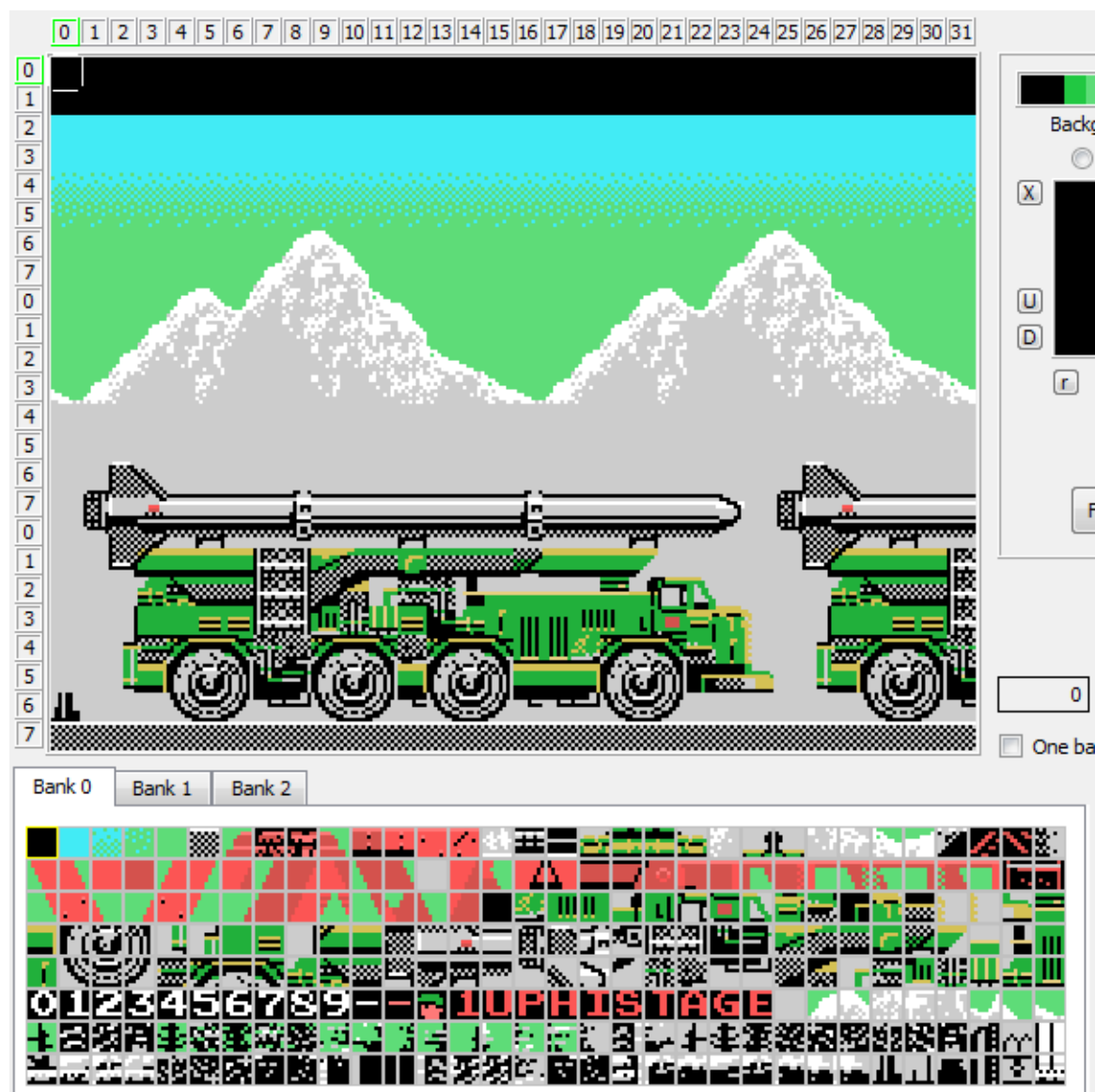
Ces données se sauvegardent dans la VRAM de la position 1800h à la position 1AFFh, cette zone occupe 32×24 caractères = 768 octets – 300h Hexa. Avec cette technique nous pouvons créer beaucoup de tableaux si on comprime les données associées à un meilleur mappage.

Maintenant la partie la plus difficile à expliquer. Si on s'arrête de nouveau à l'image de droite, on peut voir que les CHR ou Tuiles se répètent 3 fois dans la **Character pattern Table** et dans la **Colour Table**, c'est dû à la limitation du MSX, nous savons que l'écran se compose de 32 CHR de large sur 24 CHR de haut pour un total de 768 caractères, **mais ce que nous savons c'est qu'à son tour il est divisé en 3 parties** ou 3 blocs de 256 CHR. Observons dans l'image de droite, la marge à gauche **p0, p1, p2** ce sont les 3 blocs de caractères de caractères, chaque bloc est composé de 32 CHR de large x 8 CHR de haut = 256 CHR.

Question : Alors on ne peut pas utiliser plus de 256 CHR parce qu'on doit les copier dans les 3 blocs ?

Réponse : NON. Si un caractère doit être utilisé sur tout l'écran, il doit être dans les 3 blocs mais si on utilise ce caractère uniquement dans la zone du premier bloc alors on peut le mettre uniquement dans ce bloc. Il faut placer les CHR dans les blocs en fonction de la zone de l'écran dans laquelle on veut les utiliser. De plus, à mesure que nous changeons de niveaux nous pouvons intégrer de nouveaux blocs de CHR ou les remplacer dans la VRAM quand on le nécessite. Avec les images il n'y aura rien de plus à comprendre qu'avec le texte.

J'espère que tout est bien compris sinon revoyez les principes autant de fois qu'il le faut car il est important de comprendre cette partie. Comme on peut le voir dans les écrans du mappage les données se répètent beaucoup et peuvent être compressées pour occuper peu d'espace.



On observe ici un écran complet de 3 blocs de CHR utilisés, avec de petits caractères on peut réaliser une page graphique très grande . Images de nMSXtiles.

RESUME

Character Pattern Table: Occupe 6Kb dans la VRAM de **0000h** à **17FFh** surnommée **CHRTBL**
C'est ici que sont situés les CHR que nous utiliserons pour créer les écrans.

Colour Table: Occupe 6Kb dans la VRAM de **2000h** à **37FFh** surnommée **CLRTBL**
C'est ici que nous appellerons les couleurs qu'auront les CHR qui sont dans la CHRTBL.

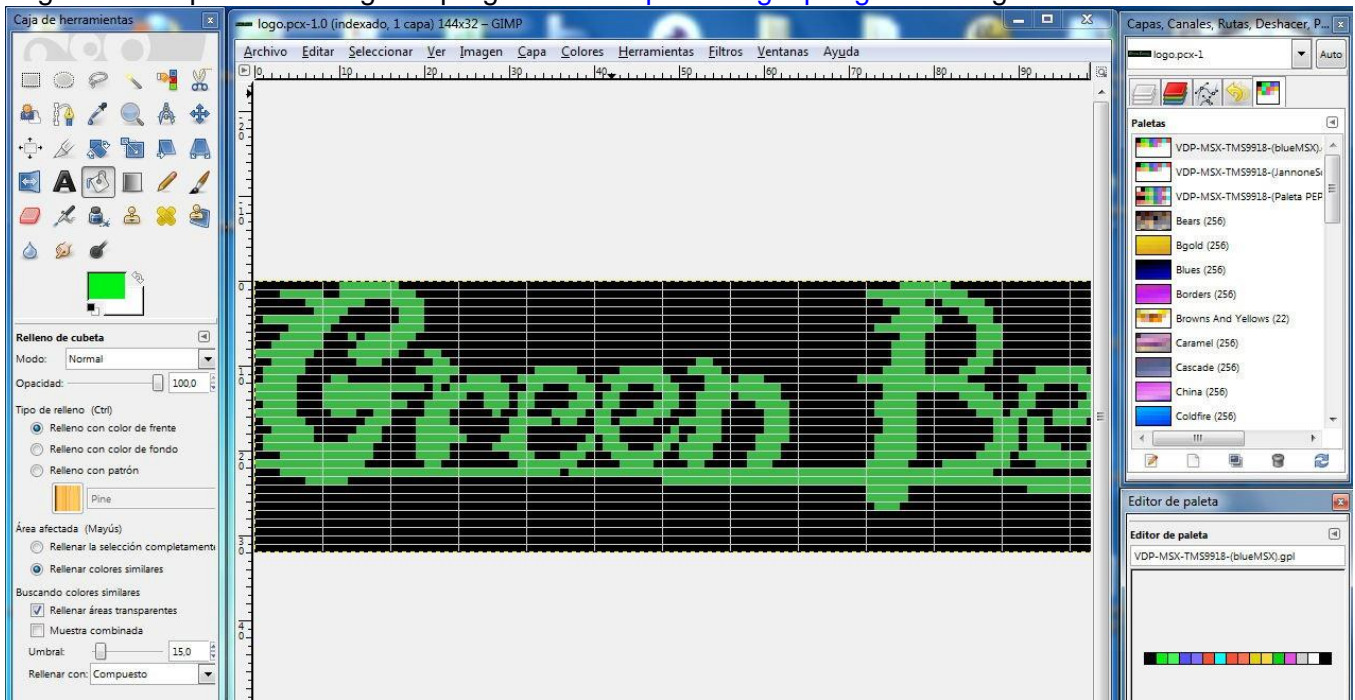
Name Table: Occupe 768 octets dans la VRAM de **1800h** à **1CFFh** surnommée **NAMTBL**
C'est ici qu'est situé le mappage avec les numéros de CHR qui génèrent les écrans.

Toutes ces données, nous les utiliserons dans le code [Assembleur](#) de [Hola Mundo Grafico](#).

Laissons de côté la **Sprite Pattern Table - SPRTBL** et la **Sprite Attribute Table - SPRATR** que nous verrons plus tard dans un tutoriel dédié au monde des sprites.

La première chose qu'il nous faut pour débiter, c'est un programme de dessin qui nous permettra de réaliser les graphiques que nous importerons dans notre code. Il y a de multiples possibilités mais personnellement j'utilise GIMP qui est un programme gratuit sous Windows et Linux et très complet.

Page officielle pour télécharger le programme : <http://www.gimp.org> Téléchargez le et installez le.



Pour configurer GIMP de manière correcte pour créer des graphiques pour MSX, nous devons configurer la résolution, la palette des couleurs et une grille appropriée et enregistrer le fichier d'image au format PCX pour l'utiliser ensuite avec les autres outils. Le fantastique manuel de aOrante nous permet de la faire.

2^a Partie:

aOrante-----

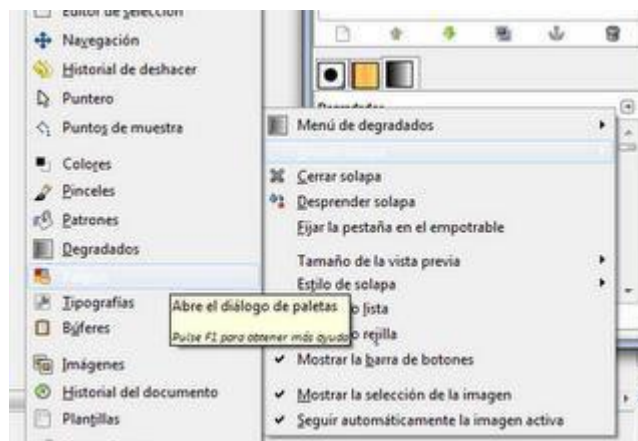
Une fois que le programme est ouvert, la première chose à faire est de créer une nouvelle image et pour cela nous irons dans le menu "Fichier" de la fenêtre principale et choisirons "Nouvelle image". On peut y accéder plus rapidement en appuyant sur [CTRL]+[N]. Une fenêtre s'ouvre où on pourra définir la taille de l'image que nous positionnerons à 256 de large sur 192 de haut.



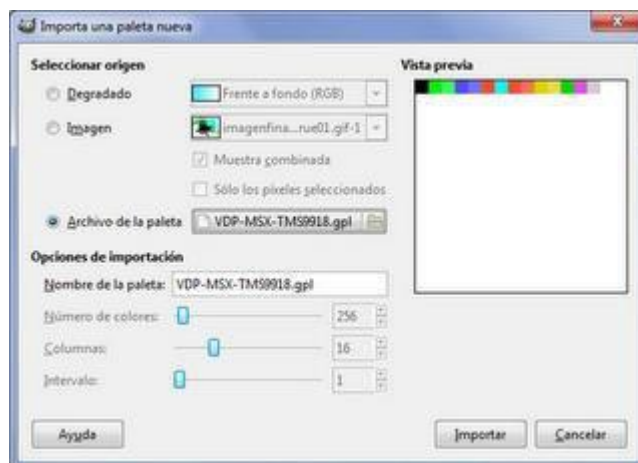
NOTE: Si on doit utiliser GIMP pour dessiner dans ce format on peut créer un gabarit dans "Fichier>Enregistrer comme Modèle". Quand on commencera une image, nous aurons seulement besoin d'indiquer notre gabarit depuis le sélecteur qui se trouve dans la partie supérieure de la fenêtre pour créer une nouvelle image.

La seconde étape sera d'indiquer la palette de couleur à utiliser. Dans notre cas, il faudra indexer une palette de 16 couleurs conforme à celle du VDP des MSX. Pour cela nous devons disposer de la palette. Nous pouvons la créer depuis "Fenêtre>Dialogues ancrable>Palette", mais pour simplifier la tâche il en existe 2 qui peuvent être téléchargées depuis cet article. Une palette qu'utilisent les émulateur blueMSX et OpenMSX et l'autre de MSX Screen Conversor de Jannone, utile pour ceux qui veulent utiliser l'application (les palettes sont en bas à la fin).

Pour la charger, il est recommandé de suivre le chemin suivant pour changer la configuration de la fenêtre de dialogue ancrable (qui se positionne à droite). Premièrement, on ajoute un nouvel onglet avec le gestionnaire de palette. Il y a 2 formats, l'ouvrir depuis le menu "Fenêtre>Dialogues ancrables>Palette" et la faire glisser directement dans la zone inférieure de la fenêtre. La seconde option serait de cliquer sur le petit bouton avec la flèche pointant vers la gauche, situé dans la partie supérieure droite où se trouvent les onglets. Apparaît un menu et on sélectionne l'option d'ajouter un onglet, dans lequel on choisit "Palette".



Une fois insérée, un clique à nouveau sur le petit bouton pour accéder à la première option "Menu de palette" et on sélectionne l'option "Importer palette..". Ensuite nous utiliserons l'option "Archive de palette" pour charger la palette.

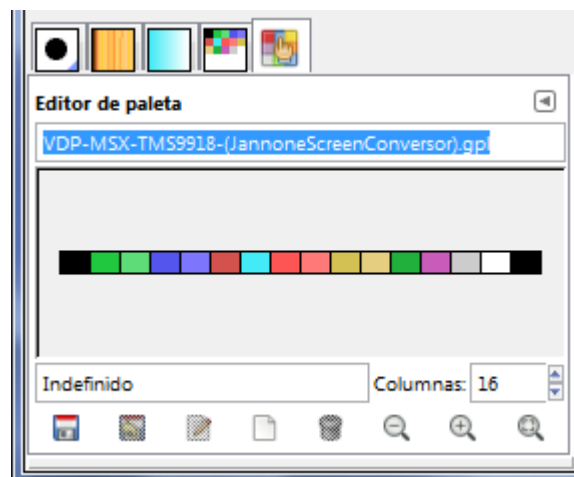


Maintenant que nous avons la palette, nous pouvons indiquer le type d'image. Sélectionnons "Image>Mode>Indexé". Une fenêtre s'affiche, dans laquelle on sélectionne l'option "Utiliser une palette personnelle" et on appuie sur le bouton qui sur la ligne suivante pour sélectionner la palette.

Important : Désactiver le control qui indique "Supprimer les couleurs non utilisées de la palette finale", son il serait impossible de peindre à l'écran.



L'étape suivante est d'accéder à notre palette de couleur et pour cela, nous devons ajouter un onglet avec l'éditeur de palette. Cela se fait en passant le curseur au-dessus de notre palette et avec le bouton droit ouvrir le menu contextuel sur lequel on appuie sur l'option "Editer Palette". Une fenêtre s'ouvre, avec nos couleurs, que l'on va ajouter à la liste des onglets juste à côté du sélecteur de palettes.



NOTE: Les modifications de l'interface utilisateur de GIMP, restent d'une session à l'autre, de ce fait, on a pas besoin de répéter ces opérations de configuration, à l'exception de la palette car c'est la palette par "défaut" qui est chargée.

Pour nous aider à respecter les caractéristiques du mode d'écran du SCREEN 2 (2 couleurs chaque 8 pixels) nous allons utiliser la grille. Avant de l'activer, nous allons la paramétrer via "Image>Configurer la grille". Dans la fenêtre, on va changer les valeurs "d'espacement", d'abord cliquer sur l'icône du cadenas pour désactiver la proportion, et ensuite, indiquer dans "Largeur" 8 et dans "Hauteur" 1. On peut maintenant l'activer. Nous allons dans le menu "Affichage" et activons l'option "Afficher grille".



Pour voir correctement la grille, nous devons visualiser les images avec un minimum de 400% de zoom. Pour cela on ajuste dans "Affichage>Zoom" ou dans le marqueur inférieur de la fenêtre principale. Une autre façon de travailler serait d'utiliser une grille de 8x8.



Maintenant, on peut dessiner. Une fois terminé, sauvegarder le fichier et traitez les données comme on va l'expliquer maintenant.

Palette pour GIMP (basée sur celle de blueMSX modifiée pa PEPE)

Fichier [VDP-MSX-TMS9918-\(Paleta PEPE\).gpl](#)



Package

aOrante-----

Dans le manuel daOrante il est fait mention de créer une image de 256x192 mais on peut créer la dimension que l'on souhaite, cela dépend du ou des graphiques que l'on veut réaliser tout en ne dépassant pas 256x192, comme on peut le voir dans la 1ere capture d'écran pour le logo Green Beret, la résolution spécifiée est 144x32 parce qu'on a besoin que de cette dimension à intégrer au code.

Nous allons passer à la création du set de caractères que nous allons utiliser pour afficher le [Hola mundo](#) graphique.

Suivant la table ASCII on peut créer un jeu de caractères en suivant cette norme.

Nous allons commencer au caractère 32 encadré en rouge dans le tableau suivant et qui représente un 'espace' et se termine par le caractère 127. Comme on le verra dans le set de caractères que j'ai dessiné à la main avec GIMP, l'ordre de chaque caractère est créé suivant la norme de la table ASCII.

32–Espace, 33–Exclamation, 34–guillemets, 35–Hash, 36–Dollar, 37–Pourcentage, Etc. Etc. jusqu'au 127.

Comme on peut le voir ici, chaque lettre correspond au numéro de caractère. De telle manière que lorsqu'on écrira une lettre 'A' majuscule, encadré en vert dans le tableau, l'ordinateur appellera un 65.

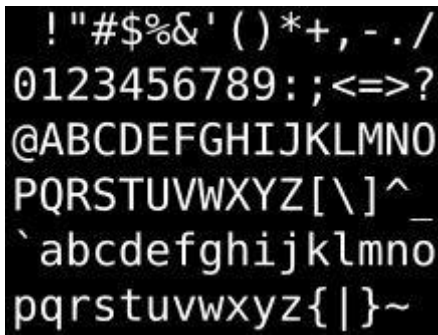
La raison de suivre cette méthode est que dans notre code assembleur, on va mettre les textes que l'on utilise dans le tutoriel en ASCII.

Au moment de positionner les CHR de notre jeu de caractères dans la VRAM, nous appellerons le premier CHR à l'adresse 32 de la CHRTBL, pour que cela coïncide avec la numérotation de la table ASCII, de cette manière appeler dans la table NAMTBL un texte en ASCII coïncidera avec les mêmes numéros de caractères que ceux de la table **Character Pattern Table - CHRTBL**.

Comme notre premier graphique est un 'espace' dans le jeu de caractères que j'ai créé et son ASCII est le 32, ce sera le numéro à partir duquel on doit commencer à appeler les CHR dans la table CHRTBL en VRAM.

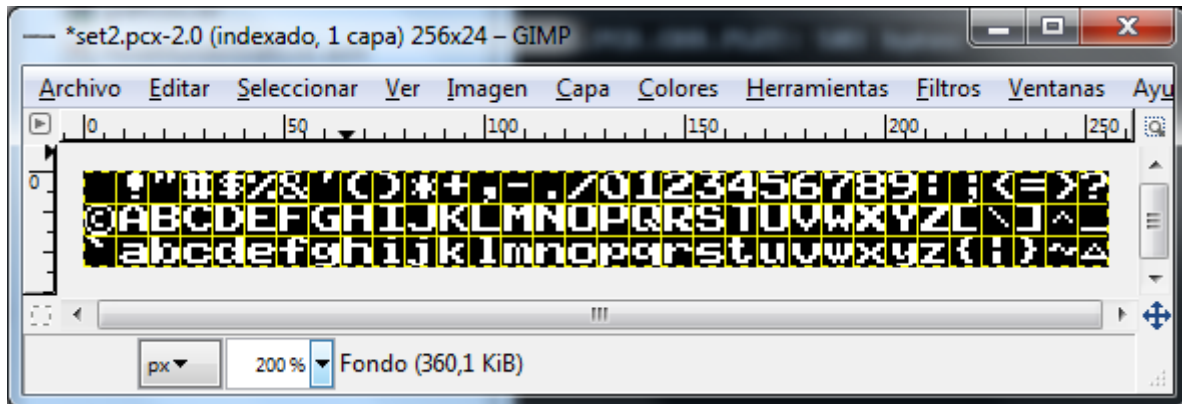
	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	` 96	p 112
0001 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0010 2	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0011 3	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0100 4	EOT 4	DC4 20	\$ 36	4 52	D 68	T 84	d 100	t 116
0101 5	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0110 6	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0111 7	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1000 8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1001 9	HT 9	EM 25) 41	9 57	I 73	Y 89	i 105	y 121
1010 A	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1011 B	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1100 C	FF 12	FS 28	` 44	< 60	L 76	\ 92	l 108	 124
1101 D	CR 13	GS 29	- 45	= 61	M 77] 93	m 109	} 125
1110 E	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
1111 F	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	DEL 127

Table de code ASCII 7 bits



Voici les 95 caractères imprimables ASCII et son ordre commence avec le caractère 20h ou 32 qui est l'espace (SP) suivis de symboles puis les chiffres et l'alphabet en majuscule et minuscule et d'autres symboles

Voici le dossier [Tutorial3.rar](#) con avec les exemples pour ce tutoriel, incluant les codes graphiques et des programmes.



On peut voir ici comment on dessine un jeu de caractères avec GIMP, vous pouvez créer le votre ou ouvrir le fichier [SET2.PCX](#) qui a été réalisé pour l'exemple. (Il est préférable de faire le votre pour vous familiariser avec GIMP). J'ai créé une image de 256 de large, qui est la largeur de l'écran en SCREEN 2, pour 24 de hauteur qui sont les 3 lignes de caractères qu'il nous faut, avec une grille de 8x8 pixels. (Le résultat est l'image ci-dessus, notez qu'il n'y a pas de couleur). Ce que l'on pourra faire plus tard via le code). Observez comment j'ai créé le jeu de caractères, si vous voulez réaliser le votre n'utilisez pas la totalité des 8 pixels, laissez une ligne vierge à droite et une en bas pour qu'elles ne se melange pas ou ne se touchent pas à l'affichage.

Nous allons voir une autre partie très importante. Nous avons les graphiques dans un fichier d'image, comment les incorporer à notre code pour le compiler avec l'assembleur [asMSX](#). Toutes ces actions nous allons les réaliser avec [MS-DOS](#) ou [Invite de commande](#). Copiez le fichier SET2.PCX dans le dossier ou vous allez créer le code, dans mon cas [C:\msx\asmsx012e\dist012e\Tools\](#)

Dans le [pack-MSX.rar](#) de la 1ere partie du tutoriel nous avons décompressé [asMSX0.12](#) dans [C:\MSX](#), depuis Windows ouvrez [Demarrer-Tous les programmes-Accessoires-Invite de commande](#)

Maintenant, tapez , [cd msx](#) [Entrée] [cd asmsx012e](#) [Entrée] [cd dist012e](#) [Entrée] [cd Tools](#) [Entrée]. Vous arrivez à l'image ci-dessous :



Nous allons maintenant convertir l'image [PCX](#) en un fichier [BINAIRE](#), pour pouvoir ensuite compresser les données avec l'[utilitaire de compression](#). Et avec le programme [binDB.exe](#) convertir le fichier [BINAIRE](#) en [TEXTE](#) au format [DB](#) pour notre assembleur. (la nécessité de compresser les données...l'espace en ROM peut paraître grand mais les garphiques se montent à 60% de la capacité de notre [ROM](#)). Ces 3 outils nous sont fournis par [E.Rosby](#), mais dans mon cas, je compresses les données avec [PLETTER](#) du groupe XL2S Entertainment car la compression est la plus pertinentequ'avec RLE. Tapez [PCX2MSX.exe SET2.PCX](#) puis validez par [Entrée].

(Vous pouvez utiliser la méthode de compression que vous voulez comme [BITBUSTER](#), [MSX-o.-Mizer](#) etc.Ce qui a de la valeur c'est la rapidité, que la compression soit bonne et utilise peu de mémoire)

```

C:\msx\asmsx012e\dist012e>cd Tools
C:\msx\asmsx012e\dist012e\Tools>PCX2MSX.exe SET2.PCX









PCX2MSX v.0.10. PCX files to TMS9918 format. Edward A. Robsy Petrus [25/12/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

Original size: 256x24 pixels
TMS9918 size: 256x24 pixels
32x3=96 converted blocks

C:\msx\asmsx012e\dist012e\Tools>

```

On a ici le résultat de l'opération réalisée en sortie, quand vous travaillez avec des CHR ou des graphiques, faites attention à ce qui suit **ERROR Color collision at line (X,Y)** , cela veut dire que nous utilisons plus de 2 couleurs par octet, nous indiquant la ligne et la colonne d'erreur.

	binDB.exe	29/12/2004 23:06	Aplicación	On peut voir ici comment a été généré les2 fichiers binaires. SET2.PCX.chr qui contient les caractères et SET2.PCX.clr qui contient les couleurs des caractères que l'on va utiliser.
	MSXwav.exe	24/08/2004 20:51	Aplicación	
	PCX2MSX.exe	29/12/2004 23:31	Aplicación	
	PCX2MSXi.exe	29/12/2004 23:36	Aplicación	
	RLEpack.exe	29/11/2004 19:26	Aplicación	
	set2.pcx	17/09/2011 13:55	PhotoshopElemen...	
	SET2.PCX.chr	17/09/2011 15:55	Archivo CHR	
	SET2.PCX.clr	17/09/2011 15:55	Archivo CLR	

On va maintenant compresser le fichier BINAIRE [SET2.PCX.CHR](#) qui sont les caractères de notre set.

```

C:\msx\asmsx012e\dist012e\Tools>RLEpack.exe SET2.PCX.CHR

RLEpack v.0.10. Run-length encode packer. Edward A. Robsy Petrus [29/11/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

SET2.PCX.CHR: 768 bytes
SET2.PCX.CHR.rle: 707 bytes [92%]

C:\msx\asmsx012e\dist012e\Tools>

```

On peut voir qu'il a seulement réussi à réduire sa taille de **8%** , l'algorithme de compression de RLE est pauvre et compresse peu les données. Ceci est juste pour vous montrer.

Pour voir la différence, nous allons compresser avec PLETTER Ver 5c1- utilisant [pletter.exe](#) de XL2S. Dans les exemples de ce tutoriel du dossier Tutorial3.rar vous trouverez le fichier [pletter5c1.rar](#), décompressez-le et copier le fichier [pletter.exe](#) dans le répertoire de travail de : [C:\msx\asmsx012e\dist012e\Tools\](#)

```

C:\Windows\system32\cmd.exe

C:\msx\asmsx012e\dist012e\Tools>pletter.exe SET2.PCX.CHR SET2.PCX.CHR.PLET
Pletter v0.5c1 - www.xl2s.tk
..... SET2.PCX.CHR.PLET: 768 -> 503

C:\msx\asmsx012e\dist012e\Tools>_

```

Tapez [pletter.exe SET2.PCX.CHR SET2.PCX.CHR.PLET](#). Vous pouvez voir la différence, avec **RLE** les 768 octets sont passés à 707 octets alors qu'avec **PLETTER**, les 768 octets sont passés à 503 octets.

binDB.exe	29/12/2004 23:06	Aplicación
BITpack.exe	24/11/2003 13:12	Aplicación
MSX-O-Mizer.exe	12/05/2008 20:59	Aplicación
MSXwav.exe	24/08/2004 20:51	Aplicación
PCX2MSX.exe	29/12/2004 23:31	Aplicación
PCX2MSXi.exe	29/12/2004 23:36	Aplicación
pletter.exe	28/02/2008 10:47	Aplicación
RLEpack.exe	29/11/2004 19:26	Aplicación
set2.pcx	01/10/2011 23:51	PhotoshopElemen...
set2.pcx.chr	05/10/2011 22:26	Archivo CHR
SET2.PCX.CHR.PLET	05/10/2011 22:58	Archivo PLET
set2.pcx.chr.rle	05/10/2011 22:31	RLE File
set2.pcx.clr	05/10/2011 22:26	Archivo CLR

Nous avons gagné 265 octets dans la compression.

En exécutant [RLEpack.exe](#) Il nous génère un nouveau fichier se terminant par [.rle](#)

Alors qu'avec [pletter](#) nous lui avons dit le nom du fichier à compresser [SET2.PCX.CHR](#) et le nom que nous voulions donner au fichier compressé, nous lui avons donné le même mais avec l'extension supplémentaire [.PLET](#) générant un nouveau fichier [SET2.PCX.CHR.PLET](#)

Utilisons le [.PLET](#) qui est celui de notre tutoriel.

Nous allons maintenant convertir le fichier [SET2.PCX.CHR.PLET](#) qui est un binaire compressé en un texte au format [DB](#) pour l'incorporer à notre code . Tapez [binDB.exe SET2.PCX.CHR.PLET](#)

```

C:\Windows\system32\cmd.exe

C:\msx\asmsx012e\dist012e\Tools>binDB.exe SET2.PCX.CHR.PLET

binDB v.0.10. binary to assembler DBs. Edward A. Robsy Petrus [29/11/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

SET2.PCX.CHR.PLET: 503 bytes

C:\msx\asmsx012e\dist012e\Tools>

```

Ceci est le résultat final, nous avons maintenant un fichier appelé [SET2.PCX.CHR.PLET.asm](#)

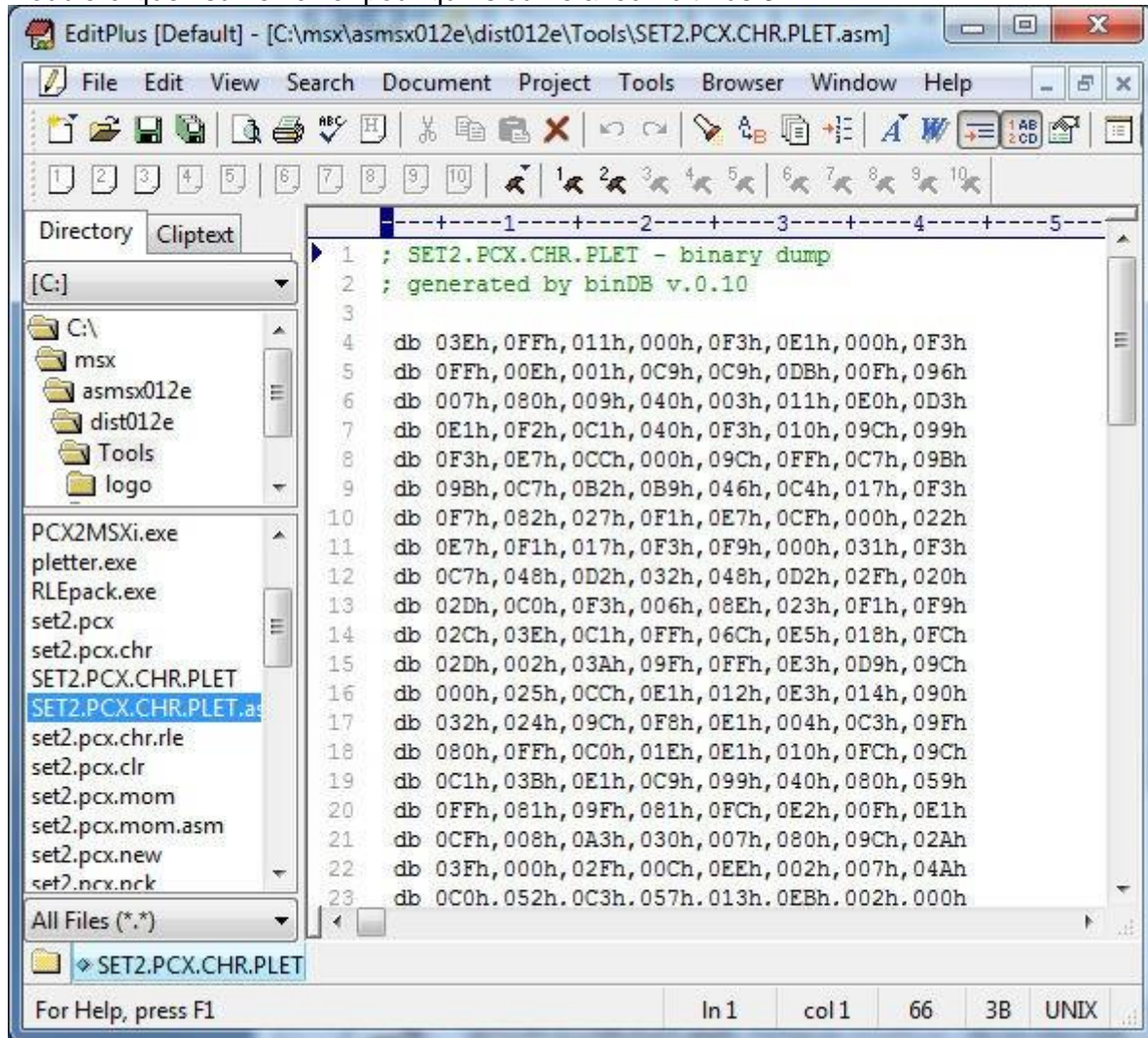
binDB.exe	29/12/2004 23:06	Aplicación
BITpack.exe	24/11/2003 13:12	Aplicación
MSX-O-Mizer.exe	12/05/2008 20:59	Aplicación
MSXwav.exe	24/08/2004 20:51	Aplicación
PCX2MSX.exe	29/12/2004 23:31	Aplicación
PCX2MSXi.exe	29/12/2004 23:36	Aplicación
pletter.exe	28/02/2008 10:47	Aplicación
RLEpack.exe	29/11/2004 19:26	Aplicación
set2.pcx	01/10/2011 23:51	PhotoshopElemen...
set2.pcx.chr	05/10/2011 22:26	Archivo CHR
SET2.PCX.CHR.PLET	05/10/2011 22:58	Archivo PLET
SET2.PCX.CHR.PLET.asm	06/10/2011 0:41	EditPlus asm Z80 (...)
set2.pcx.chr.rle	05/10/2011 22:31	RLE File
set2.pcx.clr	05/10/2011 22:26	Archivo CLR

On peut voir le fichier se terminant en [.asm](#) qui est éditible par [EditPlus](#).

Double cliquez sur le fichier pour l'ouvrir dans l'éditeur et copier/coller l'information dans notre projet.

La méthode d'ajouter les extensions est pour connaître l'état du fichier image durant tout le processus. SET2.PCX est l'IMAGE.CHR qui sont les caractères, [.PLET](#) est le fichier compressé avec [pletter](#) et [.asm](#) le fichier au format DB utilisable par notre assembleur.

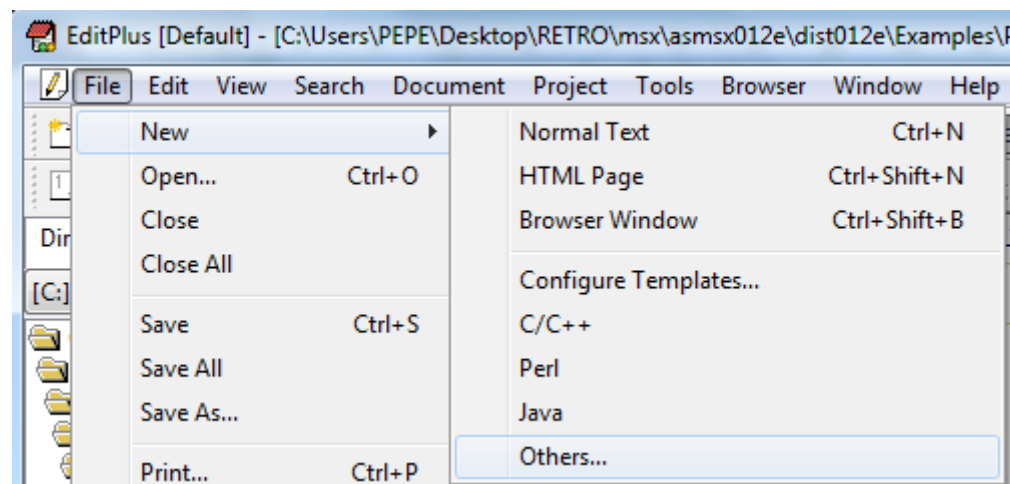
Double-cliquez sur le fichier pour qu'il s'ouvre avec EditPlus 3.



Ce sont les caractères au format DB pour l'assembleur, et ce sont les octets des CHR compressés qui le forme, nous verrons plus tard comment les manipuler.

3eme partie:

Le code de Hola Mundo Grafico. Vous vous attendez à rencontrer le même que dans la première partie du code en assembleur ? Vous savez une ROM, etc, etc... comme on a vu dans dans le premier et second tutoriel.



Allez dans menu
File – New – Others... -

asm Z80 et bouton OK

Ensuite dans File –

Save

As... y et on le met dans
le répertoire que l'on veut
(idem pour le nom). Je
l'ai nommé
HolaMundoGrafico1.asm

Le graphique sera dans
un autre onglet.

Commençons avec le code Hola Mundo Grafico

Fichier "HolaMundoGrafico1.asm" est dans le dossier **Tutorial3.rar**

```
;-----  
; Nom du programme  
; Hola Mundo Grafico - 24/09/2011  
; Version 1  
;-----  
  
;-----  
; CONSTANTES  
;-----  
; Pas de constantes déclarées  
  
;-----  
; VARIABLES DU SYSTEME  
;-----  
; Ici nous définissons avec des noms les adresses en VRAM comme expliqué dans la théorie de la première partie du tutoriel.  
  
; Adresses de la VRAM  
    CHRTBL      equ      0000h    ; Table des caracteres  
    NAMTBL      equ      1800h    ; Table des Noms  
    CLRTBL      equ      2000h    ; Table des couleurs des caractères  
    SPRATR      equ      1B00h    ; Table des attributs des sprites  
    SPRTBL      equ      3800h    ; Table des Sprites  
; Variables du Systeme MSX  
    CLIKSW      equ      $F3DB    ; Keyboard click  
    FORCLR      equ      $F3E9    ; Foreground colour  
  
;-----  
; DIRECTIVES POUR L ASSEMBLEUR ( asMSX )  
;-----  
  
    .bios        ; Definit les Noms des appels au BIOS  
    .page 2      ; Definit l'adresse de début du code en 8000h  
  
    .rom         ; C'est pour indiquer que nous créons une ROM  
    .start INICIO ; Début du code de notre programme  
  
; Suivre la norme du Standard MSX pour une ROM  
dw 0,0,0,0,0,0; 12 zeros  
  
;-----  
; DEBUT DU PROGRAMME  
;-----  
INICIO:  
  
    call  INIT_MODE_SCx ; Demarre le mode écran x  
    call  INIT_GRAFICOS ; appelle les graphiques en VRAM  
FIN:  
  
    jp     FIN          ; C'est identique à 100 goto 100 pour faire une boucle infinie.  
  
;-----  
; Tout ceci nous l'avons vu dans le précédent tutoriel ainsi il n'y a pas d'explications supplémentaires.  
; L'unique différence est que nous sommes ici en SCREEN 2 c'est pour cela que l'on active l'option - call INIGRP  
; Cette routine sert à activer n'importe lequel des modes décrits dans le texte de commentaire.  
;-----  
  
; INITIALISE LE MODE ECRAN ET LES COULEURS  
  
;-----  
; BASIC: COLOR 15,1,1  
; Etablit les couleurs  
  
INIT_MODE_SCx:  
  
    ld     hl,FORCLR      ; Variable du système  
  
    ld     [hl],15        ; Couleur du premier plan 15=Blanc  
    inc    hl             ; FORCLR+1  
    ld     [hl],1         ; Couleur du fond 1=noir  
    inc    hl             ; FORCLR+2  
    ld     [hl],1         ; Couleur du bord 1=noir  
  
; call INITXT      ; BIOS set SCREEN 0  
; call INIT32      ; BIOS set SCREEN 1  
  
    call  INIGRP          ; BIOS set SCREEN 2
```

```

; call INIMLT ; BIOS set SCREEN 3
;
; SCREEN 0 : texte de 40 x 24 avec 2 couleurs
; SCREEN 1 : texte de 32 x 24 avec 16 couleurs
; SCREEN 2 : graphiques de 256 x 192 pixels avec 16 couleurs
; SCREEN 3 : graphiques de 64 x 48 pixels avec 16 couleurs

    ret

;-----
;-----
; ENVOIT LES GRAPHIQUES EN VRAM
;-----

INIT_GRAFICOS:

; Nous faisons cela pour que rien ne se voit à l'écran
; Pendant que l'on charge les CHR et les couleurs en VRAM
    call    DISSCR        ; BIOS inihibe l'écran

; Ceci est pour enlever le son émis par le MSX quand on appuit sur une touche

    xor     a              ; ld a,0
    ld      [CLIКСW],a     ; Variable BIOS désactive le son des touches

; Remplit les 768 Octets de la Name Table - NAMTBL en VRAM
; Initialement le BIOS remplit la NAMTBL avec les valeurs 0 a 255 dans chaque tiers
; pour cela on remet tout à 0
    ld      hl,NAMTBL      ; Adresse de début en VRAM
    ld      bc,768         ; n° de CHRs à remplir
    xor     a              ; a=0 - valeur à renseigner
    call    FILVRM         ; BIOS -Fill block of VRAM with data byte ; Je vous expliquerez
plus tard le fonctionnement de FILVRM

```

Ici entre en action tout ce que l'on a appris dans les tutoriel précédents, cette partie est la liste des caractères que l'on a créé, compressé au format DB dans la VRAM, voici un fragment du code pour comprendre

<pre> ;----- ; SET2.PCX.CHR.PLET - binary dump ; generated by binDB v.0.10 ;----- SET2_PLET: db 03Eh,000h,011h,000h,018h,03Ch,000h,018h db 000h,00Eh,001h,06Ch,06Ch,048h,00Fh,096h db 007h,0FEh,009h,000h,003h,000h,030h,07Ch db 0B0h,078h,034h,0F8h,040h,030h,010h,0C6h </pre>	<p>; C'est le fichier généré par le programme binDB.exe</p> <p>; obtenu de l'image PCX compressée avec PLETTER</p> <p>; Vous devez copier/coller le texte dans le code final</p> <p>; Nous devons appeler une étiquette pour qu'elle pointe au</p> <p>; début des données de notre jeu de caractères.</p> <p>; Je l'ai appelé SET2_PLET:</p> <p>; juste en dessous des commentaires du binDB pour savoir</p> <p>; où débutent les données</p>
---	--

Si vous ne voulez pas que votre code soit trop long quand on crée un jeu vidéo, on peut enlever le code de ces données et appeler directement le fichier binaire que l'on a compressé de la manière suivante.

```

;-----
; Jeu de caracteres BINaire compressé avec PLETTER
SET2_PLET:
    .INCBIN        "SET2.PCX.CHR.PLET"

```

Ceci est le code qui va charger et décompresser les octets de notre jeu de caractères en VRAM dans la **Character Pattern Table** .

```

; Charger les graphiques des caractères en VRAM dans la CHRTBL

    ld      hl,SET2_PLET    ; jeu de CHR des caractères-Origine
    ld      de,CHRTBL+(32*8) ; Débute au CHR 32 - Destination
    call    DEPLET         ; Décompresser en VRAM

```

La routine **DEPLET** est situé dans notre code et a un paramètre d'entrée que je vais décrire : Le registre **HL** doit porter l'adresse mémoire où débute les octets des données compressées.

Dans l'exemple qui nous occupe c'est l'adresse **SET2_PLET** où sont les octets compressés du jeu de caractères.

```
ld    hl,SET2_PLET          ; set de CHR des caractères
```

Le registre **DE** doit pointer à l'adresse VRAM où l'on veut décompresser les octets.

Nous voulons charger les octets à partir du **CHR 32** de la **CHRTBL** en VRAM pour cela nous calculons pour la CHRTBL, 32 x 8 octets qui composent chaque caractère. La raison pour laquelle le jeu de caractères débute au CHR 32 est expliqué en page 9 de ce tutoriel.

```
ld    de,CHRTBL+(32*8)      ; Débute au CHR 32
```

Ececi est l'appel à la routine qui prend les octets compressés et les décompressent en VRAM suivant des valeurs passées en **HL** et en **DE**. Commence par chercher les octets du **SET2_PLET** les décompresse et les charge dans la **Character Pattern Table - CHRTBL** a partir du CHR 32.

```
call  DEPLET                ; Décompresser en VRAM
```

; Charge les couleurs des lettres en VRAM dans la CLRTBL

```
ld    hl,CLRTBL+(32*8)      ; Débute au CHR 32 de la CLRTBL
ld    bc,(32*24)            ; numero de CHRs
ld    a,0Fh                ; Valeur à charger
call  FILVRM                ; BIOS -Fill block of VRAM with data
                                byte
```

La routine **FILVRM** est situé dans le BIOS et a des paramètres en entrée : Le registre **HL** doit pointer à l'adresse mémoire de la VRAM, où nous voulons charger les octets.

Dans cet exemple, nous voulons charger les octets au CHR 32 de la **Colour Table – CLRTBL**

```
ld    hl,CLRTBL+(32*8)      ; Débute au CHR 32 de la CLRTBL
```

Dans le registre **BC** nous lui donnons le nombre d'octets à charger.
















Le jeu de caractères occupe 32 CHR de large sur une hauteur de 3 CHR, mais souvenons-nous que chaque caractère a une hauteur de 8 octets, ainsi il nous faut multiplier 3 x 8=24 octets de haut, de ce fait, on charge le registre **BC** avec 32 x 24

```
ld    bc,(32*24)            ; Nombre de CHR
```

Dans le registre A nous positionnons l'octet qui va remplir la zone sélectionnée.

Dans l'exemple, la couleur du caractère est le blanc F et le fond noir 0, qui est le même que **0Fh** en hexa. 16 couleurs de fond et 16 couleurs de caractères, rappelez-vous que les couleurs débutent à 0 et terminent à 15 – 15 en hexadécimal c'est F et le noir 0, résultat 0Fh, si vous voulez changer les couleurs, vous savez ce qu'il vous reste à faire.

```
ld    a,0Fh                ; Valeur à charger
```

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
															

Ceci est l'appel à la routine qui commence à écrire l'octet 0Fh en VRAM débutant en CLRTBL+256 et ne s'arrête que lorsqu'elle a rempli les 32x24=768 Octets que sont les 3 lignes de 32 CHR qui compose le jeu de carctères dans la **Color Table - CLRTBL**

```
call  FILVRM                ; BIOS -Fill block of VRAM with data byte
```

; Charge la chaîne de texte à l'écran

; LOCATE 6,2: PRINT "Hola Mundo Grafico"

```
ld    hl,TXT_HOLA           ; Direction où se trouve le texte
ld    de,NAMTBL+6+(2*32)    ; LOCATE 6,2 : adresse de la NAMBTl dans la VRAM
ld    bc,18                 ; Nombre de CHR que contient le texte
call  LDIVRM                ; BIOS - Copy block to VRAM, from memory
```

La routine **LDIVRM** est dans le BIOS et a des paramètres d'entrée suivant :

Le register **HL** indique l'endroit de la RAM-ROM où sont les octets que nous souhaitons transmettre à la VRAM.

Dans l'exemple, c'est la chaine de texte ASCII que nous voulons envoyer à la VRAM.

```
ld    hl,TXT_HOLA           ; Direction où se trouve le texte
```

Le registre **DE** pointe la direction en VRAM où nous voulons charger les octets qui sont dans la RAM-ROM.

Nous voulons afficher les octets dans la **Name Table – NAMTBL** en ligne 2 – Colonne 6 de l'écran.

Ainsi nous calculons la direction. NAMTBL la ligne n° 2 x32 CHR que comporte chaque ligne et on ajoute les 6 de la colonne.

```
ld    de,NAMTBL+6+(2*32)    ; LOCATE 6,2
```

Le register **BC** est chargé avec le nombre d'octets que l'on transfère de la RAM-ROM vers la VRAM.

Dans l'exemple, 18 octets qui sont les CHR qui composent le texte "Hola Mundo Grafico"=18 caractères.

```
ld    bc,18                ; Nombre de CHR que contient le texte
```

Ceci est la routine qui commence à transférer les 18 octets de la RAM-ROM vers la VRAM en NAMTBL+70. Dans notre exemple, la valeur des code ASCII du texte, sont situés dans la [Name Table - NAMTBL](#)

```
call  LDIRVM               ; BIOS - Copy block to VRAM, from memory
```

```
-----  
; Nous appelons cette routine pour afficher de nouveau l'écran.
```

```
call  ENASCR               ; BIOS afficher l'écran
```

```
; Sortir de a routine INIT GRAFICOS
```

```
ret
```

```
-----
```

Continuons avec notre texte, facile de taper **db** "HOLA MUNDO Grafico" il suffit de regarder la table ASCII de la page 9 et de regarder les valeurs qui correspondent à chaque caractère, nous pourrions l'interpréter ainsi **db** 72,79,76,65,32,77,85,78,68,79,32,71,114,97,102,105,99,111 mais ce travail asMSX va le faire pour nous au moment de la compilation, cependant ce sera les valeurs qui s'inscriront (les valeurs ASCII), voir la page 9 du tutorial qui explique plus en détail cette partie.

```
; Chaîne de texte à afficher à l'écran
```

```
TXT_HOLA:
```

```
db    "HOLA MUNDO Grafico"
```

```
-----  
Ceci est la routine chargée de décompresser les octets en VRAM, dans mon cas j'avais choisi la compression PLETTER, mais vous avez plusieurs outils sur la marché de compression-décompression comme RLE, BITBUSTER,MSX-o-Mizer, Exomizer, etc. Vous pouvez sélectionner le système de décompression que vous voulez en modifiant cette routine par le code adapté au système de compression que vous avez choisi.
```

Voici la routine de décompression Pletter v0.5b de la RAM-ROM vers la VRAM directement. Vous pouvez l'étudier si vous voulez pour voir comment elle fonctionne ou l'utiliser simplement, je vous ai expliqué que les paramètres d'entrée de cette routine doivent être fournis.

```
-----  
; Pletter v0.5b VRAM Depacker v1.1 - 16 Kb version
```

```
; HL = RAM/ROM source
```

```
; DE = VRAM destination
```

```
-----
```

```
DEPLET:
```

```
di
```

```
; VRAM address setup
```

```
ld    a,e
```

```
out   (099h),a
```

```
ld    a,d
```

```
or    040h
```

```
out   (099h),a
```

```
; Initialization
```

```
ld    a,[hl]
```

```
inc   hl
```

```
exx
```

```
ld    de,0
```

```
add   a,a
```

```
inc   a
```

```
rl    e
```

```
add   a,a
```

```
rl    e
```

```
add   a,a
```

```
rl    e
```

```
rl    e
```

```
ld    hl,modes
```



```

        add    hl,de

        ld     e,[hl]
        ld     ixl,e

        inc    hl
        ld     e,[hl]
        ld     ixh,e

        ld     e,1
        exx

        ld     iy,loop
; Main depack loop
literal:

        ld     c,098h
        outi

        inc    de
loop:    add    a,a

        call   z,getbit
        jr     nc,literal

; Compressed data
        exx
        ld     h,d
        ld     l,e
getlen: add    a,a

        call   z,getbitexx
        jr     nc,lenok
lus:
        add    a,a
        call   z,getbitexx
        adc    hl,hl

        ret    c

        add    a,a
        call   z,getbitexx
        jr     nc,lenok

        add    a,a
        call   z,getbitexx
        adc    hl,hl
lenok:  jp     c,Depack_out
        add    a,a

        call   z,getbitexx
        jp     c,lus

        inc    hl
        exx

        ld     c,[hl]

        inc    hl
        ld     b,0

        bit    7,c
        jp     z,offsok
        jp     ix

mode7:

mode6:

mode5:

```

```

add    a,l,a    b
call   z,getbit
add    a,a
rl     b
call   z,getbit
add    a,l,a    b
call   z,getbit
mode4: add    a,a
        call   z,getbit
        rl     b
mode3:  add    a,a
        call   z,getbit
        rl     b
mode2:  add    a,a
        call   z,getbit
        rl     b
        add    a,a
        call   z,getbit
        jr     nc,offsok
offsok: or     a
        inc    b
        res    7,c
        inc    bc
        push   hl
        exx
        push   hl
        exx
@@loop: ld     l,e
        ld     h,d
        sbc    hl,bc
        pop    bc
        push   af
        ld     a,l
        out    (099h),a
        ld     a,h
        nop                    ; VDP timing
        out    (099h),a
        nop                    ; VDP timing
        in     a,(098h)
        ex     af,af'
        ld     a,e
        nop                    ; VDP timing
        out    (099h),a
getbit: ld     a,d
        or     040h
        out    (099h),a
        ex     af,af'
        nop                    ; VDP timing
        out    (098h),a
        inc    de
        cpi
        jp     pe,@@loop
        pop    af
        pop    hl

```

```

jp      ld      a,[hl]

inc     hl
iy      rla

ret

getbitexx:
exx

ld      a,[hl]
inc     hl
exx

rla
ret

; Depacker exit
Depack_out:

ei
ret

modes: dw      offsok

dw      mode2
dw      mode3

dw      mode4
dw      mode5

dw      mode6
dw      mode7

```

Là , c'est la partie du code au format DB du jeu de caractères que nous avons au préalable compressé et converti au format DB avec binDB.exe et que j'ai expliqué plus tôt.

```

SET2_PLET:
db  03Eh,0FFh,011h,000h,0F3h,0E1h,000h,0F3h
db  0FFh,00Eh,001h,0C9h,0C9h,0DBh,00Fh,... .

```

```

;-----

```

Ou bien enlever tous ces DB et les inclures dans le fichier BINAIRE

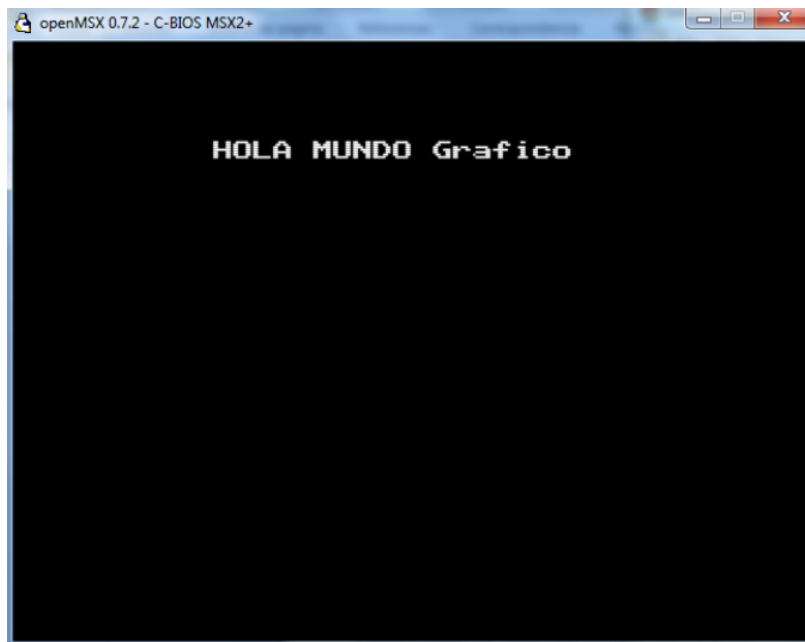
Directement dans le code de cette manière

Le fichier SET2.PCX.PLET doit être dans le même répertoire que celui du code assembleur.

```

;-----
; SET2.PCX.CHR.PLET
; Jeu de caracteres en BINaires compressés avec PLETTER
;SET2_PET:
; .INCBIN"SET2.PCX.CHR.PLET"
;-----
; FIN DU CODE EN ASSEMBLEUR.
;-----

```



Ceci est le résultat final si tout c'est bien passé



Tout cela est triste sans couleur, dans la seconde partie du “monde des graphiques couleurs” nous allons voir ce processus.

L'image de gauche, nous verrons comment la réaliser dans le prochain tutoriel.

Vous pouvez créer la même chose avec ce que je vous ai appris dans ce tutoriel.

J'espère que cela vous a plu et nous nous verrons au prochain tutoriel. Je vous expliquerai comment donner une couleur spécifique à nos caractères via le code, comment intégrer des graphiques à notre Hola Mundo Grafico et tous les liens entre la création d'écrans ou ce qui revient au même travailler avec la [NAMTBL](#) pour mapper avec [nMSXtiles](#).

José Vila Cuadrillero