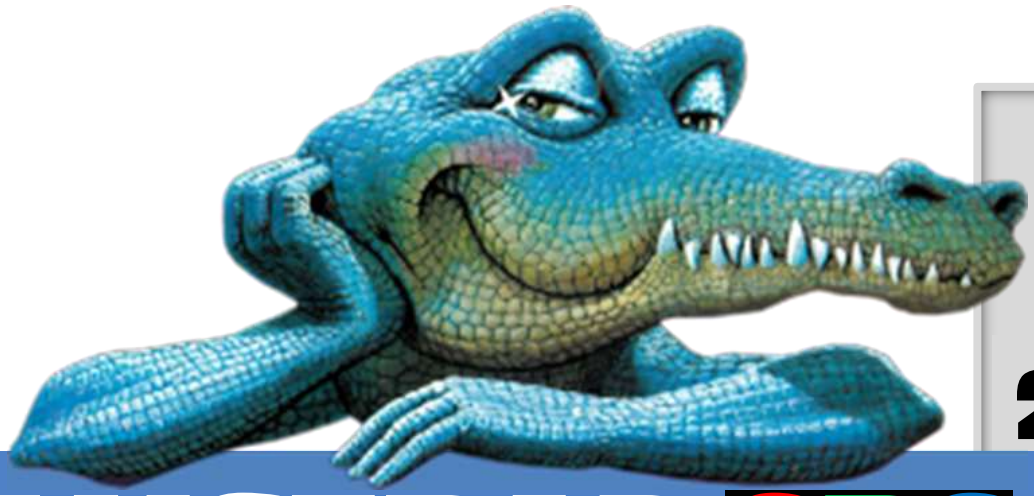
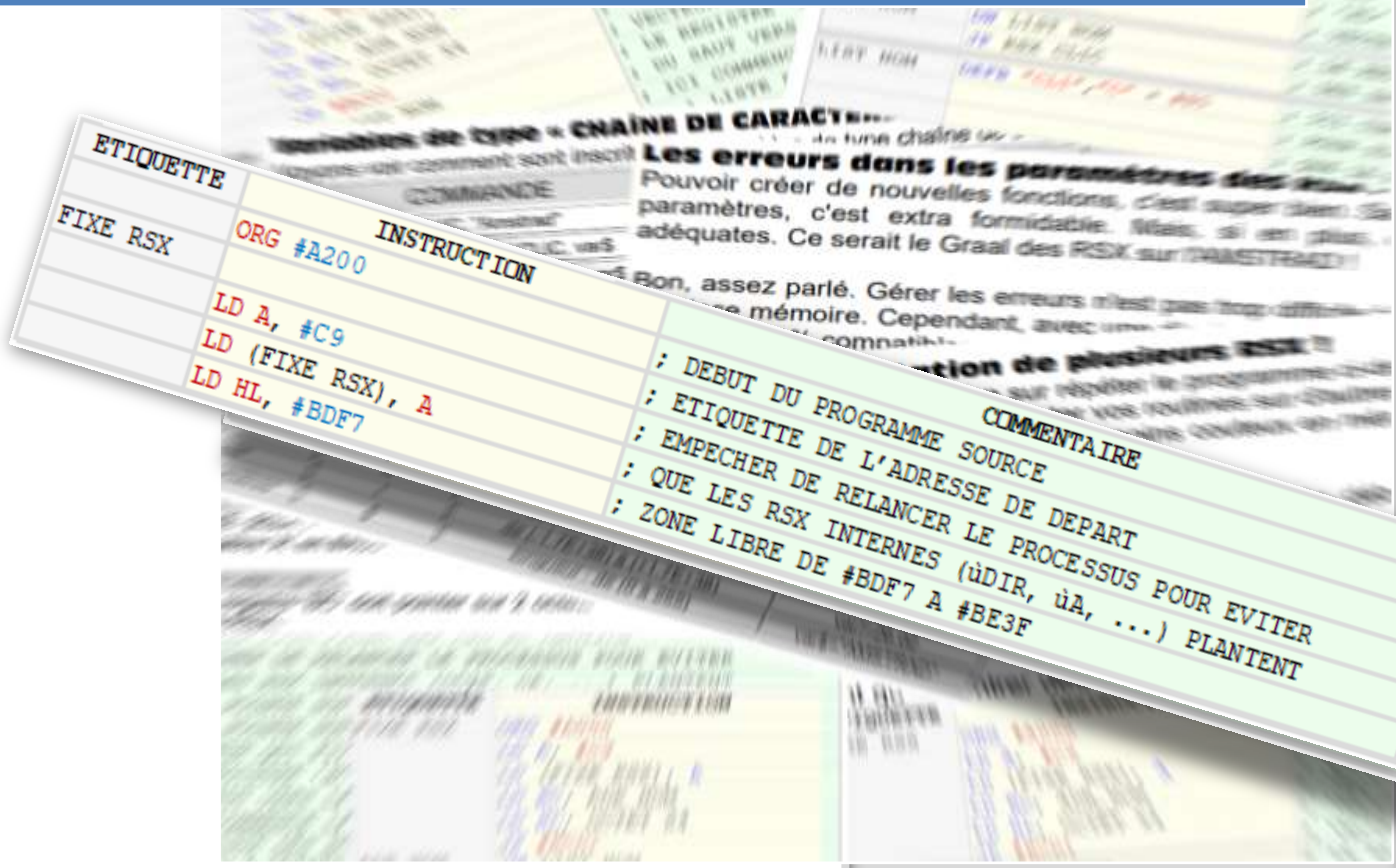


2023

AMSTRAD CPC

RSX : Les mini-caractères



Ce tutoriel s'adresse aussi bien aux débutants qu'aux plus confirmés désirant se perfectionner en langage Assembleur. A travers de nombreux exemples et explications très détaillées, vous apprendrez à créer de nouvelles commandes RSX qui permettront d'afficher des mini-caractères très colorés tout en vous dévoilant les secrets des couleurs du MODE 0 ainsi que le fameux codage des adresses écran de l'AMSTRAD CPC.

Philippe MOULIN

AMSTRAD-CPC

TUTORIEL RSX - Les mini-caractères en folie

Ce didacticiel fait suite à mon guide : AMSTRAD CPC – Les RSX et leurs paramètres.

- S O M M A I R E -

Partie Assembleur

Matière à réflexion :

Si vous pensez que l'Amstrad CPC est un ordinateur trop ancien pour le monde d'aujourd'hui et que ce tutoriel est complètement dépassé, alors je vous répondrai que c'est un fait, le monde des ordinateurs évolue trop rapidement à mon goût. Pourtant, on dit bien que c'est dans les vieux pots que l'on fait les meilleures soupes et l'AMSTRAD CPC est pour moi, la recette de base dans le domaine de la programmation en Assembleur, mais aussi en BASIC. Les langages de programmation d'aujourd'hui, sont bien trop sophistiqués pour le plaisir de programmer et de partager nos acquis tout en s'amusant.

LE RETOUR DES MINI-CARACTÈRES EN FOLIE !

À travers les 370 octets que composent les routines de ce programme, découvrez comment créer des commandes RSX qui vous permettront d'afficher des petits caractères de plusieurs couleurs, dans le **MODE 0** de l'Amstrad CPC.
Les nombreux exemples et illustrations de ce tutoriel, vous permettront de suivre pas à pas le déroulement de chacune des étapes de la programmation en assembleur, tout en vous dévoilant les astuces et les pièges à éviter sur l'Amstrad CPC.
Je vous conseille de le lire une première fois avant de vous plonger dans la programmation afin d'obtenir une vue plus orientée sur l'ensemble de cet ouvrage.

Pourquoi « le retour » ?

C'est une version très améliorée de mon ancien programme de 2019 portant le même nom et dont vos nombreux commentaires m'ont encouragé à créer ce tutoriel en tenant compte de vos remarques**.**

Les plus de cette nouvelle version :

* Plus rapides : toutes les routines ont été revitalisées et améliorées.
* Plus nombreux : de 54 mini-caractères, ils sont à présent 65 sous vos ordres.
* Plus sûr : ils sont armés de nombreux tests anti-plantages.
* Moins gourmands : soit un gain de 57 octets par rapport à la version précédente.
* Toujours aussi FUN : ils n'auront plus de limites en votre imagination.

La possibilité de redéfinir vos propres mini-caractères et de pouvoir les enregistrer dans le même fichier que les commandes RSX, facilitera la programmation et rendra vos programmes plus clairs et légers.
En plus d'être entièrement compatibles sur toute la gamme des AMSTRAD CPC, les 65 mini-caractères n'occupent que 569 octets **tout compris**. Ils vous en feront toujours autant voir 

À savoir sur les RSX de ce tutoriel :

- Ce tutoriel ce base sur les ordinateurs Amstrad CPC avec un clavier AZERTY. C'est pourquoi certains caractères utilisés ici, ne correspondront pas sur les autres types de claviers. C'est surtout valable pour le caractère [**ù**] situé devant le nom des commandes RSX de ce tutoriel qui devra être remplacé par le caractère [**|**] et c'est aussi valable pour le caractère [**à**] situé devant le nom des variables qui lui, correspond à une arobase sur les autres types de claviers.

- Avant d'installer des commandes RSX, il faut d'abord leur réserver un emplacement dans la mémoire RAM pour éviter qu'ils soient écrasés par les autres programmes.

- Pour que le BASIC de l'AMSTRAD CPC puisse reconnaître les noms donnés aux commandes RSX, il faut les initialiser une première fois en utilisant l'instruction `CALL adresse` du BASIC Locomotive.

- Si le nombre des paramètres envoyés depuis une commande RSX, ne correspond pas, la commande sera annulée et rien ne se passera. C'est aussi valable si la valeur d'un paramètre sort des limites attendues. Cela est nécessaire pour éviter que l'ordinateur ne se plante sans que l'on ne sache le pourquoi, d'où ça vient.

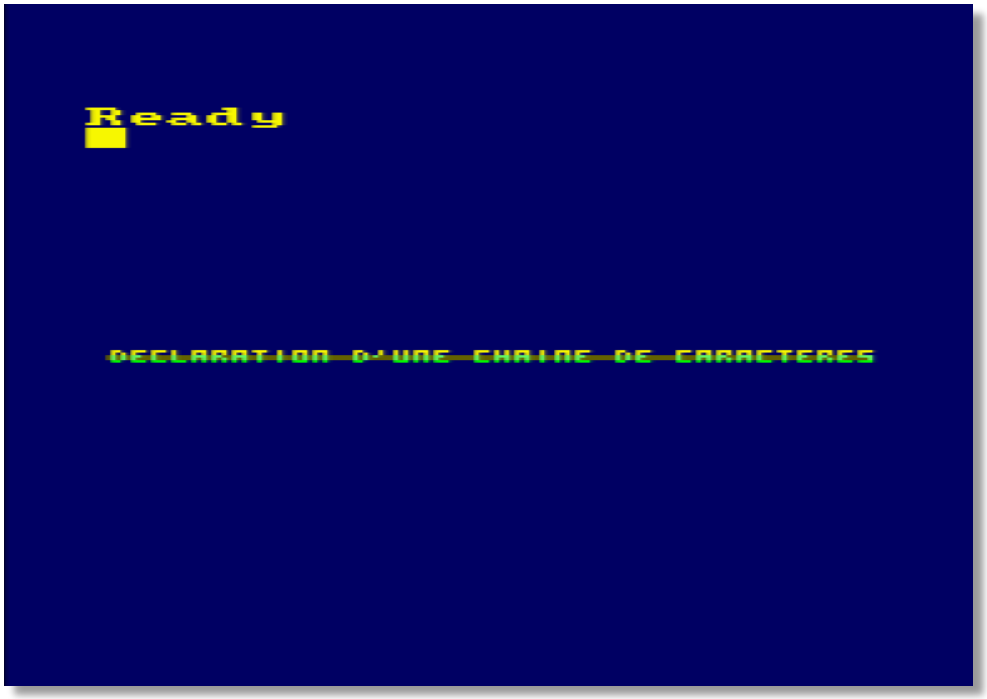
- Les chaînes de caractères utilisées dans les paramètres des commandes RSX, doivent être

déclarées avant leur utilisation et leurs noms doivent être précédés du symbole arobase pour rendre vos programmes 100 % compatibles sur toute la gamme des Amstrad CPC.

Voici un exemple de toutes les explications citées ci-dessus :

```
10 MEMORY &A1FF
20 LOAD"mini2.bin", &A200
30 CALL &A200
40 chaine$="Declaration d'une chaine de caracteres"
50 MODE 0
60 ùENCRE, 1, 13, 12
70 ùPAPIER, 0, 9, 0
80 ùCURSEUR, 1, 16
90 ùFONTE, àchaine$
```

Résultat :



Explications de cet exemple :

`10 MEMORY &A1FF` Réserve l'emplacement des commandes RSX

`20 LOAD "mini2.bin", &A200` Installe les commandes RSX à l'adresse &A200

`30 CALL &A200` Initialise les noms des RSX

`40 chaine$= "declaration d'une chaine de caractere"` Déclare une variable de type chaîne de caractères

`50 MODE 0` Sélectionne le **MODE 0** sur l'écran

`60 ùENCRE, 1, 13, 12` Initialise les 3 encres à utiliser pour les mini-caractères

`70 ùPAPIER, 0, 9, 0` Comme [**ùENCRE**] mais pour les 3 couleurs du papier

`80 ùCURSEUR, 1, 16` Positionne le curseur des mini-caractères sur l'écran

`90 ùFONTE, àchaine$` Affiche la chaîne de caractères sur l'écran et déplace le curseur à la suite du dernier caractère affiché

SYNTAXE DES NOUVELLES COMMANDES RSX

ùCURSEUR, Colonne, Ligne

Déplace le curseur des mini-caractères vers une position relative au coin supérieur gauche de l'écran.

* Colonne (de 0 à 39) représente l'axe X du curseur.
* Ligne (de 0 à 32) représente l'axe Y du curseur.
* Les coordonnées 0, 0 indiquent le coin supérieur gauche de l'écran.
* Les coordonnées 39, 32 correspondent au coin inférieur droit de l'écran.

ùENCRE, Encre1 [, Encre2] [, Encre3]

Encre1, Encre2 et Encre3 (de 0 à 15) indiquent les couleurs des encres à utiliser pour les mini-caractères.
Les paramètres (Encre2 et Encre3) sont facultatifs et en cas d'omission, les encres concernées garderont leur ancienne valeur.

ùPAPIER, Papier1 [, Papier2] [, Papier3]

Papier1, Papier2, Papier3 (de 0 à 15) indiquent les couleurs des papiers à utiliser pour les mini-caractères.
Les paramètres (Papier2 et Papier3) sont facultatifs. Leurs valeurs resteront inchangées en cas d'omission.

NB : (**ùENCRE** et **ùPAPIER**)
Les mini-caractères sont composés de 6 lignes. Les couleurs changeront automatiquement toutes les 2 lignes. Ainsi, on pourra utiliser jusqu'à 6 couleurs différentes pour afficher les mini-caractères.

ùROULE, àchaine$

Cette instruction fait faire une rotation vers la gauche du contenu de la chaîne de caractère. Cette méthode très rapide en langage machine, était très utilisée dans les anciens listings écrits en BASIC pour faire défiler un message en bas de page.

Exemple :

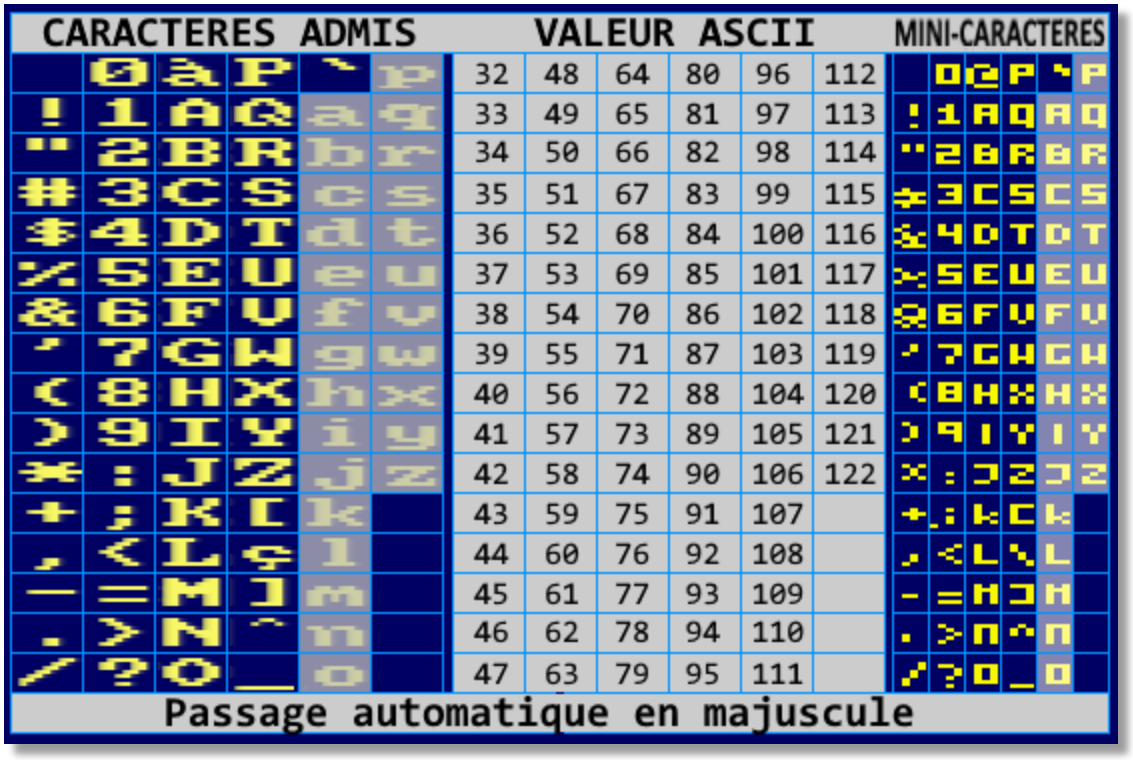
```
a$= "Roulez jeunesse ! "
ùROULE, àa$
PRINT a$

oulez jeunesse ! R
```

ùFONTE, àchaine$

Affiche la chaine de caractères à la position du curseur des mini-caractères.

Exemple :

LISTE DES 65 CARACTÈRES AUTORISÉS



Tous les autres caractères présents dans la chaîne, seront remplacés par un caractère vide (code ASCII 32).

```
ùDEFCAR, Caractère, Octet1, Octet2, Octet3
```

Redéfinis la forme d'un mini-caractère.

- 'Caractère' doit contenir le code ASCII du mini-caractère à redéfinir.
- Octet1, Octet2, Octet3 (de 0 à 255) représentent chacun 2 lignes d'un caractère.

Cette commande peut aussi être utilisée de cette manière :

```
ùDEFCAR, Caractère, Lig1, Lig2, Lig3, Lig4, Lig5, Lig6
```

- Lig1 à Lig6, (de 0 à 15) représentent les 4 BITs de chacune des lignes composant le mini- caractère.

EXEMPLE :

Nous allons représenter la matrice du mini-caractère [**1**] par un petit carré.

```
ùDEFCAR, 49, &F, &9, &9, &9, &9, &F
```

Ou dans sa version simplifiée :

```
ùDEFCAR, 49, &F9, &99, &9F
```

* Le code ASCII dont le numéro est 49 représente le caractère [**1**].

Résultat :

ASTUCE :

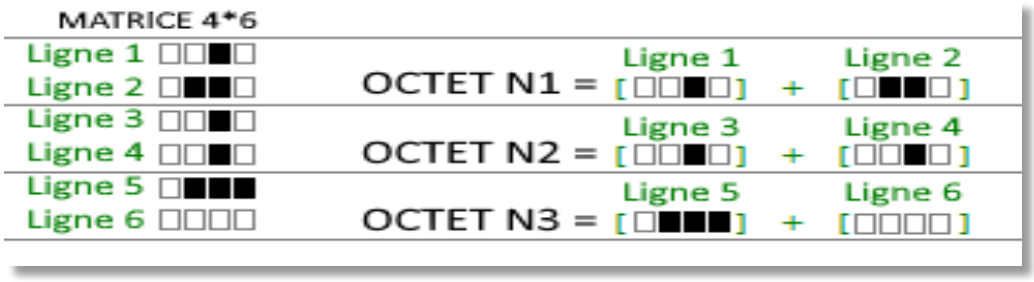
Le code ASCII du caractère à redéfinir peut être obtenu avec la fonction **ASC("")** du BASIC. Ça peut toujours être utile si on n'en connait pas sa valeur.

Ainsi, `ùDEFCAR, ASC("1"), 49, &F9, &99, &9F` reviendra au même que les 2 exemples cités ci-dessus.

MATRICE DES MINI-CARACTÈRES

Étudions comment sont constitués les mini-caractères avant de redonner forme au mini-caractère [**1**] qui nous a été bien utile pour les exemples.
La largeur des mini-caractères est de 4 pixels et sa hauteur est de 6 pixels.
Ce qui nous donne le nombre de 4*6 = 24 pixels. Les 24 pixels seront enregistrés dans 3 octets.
En sachant que 1 octet = 8 BITs, le calcul est vite fait 3*8 = 24. Tout correspond bien.
Comme 2 lignes de 4 pixels rentrent dans 1 octet, il suffit alors d'enregistrer les 2 premières lignes dans le premier octet, puis les deux lignes suivantes dans le second octet et les deux dernières lignes dans un troisième octet.

Schéma représentant le mini-caractère [**1**].



La commande pour restituer le mini-caractère [**1**] sera donc :

```
ùDEFCAR, 49, &26, &22, &70
```

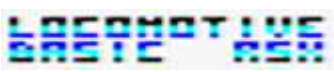
Pour simplifier les choses, on pourra aussi l'écrire de cette manière :

```
ùDEFCAR ASC("1"), 2, 6, 2, 2, 7, 0
```

Dans ce dernier exemple, chaque paramètre d'une valeur (de 0 à 15), représente une ligne de 4 pixels d'un mini-caractère. La routine RSX se chargera de reconstituer automatiquement les valeurs des 6 lignes pour les transférer dans le 3 octets utilisés pour la matrice du mini-caractère.
Exemple d'un petit programme qui utilise les mini-caractères pour dessiner un petit train :

```
10 MEMORY &A1FF
20 LOAD"mini2.bin", &A200
30 CALL &A200
40 MODE 0
50 ùDEFCAR, ASC("0"), &72, &F8, &F3
60 ùDEFCAR, ASC("1"), &72, &E3, &F1
70 ùDEFCAR, ASC("2") ,&F2, &2E, &F8
80 ùDEFCAR, ASC("3"), &76, &2B, &F3
90 ùDEFCAR, ASC("4"), &FB, &AE, &F6
100 a$="012": b$="34"
110 ùCURSEUR, 0, 5
120 ùENCRE, 3, 4, 1:ùPAPIER, 0, 0, 5
130 ùFONTE, @a$
140 ùENCRE, 3, 2
150 ùFONTE, @b$
160 ùENCRE, 3, 7
170 ùFONTE, @b$
180 ùENCRE, 6, 2
190 ùFONTE, @b$
```

RÉSULTAT :

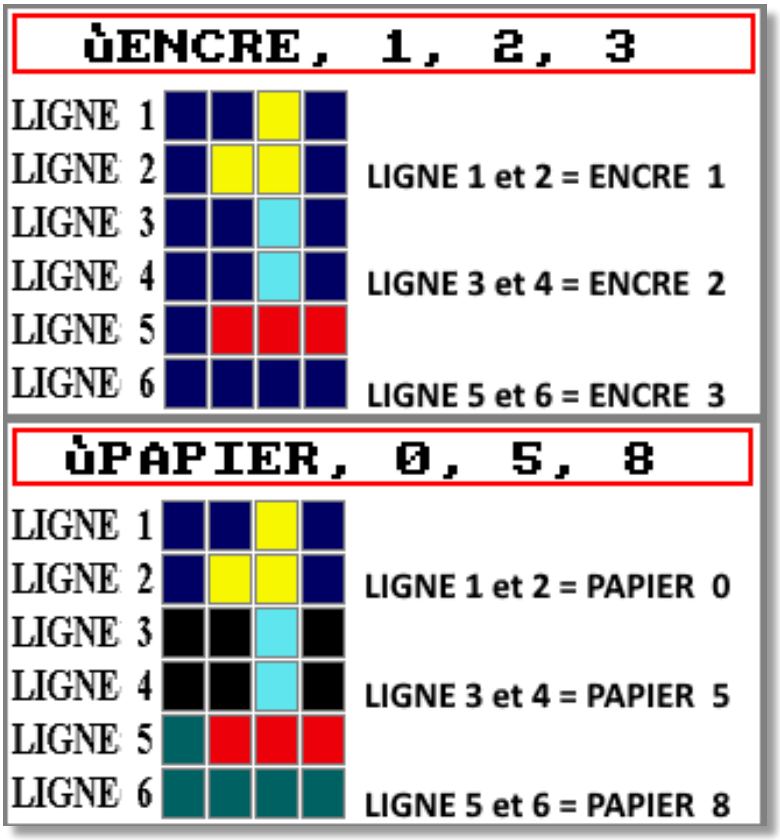


COULEURS DES ENCRES ET DES PAPIERS

Nous allons mieux nous pencher sur les couleurs des mini-caractères afin de bien comprendre comment fonctionnent les 2 commandes RSX des couleurs.
Je vous ai parlé plus haut que les couleurs tournaient toutes les deux 2 lignes du mini-caractère. C'est-à-dire que l'affichage d'un mini-caractère démarre du haut et va vers le bas. Il y a donc un compteur pour savoir sur quelle ligne il faut envoyer les données pour former le mini-caractère. Et c'est ce compteur de lignes qui nous servira pour en changer les couleurs.

Un mini-caractère est constitué de 6 lignes, comme on changera la couleur des encres et des papiers toutes les 2 lignes, il nous faudra donc 3 couleurs pour les encres et 3 autres couleurs pour les papiers.
Rien n'oblige à utiliser 3 couleurs différentes. Un mini-caractère peut aussi être affiché en une seule couleur d'encre et de papier, mais il peut être embelli en plusieurs couleurs différentes avec ce programme !

Examinons les deux commandes RSX des couleurs :



Comme vous pouvez le voir, il est très simple d'afficher des mini-caractères en plusieurs couleurs.
Les 2 premières lignes du mini-caractère auront la couleur du premier paramètre, les 2 lignes du milieu auront la couleur du deuxième paramètre et enfin les 2 dernières lignes auront la couleur du troisième paramètre.

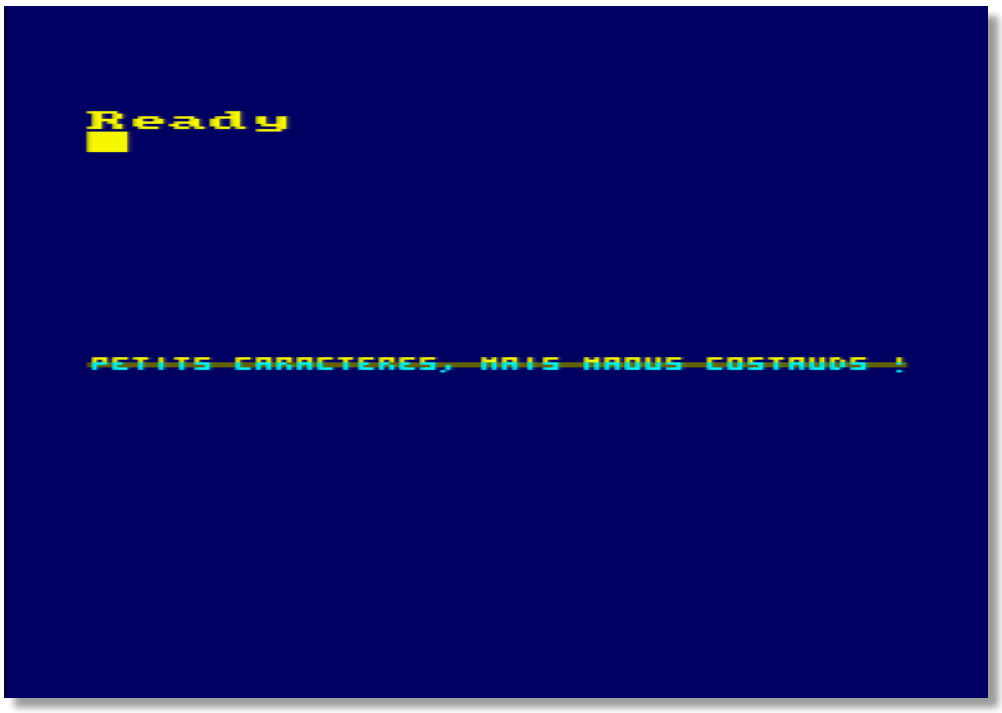
Notez que seuls, les paramètres qui seront présents dans les commandes RSX { **ùENCRE**] et [**ùPAPIER**] seront pris en compte. S'il en manque, leurs valeurs ne seront pas modifiées.

Ces 2 commandes RSX ressemblent aux commandes [**PEN**] et [**PAPER**] du BASIC Amstrad, mais ne concernent que les mini-caractères et ne modifient pas les valeurs des commandes BASIC.

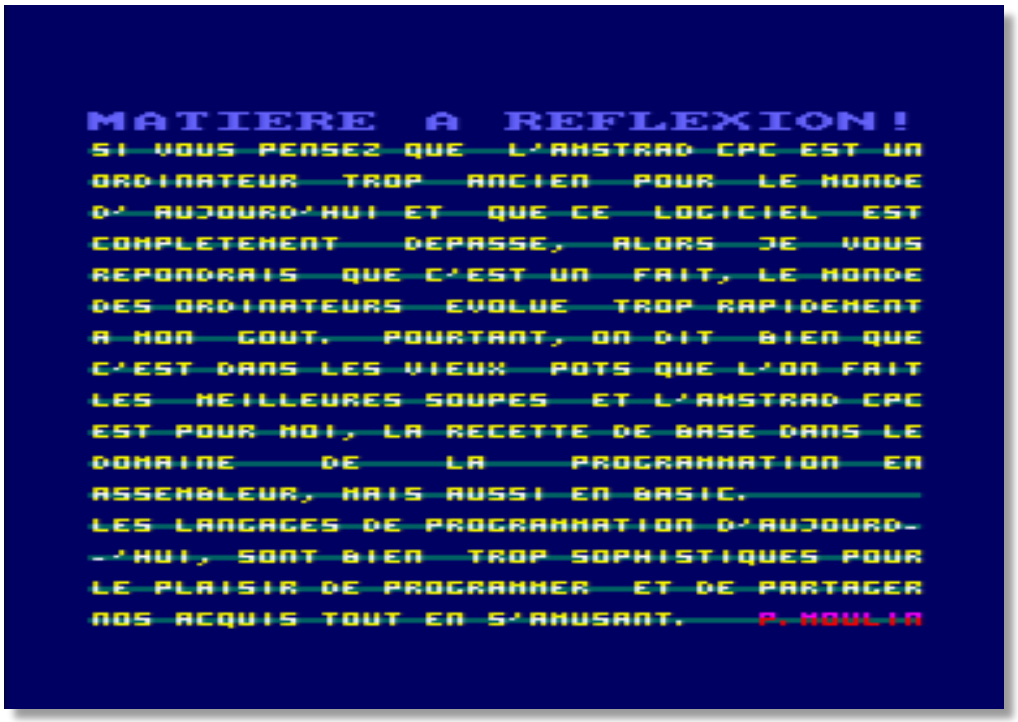
A présent, testons les commandes RSX que nous venons de voir :

```
10 MEMORY &A1FF
20 LOAD"mini2.bin", &A200
30 CALL &A200
40 MODE 0
50 ùENCRE, 1, 2, 2
60 ùPAPIER, 0, 9, 0
70 a$="Petits caracteres, mais maous costauds !"
80 ùCURSEUR, 0, 16
90 ùFONTE, @a$
```

Résultat :



On a fait le tour des commandes RSX de ce tutoriel. Avouez qu'ils ont une grande gueule ces 65 mini-caractères !



SAUVEGARDE ET LECTURE DES MINI-CARACTÈRES

Vous venez de redéfinir tous vos mini-caractères à l'aide de la commande RSX [**ùDEFCAR**] et vous désirez les enregistrer avec les commandes RSX afin de les incorporer dans vos programmes. Alors pas de problème, c'est simple à faire.

Sauvegarde des mini-caractères :

Tapez les 2 lignes ci-dessous en remplaçant le nom "**mini2.bin**" par celui qui vous convient :

```
POKE &A200, &3F
Ready
SAVE"mini2.bin", b, &A200, &239
Ready
```

Pourquoi ce `POKE &A200, &3F` allez-vous me dire ?
Pour plus de sécurité, le programme condamne automatiquement l'accès pour empêcher de relancer le processus d'initialisation des RSX en inscrivant la valeur &C9 à l'emplacement &A200. Il faut donc lui en redonner l'autorisation et la valeur &3F représente l'opcode de l'instruction [**LD A, valeur**] qui a été remplacé par l'opcode &C9 [**RET**] lors du premier lancement d'installation des RSX.

Ne vous inquiétez pas si cela vous semble obscur, on y reviendra au tout début de la prochaine partie concernant la programmation en Assembleur.

La longueur totale du programme est de &239 octets, soit 569 octets seulement, dans lesquels sont compris vos 65 mini-caractères ainsi que toutes les commandes RSX.

Lecture des mini-caractères :

La procédure pour pouvoir inclure vos mini-caractères et les RSX dans vos programmes, est identique à la méthode utilisée au début de l'ouvrage. Remplacez seulement le nom du fichier "**MINI2.BIN**" par celui de votre sauvegarde.

Programme pour inclure les commandes RSX et vos mini-caractères :

```
10 MEMORY &A1FF
20 LOAD"mini2.bin", &A200
30 CALL &A200
```

Cette partie sur les explications des nouvelles commandes RSX est terminée. La prochaine partie concernant le programme en Assembleur est très documentée dans les lignes des routines et aussi dans les explications qui en suivent.

NOTE :

ASSEMBLEUR : NOMS DES COMMANDES RSX

Cette partie « ASSEMBLEUR », sera étudiée par étapes pour que le programme soit plus facile à comprendre. Les explications seront fournies après le code source de chacune des étapes.
NB : Seul le programme dans sa totalité peut être compilé. Les étapes ne servent que pour les explications et les exemples.

Code source : noms donnés aux commandes RSX

ETIQUETTE	INSTRUCTION	COMMENTAIRE
	ORG #A200	; DEBUT DU PROGRAMME SOURCE
FIXE_RSX		; ETIQUETTE DE L'ADRESSE DE DEPART
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE_RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD HL, #BDF7	; ZONE LIBRE DE #BDF7 A #BE3F
	LD BC, ADR_NOM_RSX	; 'BC'=ADRESSE DES NOMS DES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
		;
ADR_NOM_RSX	DW NOM_RSX	; 2 OCTETS PRIS PAR LE VECTEUR #BCD1
	JP RSX_CURSEUR	; SUIVIS DES SAUTS VERS LES ROUTINES
	JP RSX_ENCRE	; DANS L'ORDRE DES NOMS DONNES AUX RSX
	JP RSX_PAPIER	;
	JP RSX_DEFCAR	;
	JP RSX_FONTE	;
	JP RSX_ROULE	;
		;
NOM_RSX		; LISTE DES NOMS DONNES AUX RSXs
	STR "CURSEUR"	; LA DERNIERE LETTRE DES NOMS DOIT
	STR "ENCRE"	; SE TERMINE AVEC LE BIT 7 DE MIS (=1)
	STR "PAPIER"	; HEUREUSEMENT, WINAPE LE FAIT
	STR "DEFCAR"	; AUTOMATIQUEMENT, SINON VOICI DES
	STR "FONTE"	; EXEMPLE1> DEFB "PAPIE", "R" + #80
	STR "ROULE"	; EXEMPLE2> DEFB "ROUL", "E" + #80
	DB #00	; LA LISTE DOIT SE TERMINER PAR UN ZERO

Pour ce tutoriel, les RSX seront implantés en haut de la mémoire, à partir de l'adresse &A200. Rien ne vous empêche de le recompiler à l'adresse de votre choix à condition d'adapter les commandes [**MEMORY**], [**LOAD**] et [**CALL**] du BASIC Locomotive.

Explications de cette routine :

Au début, on empêchera le programme de s'exécuter par plus d'une fois, en plaçant la valeur &C9, qui est l'opcode de l'instruction [**RET**] de l'Assembleur, à cette même adresse d'implantation &A200.

J'ai constaté en effet, que les autres instructions RSX internes comme celles du lecteur de disquettes (**ùA**, **ùB**, **ùDIR** …etc…) faisaient planter l'ordinateur si on exécutait l'implantation de nos RSX par plus d'une fois.
Il est plus simple d'installer un retour au BASIC dès le premier appel. De cette manière, si une instruction [**CALL** **&A200**] est une nouvelle fois exécutée, le programme se redirigera directement vers le BASIC sans en re-exécuter le code. Vous voyez que mon charabia du paragraphe précédent devient moins obscur.

Pour fixer les commandes RSX en mémoire RAM, le vecteur système &BCD1 demande quelques paramètres.
- Le registre '**HL**' doit pointer sur une zone de 4 octets de libre. C'est pourquoi j'utilise l'emplacement d'une zone libre du système qui n'est pas utilisée par le BASIC. Cette zone commence à l'adresse &BDF7 et se termine à l'adresse &BE3F.

ASTUCE :
Vous pouvez aussi utiliser la partie restant de la zone, qui se situe après les 4 octets que l'on vient de réserver, C'est à dire, en partant de l'adresse &BDFB, jusqu'à l'adresse &BE3F incluse. Cette zone mémoire n'est pas effacée lorsque l'on exécute l'instruction [**CALL 0**].

Le vecteur &BCD1 demande aussi de faire pointer le registre '**BC**' sur deux octets contenant l'adresse ou commence les noms des RSX. C'est assez facile à faire, mais assez complexe à expliquer. Le plus simple est de suivre mon modèle ci-dessus.

Tout de suite après ces 2 octets, il faut inscrire l'adresse des sauts vers vos routines puis les noms de vos commandes RSX dans le même ordre que celui de vos sauts.
La valeur zéro inscrite en dernier est aussi obligatoire afin de signaler au vecteur &BCD1 que la liste des noms est terminée.

À propos des noms de vos RSX, pour que le vecteur puisse aussi connaître la fin de chacun des noms donnés, il faut ajouter la valeur de &80 au dernier caractère de chaque nom. La plupart des Assembleurs d'aujourd'hui le font automatiquement, mais je vous donne la solution pour une commande RSX sous le nom de "ROUTINE" :
`DEFB "ROUTIN", "E" + #80` Tout simplement.

Cette première partie est terminée, mais un dernier truc à savoir au sujet des noms donnés aux commandes RSX.
Vous pouvez appeler vos routines depuis le BASIC avec le nom des commandes RSX mais rien ne vous empêche de le faire directement à partir de l'adresse de la routine si vous la connaissez.
Par exemple, l'instruction `CALL &A275, 2, 6` est identique à la commande `|CURSEUR, 2, 6`
Il y a cependant plus de risque de planter l'ordinateur avec l'instruction [**CALL**].

Adresses des routines de ce programme implanté en &A200 :

COMMANDES RSX	ADRESSES ROUTINES
|CURSEUR, Colonne, Ligne	CALL &A275, Colonne, Ligne
|ENCRE, Encre1, Encre2, Encre3	CALL &A297, Encre1, Encre, Encre3
|PAPIER, Papier1, Papier2, Papier3	CALL &A29C, Papier1, Papier2, Papier3
|DEFCAR, caractère, V1, V2, V3	CALL &A2BB, V1, V2, V3
|FONTE, |Chaine$	CALL &A2F0, |Chaine$
|ROULE, |Chaine$	CALL &A361, |Chaine$

ASSEMBLEUR : DÉCLARATION DES VARIABLES

Les variables ne concernent que les couleurs utilisées par les mini-caractères. En les déclarant au début du programme, on comprendra mieux comment fonctionnent les routines RSX.

Code source : déclaration des variables

ETIQUETTE	INSTRUCTION	COMMENTAIRE
ENCRE	DB #40, #40, #40	; EMPLACEMENT DES 3 ENCRES (VALEURS CODEES)
PAPIER	DB #00, #00, #00	; EMPLACEMENT DES 3 PAPIERS (VALEURS CODEES)
	;PEN 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	
COULEUR CODEE	DB #00,#40,#04,#44,#10,#50,#14,#54,#01,#41,#05,#45,#11,#51,#15,#55	

Explications des variables :

Tout cela a l'air très barbare… Mais non, une fois ce système bien compris, ça sera clair comme la météo d'aujourd'hui. Et croyez-moi, si j'ai mis du temps à comprendre le principe du codage des couleurs, avec mes explications, ce sera bien plus facile pour vous.

En premier lieu, on réserve 6 emplacements qui serviront aux routines pour savoir quelle couleur il faudra prendre pour allumer les pixels d'un mini-caractère. Notez pour plus tard, que l'emplacement des papiers, se situe 3 octets plus loin que celui des encres.

En dessous vous avez, (comme commentaire), le numéro des 16 encres du **MODE 0**, et encore en dessous, vous avez leur correspondance en couleurs codées.
Des couleurs codées ? Alors venons-en justement à résoudre ce fameux codage des couleurs du **MODE 0** de l'AMSTRAD CPC.

En **MODE 0**, un octet sur l'écran contient 2 pixels. Si vous tapez : `POKE &C000, &40`. Le pixel droit de l'octet &C000 situé en haut à gauche de l'écran, s'allumera dans la couleur 1 (**PEN 1**).

Cette valeur &40, comme vous pouvez le voir, correspond à l'encre numéro 1.

Écrire `POKE &C000, 1` pour la couleur 1 (**PEN 1**), aurait peut-être été plus facile, mais ce n'est pas aussi simple, comme nous allons le voir un peu plus loin !

Si les valeurs du tableau ci-dessus concernent le pixel de droite de tous les octets de l'écran, alors quelles sont les valeurs des couleurs codées pour le pixel de gauche ?
C'est simple, pour que le pixel s'allume de la couleur 1 du côté gauche d'un octet sur l'écran, il suffit de multiplier sa valeur codée par 2.

Donc, la couleur 1 pour le côté gauche d'un octet sur l'écran, correspond à 2*&40, soit &80.
Essayons : TAPEZ : `POKE &C000, &80` Ça marche, mais étrange non, le pixel de droite s'est effacé !

Et oui, l'adresse &C000 désigne 1 octet et dans le **MODE 0**, 1 octet contient 2 pixels sur l'écran. Si on place une valeur dans cet octet, l'ancienne valeur n'est pas conservée. Ainsi, en plaçant seulement la couleur pour le côté gauche de l'octet, la couleur pour le côté droit du même octet se retrouve avec la valeur zéro.

Mais alors, une question se pose et je suis sûr vous avez déjà la réponse !
Comment faire pour afficher le pixel gauche et le pixel droit de la couleur 1 d'un octet sur l'écran ?
C'est aussi simple que 2 et 2 font 4… Si on additionne le pixel gauche d'une valeur &80 avec le pixel droit d'une valeur &40, les deux pixels de la couleur 1 (PEN 1) s'allumeront !
Essayons de voir ça… Tapotons `POKE &C000, &40 + &80` Et toc, ça marche.

Pour toutes les autres couleurs, c'est le même principe. Afin de vérifier mes dires, essayez d'allumer le pixel gauche de la 3ème couleur (En principe de la couleur rouge si vous n'avez pas utilisé la commande **INK**). La correspondance pour la couleur 3 (**PEN 3**) est &44 (Voir le graphisme des correspondances ci-dessus).

Multiplions par deux cette valeur pour le pixel gauche. Soit 2*&44 = &88.
Puis tapons `POKE &C000, &88` BINGO, en plein dedans, la lumière fuse en rouge.

A vous d'allumer le pixel droit en rouge puis les deux pixels de cet octet, en rouge aussi ! Vous voyez que ce n'est pas la mer à boire.

Cependant, une autre question pourrait alors surgir.
Et si on désire allumer le pixel de gauche dans la 2ème couleur (en bleu) et le pixel de droite dans la 4ème couleur qui est logiquement le blanc ?
C'est encore le même principe et la formule est toujours (2 * couleur codée gauche + couleur codée droite).
Tapons, vite fait, ce petit calcul :

```
BLEU= &04: BLANC= &10
OCTET= 2 *BLEU +BLANC
POKE &C000, OCTET
```

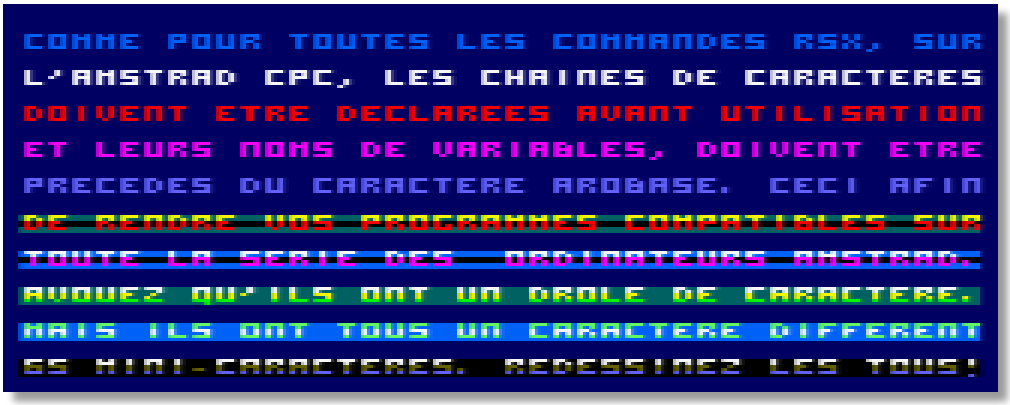
Et voici le résultat :

Pour en revenir à la facilité d'envoyer directement la couleur de l'encre non codée dans un octet sur l'écran en tapant simplement : `POKE &C000, 1` pour l'encre numéro 1.
Vous vous imaginez bien que ce procédé ne pourrait afficher qu'un pixel par octet sur l'écran et que pour afficher le deuxième pixel du même octet, il faudrait multiplier la couleur par le nombre total de couleurs du **MODE 0**. C'est-à-dire par 16.
Le microprocesseur Z80 n'a pas d'instruction pour multiplier directement un chiffre par 16. Ce qui entraînerait des pertes de vitesse d'affichage, car les multiplications par 16 prennent plus de ressources et c'est sûrement pourquoi, à mon avis qu'ils ont choisi cette judicieuse méthode de la multiplication par 2, car il suffit d'une seule instruction Assembleur pour y parvenir.

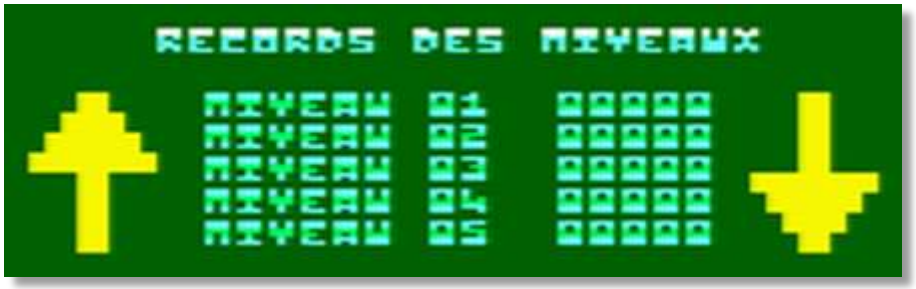
J'espère que le monde mystérieux des couleurs du **MODE 0** de l'Amstrad CPC, n'a plus de secret pour vous !
Pour en conclure avec le codage des couleurs, le **MODE 1** de l'AMSTRAD CPC, a aussi un codage assez spécial que l'on étudiera dans un prochain tuto, si mon temps le permet.

Si vous êtes toujours là, je m'en félicite et je me dis que je n'ai pas expliqué tout ça pour payer les prunes de ma voisine.

Exemple d'utilisation des mini-caractères :



Un autre exemple qui vient de mon jeu « JOSH » pour afficher la liste des records :



L'affichage des deux flèches XXL en jaune ne fait pas partie des commandes RSX de ce tutoriel. C'est une autre routine que j'ai créée uniquement pour mon jeu et qui est un de mes futurs projets de tuto.

NOTE :

ASSEMBLEUR : SOUS-ROUTINE INIT_CARACTERE

Avant de programmer les commandes RSX, on va d'abord créer une sous-routine qui permettra de connaître l'adresse ou est stockée la matrice d'un mini-caractère. La raison en est simple, puisque cette sous routine sera appelée par plusieurs routines RSX, cela fera économiser de la place et rendra la lecture plus facile par la suite.

En premier lieu, regardons les conditions d'entrée '**CE**' et de sortie '**CS**' qu'il nous faudra.

- **CE** : il faut que le registre '**A**' contienne le code ASCII du caractère à rechercher.
- **CS** : le registre '**HL**' contiendra l'adresse de la matrice correspondante au mini-caractère et si le caractère ne figure pas dans la liste des 65 mini-caractères, alors '**HL**' devra pointer sur le premier mini-caractère représentant un ESPACE.

Code source : sous-routine INIT_CARACTERE

ETIQUETTE	INSTRUCTION	COMMENTAIRE
INIT_CARACTERE		;
	SUB A, 32	; ELIMINER LES CARACTERES EN DESSOUS DE 32
	JR C, CARACTERE_ERREUR	; SI LE CARACTERE < ESPACE (32) ALORS ERREUR
	CP A, #41	; REGARDER SI LE CARACTERE EST EN MINUSCULE
	JR C, CARACTERE_OK	; NON ALORS ON POURSUIT
	SUB A, 32	; SINON REAJUSTER LES MINUSCULES EN MAJUSCULES
		;
CARACTERE_OK		;
	CP A, #41	; REGARDER SI LE CARACTERE EST DANS LA LISTE
	JR C, CARACTERE_AJUSTE	; OUI ALORS ON POURSUIT
		; SINON
CARACTERE_ERREUR		; ON AJUSTE LE CARACTERE AU DELA DE LA TABLE
	XOR A	; PAR LE CARACTERE ESPACE
		;
CARACTERE_AJUSTE		; FAIRE POINTER 'HL' SUR LE CARACTERE TROUVE
	LD HL, TABLE_CARACTERES	; 'HL' POINTE SUR LE DEBUT DE LA TABLE
	CP D, #00	; IL FAUT LUI AJOUTER 3* LA VALEUR DU CARACTERE
	LD E, A	; 'DE' CONTIENT LE NUMERO DE CARACTERE
	ADD A, A	; MULTIPLIER PAR 3 OCTETS PAR CARACTERE
	ADD A, E	; 'A' = 'A' * 3
	LD E, A	; 'DE' CONTIENT L'INDEX DU CARACTERE
	ADD HL, DE	; 'HL' POINTE SUR LA POSITION DU CARACTERE
	RET	;

Commençons par le début… Un jour, peut-être, j'essaierai de commencer pas la fin.
Pour bien faire, il faut que le premier mini-caractère qui représente un ESPACE, ait pour index la valeur zéro.
Sachant que d'origine, le code ASCII du caractère qui représente un ESPACE est 32. Il faut donc déduire 32 du registre '**A**' afin de ramener l'index sur le premier mini-caractère.
Mais si le code ASCII envoyé est plus petit que 32, c'est qu'il ne figure pas dans la liste des mini-caractères, et dans ce cas, on initialise sa valeur à zéro pour qu'il corresponde au mini-caractère qui représente un ESPACE.

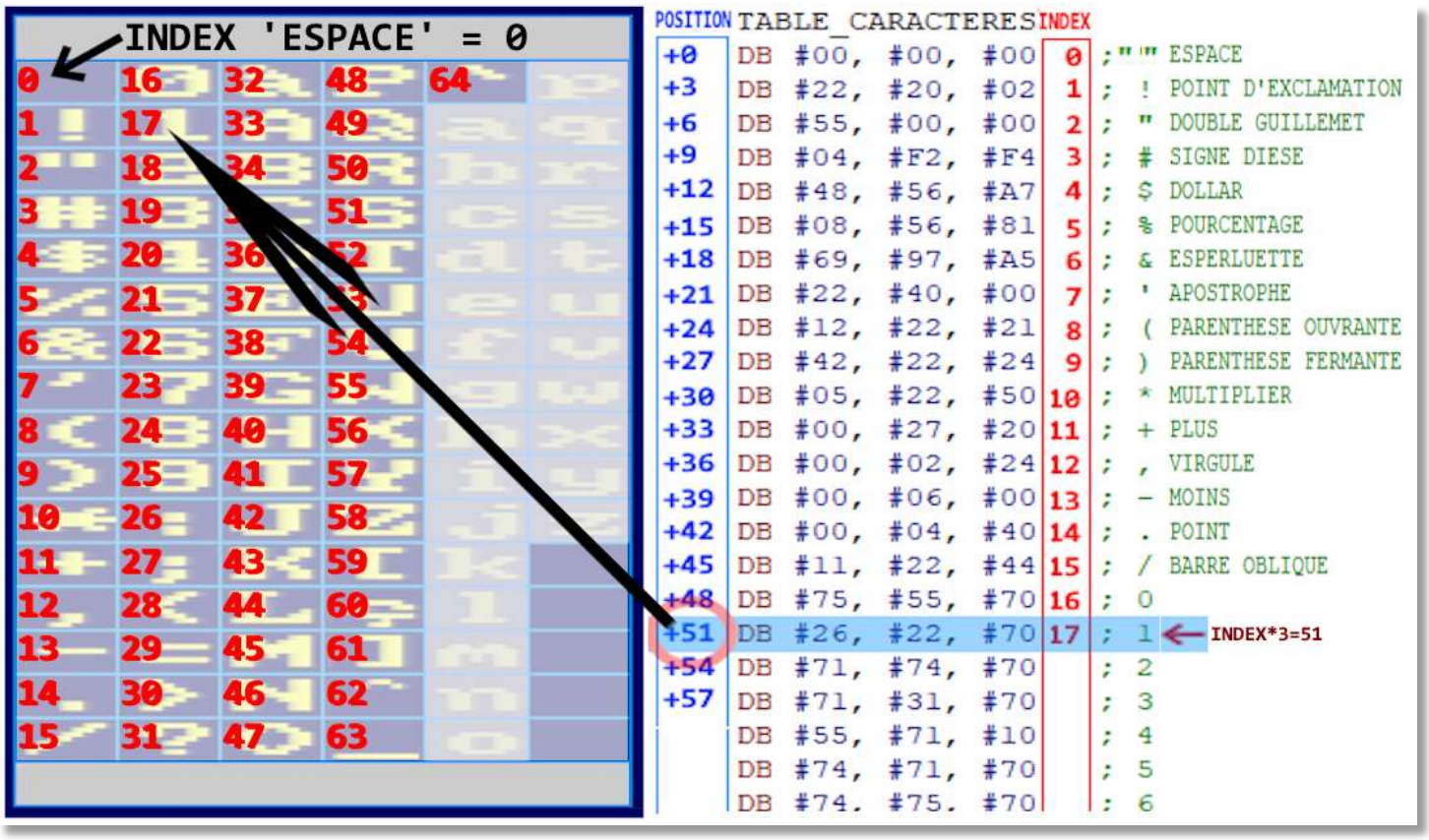
Si après cette déduction, la valeur obtenue est plus grande que 65, soit le nombre total des mini-caractères, c'est qu'il est possible qu'il s'agisse d'un caractère en minuscule. Dans ce cas, il faut déduire 32 de sa valeur, afin de faire pointer l'index des caractères minuscules sur celui des caractères majuscules.
Ainsi le [**a**] pointera sur le [**A**] des mini-caractères, le [**b**] pointera sur le [**B**] etc. … etc. …

Après ça, tous les caractères plus grands que 65, seront initialisés à la valeur zéro car ils sortent de la table des matrices.
On en arrive maintenant à ce que le registre '**A**' contienne l'index correspondant au numéro du mini-caractère.
Comme on l'a vu plus haut, un mini-caractère est composé de 3 octets, on va donc multiplier la valeur obtenue par 3 afin de faire pointer le registre '**A**' sur la matrice correspondante au mini-caractère.

Il ne reste plus qu'à faire pointer le registre '**HL**' sur l'adresse de la table des mini-caractères et d'ajouter l'emplacement de la matrice obtenue. C'est aussi simple que bonjour les crocos !

A la sortie de la routine, le registre '**A**' et le registre '**DE**' sont détruits et le registre '**HL**' pointe sur l'adresse du premier octet du mini-caractère recherché.

EXEMPLE :
Comme exemple, nous allons rechercher l'adresse de la matrice du mini-caractère [**1**].
On sait maintenant que le code ASCII du caractère [**1**] est 49. Donc avant appel de la routine, le registre '**A**' contient la valeur 49 et après l'exécution de l'instruction '**SUB A, 32**' le registre '**A**' aura pour valeur 17.
Les différents tests ne s'appliqueront pas et la routine continuera directement vers l'étiquette [**CARACTERE_AJUSTE**] puisque le registre '**A**' est plus grand que 0 et qu'il est plus petit que 65.
Puis on charge dans le registre '**HL**' l'emplacement de la table des mini-caractères et ensuite il faut multiplier l'index obtenu, par le nombre d'octets composant un mini-caractère.
En additionnant 3 fois le registre '**A**', on obtient finalement, le déplacement de la matrice du mini-caractère [**1**] par rapport à la position 0. Le registre '**A**' contient maintenant la valeur 51.



On utilise le registre '**DE**' pour ajouter le déplacement obtenu avec l'emplacement de la table des mini-caractères car, on ne peut pas additionner un registre de 8 bits avec un registre de 16 bits.
Pour finir, Le registre '**HL**' pointe bien sûr le premier octet du mini-caractère recherché.

ASSEMBLEUR : RSX ùCURSEUR

APPEL : ùCURSEUR, Colonne, Ligne

- Colonne (de 0 à 39) et Ligne (de 0 à 32) représentent les coordonnées ou s'affichera le texte des mini-caractères. Les coordonnées 0, 0 correspondent au coin supérieur gauche de l'écran.

Notes concernant le **MODE 0** des AMSTRAD CPC.

→ En largeur, il y 80 octets qui peuvent contenir 2 couleurs chacun soit 160 pixels de couleur.
→ En hauteur, il y a 200 octets (de 0 à 199) et non pas 400… malheureusement !
→ L'adresse de l'écran débute en &C000 et se termine en &FFFF, soit un total de &4000 octets.

Code source : RSX ùCURSEUR

ETIQUETTE	INSTRUCTION	COMMENTAIRE
RSX_CURSEUR		; ùCURSEUR, Colonne, Ligne
	DEC A	;
	DEC A	; REGARDER S'IL Y A BIEN 2 PARAMETRES
	RET NZ	; SI LE NOMBRE DE PARAMETRE NE CORRESPOND PAS
	LD H, A	; 'H'= 0
	LD A, E	; 'A'= Ligne >>> DERNIER PARAMETRE DANS 'DE'
	CP A, 33	;
	RET NC	; SI Ligne > 32 ALORS ON SORT
	ADD A, A	; x2 IL FAUT MULTIPLIER LES LIGNES PAR 6 POUR
	ADD A, E	; x3 L'AJUSTER AU VECTEUR ECRAN #BC1D
	ADD A, A	; x6 PUIS SOUSTRAIRE 199 AFIN DE FIXER L'INDEX
	ADD A, 56	; EN HAUT DE L'ECRAN.
	CPL	; EQUIVALENT A (255-56)-'A'
	LD L, A	; HL'= AXE_Y
	LD A, (IX + 2)	; 'A' = Colonne
	CP A, 40	; AJUSTER LES COLONNES POUR LE MODE 0
	RET NC	; SI Colonne > 39, ON REPARD VERS LE BASIC
	LD (FIN_DE_COLONNE +1), A	; VARIABLE QUI INDIQURA UNE FIN DE COLONNE
	ADD A, A	; x2 MULTIPIER SA VALEUR PAR 4
	ADD A, A	; x4 POUR OBTENIR L' AXE X DE L'ECRAN
	LD E, A	; 'DE'= AXE_X
	CALL #BC1D	; VECTEUR #BD1D (ADRESSE ECRAN)
	LD (CURSEUR + 1), A	; INSCRIRE LE RESULTAT POUR COMANNDE RSX ùFONTE
	RET	;

A SAVOIR : lorsque l'on exécute une commande RSX depuis le BASIC de l'Amstrad CPC, le registre '**A**' contient le nombre de paramètres envoyés, le registre '**DE**' contient la valeur du dernier paramètre et le registre '**IX**' pointe sur l'adresse du dernier paramètre.

Le vecteur système &BCD1 ne sera utilisé que dans cette routine pour fixer la position de départ d'un mini-caractère sur l'écran. Aucune autre routine n'utilise de vecteurs systèmes et la routine qui affiche les mini-caractères, possède son propre système de calcul de déplacement sur l'écran ce qui accélère énormément l'affichage des mini-caractères par rapport à la version précédente.

Déroulement de la routine :

Pour cette routine, on a besoin de 2 paramètres. En décrémentant par 2 fois le registre '**A**', on lui donne la valeur de zéro et si ce n'est pas le cas, c'est que le nombre de paramètres ne correspond pas. Dans cette condition, on n'exécutera pas la routine et on fera un retour vers le BASIC pour éviter des plantages de l'ordinateur.
Ensuite, on initialise le registre '**H**' avec la valeur du registre '**A**', c'est-à-dire que '**H**'= 0.
On charge le paramètre [Ligne] dans le registre '**A**' et on regarde si sa valeur ne sort pas de l'écran. Sinon, on arrête aussi le processus de la routine en retournant au BASIC.

Sachant que les mini-caractères occupent 6 lignes en hauteur, il faut donc multiplier le registre '**A**' par 6 afin de le faire correspondre à l'axe Y de l'écran demandé par le vecteur &BC1D.
Comme on se base sur des coordonnées haut/gauche de l'écran et que le vecteur &BC1D calcule depuis le bas/gauche de l'écran, il faut donc inverser la valeur en lui ôtant le nombre d'octets en hauteur. Soit la formule (199 – Ligne*6).

Le plus simple aurait été de coder :

```
        LD L, A         ; 'L'= LIGNE
        LD A, 199       ; 'A'= INDEX Y DU HAUT DE L'ECRAN
        SUB A, L        ; 'A'= POSITION Y DES LIGNES
```

Mais j'ai utilisé une astuce plus rapide en exécution et moins gourmande en octets en remplaçant ce code par :

```
        ADD A, 56       ; FIXER L'INDEX EN HAUT DE L'ECRAN
        CPL             ; EQUIVALENT A (255-56)-'A'
```

Tout le monde sait qu'un octet peut contenir une valeur de 0 à 255. En additionnant la valeur manquante pour aller de 0 à 199, soit (255-199) = 56, on obtient l'inverse du résultat attendu. Il suffit alors d'exécuter l'instruction '**CPL**' du Z80 pour obtenir le bon résultat.
Une fois la valeur obtenue, on la charge dans le registre '**L**', ainsi le registre '**HL**' contient le numéro de la ligne demandée par le vecteur système &BC1D.

Ensuite, on charge le paramètre des [Colonne] dans le registre '**A**' et on regarde si sa valeur ne dépasse pas 39, soit le nombre total de mini-caractères dans la largeur de l'écran en partant du 0. Si c'est le cas, on retourne au BASIC sans rien modifier.
Ce paramètre nous servira aussi à compter les colonnes lors de l'affichage d'une chaîne de caractère sur l'écran. Cela permettra de savoir si on ne doit pas passer à la ligne suivante.

On l'inscrit donc comme compteur de colonne pour l'usage de la commande RSX [**ùFONTE**].
Toujours pour le vecteur &BC1D, il faut multiplier les colonnes par 4 afin d'obtenir une valeur comprise entre (0 et 159) correspondant aux 160 pixels des 80 octets en largeur de l'écran.

Une fois que le registre '**HL**' contient les lignes et que le registre '**DE**' contient les colonnes, on appelle le vecteur système &BC1D qui renvoie dans le registre '**HL**', l'adresse sur l'écran, correspondante aux coordonnées fournies. Pour finir, on inscrit la position ainsi obtenue, dans l'emplacement qui servira à positionner le premier mini-caractère à l'usage de la commande RSX [**ùFONTE**].

ASSEMBLEUR : RSX ùENCRE ET ùPAPIER

APPEL : ùENCRE, Encre1 [, Encre2] [, Encre3]
APPEL : ùPAPIER, Papier1 [, Papier2] [, Papier3]

- Sélectionne les couleurs pour afficher les mini-caractères. Chacun des paramètres prend une valeur entière de 0 à 15.
- Seul le premier paramètre est obligatoire. Les deux autres sont optionnels, mais s'ils font de l'absentéisme, leurs valeurs ne seront pas modifiées.

Ces deux RSX fonctionnent de la même manière et il n'y a que l'emplacement des couleurs qui change. Je les ai regroupés pour éviter d'en faire un doublon.

Code source : RSX ùENCRE & ùPAPIER

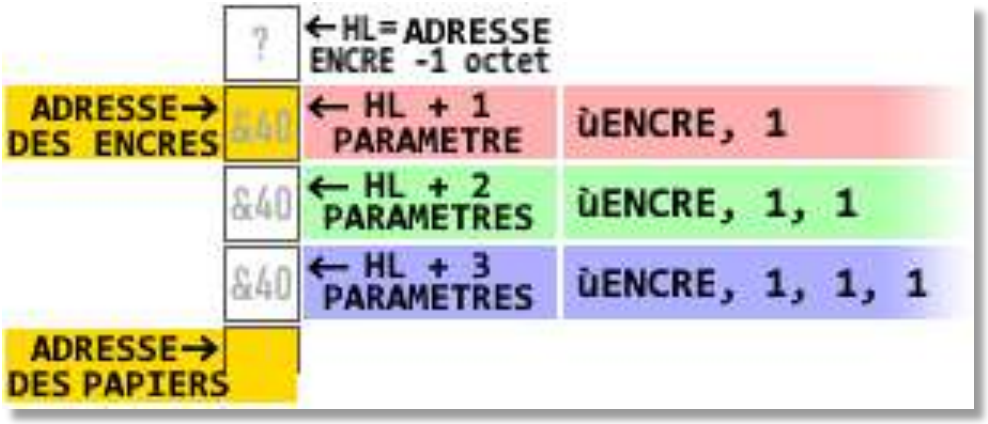
```
ETIQUETTE       INSTRUCTION              COMMENTAIRE
RSX_ENCRE                                ; ùENCRE, Pen1 [, Pen2] [, Pen3]
                LD HL, ENCRE - 1         ; 'HL'= ADRESSE DES ENCRES -1
                JR ENCRE_OK              ; ALLER VERS LA SUITE DE LA ROUTINE
                                         ; ET FERA GAGNER DES OCTETS
RSX_PAPIER                               ; ùPAPIER, Papier1 [, Papier2] [, Papier3]
                LD HL, PAPIER - 1        ; 'HL'= ADRESSE DES PAPIERS -1
                                         ; 'SUIVANT L'APPEL, 'HL' POINTE SUR L'ADRESSE
      ENCRE_OK                           ; DES ENCRES OU DES PAPIERS -1 OCTET
                CP A, 4                  ; SI LE NOMBRE DE PARAMETRE EST >3 ALORS SORTIR
                RET NC                   ;
                OR A                     ; ET SORTIR AUSSI S'IL N'Y A PAS DE PARAMETRE
                RET Z                    ;
                LD B, #00                ; FAIRE POINTER 'HL' SUR LA DERNIERE COULEUR
                LD C, A                  ; 'BC'= NOMBRE DE PARAMETRE DE 1 A 3
                ADD HL, BC               ; 'HL' POINTE SUR LA DERNIERE COULEUR A TRAITER
                EX HL, DE                ; 'DE' =DERNIERE COULEUR (ADR. DESTINATION)
                                         ;
  ENCRE_PAPIER                           ;
                LD HL, COULEUR CODEE     ; 'HL' POINTE SUR L'ADRESSE DES COULEURS CODEES
                LD C, (IX + 0)           ; 'C'= VALEUR PARAMETRES EN PARTANT DU DERNIER
                ADD HL, BC               ; 'HL' POINTE SUR LA COULEUR CODEE A TRANSFERER
                LDD                      ; TRANSFERT LA VALEUR DE 'HL' DANS 'DE'
                INC BC                   ; IL FAUT REMETTRE 'B' A ZERO SUITE A 'LDD'
                INC IX                   ; REMONTER D'UN PARAMETRE EN INCREMENTANT 'IX'
                INC IX                   ; EN DEUX FOIS (CHAQUE PARAMETRE =2 OCTETS)
                DEC A                    ; PUIS DECREMENTER LE COMPTEUR DE PARAMETRE
                JR NZ, ENCRE_PAPIER      ; ET BOUCLER JUSQU'AU DERNIER
                RET                      ;
```

.
Le premier objectif est de faire pointer le registre '**HL**' sur l'emplacement de la dernière couleur (encre ou papier) correspondant au nombre de paramètres envoyés, car le registre '**IX**' pointe toujours sur le dernier paramètre et l'on va traiter les données en partant du dernier paramètre pour revenir jusqu'au premier. Ce qui nous fera gagner quelques octets.

Au début, suivant la commande RSX, on charge soit l'emplacement des encres ou soit celui des papiers dans le registre '**HL**' mais, on lui ôte 1 octet car en lui rajoutant (voir plus loin) le nombre de paramètre, celui-ci pointera sur le dernier emplacement correspondant à la couleur par rapport au nombre de paramètres envoyés.

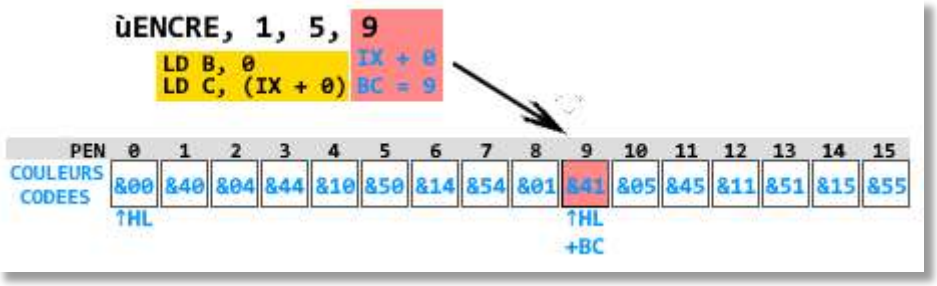
Après, vient le test du nombre de paramètres. On arrête la routine si celui-ci est plus grand que 3, ou égal à 0.
Puis, on additionne le nombre de paramètres envoyés au registre '**HL**'. Et c'est là que l'emplacement du registre '**HL**' sera positionné sur la dernière couleur à traiter.

Exemple avec la commande RSX [**ùENCRE**] :



On place ensuite cette adresse dans le registre '**DE**' car on se servira de l'instruction '**LDD**' qui a l'avantage de transférer le contenu de '**HL**' dans l'emplacement pointé par '**DE**'.

Il ne faut pas oublier que dans le **MODE 0** des AMSTRAD CPC, les couleurs sont codées.
Il faut donc faire pointer le registre '**HL**' sur l'adresse des couleurs codées et de lui rajouter la valeur transmise dans les paramètres pour que ce registre se positionne sur la couleur codée correspondante à la couleur demandée.
La formule idéale du microprocesseur Z80 serait : **ADD HL, (IX + 0)**
Mais comme cette instruction n'existe pas, c'est donc le registre '**BC**' qui servira pour l'addition en chargeant la valeur de 0 dans le registre '**B**' et la valeur de la couleur contenue du paramètre en cours dans le registre '**C**'.



On peut maintenant transférer la couleur codée que contient le registre '**HL**' dans l'emplacement pointé par le registre '**DE**'.
L'instruction '**LDD**' a aussi l'avantage de décrémenter le registre '**DE**' et donc de l'orienter sur sa prochaine destination, puisque l'on traite les couleurs en remontant dans les paramètres.

En programmant cette routine, j'ai commis l'erreur d'oublier que l'instruction '**LDD**' décrémentait aussi le registre '**BC**' et si l'un des paramètres est égal à zéro, alors le registre 'BC' passait de &0000 à &FFFF. Le registre '**B**' n'étant plus égal à zéro, ce qui avait pour conséquence de ne pas obtenir les bonnes couleurs d'affichage. Bref, un simple '**INC BC**' fesse cette erreur…
Puis, on passe du paramètre de droite à celui vers la gauche en incrémentant le registre '**IX**' par deux fois, car chaque paramètre envoyé occupe 2 octets. Et pour finir, on boucle sur la totalité du nombre de paramètres.

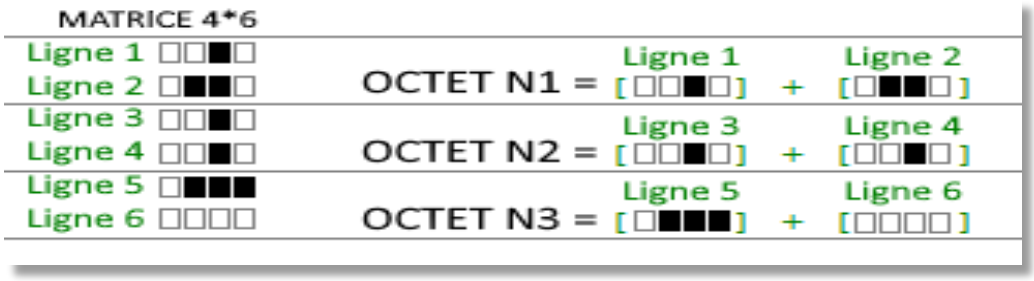
ASSEMBLEUR : RSX ùDEFCAR

APPEL : ùDEFCAR, Numéro du caractère, V1, V2, V3, V4, V5, V6
APPEL : ùDEFCAR, Numéro du caractère, Val1, Val2, Val3

- Redéfinie la forme d'un mini-caractère.
- Numéro du caractère correspond au code ASCII du mini-caractère à redéfinir.
- V1, V2, V3, V4, V5 et V6 (De 0 à 15) représentent les 6 lignes du mini-caractère.
- Val1, Val2 et Val3 (De 0 à 255) représentent les 6 lignes sur 3 octets du mini-caractère.

Cette commande est similaire à la commande [**SYMBOL**] du BASIC LOCOMOTIVE, mais concerne uniquement les mini-caractères.
Pour les exemples à suivre, on utilisera souvent le caractère [**1**] dont son code ASCII est 49.

Graphisme du mini-caractère [1].



Un mini-caractère est composé de 6 lignes de 4 pixels. Soit 6*4 = 24 pixels.
Un octet est composé de 8 BITs. Il nous faudra donc 3 octets pour faire rentrer les 24 pixels.
Le premier octet contiendra les 2 premières lignes. Le second octet contiendra la ligne 3 et la ligne 4. Le troisième octet contiendra les 2 dernières lignes.

Pour simplifier les choses, il y a deux façons d'utiliser cette commande RSX.
- De ligne par ligne sur une valeur de 4 BITs pour un total de 7 paramètres.
- Ou par 2 lignes à la fois sur une valeur de 8 BITs pour un total de 4 paramètres.

Ainsi, les deux commandes suivantes sont identiques :
ùDEFCAR, 49, &2, &6, &2, &2, &7, &0
ùDEFCAR, 49, &26, &22, &70

NB : n'oubliez pas d'utiliser la fonction **ASC("caractère")** du BASIC, au cas où vous ne connaissez pas le code ASCII du caractère à redéfinir.

Code source : RSX ùDEFCAR

ETIQUETTE	INSTRUCTION	COMMENTAIRE
RSX_DEFCAR		; ùDEFCAR, Numero caractere, Val1, Val2, Val3
		; ùDEFCAR, Numero caractere, V1, V2, V3, V4, V5, V6
	LD C, A	; 'C' SERA UTILISE POUR TRAITER LES VALEURS
	CP A, 4	; FAIRE POINTER 'HL' SUR LE PREMIER PARAMETRE
	JR NZ, SEPT_PARAMETRES	; CORRESPONDANT AU CODE ASCII DU CARACTERE
	LD B, (IX + 6)	; 'B' =CODE ASCII DU MINI-CARACTERE A REDEFINIR
	JR DEFCAR_HL_OK	; SAUTER LE TEST POUR LES 7 PARAMETRES
		; 'SUIVANT L'APPEL, 'HL' POINTE SUR L'ADRESSE
SEPT_PARAMETRES		;
	CP A, 7	; SI LE NB DE PARAMETRES EST DIFFERENT DE 4 OU 7
	RET NZ	; ARRETER LA ROUTINE

```
                    LD B, (IX + 12)          ; 'B' =CODE ASCII DU MINI-CARACTERE A REDEFINIR
                                             ;
    DEFCAR_HL_OK                             ;
                    LD A, B                  ; 'A' =CODE ASCII DU MINI-CARACTERE A REDEFINIR
                    CALL INIT CARACTERE      ; 'HL'=ADRESSE MATRICE DU CARACTERE + 3 OCTETS
                    INC HL                   ; FAIRE POINTER 'HL' SUR LE DERNIER OCTET
                    INC HL                   ; 'HL' POINTE SUR LA FIN DE LA MATRICE DU CARAC
                                             ;
                    CP B, 3                  ; 'B' =BOUCLE DE 3 OCTETS DU MINI-CARACATERE
          DEFCAR B                           ;
                    LD A, (IX + 0)           ; 'A'= VALEUR PARAMETRES EN PARTANT DU DERNIER
                    BIT 1, C                 ; TESTER SI ON A ENVOYER 4 OU 7 PARAMETRES
                    JR Z, DEFCAR_VALEUR_OK   ; SI 4 PARAMETRES, ENVOYER VALEUR DIRECTEMENT
                    LD E, A                  ; SINON ON DOIT TRAITER 2 PARAMETRES DE 4 BITS
                    INC IX                   ; ALLER CHERCHER LA VALEUR 4 BITS DE GAUCHE
                    INC IX                   ; DANS LES PARAMETRES TRANSMIS EN (IX + x)
                    LD A, (IX + 0)           ; POUR EN FAIRE UNE VALEUR DE 8 BITS
                    ADD A, A                 ; x2 EN MULTIPLIANT x16 LE PARAMETRE DE GAUCHE
                    ADD A, A                 ; x4 ET EN ADDITIONNANT LE PARAMETRE DE DROITE
                    ADD A, A                 ; x8
                    ADD A, A                 ; x16
                    ADD A, E                 ; 'A' =VALEUR SUR 8 BITS DE DEUX PARAMETRES
                                             ;
  DEFCAR_VALEUR_OK                           ;
                    LD (HL), A               ; INSCRIRE LA VALEUR OBTENUE DE 8 BITS
                    DEC HL                   ; ET FAIRE POINTER 'HL' SUR UN OCTET EN MOINS
                    INC IX                   ; PUIS INCREMENTER IX DEUX FOIS POUR PASSER
                    INC IX                   ; AU PARAMETRE SUIVANT
                    DJNZ, DEFCAR_B           ; BOUCLER LES 3 OCTETS
                    RET                      ;
```

Dès le début, on utilise le registre '**C**' qui nous servira à identifier si on a 4 ou 7 paramètres pour traiter les valeurs à inscrire. Puis, suivant ce nombre, on charge dans le registre '**B**', le code ASCII du caractère à redéfinir que l'on obtient dans le premier paramètre envoyé (celui qui se situe tout à gauche).
Ce code ASCII est ensuite transmis dans le registre '**A**' afin de trouver l'adresse de la matrice correspondante au mini-caractère que la sous-routine [**INIT_CARACTERE**] chargera dans le registre '**HL**'.

Pour cette routine aussi, on va faire pointer le registre '**HL**' sur le dernier octet du mini-caractère à redéfinir afin de traiter les valeurs dans le même sens que les paramètres transmis dans le registre '**IX**'. Ensuite, il faut faire une boucle des 3 octets à inscrire et regarder si on doit ajuster les valeurs suivant le nombre de paramètres transmis.

Si on envoie 4 paramètres, c'est que les valeurs à transférer sont représentées en 8 BITs et donc il n'y a rien à ajuster. Mais si le nombre de paramètres est de 7, alors il faut ajuster les valeurs de 4 BITs envoyées sur deux paramètres pour n'en faire qu'une d'une valeur de 8 BITs. D'où la formule : (16 * Valeur du paramètre de gauche) + valeur du paramètre de droite.

Après ça, on inscrit la valeur obtenue dans les adresses de la matrice du mini-caractère. On repositionne le registre '**HL**' sur un octet en arrière, et on répète 3 fois l'opération. Ce qui correspond aux 3 octets d'une matrice d'un mini-caractère.

ASSEMBLEUR : RSX ùFONTE

APPEL : a$= "Bonjour d'ici ! " : ùFONTE, àa$

- Affiche la chaîne de caractère d'après les matrices des mini-caractères.
- Les chaînes de caractères doivent être déclarées et précédées d'une arobase avant leurs utilisations dans les paramètres des commandes RSX.

Code source : RSX ùFONTE

ETIQUETTE	INSTRUCTION	COMMENTAIRE
RSX FONTE		; a$ DOIT ETRE DECLARER ET PRECEDE DE [@]
	DEC A	; IL NE FAUT QU'UN SEUL PARAMETRE
	RET NZ	; SINON ON ARRETE LA ROUTINE
	EX HL, DE	; 'HL' CONTIENT LE DESCRIPTEUR DE LA CHAINE
	LD A, (HL)	; 'A'= NB DE CARACTERES DANS LA CHAINE
	OR A	; SI LA CHAINE EST VIDE, ON ARRETE LA ROUTINE
	RET Z	; POUR EVITER LE PLATAGE D'UNE BOUCLE A ZERO
	INC HL	; RECUPERER L'ADRESSE OU SONT INSCRITS LES
	LD E, (HL)	; CARACTERES DE LA CHAINE
	INC HL	;
	LD D, (HL)	; 'DE' CONTIENT L'ADRESSE DU PREMIER CARACTERE
	LD B, A	; 'B' =LONGUEUR_CHAINE ET SERVIRA DE BOUCLE
BOUCLE_CHAINE		;
	LD A, (DE)	; 'A'= CARACTERE DE LA CHAINE
	INC DE	; SERA UTILISE POUR LE CARACT. SUIVANT
	PUSH BC	; SAUVEGARDER LA LONGUEUR DE LA CHAINE
	PUSH DE	; SAUVEGARDER L'EMPLACEMENT DES CARACTERES
	CALL INIT CARACTERE	; 'HL' POINTE SUR LES 3 OCTETS A TRAITER
	LD IY, ENCRE - 1	; 'IY' EST INITIALISE VERS LES ENCRES -1 OCTET
	EX HL, DE	; 'DE'= MATRICE 4x6 DU CARACTERE A TRAITER
CURSEUR	LD HL, #C000	; 'HL'= EMPLACEMEMNT OU AFFICHER LE CARACTERE
	PUSH HL	;
		;
	LD B, 6	; 6 LIGNES A AFFICHER PAR CARACTERE
SIX_LIGNES		;
	BIT 0, B	; 1 OCTET CONTIENT 8 BITS ET LA MATRICE EN A 4
	JR NZ, COULEUR OK	; IL FAUT DONC FAIRE 2 LIGNES POUR LIRE 1 OCTET
	LD A, (DE)	; ET AUSSI POUR CHANGER LES COULEURS.
	INC DE	; LE CHIFFRE PAIRE DE 'B' PERMETTRA CE CONTROLE
	LD C, A	; 'C'= OCTETS DU CARACTERE (1 OCTET=2 LIGNE)
	INC IY	; 'IY' POINTE SUR LES ENCRES
		;
COULEUR_OK		;
	PUSH DE	;
	LD E, (IY + 0)	; 'E' CONTIENT COULEUR DE L'ENCRE
	LD D, 2	; COMPTEUR DE 2 OCTETS (UNE LIGNE DE 4 PIXELS)
		;
LIGNE_2OCTETS		;
	LD A, (IY + 3)	; 'A'= COULEUR DU PAPIER
	SLA C	; ROTATION [Carry]<- [76543210]<- [0]
	JR NC, PAPIERG_A_GAUCHE	; SI LE DERNIER BIT EST =0, ON GARDE LE PAPIER
	LD A, E	; SINON 'A'= COULEUR DE L'ENCRE
		;

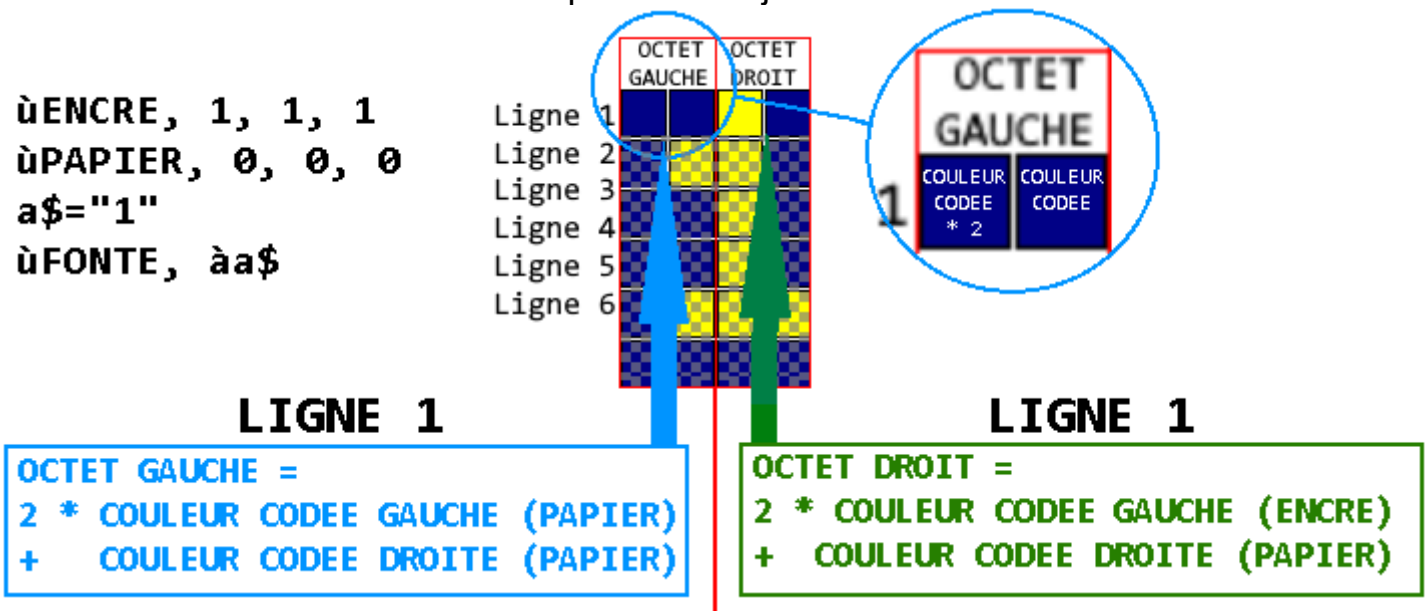
```
PAPIERG_A_GAUCHE INC IX                    ; 'A'= COULEUR CODEE DU PIXEL
                 ADD A, A                  ; IL FAUT *2 POUR LE PIXEL DE GAUCHE
                 SLA C                     ; FAIRE DE MEME POUR LE PIXEL DE DROITE
                 JR NC, PAPIERG_A_DROITE   ; 'A'= PIXEL DE GAUCHE *2
                 ADD A, E                  ; SI BIT=1,  'A'= PIXEL DE GAUCHE *2 + ENCRE
                 JR OCTET_OK               ; ON A LES DEUX COULEURS POUR L'OCTET ECRAN
                 
PAPIERG_A_DROITE                           ; SAUF SI LE BIT ETAIT A ZERO
                 ADD A, (IY + 3)           ; 'A'= PIXEL DE GAUCHE *2 + PAPIER
                                           ;
       OCTET_OK                            ; ON A LES DEUX COULEURS POUR L'OCTET ECRAN
                 LD (HL), A                ; INSCRIRE L'OCTET SUR L'ECRAN
                 INC HL                    ; ET PASSER AU 2EME OCTET DU CARACTERE
                 DEC D                     ;
                 JR NZ, LIGNE_2OCTETS      ;
                                           ; UNE LIGNE DE 2 OCTETS EST FINIE, IL FAUT
                 LD DE, #800 - 2           ; DESCENDRE D'UNE LIGNE SUR L'ECRAN - 2 OCTETS
                 ADD HL, DE                ; POUR DESCENDRE D'1 LIGNE, AJOUTER #800 A 'HL'
                 JR NC, LIGNE_2OCTETS      ;
                 LD DE, #C050              ; MAIS SI ON TOMBE SUR UNE CASSURE DES 8 LIGNES
                 ADD HL, DE                ; IL FAUT RECTIFIER LA VALEUR EN AJOUTANT #C050
                                           ;
 CORRIGE_ADR_ECR                           ;
                 POP DE                    ; RECUPERER L'ADRESSE DE LA CHAINE
                 DJNZ, SIX_LIGNES          ; FAIRE LES 6 LIGNES QUI COMPOSENT LE CARACTERE
                 POP HL                    ; IL FAUT DEPLACER LE CURSEUR D'UNE COLONNE
                 INC HL                    ; RIEN DE PLUS SIMPLE, UN CARACTERE OCCUPE
                 INC HL                    ; 2 OCTETS. MAIS..
                                           ; S1 ON ARRIVE A DROITE DE L'ECRAN, IL   FAUT
  FIN_DE_COLONNE LD A, #00                 ; DESCENDRE DE 6 LIGNES ET REVENIR A GAUCHE
                 INC A                     ; J'UTILISE UNE VARIABLE FIN_DE_COLONNE  AFIN
                 CP A, 40                  ; D'EFFECTUER CE CONTROLE ET SI 'A' > 39 ALORS
                 JR C, COLONNE_OK          ; IL FAUT EXECUTER LA ROUTINE, SINON ON CONTINU
                 LD DE, #2FB0              ; POUR DESCENDRE D'UN CARACTERE ET REVENIR A
                 ADD HL, DE                ; GAUCHE DE L'ECRAN, LA FORMULE EST (6*#800)-80
                 JR NC, CORRIGE_COLONNE    ; (HAUTEUR 6 * #800)-(LARGEUR 2*40)= #2FB0
                 LD DE, #C050              ; MAIS, EN PARTANT DU HAUT DE L'ECRAN, TOUTES
                 ADD HL, DE                ; LES 8 LIGNES IL Y A UNE CASSURE QU'IL FAUT
                                           ; REAJUSTER POUR SE RETROUVER 1 LIGNE PLUS BAS
 CORRIGE_COLONNE                           ;
                 XOR A                     ; NE PAS OUBLIER DE REMETTRE LA VARIABLE
                                           ; FIN_DE_COLONNE A ZERO
      COLONNE_OK                           ;
                 LD (FIN_DE_COLONNE +1), A ; INSCRIRE L'INDICATEUR DES COLONNES
                 LD (CURSEUR + 1), HL      ; ET LA POSITION SUR L'ECRAN POUR LE PROCHAIN
                 POP DE                    ; CARACTERE. RECUPERER L'ADRESSE DE LA CHAINE
                 POP BC                    ; ET SA LONGUEUR AUSSI
                 DJNZ, BOUCLE_CHAINE       ; TERMINER L'AFFICHAGE DE TOUTS LES CARACTERES
                 RET
```

Après les contrôles de sécurité qui doivent vous être familiers maintenant, on fait pointer le registre '**DE**' sur l'emplacement du premier caractère de la chaîne envoyée en paramètre, tandis que la longueur de la chaîne est chargée dans le registre '**B**'.
On utilise une étiquette [**BOUCLE_CHAINE**] pour décompter les caractères de la chaîne au fur et à mesure qu'ils seront affichés sur l'écran.

Ensuite on fait pointer le registre '**HL**' sur l'emplacement de la matrice du mini-caractère correspondant à l'aide de la sous-routine [**INIT_CARACTERE**] vue précédemment.
Le registre '**IY**' est utilisé pour sélectionner les couleurs des pixels du mini-caractère à afficher. Celui-ci sera incrémenté toutes les deux lignes afin d'en changer les couleurs.
Notez que l'adresse des couleurs des papiers est positionnée 3 octets plus loin que l'adresse des encres comme on l'a vu précédemment.
Ainsi, avec le code **(IY + 0)** on obtiendra l'encre et avec le code **(IY + 3)** on obtiendra le papier des 2 lignes du mini-caractère en cours et toutes les 2 lignes affichées, le registre '**IY**') sera incrémenté de façon à changer de couleur.

La valeur située à l'emplacement de l'étiquette [**CURSEUR**] est automodifiable et sert à connaître la position sur l'écran pour afficher le mini-caractère.

Il y a 6 lignes à faire. Chacune des lignes sont composées de 4 pixels. Un octet sur l'écran en **MODE 0** contient la valeur de 2 pixels.
Il faut donc faire une boucle pour les 6 lignes et une boucle pour afficher côte à côte sur l'écran, les 2 octets correspondant à 4 BITs d'une ligne de la matrice.
Si 1 des 4 BITs d'une ligne d'un mini-caractère est égal à zéro, alors on sélectionnera la couleur du papier et bien sûr, s'il est égal à un, on sélectionnera la couleur de l'encre.
.
Chaque octet sur l'écran, contient une couleur codée à gauche et il faudra multiplier sa valeur par deux et une couleur codée à droite qu'il faudra ajouter.



RAPPEL : (voir correspondance des couleurs codées. Page 12)

- (**PEN 0**)=0
- (**PEN 1**)=&40

L'octet situé à gauche aura pour valeur : (2 * papier) + papier, soit (2*0) + 0 = 0
L'octet situé à droite aura pour valeur (2 * encre) + papier, soit (2*&40) + 0 = &80

Une fois cette représentation bien assimilée, le reste du programme devient plus simple à comprendre.

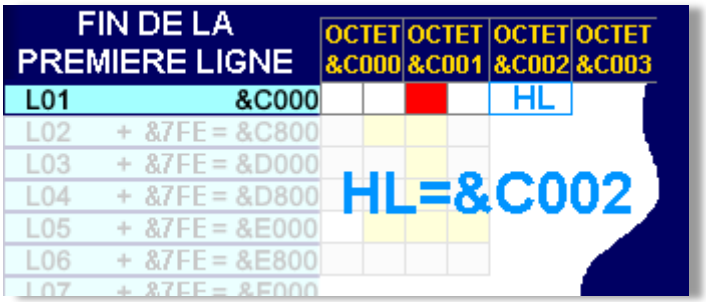
On commence par faire une boucle pour afficher les 6 lignes du mini-caractère. Le registre '**B**' étant le compteur de lignes, à chaque fois que sa valeur sera un chiffre pair, on placera dans le registre '**C**', l'octet qui correspond à 2 lignes de la matrice d'un mini-caractère et on incrémentera le registre '**IY**' pour passer d'une couleur à l'autre. Si la valeur du registre '**B**' est un chiffre impair, alors on sautera directement à l'étiquette [**COULEUR_OK**] puisque cela

signifiera qu'il faut garder les mêmes couleurs des encres et papiers et que le registre '**C**' contient encore des données non traitées.
Maintenant que le registre '**C**' contient les 8 BITs de deux lignes. C'est lui que l'on va faire tourner pour savoir la couleur (encre ou papier) que l'on devra utiliser. Comme on affiche les pixels de gauche vers la droite, il est logique que le BIT à tester de ce registre soit celui de gauche.
Justement, l'instruction [**SLA C**] provoque une rotation vers la gauche du registre '**C**' et place la valeur du BIT 7 (Le plus à gauche) de ce registre, dans le **Carry** « Bit 0 du registre '**F**' ». Ce qui permettra de sélectionner l'encre si la valeur du **Carry** vaut 1, sinon ce sera le papier.

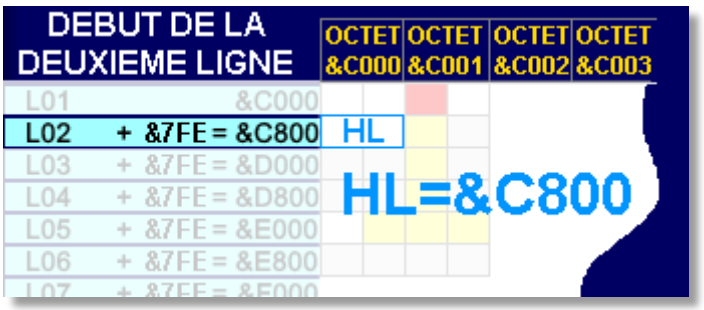
Pour sélectionner la couleur qui sera affectée en rapport aux BITs du registre '**C**', on place la couleur de l'encre dans le registre '**D**' et la couleur du papier dans le registre '**A**'. Si le **Carry** est égal à zéro, alors on garde le papier, sinon on charge le registre '**A**' avec l'encre.
Comme c'est la couleur du côté gauche, il faut la multiplier par deux. Après, on refait une rotation du registre '**C**' pour connaître la couleur qui ira à droite, sauf que l'on additionne soit l'encre, si le **Carry** vaut 1, soit le papier, si le **Carry** vaut zéro.

Une fois que cela est terminé, on affiche l'octet des deux couleurs obtenues sur l'écran, on décale la position sur l'écran d'1 octet sur la droite et on refait pareil pour le second octet de la ligne.

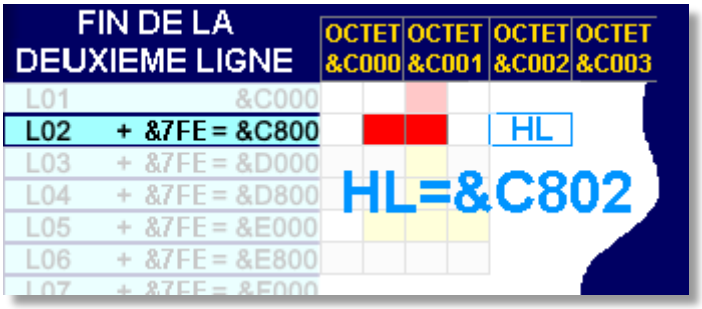
Lorsque-que la ligne est terminée, on doit positionner le registre '**HL**' sur l'emplacement écran d'une ligne en dessous et revenir de 2 octets en arrière correspondant à la ligne qui vient d'être finie. Dans l'exemple suivant, le mini-caractère [**1**] sera positionné en haut et à gauche de l'écran (ùCURSEUR, 0, 0).



La formule pour calculer le passage de ligne est donc : (position actuelle) + &800 – 2 octets.
On utilisera le registre '**DE**' pour effectuer cette opération, en le chargeant avec la valeur (&800 – 2), soit &7FE.



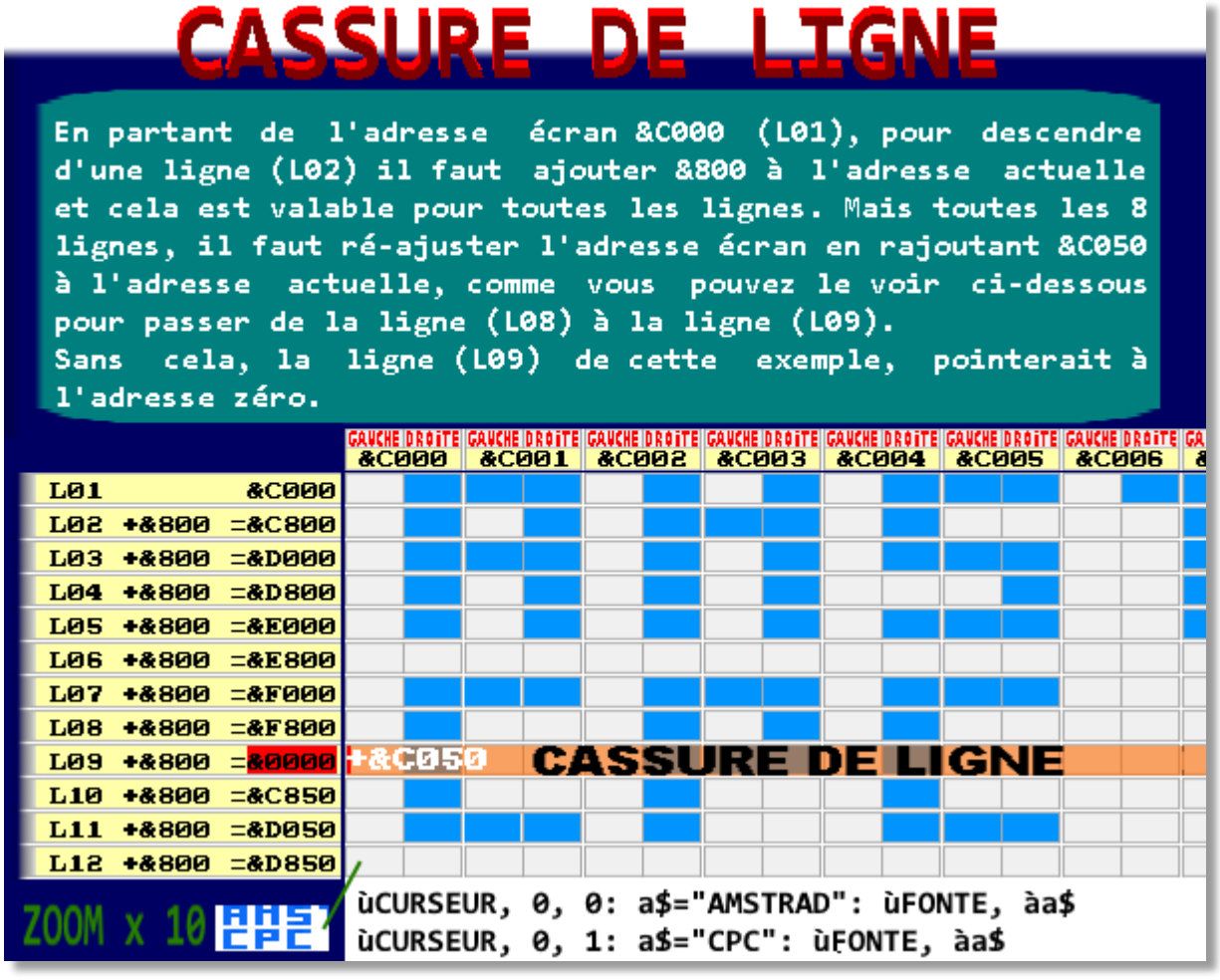
On procédera de la même manière pour le restant des lignes en ajoutant &7FE au registre '**HL**' à chaque fois qu'une ligne sera terminée.



IMPORTANT : en additionnant cette valeur avec la position actuelle, on peut se trouver dans une cassure de ligne… Mais qu'est-ce que c'est, qu'une cassure de ligne, allez-vous me demander ?
Ce que j'appelle une cassure de ligne, c'est lors-ce qu'il faut réajuster la valeur de l'adresse écran quand celle-ci pointe en dehors de l'écran. Ce qui provoquerait un plantage de l'ordinateur à coup sûr.

Examinez bien le schéma ci-dessous, et vous comprendrez que lorsqu'on désirera afficher un mini-caractère entre deux lignes multiples de huit, qu'il nous faudra recalculer l'adresse écran pour ne pas envoyer les données à une adresse autre que l'écran.

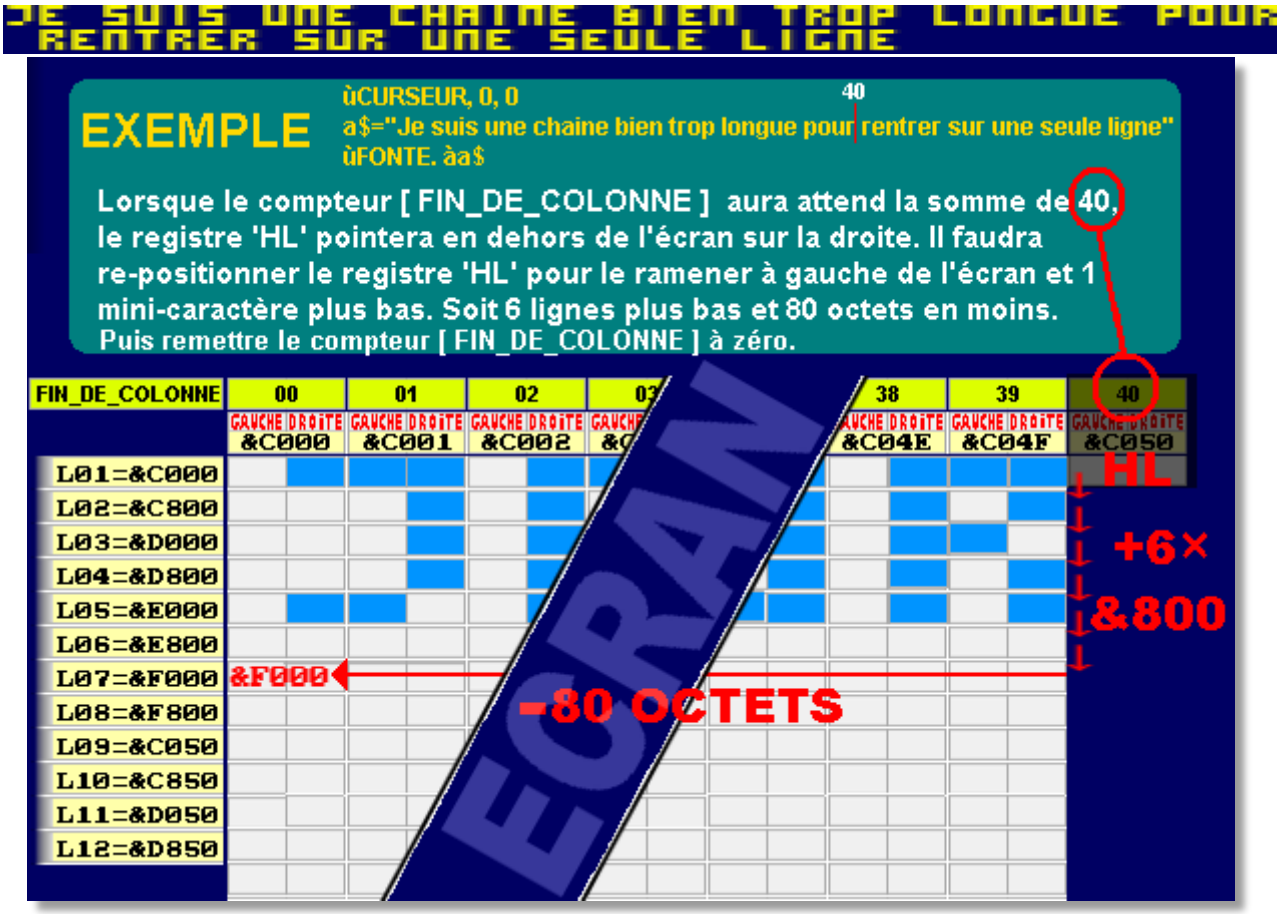
Ci-dessous, la chaîne de mini-caractères "**AMSTRAD**" est affichée dans le **MODE 0**, et en haut à gauche de l'écran puis en dessous, c'est la chaîne de mini-caractères "**CPC**" qui est affichée. Le zoom permet de bien situer la cassure qu'il faudra réajuster après la 8ème ligne.



Pour savoir s'il y a une cassure de ligne, on teste tout simplement le **Carry** qui signalera un débordement après l'addition des registres '**HL**' et '**DE**' et dans ce cas, on fait une autre

addition en chargeant la valeur &C050 dans le registre '**DE**' afin de replacer la position du registre '**HL**' sur l'écran et juste une ligne en dessous. Comme le montre l'exemple pour passer de la ligne L08 à la ligne L09 (de &F800 on passe en &C050).
L'ajout de la valeur &C050 à l'adresse écran actuelle, sera effectué toutes les fois où l'on tombera sur le numéro d'une ligne écran multiple de 8.
A la suite de tous les calculs, il ne reste qu'à boucler sur l'affichage des 6 lignes du mini-caractère.

Une fois que le mini-caractère est complètement constitué, pour passer au caractère suivant de la chaîne, il faut repositionner le curseur d'une colonne (2 octets) à droite et recommencer la procédure jusqu'à terminer tous les caractères de la chaîne.
Le compteur de caractère [**FIN_DE_COLONNE**] qui s'incrémente après l'affichage d'un mini-caractère, permet de savoir si le curseur arrive tout à droite de l'écran. Si sa valeur attend la somme de 40 (colonnes), c'est qu'il faudra descendre de 6 lignes et revenir tout à gauche de l'écran.



La formule est simple : (6 lignes * &800) – (les 80 octets pour revenir sur la gauche) = &2FB0.
C'est le registre '**DE**' qui assurera l'addition mais là encore, il faut se sécuriser en réajustant l'adresse écran si au cas où l'on se retrouve au milieu d'une cassure de ligne et Il ne faut pas non plus oublier de remettre la valeur du compteur [**FIN_DE_COLONNE**] à zéro pour indiquer que l'on se trouve tout à gauche sur l'écran.

Pour terminer cette routine, on mémorise la nouvelle position du curseur pour l'affichage du prochain mini-caractère, et il ne reste plus qu'à procéder de la même manière pour tous les autres caractères contenus dans la chaîne.

ASSEMBLEUR : RSX ùROULE

APPEL : a$= "Bonjour d'ici ! " : ùROULE, àa$

- Exécute une rotation d'un caractère de la droite vers la gauche à n'importe quelle chaîne de caractère.
- Les chaînes de caractères doivent être déclarées et précédées d'une arobase avant leurs utilisations dans les paramètres des commandes RSX.

NB : Si on affiche la chaîne de caractère toujours au même endroit sur l'écran, à l'aide de la commande RSX [**ùFONTE**] ou avec un simple **PRINT** du BASIC, on obtiendra un texte défilant, comme dans de nombreux jeux ou démo sur l'AMSTRAD CPC.

EXEMPLE :

```
10 MEMORY &A1FF
20 LOAD"mini2.bin", &A200
30 CALL &A200
40 a$="Bonjour le monde "
50 MODE 0
60 FOR a=1 TO 12
70 ùENCRE, a, a, a
80 ùCURSEUR, 12, a +5
90 ùFONTE, @a$
100 ùROULE, @a$
110 NEXT
```

Résultat :

Code source de la routine :

ETIQUETTE	INSTRUCTION	COMMENTAIRE
RSX FONTE		; a$ DOIT ETRE DECLARER ET PRECEDE DE [@]
	DEC A	; TESTER S'IL Y A BIEN 1 SEUL PARAMETRE
	RET NZ	; SINON, ON SORT SANS RIEN MODIFIER
	LD B, A	; 'B'= 0
	EX HL, DE	; 'HL'= DESCRIPTEUR DE LA CHAINE
	LD A, (HL)	; 'A'= NB DE CARACTERES DANS LA CHAINE
	OR A	; SI LA LONGUEUR EST NULLE, ON SORT AUSSI
	RET Z	;

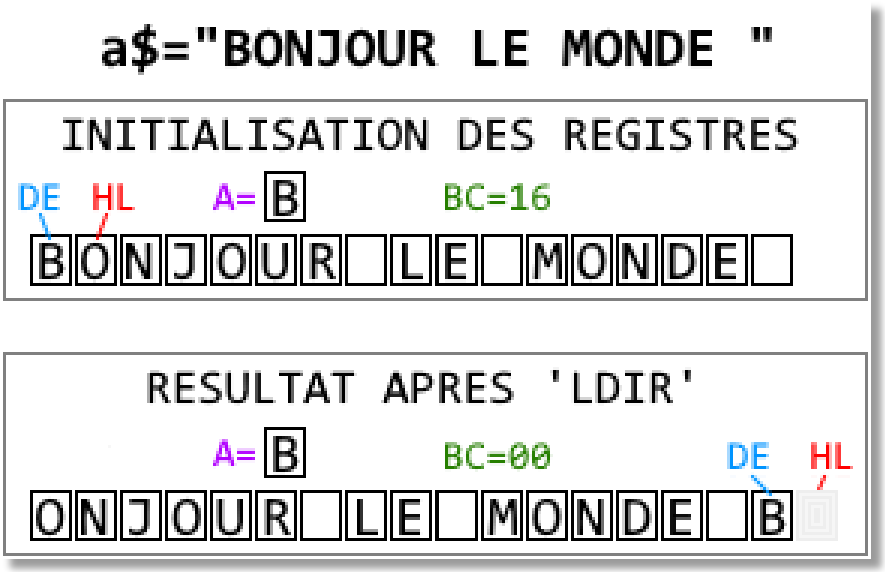
```
        INC HL          ; FAIRE POINTER 'DE' SUR LE 1er CARACTERE
        LD E, (HL)      ;
        INC HL          ;
        LD D, (HL)      ; 'DE' POINTE SUR LE PREMIER CARACTERE
        LD C, A         ; 'BC'= NOMBRE DE CARACTERE DE LA CHAINE
        DEC C           ; -1 CARACTERE
        LD A, (DE)      ; 'A'= CODE ASCII DU 1er CARACTERE DE LA CHAINE
        LD H, D         ; FAIRE POINTER 'HL' SUR LE 2eme CARACTERE
        LD L, E         ;
        INC HL          ; 'HL' POINTE SUR LE DEUXIEME CARACTERE
        LDIR            ; DEPLACE TOUS LES CARACTERES SAUF LE PREMIER
        LD (DE), HL     ; LE PREMIER CARACTERE VA DANS LE DERNIER
        RET             ;
```

Je ne vais pas expliquer les tests effectués au début qui devraient être compréhensibles maintenant.
Ce qu'il nous aurait fallu pour bien simplifier cette commande RSX, c'est la même fonction que l'instruction [**LRC/LRCA**] du Z80, mais étudiée pour les chaînes de caractères, car le principe en est identique. On doit donc créer une routine pour combler ce manque, et pour cela, on utilisera l'instruction de copie [**LDIR**] du Z80 qui effectue une copie du contenu du registre '**HL**' pour le coller dans l'emplacement du registre '**DE'** du nombre d'octets contenus dans le registre '**BC**'.

Le principe du programme est assez simple :

→ 1 - Mémoriser le premier caractère de la chaîne.
→ 2 - Décaler tous les autres caractères vers la gauche.
→ 3 - Copier le caractère mémorisé dans l'emplacement du dernier caractère.

Voici un schéma avant et après que l'instruction '**LDIR**' soit exécutée :



Une fois que le registre '**DE**' pointe sur l'adresse du premier caractère de la chaîne et que le registre '**BC**' contient le nombre de caractères qu'il faudra déplacer, on mémorise dans le registre '**A**' le code ASCII du premier caractère.
Ensuite, on positionne le registre '**HL**' sur le deuxième caractère (Adresse source) et on effectue le copier/coller de tous les caractères restants. Pour finir, on remplace le dernier caractère de la chaîne, par celui qui a été mémorisé dans le registre '**A**'.

Pour finir le programme des commandes RSX, voici les matrices de 65 mini-caractères provisoires, qui n'attendent que votre talent pour être redessiné.

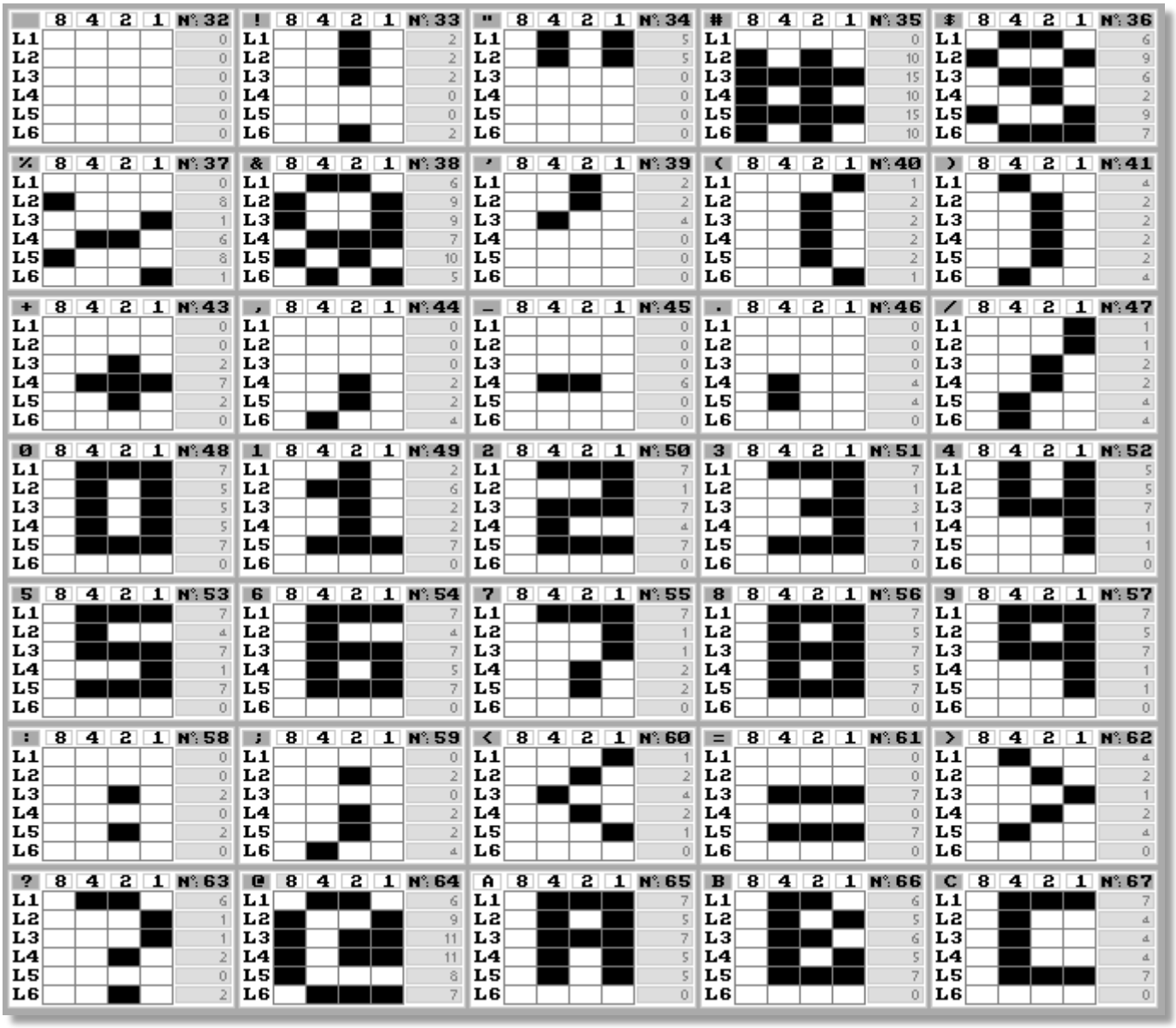
TABLE DES 65 MINI-CARACTÈRES

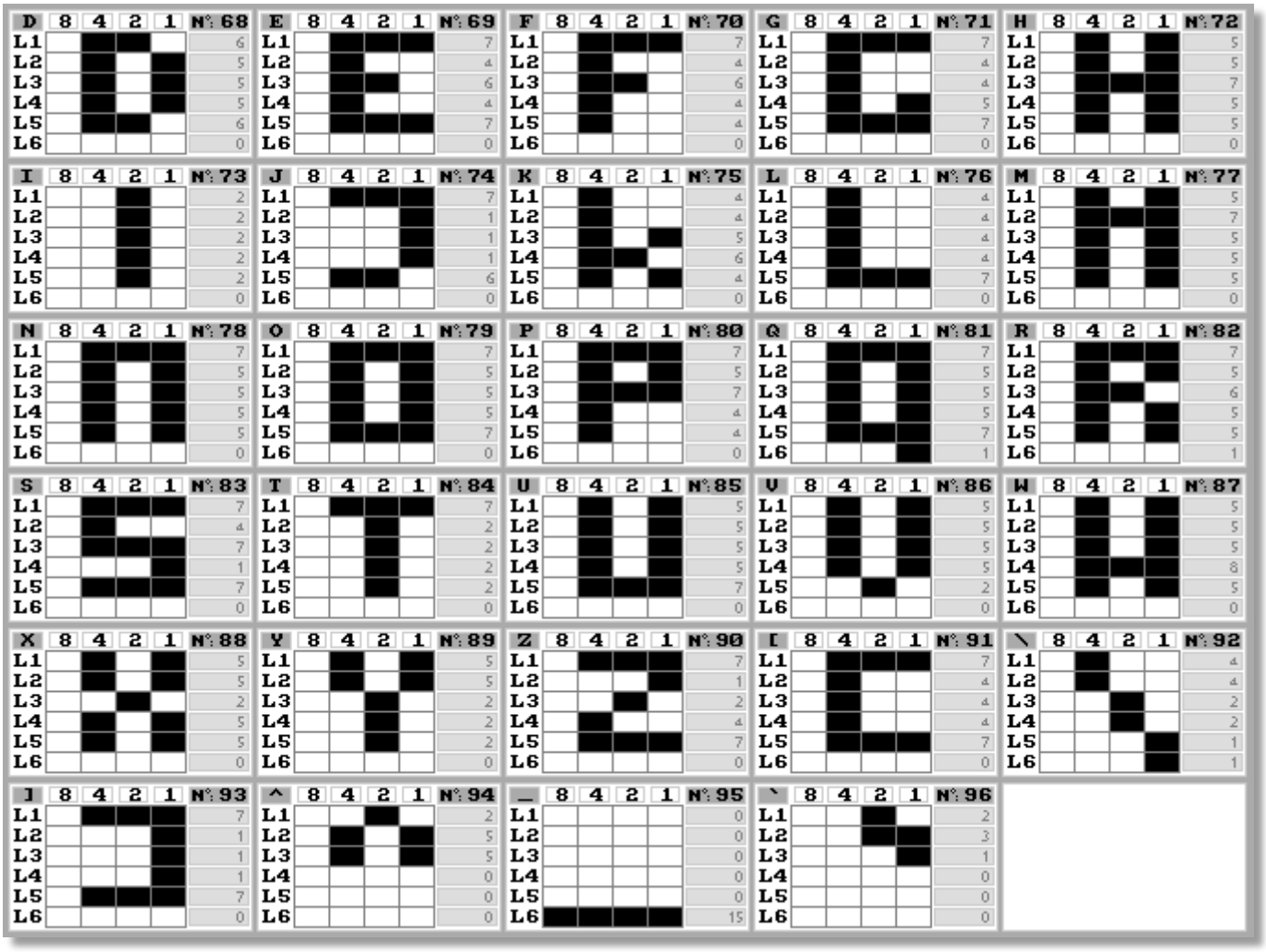
ETIQUETTE	INSTRUCTION	COMMENTAIRE
TABLE_CARACTERES		
	DB #00, #00, #00	; " " ESPACE
	DB #22, #20, #02	; ! POINT D'EXCLAMATION
	DB #55, #00, #00	; " DOUBLE GUILLEMET
	DB #0A, #FA, #FA	; # SIGNE DIESE
	DB #69, #62, #97	; $ DOLLAR
	DB #08, #16, #81	; % POURCENTAGE
	DB #69, #97, #A5	; & ESPERLUETTE
	DB #22, #40, #00	; ' APOSTROPHE
	DB #12, #22, #21	; (PARENTHESE OUVRANTE
	DB #42, #22, #24	;) PARENTHESE FERMANTE
	DB #05, #22, #50	; * MULTIPLIER
	DB #00, #27, #20	; + PLUS
	DB #00, #02, #24	; , VIRGULE
	DB #00, #06, #00	; - MOINS
	DB #00, #04, #40	; . POINT
	DB #11, #22, #44	; / BARRE OBLIQUE
	DB #75, #55, #70	; 0
	DB #26, #22, #70	; 1
	DB #71, #74, #70	; 2
	DB #71, #31, #70	; 3
	DB #55, #71, #10	; 4
	DB #74, #71, #70	; 5
	DB #74, #75, #70	; 6
	DB #71, #12, #20	; 7
	DB #75, #75, #70	; 8
	DB #75, #71, #10	; 9
	DB #00, #20, #20	; : ; DEUX POINTS
	DB #02, #02, #28	; ; POINT VIRGULE
	DB #12, #42, #10	; < PLUS PETIT QUE
	DB #00, #70, #70	; = EGAL
	DB #42, #12, #40	; > PLLUS GRAND QUE
	DB #61, #12, #02	; ? POINT D'INTERROGATION
	DB #69, #BB, #87	; @ AROBASE
	DB #75, #75, #50	; A
	DB #65, #75, #70	; B
	DB #74, #44, #70	; C
	DB #65, #55, #60	; D
	DB #74, #64, #70	; E
	DB #74, #64, #40	; F
	DB #74, #45, #70	; G
	DB #55, #75, #50	; H
	DB #22, #22, #20	; I
	DB #71, #11, #60	; J
	DB #44, #56, #50	; K
	DB #44, #44, #70	; L
	DB #57, #55, #50	; M
	DB #75, #55, #50	; N
	DB #75, #55, #70	; O
	DB #75, #74, #40	; P
	DB #75, #55, #71	; Q
	DB #75, #65, #50	; R
	DB #74, #71, #70	; S

```
        DB #72, #22, #20        ; T
        DB #55, #55, #70        ; U
        DB #55, #55, #20        ; V
        DB #55, #57, #50        ; W
        DB #55, #25, #50        ; X
        DB #55, #22, #20        ; Y
        DB #71, #24, #70        ; Z
        DB #74, #44, #70        ; [   CROCHET GAUCHE
        DB #44, #22, #11        ; \   BARRE OBLIQUE INVERSEE
        DB #71, #11, #70        ; ]   CROCHET FROIT
        DB #25, #50, #00        ; ^   FLECHE HAUT
        DB #00, #00, #0F        ; _   SOULIGNEMENT
        DB #23, #10, #00        ; `   APOSTROPHE INVERSEE
FIN
```

Et voilà le travail. Ce qui me locomotive, c'est de savoir que vous avez suivi mon tutoriel jusqu'à la fin.
Cependant, les erreurs sont humaines. Si vous en trouvez dans ce tutoriel, cela pourrait signifier que je suis un humain mais, pour toutes vos questions, corrections, améliorations, suggestions, n'hésitez pas à me contacter.

Planche des 65 mini-caractères :





LISTE DE LIEN

Liens concernant l'AMSTRAD CPC :

https://www.cpc-power.com/
https://amstrad.eu/
https://asmtradcpc.zilog.fr/
https://acpc.me/
https://www.amstariga.net/
https://jerres12.net/border0/
https://www.amstradtoday.com/
https://cpcrulez.fr/
http://tj.gpa.free.fr/
http://cpc.sylvestre.org/
http://cpccrackers.free.fr/
https://www.velus.be/
https://www.genesis8bit.fr/
http://memoryfull.net/
http://quasar.cpcscene.net/doku.php

Au sujet de l'auteur :

Blog : http://retropoke.canalblog.com/
E-mail : philippe.moulin.fr@laposte.net

Dans la même collection :

AMSTRAD CPC : Les RSX et leurs paramètres 06/2022

Philippe Moulin, 07/2023

Matrices à imprimer :