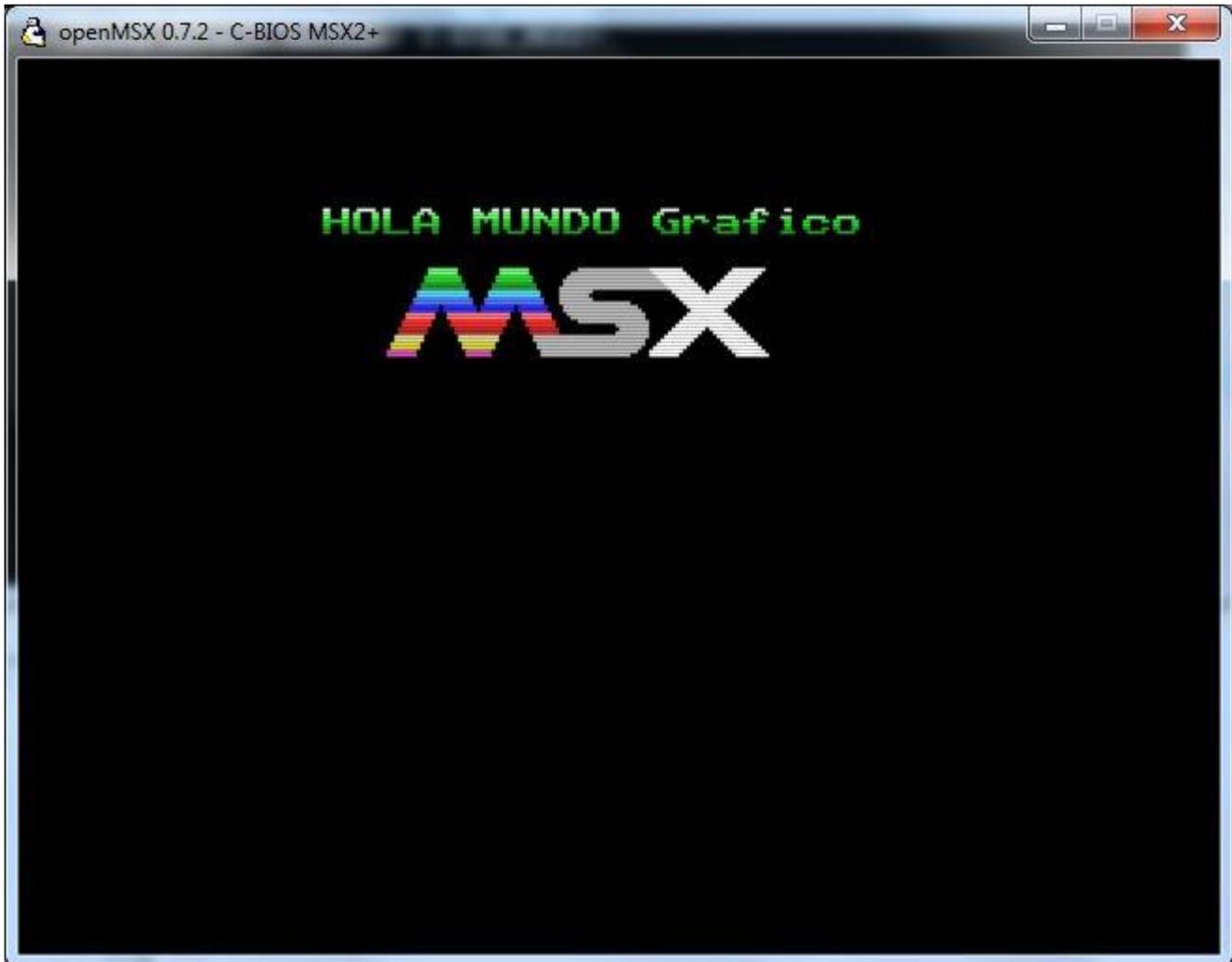


TUTORIEL D'INITIATION A LA PROGRAMMATION EN LANGAGE ASSEMBLEUR POUR MSX

PARTIE 4 – LE MONDE DES GRAPHIQUES 2

Dans cette partie du tutoriel nous allons revenir au code d' [Hola Mundo Grafico](#) de la partie 3 et lui ajouter une couleur spéciale à notre jeu de caractères, en plus nous rajouterons un graphique ou logo et nous allons voir comment travailler depuis le code afin d'obtenir ce que l'on voit dans l'écran ci-dessous, nous repasserons sur les principes de la partie 3 afin d'obtenir l'image ci-dessous

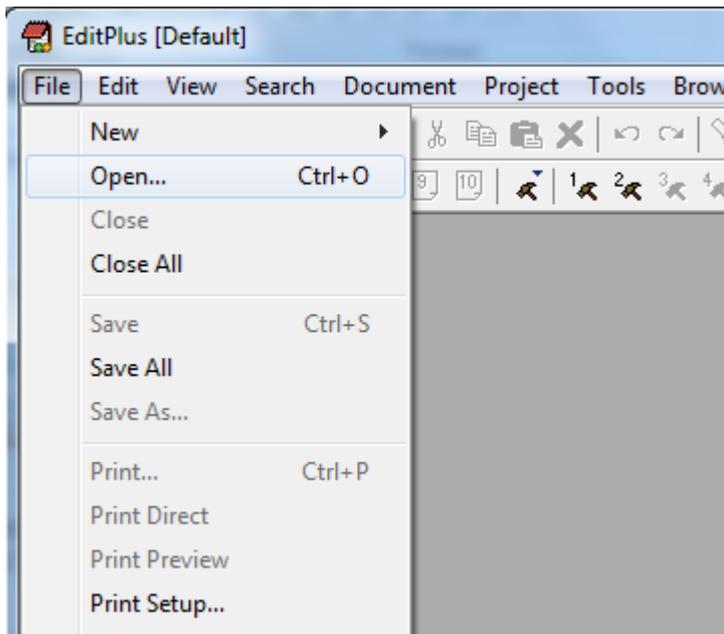


Ceci sera le résultat de notre nouveau tutoriel, mais avant de démarrer nous allons voir une option intermédiaire. Cette version intermédiaire nous l'appellerons " HolaMundoGrafico2.asm" qui fait partie du paquet de ce tutoriel [Tutorial4.rar](#)
<http://www.dimensionzgames.com/wp-content/uploads/downloads/2012/02/Tutorial4.rar>

Je voudrais faire remarquer que le graphique du logo MSX je l'ai créé et dessiné sous GIMP et nous allons voir ensuite comment importer nos graphiques dans le programme nMSXtiles.
Les couleurs qui figurent sur le logo MSX est une référence à la palette originale du MSX comportant 16 couleurs.

Tous ce que l'on va voir ici, nous servira pour créer des menus ou écrans d'initialisation de vos ROM, si votre objectif est de créer les écrans d'introduction de vos jeux.

Allons-y pour le code d' [HolaMundoGrafico2.asm](#)



Lancer [EditPlus 3](#), depuis le menu

Appuyez sur ...

[File – Open](#)

Ouvrez le fichier [HolaMundoGrafico1.asm](#) de la partie 3 des tutoriels..

Quand il est ouvert, avant de la modifier , faites la chose suivante

Appuyez sur [File – Save As...](#) et enregistrez le dans le répertoire que vous voulez avec le nom que vous voulez, moi j'ai choisi [HolaMundoGrafico2.asm](#)

Voici le code d' [HolaMundoGrafico1](#) que nous avons enregistré sous le nouveau nom de [HolaMundoGrafico2.asm](#)

```
-----  
; Nom de notre programme  
; Hola Mundo Grafico - 24/09/2011  
; Version 1  
-----
```

La premiere chose est de changer le nom du projet , la date et le numéro de la version.

```
-----  
; Nom de notre programme  
; Hola Mundo Grafico - 28/09/2011  
; Version 2  
-----
```

La seconde chose que nous allons modifier c'est que nous voulons donner une couleur spéciale à notre jeu de caractères et c'est cette partie du code que je modifie et que je vous expliquerai ensuite.

```
; Charger les couleurs des caractères en VRAM de la CLRTBL  
ld hl,CLRTBL+(32*8) ; Débuter au CHR 32 de la CLRTBL  
ld bc,(32*24) ; nombre de caracteres  
ld a,0Fh ; Valeur à charger  
call FILVRM ; BIOS -Fill block of VRAM with data byte
```

La routine [FILVRM](#) est située dans le BIOS et à des paramètres d'entrée : Le registre **HL** qui pointe en direction de la mémoire VRAM à l'endroit où nous voulons commencer à charger les caractères.

Dans l'exemple, nous voulons charger les caractères à partir du caractère 32 de la [Colour Table – CLRTBL](#)

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

Dans le registre **BC** nous lui indiquons combien d'octets nous voulons charger.

Le jeu de caractères occupe 32 caractères de large sur 3 caractères de haut, mais souvenez-vous que chaque caractère a une hauteur de 8 octets, il nous faut donc multiplier 3 x 8=24 octets de haut, pour cela on met 32 x 24 dans le registre **BC**.

```
ld bc,(32*24) ; nombre de caracteres
```

Dans le registre **A** nous lui indiquons l'octet à utiliser pour remplir la zone sélectionnée.

Dans l'exemple, la couleur de caractère est blanc F et le fond noir 0, qui est la même chose que **0Fh** en hexa. 16 couleurs de fond et 16 couleurs de caractères, qui débutent à 0 et terminent à 15, si on regarde la palette MSX, le blanc correspond à 15 et le noir à 0- 15 c'est F en hexadécimal et le noir 0 est 0Fh, si vous voulez changer les couleurs, vous savez comment maintenant.

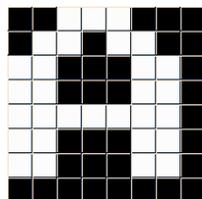
```
ld a,0Fh ; Valeur à charger
```



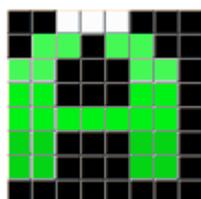
Ceci appelle la routine qui commence à écrire l'octet 0Fh en VRAM commençant en CLRTBL+256 et ne s'arrêtant que lorsque les 32x24=768 octets seront remplis, qui sont les 3 lignes de 32 CHR qui représentent le jeu de caractères dans la [Color Table - CLRTBL](#)

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

Nous allons maintenant pouvoir charger les 96 CHR de 8 octets chacun dans la **Colour Table - CLRTBL** pour donner la couleur blanche fond noir à toutes les lettres de notre jeu de caractères.



Byte 0 - 0Fh
Byte 1 - 0Fh
Byte 2 - 0Fh
Byte 3 - 0Fh
Byte 4 - 0Fh
Byte 5 - 0Fh
Byte 6 - 0Fh
Byte 7 - 0Fh



Byte 0 - 0Fh
Byte 1 - 03h
Byte 2 - 03h
Byte 3 - 02h
Byte 4 - 02h
Byte 5 - 0Ch
Byte 6 - 0Ch
Byte 7 - 0Ch

Ceci est la nouvelle couleur de notre jeu de caractères.

On a une combinaison de couleurs, vous pouvez le modifier. Mais vous devrez aussi modifier le code et les valeurs dans **TBL_MICLR** et donc mettre les vôtres.

Nous allons maintenant procéder au codage de tout ceci en assembleur.

Ceci est la partie à enlever du code, ou bien modifiez-le avec le code d'en dessous.

```
; Charger les couleurs des lettres en VRAM de la CLRTBL
ld    hl,CLRTBL+(32*8)      ; Débuter au CHR 32 de la CLRTBL
ld    bc,32*24             ; Nombre de caracteres
ld    a,0Fh                ; Valeur à renseigner
call  FILVRM               ; BIOS -Fill block of VRAM with data byte
```

Voici la nouvelle partie du code qui remplace la précédente.

```
; Charger en multi-couleurs les lettres en VRAM de la CLRTBL
ld    hl,TBL_MICLR         ; Table avec les couleurs des CHR
ld    de,CLRTBL+(32*8)    ; Débuter au CHR 32 de la CLRTBL
ld    b,(32*3)            ; Nombre de caracteres
call  COPY_BLOCK          ; Routine appelée pour réaliser le process
```

Vous devez vous douter de ce que va réaliser cette partie du code.

On charge **HL** avec la table de 8 octets de la couleur que nous voulons pour nos CHR.

On charge **DE** pour pointer à l'adresse **CLRTBL+(32*8)** en VRAM pour que cela débute au CHR n°32. On charge le registre **B** avec le nombre de CHR à mettre à jour avec la table de couleur.

Avec ces paramètres d'entrée nous appelons une nouvelle routine que nous allons créer dans notre code appelée **COPY_BLOCK** qui aura la responsabilité de trouver les 8 octets de couleur qui composent la table, et que nous allons appliquer au set de caractères dans la **Colour Table - CLRTBL** en VRAM.

A ce niveau, nous allons utiliser les noms courts dans les descriptions de zones de la VRAM.

Juste en dessous dans la partie du **TXT_HOLA** nous allons ajouter la table multi-couleurs que nous voulons donner à notre jeu de caractères, ajoutons ce code :

; Chaîne de texte à visualiser dans l'écran

```
TXT_HOLA:
db    "HOLA MUNDO Grafico"
```

; Table avec le multi-couleur pour les CHR

```
TBL_MICLR:
db    0Fh,03h,03h,02h,02h,0Ch,0Ch,0Ch ; Ce sont les codes couleurs que nous avons vu plus haut sur les octets de la lettre A
```

Ceci est la routine [COPY_BLOCK](#) qu'il faut ajouter juste en dessous de ce que nous venons d'ajouter.

```
-----  
; Copie de manière séquentielle les blocs de 8 octets en VRAM  
; Paramètres: HL = Direction de l'origine en RAM-ROM  
;             DE = Direction de la destination en VRAM  
;             B  = Nombre de CHR à copier maximum 256 CHR  
-----  
COPY_BLOCK:  
  push  bc           ; On garde en mémoire combien de CHR à remplir  
  push  hl           ; On garde en mémoire la direction de l'origine  
  push  de           ; On garde en mémoire la direction de la destination  
  ld    bc,8         ; On copie les premiers 8 octets  
  call  LDIRVM       ; BIOS - Copy block to VRAM, from memory  
  pop   hl           ; On récupère la direction de la destination  
  ld    bc,8         ; 8 octets rajoutés à la direction  
  add   hl,bc        ; La destination en VRAM sera 8 octets plus loin  
  ex    de,hl        ; Passons le registre hl dans le registre de  
  pop   hl           ; On récupère la direction de la destination  
  pop   bc           ; On récupère le nombre de CHR restants  
  djnz  COPY_BLOCK   ; S'il reste au moins un CHR on répète...  
                    ; ...tout le process jusqu'à obtenir 0 CHR  
  ret                ; On quitte la routine quand c'est fini  
-----
```

Comme on peut l'observer dans la routine qui est très commentée, elle fait appel à la routine [BIOS LDIRVM](#) chargée d'emmener les octets de la ROM-RAM à la VRAM, dans ce cas, elle va copier 8 par 8 octets la table des couleurs à chaque caractères de la [CLRTBL](#) finissant la boucle quand elle arrive au CHR 0.

La [CLRTBL](#) à la fin de l'appel de [COPY_BLOCK](#) ressemblera à ceci. Bloc 0



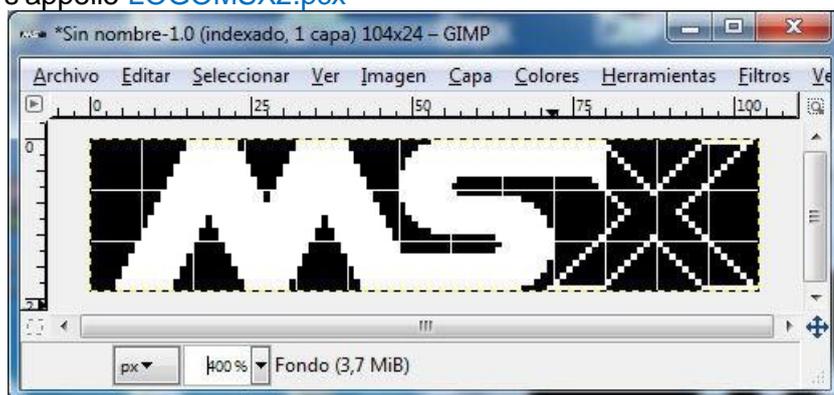
On peut maintenant compiler et exécuter notre code [HolaMundoGrafico2.asm](#)



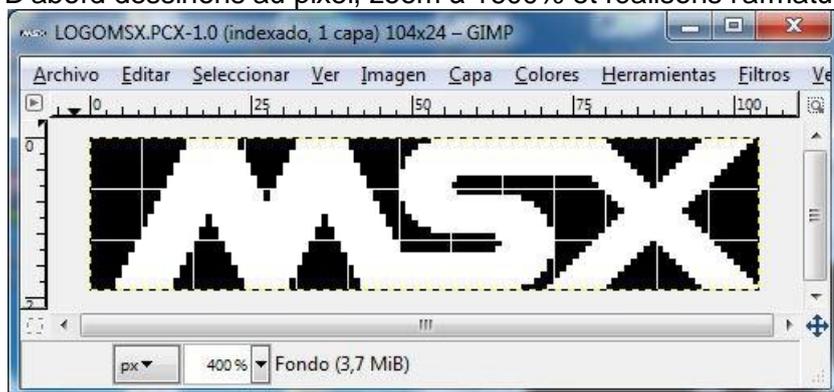
Et voilà, à partir de maintenant nos textes sont en plusieurs couleurs. Nous avons vu comment il est facile d'implémenter de la couleur.

Nous allons maintenant voir le logo ou graphique à côté du Hola Mundo.

Il faudra en premier créer le logo (ou graphique) que nous voulons introduire à l'écran. Ouvrons ainsi GIMP et mettons nous à l'oeuvre, créons un graphique comme vu dans la partie 3 quand nous avons vu la création de jeu de caractères. J'ai mis une copie-écran du processus de création. Le fichier s'appelle [LOGOMSX2.pcx](#)



D'abord dessinons au pixel, zoom à 1600% et réalisons l'armature du dessin.



Ensuite avec l'outils de remplissage, remplissons en blanc l'intérieur.

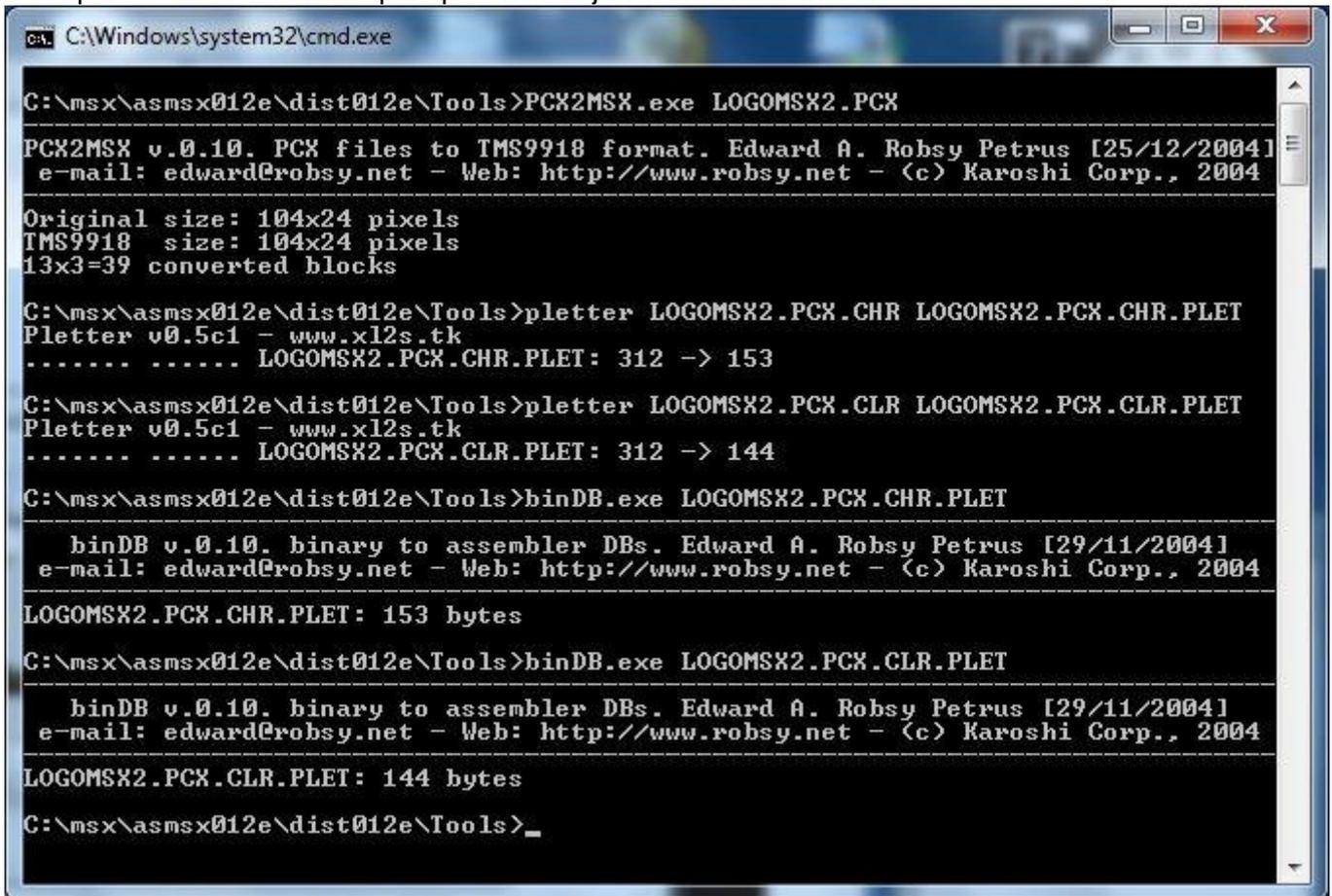


Finalement, la mise en couleur se fait en faisant attention de ne pas mettre plus de 2 couleurs par CHR à l'aide de la grille 8x1 de GIMP. Le PCX2MSX vous informera de l'erreur si vous utilisez plus de 2 couleurs par octet.

Attention, à chaque fois que nous créons une nouvelle image, GIMP charge la palette par défaut, dans le menu [Couleurs-Map-Etablir le mappage des couleurs](#) appuyez sur DEFAULT et sélectionnez dans la liste déroulante [VDP-MSX-TMS9918-\(Palette PEPE\)](#) appuyez sur le bouton accepter et quittez. Ceci, je vous l'ai déjà expliqué si vous ne voulez pas avoir de surprise à l'écran plus tard.

On conserve le format de fichier PCX avec le nom [LOGOMSX2.PCX](#) dans le répertoire où sont les outils, pletter.exe, pcx2msx.exe et le bindb.exe.

Cette partie devrait être facile puisque on l'a déjà vu dans le tutoriel 3.



Cette capture d'écran résume tout le processus.

- 1 – On génère les fichiers binaires séparant les **CHRs** et les **CLRs** avec PCX2MSX.exe
- 2 – On compresse le fichier binaire des caractères **LOGOMSX2.PCX.CHR** avec pletter.
- 3 - On compresse le fichier binaire des couleurs **LOGOMSX2.PCX.CLR** avec pletter
- 4 – On convertit le binaire compressé des **CHRs** au format assembleur avec le binDB.exe
- 5 - On convertit le binaire compressé des **CLRs** au format assembleur avec le binDB.exe

Nous venons de créer nos fichiers **.asm** avec les octets compressés de notre logo.

Le premier avec les caractères **LOGOMSX2.PCX.CHR.PLET.asm** que nous appellerons **LOGO_CHR**:

```
; LOGOMSX2.PCX.CHR.PLET - binary dump  
; generated by binDB v.0.10  
LOGO_CHR:  
db 01Eh,0FFh,000h,000h,0FEh,0FEh,0FCh,0FCh  
db 0F8h,0F8h,0F0h,01Ah,0F0h,001h,001h,0C1h  
-----
```

Le second avec les couleurs **LOGOMSX2.PCX.CLR.PLET.asm** que nous appellerons **LOGO_CLR**:

```
; LOGOMSX2.PCX.CLR.PLET - binary dump  
; generated by binDB v.0.10  
LOGO_CLR:  
db 01Eh,000h,000h,000h,003h,003h,002h,002h  
db 00Ch,00Ch,007h,040h,007h,007h,022h,022h  
-----
```

Ces étiquettes sont positionnées au début de chaque code pour les localiser par le code assembleur car on va les ajouter au code du **HolaMundoGrafico3.asm** que nous allons créer.

Repétons le processus du début du tutoriel [HolaMundoGrafico2.asm](#) et sauvegardons le code comme [HolaMundoGrafico3.asm](#) , modifions la date et la version, nous allons le modifier pour ajouter le logo MSX.

Nous allons à nouveau ajouter du code :
Juste en dessous de l'appel de la chaîne de texte, ajoutons la portion de code suivante.

```
; Appeler la chaîne de texte à l'écran
; LOCATE 6,2: PRINT "Hola Mundo Grafico"
    ld    hl, TXT_HOLA           ; Endroit où se trouve la chaîne de texte
    ld    de, NAMBTBL+6+(2*32)   ; LOCATE 6,2 : Adresse de la NAMBTBL en VRAM
    ld    bc, 18                 ; Nombre de CHR que comporte le texte
    call  LDIRVM                 ; BIOS - Copy block to VRAM, from memory
```

On ajoute ces 2 groupes de code à cet endroit.

```
; Appeler le logo MSX
    ld    hl, LOGO_CHR           ; Direction des CHR du logo MSX
    ld    de, CHRTBL+(128*8)     ; Nous les plaçons à partir du CHR n°128 dans la CHRTBL
    call  DEPLET                 ; Descompresser en VRAM

; Appeler les couleurs du logo MSX
    ld    hl, LOGO_CLR           ; Direction des CLR du logo MSX
    ld    de, CLRTBL+(128*8)     ; Nous les plaçons à partir du CHR n°128 dans la CLRTBL
    call  DEPLET                 ; Décompresser en VRAM
```

Les explications sont toujours les mêmes.

Basiquement, ce que l'on réalise c'est décompresser les octets des CHR du logo en appelant la CHRTBL et commençant à la 5eme ligne ou ce qui revient au même, au [CHR n°128](#).

La seconde partie décompresse les octets des couleurs des CHR et les plaçons dans la CLRTBL en débutant au premier CHR de la 5eme ligne ou [CHR n°128](#).

J'ai mis une capture d'écran avec le résultat final de la CHRTBL et CLRTBL, le 1er tiers ou bloc devrait être comme ceci.



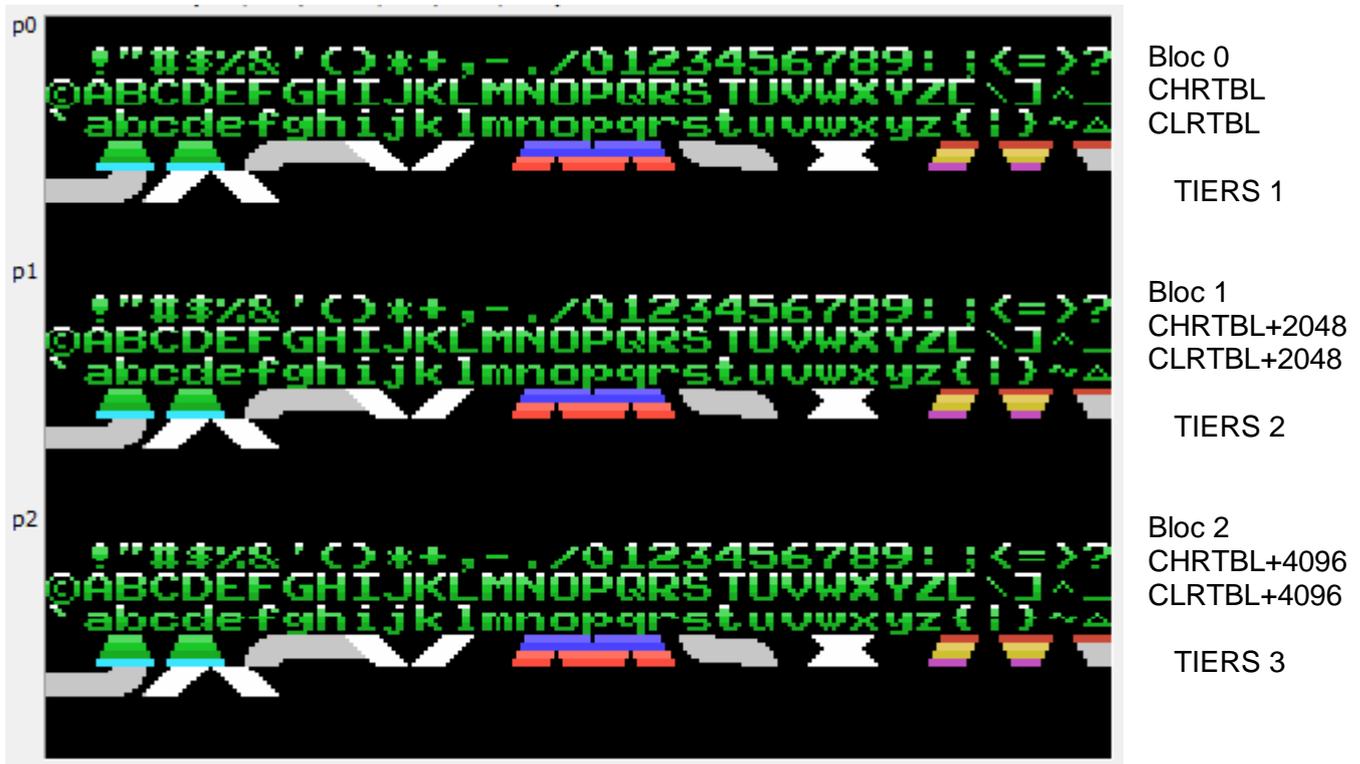
Cette partie doit être expliquée si on veut modifier le tutoriel ou si l'on veut faire apparaître les caractères ou graphiques dans n'importe quelle zone de l'écran pour vos futures créations.

Rappelez-vous que dans la partie 3, je vous ai dit que la VRAM était divisée en 3 parties pour l'écran , 3 bloc de 256 CHR chacun, si on veut afficher des caractères dans le 1er tiers de l'écran il faut utiliser le bloc 1 de jeu de caractères. Si on veut faire pareil pour les 2 autres tiers, il faut répéter l'opération pour les 2 autres blocs de la CHRTBL et de la CLRTBL.

Ce n'est pas difficile avec ce que l'on a appris, vous pouvez le faire vous même, je vous donne une piste. 256 CHR représentent un bloc multiplié par 8 octets qui compose chaque caractère nous donne un total de 2048 octets.

```
; Charger les graphiques de lettres en VRAM dans la CHRTBL dans le 2eme tiers
    ld    hl, SET2_PLET          ; jeu de CHR des lettres
    ld    de, CHRTBL+2048+(32*8) ; Débute au CHR 32 du bloc 1
    call  DEPLET                 ; Décompresse en VRAM

; Charger les graphiques de lettres en VRAM dans la CHRTBL dans le 3eme tiers
    ld    hl, SET2_PLET          ; jeu de CHR des lettres
    ld    de, CHRTBL+4096+(32*8) ; Débute au CHR 32 du bloc 3
    call  DEPLET                 ; Décompresse en VRAM
```



Ayant fait cela, nous pouvons afficher du texte et le logo MSX n'importe où à l'écran.

Continuons avec le HolaMundoGrafico3, nous avons les CHR et les CLR en VRAM, il nous reste plus qu'à les appeler dans la NAMTBL via les numéros de CHR qui correspondent au logo MSX, pour que tout s'affiche de la même façon qu'on l'a fait pour le texte.

Le logo MSX est composé de 13 CHR de large et de 3 lignes de CHR de haut et comme on a chargé le 1er CHR du logo au numéro 128, nous allons procéder ainsi :

```

Locate 8,4; Print chr$(128);chr$(129) ;chr$(130) ;chr$(131) ;chr$(132) ;chr$(133) ;chr$(134) ;chr$(135)
;chr$(136) ;chr$(137) ;chr$(138) ;chr$(139) ;chr$(140)

; Chargement des 13 premiers CHR du logo
ld hl,NAMLOGO ; 1ere ligne du mappage
ld de,NAMTBL+8+(4*32) ; LOCATE 8,4
ld bc,13 ; Nombre de CHR
call LDIRVM ; BIOS - Copy block to VRAM, from memory

Locate 8,5; Print chr$(141);chr$(142) ;chr$(143) ;chr$(144) ;chr$(145) ;chr$(146) ;chr$(147) ;chr$(148)
;chr$(149) ;chr$(150) ;chr$(151) ;chr$(152) ;chr$(153)

; Chargement des 13 seconds CHR du logo
ld hl,NAMLOGO+13 ; 2eme ligne du mappage
ld de,NAMTBL+8+(5*32) ; LOCATE 8,5
ld bc,13 ; Nombre de CHR
call LDIRVM ; BIOS - Copy block to VRAM, from memory

Locate 8,6; Print chr$(154);chr$(155) ;chr$(156) ;chr$(157) ;chr$(158) ;chr$(159) ;chr$(160) ;chr$(161)
;chr$(162) ;chr$(163) ;chr$(164) ;chr$(165) ;chr$(166)

; Chargement des 13 derniers CHR du logo
ld hl,NAMLOGO+13+13 ; 3eme ligne du mappage
ld de,NAMTBL+8+(6*32) ; LOCATE 8,6
ld bc,13 ; Nombre de CHR
call LDIRVM ; BIOS - Copy block to VRAM, from memory

```

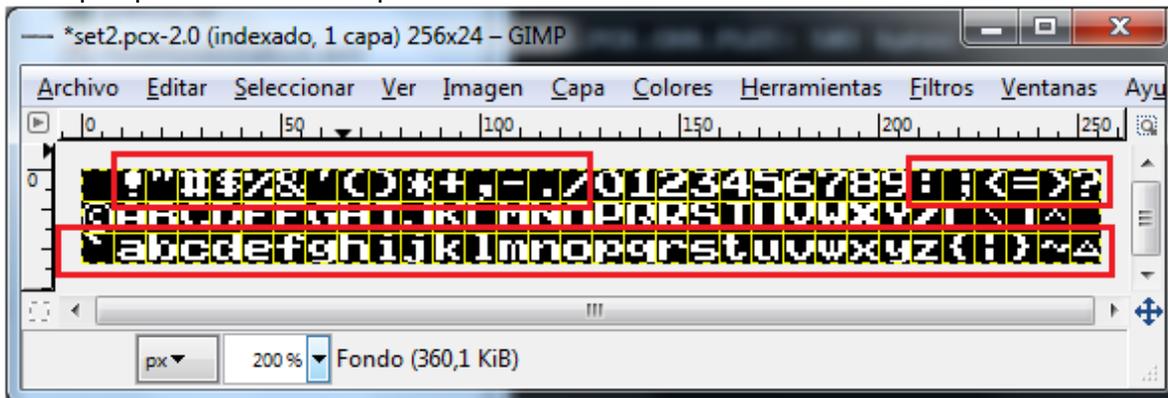
Pour cela j'ai créé une table appelée NAMLOGO qui a les valeurs des octets qui seront chargés dans la NAMTBL juste en dessous des octets compressés et des couleurs du logo MSX. ([Regardons ci-dessous](#))

A la fin des octets compressés du jeu de caractères se trouvent ceux du logo MSX.

```
-----  
; LOGOMSX2.PCX.CHR.PLET - binary dump  
; generated by binDB v.0.10  
-----  
LOGO_CHR:  
db 01Eh,0FFh,000h,000h,0FEh,0FEh,0FCh,0FCh  
db 0F8h,0F8h,0F0h,01Ah,0F0h,001h,001h,0C1h  
db 011h,07Ch,07Ch,038h,038h,0F4h,00Dh,000h  
db 07Fh,000h,07Fh,03Fh,03Fh,01Fh,01Fh,0F8h  
db 0E0h,0C0h,035h,080h,080h,0B5h,013h,080h  
db 01Eh,080h,0C0h,0E0h,0F0h,0F8h,0FCh,050h  
db 0FEh,012h,024h,00Fh,007h,003h,055h,001h  
db 04Eh,04Ch,02Ah,001h,029h,001h,003h,007h  
db 00Fh,01Fh,03Fh,0ACh,027h,0E0h,0B5h,014h  
db 000h,040h,020h,020h,070h,070h,010h,010h  
db 0D6h,042h,005h,008h,008h,01Ch,01Ch,00Fh  
db 03Ch,03Ah,03Dh,003h,085h,025h,09Fh,051h  
db 0F8h,04Ch,061h,057h,01Fh,054h,0FBh,09Bh  
db 0AEh,00Fh,01Dh,05Eh,0EEh,05Fh,06Eh,07Fh  
db 0FFh,055h,0C7h,019h,02Dh,047h,055h,04Bh  
db 00Bh,00Fh,013h,0F4h,01Dh,013h,03Eh,03Eh  
db 03Eh,07Fh,08Ch,0E6h,061h,0C0h,0E6h,051h  
db 0B4h,024h,037h,01Eh,0F3h,0BEh,05Bh,0BFh  
db 0CFh,0DFh,01Fh,0DEh,0FFh,0FFh,0FFh,0FFh  
db 0F8h  
  
; .INCBIN "LOGOMSX2.PCX.CHR.PLET"  
-----  
  
-----  
; LOGOMSX2.PCX.CLR.PLET - binary dump  
; generated by binDB v.0.10  
-----  
LOGO_CLR:  
db 01Eh,000h,000h,000h,003h,003h,002h,002h  
db 00Ch,00Ch,007h,040h,007h,007h,022h,022h  
db 0CCh,0CCh,077h,05Bh,077h,011h,01Eh,00Fh  
db 033h,033h,0CEh,00Fh,00Eh,000h,07Ch,0EEh  
db 000h,00Eh,06Dh,000h,007h,01Ch,01Dh,0FFh  
db 0EFh,000h,00Fh,01Eh,00Fh,000h,00Fh,08Fh  
db 000h,0FFh,01Fh,007h,000h,000h,000h,005h  
db 005h,004h,004h,009h,009h,088h,004h,088h  
db 055h,055h,044h,044h,007h,008h,051h,008h  
db 00Fh,007h,099h,099h,0BDh,00Fh,0D5h,01Fh  
db 007h,05Bh,0D3h,069h,05Fh,00Eh,0EFh,068h  
db 03Ch,078h,046h,0F0h,058h,0FFh,0FFh,0FBh  
db 067h,0E0h,05Fh,006h,006h,00Bh,00Bh,003h  
db 00Ah,00Ah,00Dh,00Dh,066h,066h,06Dh,007h  
db 0A8h,00Fh,017h,00Fh,0BBh,0BBh,036h,0AAh  
db 0AAh,00Fh,0EBh,027h,00Fh,067h,061h,000h  
db 0DBh,0C1h,09Ch,0D4h,000h,00Fh,0F6h,05Fh  
db 0BFh,0DFh,03Fh,0D7h,0FFh,0FFh,0FFh,0F0h  
  
; .INCBIN "LOGOMSX2.PCX.CLR.PLET"  
-----  
  
-----  
; table avec les CHR du logo MSX  
NAMLOGO:  
db 128,129,130,131,132,133,134,135,136,137,138,139,140  
db 141,142,143,144,145,146,147,148,149,150,151,152,153  
db 154,155,156,157,158,159,160,161,162,163,164,165,166  
-----  
  
-----  
; FIN DU CODE ASSEMBLEUR.  
-----
```

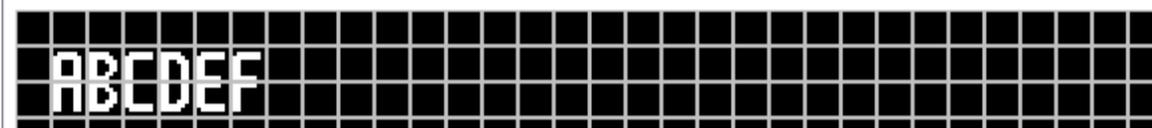
On peut maintenant compiler et exécuter le code de HolaMundo3.asm, le résultat final est visible à la première page du tutoriel.

Voici quelques améliorations possibles.



Les lettres en minuscules s'utilisent peu, nous avons ainsi 32 CHR de plus pour des graphiques, ou appeler le logo MSX dans ces caractères. Idem pour les symboles graphiques qui sont au nombre de 21 CHR que nous pouvons utiliser, le reste doit rester à sa position (suivant la norme ASCII) J'ai ajouté un CHR peu utilisé le Ñ .

On peut créer un jeu de caractères de la manière suivante, une lettre de 2 fois la taille.



Dans les majuscules on crée la partie haute des lettres et en minuscules la partie basse. Ainsi le code suivant va faire les chargements des CHR dans la NAMTBL.

```

; Chaîne de texte à visualiser à l'écran
TXT_MA:
db      "ABCDEF"
TXT_MI:
db      "abcdef"

; Charger la chaîne de texte à l'écran
; LOCATE 6,2: PRINT "ABCDEF"
ld      hl,TXT_MA          ; Endroit où se trouve le texte
ld      de,NAMTBL+6+(2*32) ; LOCATE 6,2 : Destination de la NAMTBL en VRAM
ld      bc,6              ; Nombre de CHR que contient le texte
call   LDIRVM            ; BIOS - Copy block to VRAM, from memory

; LOCATE 6,3: PRINT "abcdef"
ld      hl,TXT_MI         ; Endroit où se trouve le texte
ld      de,NAMTBL+6+(3*32) ; LOCATE 6,3 : Destination de la NAMTBL en VRAM
ld      bc,6              ; Nombre de CHR que contient le texte
call   LDIRVM            ; BIOS - Copy block to VRAM, from memory
    
```

Le résultat est en dessous sans mise en couleur.



Je vous laisse expérimenter.

J'espère que cela vous a été utile et on se retrouve pour le prochain tutoriel

José Vila Cuadrillero