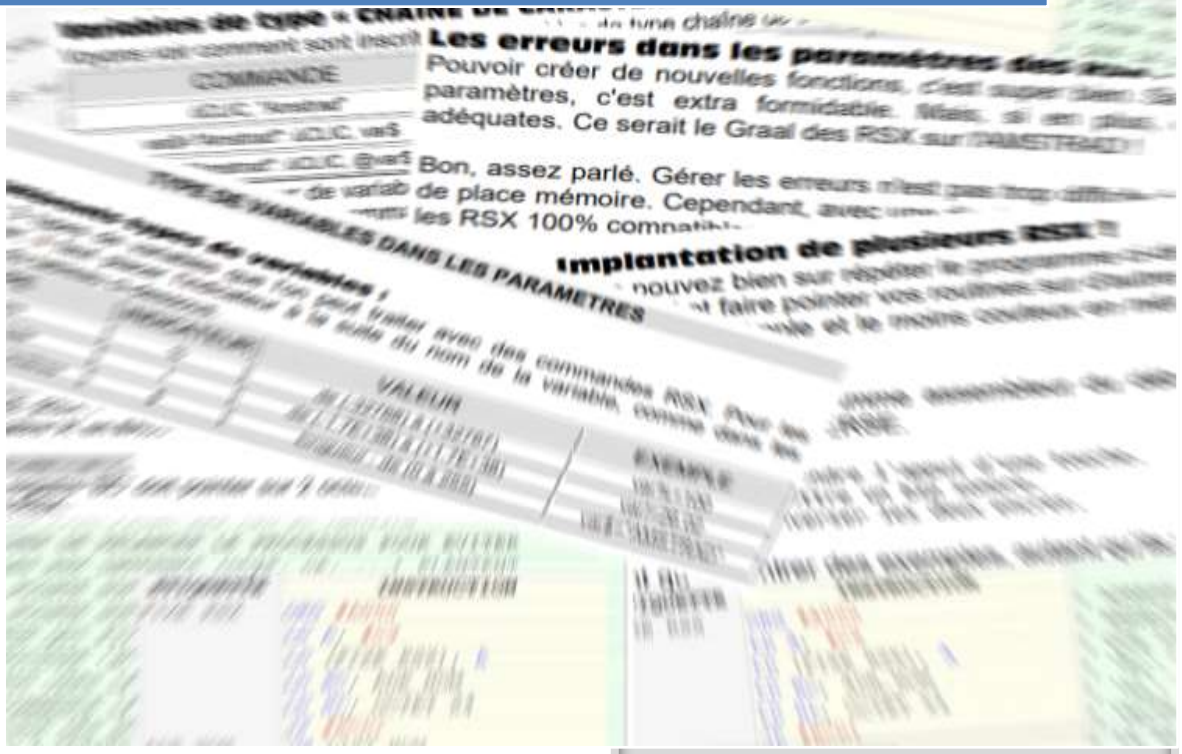
2022

AMSTRAD CPC

Les RSX et leurs paramètres



À travers, ce guide, découvrez tous les secrets sur les extensions RSX de l'AMSTRAD CPC.
Vous serez en mesure de créer rapidement des nouvelles instructions d'une qualité professionnelle avec la gestion du type de variables, des paramètres optionnels et des erreurs appropriées.
Jamais un guide, sur ce sujet, n'aura été aussi complet.

Philippe MOULIN

A M S T R A D - C P C

Les RSX et leurs paramètres

Ce guide est issu de longues recherches sur la toile et dans les livres et revues, afin de vous fournir des informations les plus exactes possibles. Néanmoins, l'auteur, PHILIPPE MOULIN n'assume aucune responsabilité dans tous les domaines suite à l'usage de ce guide. Il est gratuit et vous pouvez le copier et le partager dans les conditions de n'y apporter aucune modification sans l'accord de l'auteur.

- S O M M A I R E -

06/2022 – PREMIERE EDITION

Mais que veut dire « RSX » ?

RSX signifie : Résident System eXtension (extension des instructions BASIC)... Les RSX permettent d'ajouter des instructions au langage BASIC de l'AMSTRAD CPC. Dans une certaine façon, c'est l'ancêtre des bibliothèques du C++.

Dit-on une RSX ou un RSX ?

Les RSX n'ont pas de sexe et chacun peut dire un RSX comme pour un élargissement du langage BASIC, ou une RSX comme pour une extension au langage BASIC. L'important, c'est de se faire comprendre, non !

Les RSX sont accessibles depuis le BASIC en plaçant le caractère "|" devant leurs noms. Suivant le pays, le caractère "|" est à remplacer par le caractère "ù". Pour plus de clarté dans le manuel, j'utiliserai les deux manières.

Voici un exemple de RSX pour obtenir le catalogue de la disquette.
Tapez : « |DIR » ou « ùDIR » selon votre cas.
|DIR est une commande RSX déjà existante sur les AMSTRAD CPC équipés d'un lecteur de disquette. Sa routine d'exécution est située en mémoire ROM, mais ce n'est pas le sujet de ce guide.

Différence entre RSX et CALL :

Les RSX sont similaires à la commande CALL. La seule différence est que vous n'avez pas besoin de connaître l'adresse du programme binaire pour utiliser les RSX.
Pour notre exemple, nous allons utiliser le vecteur système &BB06 qui attend l'appui sur une touche avant de rendre la main.

Depuis la ligne de commande du BASIC, tapez : `CALL &BB06` puis appuyez sur la touche [ENTER].
Comme vous le voyez, cette commande attend que vous appuyiez sur une touche pour rendre la main.
On peut aussi remplacer l'adresse du vecteur par une variable.
Depuis le BASIC, tapez : `CLIC=&BB06: CALL CLIC` puis appuyez sur la touche [ENTER]. Le résultat est identique.

L'utilité des RSX est de ne plus se soucier de l'adresse des routines. Ainsi le fameux « CALL &BB06 » ou l'excellent « CALL CLIC » pourront être remplacés par un simple |CLIC. C'est ce que nous développerons dans la prochaine partie du guide.

La limite des RSX :

Les caractères autorisés dans le nom des RSX sont :
- Les 26 lettres majuscules de l'alphabet
- Les 10 chiffres
- Le point final

Le nom des RSX peut commencer soit par une lettre, un chiffre ou même par le point. Le point en un seul caractère est aussi admis, Ainsi ù. ù.. ù.A. sont des RSX valides.
La longueur maximum du nom des RSX est de 255 caractères. Mais il faut savoir que plus la longueur des noms des RSX ou des variables est longue et plus cela demandera des ressources au processeur.

On peut envoyer jusqu'à 32 paramètres avec les RSX. Je vous détaillerais un exemple de RSX utilisant un nombre de paramètres indéfinis un peu plus loin de ce guide.
Le type de variable dans les paramètres n'a pas d'importance, mais pour reconnaître de quel type est une variable depuis la routine des RSX, il faut impérativement faire précéder le nom de la variable par le caractère @.
De cette manière, se sera l'adresse de la variable qui sera envoyée et cela permettra d'avoir plus de renseignements sur la variable elle-même.

- N O T E -

Implantation du RSX « |CLIC ».

Dans cette partie, nous allons voir comment créer une commande RSX qui demande l'appui sur une touche avant de rendre la main au BASIC.

Commençons d'abord par le programme assembleur, les explications suivront.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
FIXE_RSX	ORG #A000	; ADRESSE POUR FIXER LES RSX EN MEMOIRE
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD BC, ADR NOM	; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
	LD HL, OCTET_X4	; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR_NOM	DW LIST_NOM	; LE REGISTRE 'BC' POINTE ICI SUIVI DE LA LISTE
	JP RSX_CLIC	; DES SAUTS VERS LES ROUTINES RSX
LIST_NOM	DEFB "CLI","C" + #80	; ICI COMMENCE LA LISTE DES NOMS DES RSX
		; POUR SIGNALER LA FIN DU NOM D'UNE RSX, IL FAUT
		; QUE LE BIT 7 DU DERNIER CARACTERE SOIT A 1
	DB 00	; LA LISTE DES NOMS DOIT FINIR PAR UN OCTET A 0
OCTET_X4	DB 00, 00, 00, 00	; LE REGISTRE 'HL' POINTE ICI SUR
		; L'EMPLACEMENT DES 4 OCTETS RESERVES
		; ###### LES ROUTINES DES RSX COMMENCENT ICI
RSX_CLIC	JP #BB06	; APPEL BASIC > |CLIC (SANS PARAMETRE)

NB : les nombreux commentaires sont volontaires afin que vous compreniez mieux le programme en lui-même.

Le caractère [#] est utilisé pour représenter les chiffres hexadécimaux, cela apporte plus de clarté au listing. Nous allons analyser ce programme en détail, mais ce ne sera pas toujours le cas par la suite.

ORG #A000

L'instruction 'ORG' permet de définir l'adresse de départ du programme.
Vous pouvez bien sur commencer à une autre adresse, ici ce n'est que pour l'exemple.

LD, A, #C9

#C9 correspond à l'instruction 'RET' en assembleur Z80. (On appelle ça un op-code).
Le registre ‘A’ est chargé avec l’op-code ‘RET’.

LD (FIXE_RSX), A

FIXE_RSX se situe au tout début du programme en #A000. On va donc inscrire la valeur du registre 'A' en #A000.

LD BC, ADR_NOM

Le registre 'BC' doit pointer sur 2 octets qui contiennent l'adresse ou sont stockés les noms des RSX.

`LD HL, OCTET_X4`
Le registre 'HL' doit pointer sur l'adresse ou 4 octets sont libres. Les 4 octets sont utilisés par le vecteur &BCD1 qui ajoute les RSX.

`JP #BCD1`
Ce vecteur fixe les RSX en mémoire.

`ADR_NOM DW LIST_NOM`
2 Octets contenant l'adresse ou débute la liste des noms des RSX utilisés par le vecteur #BCD1 dans le registre 'BC'.

`JP RSX_CLIC`
L'adresse de la liste des noms des RSX doit être suivie des sauts vers les routines RSX.

`LIST_NOM DEFB "CLI","C" + #80`
Liste des noms des RSX que l'on veut créer. Le dernier caractère des noms des RSX doit se terminer avec le bit 7 à 1.
En procédant de la manière ci-dessus, la manipulation est assez facile à réaliser.

`DB 00`
La liste des noms des RSX doit se terminer par un octet d'une valeur de '0'.

`OCTET_X4 DB 00, 00, 00, 00`
Emplacement des quatre octets de libres pointés par le registre 'HL'.

`RSX_CLIC JP #BB06`
Adresse du programme principal de la RSX 'CLIC'
Effectue un saut direct vers le vecteur #BB06

Utilisation du RSX « |CLIC ».

Tout d'abord, réservons la mémoire nécessaire pour que les variables BASIC n'écrasent pas votre routine.
Depuis la ligne de commande, tapez : `MEMORY &9FFF` et compilez votre programme puis tapez : `CALL &A000` pour fixer les noms des RSX en mémoire.

Ça y est, on peut enfin utiliser notre RSX. Toujours sous la ligne de commande BASIC, tapez : `|CLIC` ou `ùCLIC` suivant le clavier.
La routine du RSX sera lancée et tout comme le fameux « CALL &BB06 », il vous faudra appuyer sur une touche du clavier pour revenir au BASIC.

La création d'une commande RSX n'est pas plus compliqué que ça et ce sera pareil si on désire créer plusieurs RSX.

Implantation de plusieurs RSX !

Vous pouvez bien sur répéter le programme ci-dessus à plusieurs endroits de la mémoire de l'Amstrad et faire pointer vos routines sur d'autres programmes.
Mais le plus simple et le moins couteux en mémoire c'est de n'utiliser qu'une seule routine pour fixer les RSX.

Reprenons le programme assembleur du début et ajoutons deux nouvelles commandes RSX |BEEP et |INVERSE.

```
- |CLIC     : attendre l'appui d'une touche.
- |BEEP     : émettre un bip sonore.
- |INVERSE  : inverser les deux encres.
```

Tant qu'à montrer des exemples, autant qu'ils soient utiles.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
FIXE_RSX	ORG #A000	; ADRESSE POUR FIXER LES RSX EN MEMOIRE
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD BC, ADR_NOM	; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
	LD HL, OCTET_X4	; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR NOM	DW LIST NOM	; LE REGISTRE 'BC' POINTE ICI SUIVI DE LA LISTE
	JP RSX CLIC	; DES SAUTS VERS LA ROUTINE DU RSX 'CLIC'
	JP RSX_BEEP	; SAUT VERS LA ROUTINE DU RSX 'BEEP'
	JP RSX_INV	; SAUT VERS LA ROUTINE DU RSX 'INVERSE'
LIST_NOM	DEFB "CLI","C" + #80	; ICI COMMENCE LA LISTE DES NOMS DES RSX
	DEFB "BEE","P" + #80	; POUR SIGNALER LA FIN DU NOM D'UNE RSX, IL FAUT
	DEFB "INVERS","E" + #80	; QUE LE BIT 7 DU DERNIER CARACTERE SOIT A 1
	DB 00	; LA LISTE DES NOMS DOIT FINIR PAR UN OCTET A 0
OCTET_X4	DB 00, 00, 00, 00	; LE REGISTRE 'HL' POINTE ICI SUR
		; L'EMPLACEMENT DES 4 OCTETS RESERVES
		; ###### LES ROUTINES DES RSX COMMENCENT ICI
RSX_CLIC	JP #BB06	; APPEL BASIC > |CLIC (SANS PARAMETRE)
RSX_BEEP	LD A, 7	; APPEL BASIC > |BEEP (SANS PARAMETRE)
	JP #BB5A	; VECTEUR #BB5A EQUIVALENT A PRINT CHR$(7)
RSX_INV	LD A, 24	; APPEL BASIC > |INVERSE (SANS PARAMETRE)
	JP #BB5A	; LE VECTEUR #BB9C FAIT LA MEME CHOSE

Ce programme de 57 octets seulement rajoute 3 instructions et il n'est volontairement pas optimisé.

Remarquez bien les deux sauts et les 2 noms des RSX qui ont été rajoutés à la suite du premier RSX. (texte sur fond vert).
Le plus important à retenir, c'est que la liste des sauts de vos routines doit correspondre dans le même ordre que la liste des noms des RSX.

Prochaine étape : Les paramètres des RSX.

Récupération d'un paramètre.

Prenons un exemple avec une RSX **ùCLIC** que l'on a déjà étudié et ajoutons lui un paramètre qui désignera la touche qui devra être appuyée.

La commande se présentera ainsi : `ùCLIC, 32` Le paramètre '32' désignera le code ASCII de la touche à appuyer. (pour l'exemple, ce sera la touche ESPACE)

Lorsque l'on transmet des paramètres, certains registres contiennent des informations qui correspondent à cela :
- Le registre 'A' contient le nombre total des paramètres envoyés.
- Le registre 'IX' pointe sur le dernier paramètre. Ce registre utilise 2 octets par paramètre transmis.
- Le registre 'DE' contient la valeur du dernier paramètre.

Dans l'exemple (ùCLIC, 32), le registre 'DE' sera égal à 32 (valeur du dernier paramètre). J'ai vu beaucoup de programmes qui chargent le dernier paramètre avec les commandes :

```
LD D, (IX + 1)
LD E, (IX + 0)
```

Soit une perte de 6 octets sans compter le temps machine et le temps perdu à les écrire.

Avant de voir comment récupérer plusieurs paramètres, on va créer la commande RSX `ùCLIC, touche` afin de tester d'autres valeurs que '32'.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
FIXE_RSX	ORG #A000	; ADRESSE POUR FIXER LES RSX EN MEMOIRE
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE_RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD BC, ADR_NOM	; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
	LD HL, OCTET_X4	; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR_NOM	DW LIST_NOM	; LE REGISTRE 'BC' POINTE ICI SUIVI DE LA LISTE
	JP RSX_CLIC	; DES SAUT VERS LA ROUTINE DU RSX 'CLIC'
LIST_NOM	DEFB "CLI","C" + #80	; ICI COMMENCE LA LISTE DES NOMS DES RSX
		; POUR SIGNALER LA FIN DU NOM D'UNE RSX, IL FAUT
		; QUE LE BIT 7 DU DERNIER CARACTERE SOIT A 1
	DB 00	; LA LISTE DES NOMS DOIT FINIR PAR UN OCTET A 0
OCTET_X4	DB 00, 00, 00, 00	; LE REGISTRE 'HL' POINTE ICI SUR
		; L'EMPLACEMENT DES 4 OCTETS RESERVES
		; ###### LES ROUTINES DES RSX COMMENCENT ICI
RSX_CLIC	DEC A	; REGARDER SI LE NOMBRE DE PARAMETRE EST BON
	RET NZ	; SI 'A'-1 <>0 LE NOMBRE DE PARAMETRE N'EST PAS BON
CLIC_TOU	CALL #BB06	; ATTENDRE L'APPUI SUR UNE TOUCHE
	CP A, E	; COMPARER LA TOUCHE TAPEE AVEC LA VALEUR DU PARAMETRE
	JR NZ, CLIC_TOU	; LA TOUCHE TAPEE NE CORRESPOND PAS, ON RECOMMENCE
	RET	; LA BONNE TOUCHE A ETE TAPEE ON RETOURNE AU BASIC

N'oubliez pas de fixer les RSX avant de les utiliser en tapant : `CALL &A000`

Vous pouvez tester la commande `ùCLIC, valeur` avec d'autre valeurs que 32, mais

ATTENTION, Le programme ne rendra la main que si les deux valeurs sont identiques alors n'essayez pas de rentrer des valeurs qui ne peuvent pas correspondre avec une des touches du clavier, sinon un RESET de l'Amstrad s'imposera.

Ce programme ne teste pas si la valeur entrée en paramètre est supérieur à 255 et ne gère pas les `Syntax error`. Ce paragraphe sera traité à la fin du guide.

Récupération de plusieurs paramètres.

Nous allons encore prendre un exemple avec la construction d'une commande RSX qui tracera un trait d'un point (X1, Y1) vers un autre point (X2, Y2) dans la couleur indiquée en dernier paramètre.
Exemple : `ùLIGNE, x1, y1, x2, y2, couleur`

Voyons voir ce que contiendront les registres dès l'appel de cette RSX :
Le registre 'A' devra avoir la valeur 5 (nombre de paramètres)
Le registre 'DE' aura la valeur du dernier paramètre (couleur)
Le registre 'IX' pointera sur le dernier paramètre (couleur)

Pour l'exemple, ajoutons des valeurs aux paramètres : `ùLIGNE, 200, 80, 480, 260, 3`

Maintenant, regardons le tableau suivant :

ùLIGNE	x1		y1		x2		y2		couleur	
VALEUR	200		80		480		260		3	
HEXA-DECIMAL	00C8		0050		01E0		0104		0003	
REGISTRE IX	IX + 9	IX + 8	IX + 7	IX + 6	IX + 5	IX + 4	IX + 3	IX + 2	IX + 1	IX + 0
CONTENU IX	00	C8	00	50	01	E0	01	04	00	03
REGISTRE DE									D	E

NB : Le registre 'E' contient déjà la couleur (IX + 0).

Les valeurs en hexadécimales sont plus représentatives pour savoir comment sont stockées les données dans le registre ‘IX’. Prenons en exemple le paramètre y2 d’une valeur de 260 soit #0104 en hexadécimal :
Le registre ‘IX + 3’ sera égal à #01
Le registre ‘IX + 2’ sera égal à #04
Soit #0104 = 260

Pour notre exemple, il faudra aussi penser aux vecteurs systèmes qui remplace la commande ‘MOVE’, le vecteur qui remplace la commande ‘DRAW’ et le vecteur qui remplace la commande ‘GRAPHIC_PEN’.

Vecteurs utilisés pour tracer cette ligne :

	Vecteur	Equivalent BASIC
1-	#BBDE	GRAPHIC_PEN
2-	#BBC0	MOVE
3-	#BBF6	DRAW

Voyons comment procéder : `ùLIGNE, x1, y1, x2, y2, couleur`

ETIQUETTE	INSTRUCTION	COMMENTAIRE
FIXE_RSX	ORG #A000	; ADRESSE POUR FIXER LES RSX EN MEMOIRE
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE_RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD BC, ADR_NOM	; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
	LD HL, OCTET_X4	; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR_NOM	DW LIST_NOM	; LE REGISTRE 'BC' POINTE ICI SUIVI
	JP RSX_LIG	; DU SAUT VERS LA ROUTINE DU RSX 'LIGNE'
LIST NOM	DEFB "LIGN","E" + #80	; ICI COMMENCE LA LISTE DES NOMS DES RSX
	DB 00	; LA LISTE DES NOMS DOIT FINIR PAR UN OCTET A 0
OCTET_X4	DB 00, 00, 00, 00	; 'HL'= 4 OCTETS DE RESERVES
		; ###### LES ROUTINES DES RSX COMMENCENT ICI
RSX_LIG	CP A, 5	; COMPARER LE REGISTRE 'A' AVEC LE NOMBRE DE PARAMETRE
	RET NZ	; SI ERREUR
	LD A, E	; 'A'= COULEUR
	CALL #BBDE	; FIXER LA COULEUR (GRAPHIC_PEN)
	LD D, (IX + 9)	; CHARGER DANS 'DE' LA VALEUR DU PARAMETRE X1
	LD E, (IX + 8)	; 'DE'= X1 (200)
	LD H, (IX + 7)	; CHARGER DANS 'HL' LA VALEUR DU PARAMETRE Y1
	LD L, (IX + 6)	; 'HL'= Y1 (80)
	CALL #BBC0	; POSITIONNER LE CURSEUR GRAPHIQUE (MOVE)
	LD D, (IX + 5)	; CHARGER DANS 'DE' LA VALEUR DU PARAMETRE X2
	LD E, (IX + 4)	; 'DE'= X2 (480)
	LD H, (IX + 3)	; CHARGER DANS 'HL' LA VALEUR DU PARAMETRE Y2
	LD L, (IX + 2)	; 'HL'= Y2 (260)
	JP #BBF6	; TRACER LA LIGNE ET RETOURNER AU BASIC (DRAW)

Ce qui faut retenir, c'est que le registre 'IX' pointe toujours sur le dernier paramètre et que c'est en remontant dans ce registre qu'il est possible de savoir les valeurs des autres paramètres.

Les paramètres optionnels :

Il est parfois utile de rendre optionnel un des paramètres. Souvent c'est le dernier mais on peut aussi moduler les paramètres d'après le nombre total envoyé dans la commande.

Dans la dernière commande RSX que nous avons étudié, il serait bien de ne pas rendre obligatoire le paramètre COULEUR.

On sait que le registre 'A' contient le nombre total de paramètres transmis. Il suffira donc de dévier la routine si ce registre ne contient que 4 paramètres et dans le cas contraire, il ne faudra pas oublier de faire pointer le registre 'IX' sur le paramètre 'Y2' une fois que le paramètre 'couleur' sera traité.

En incrémentant le registre 'IX' on peut le faire pointer sur un autre paramètre. Ce qui permet de créer des paramètres optionnels avec peu de codes en plus. Ainsi une fois que le paramètre 'couleur' sera traité, on fera pointer le registre 'IX' sur le paramètre 'y2' comme dans l'exemple suivant.

Regardons le programme ci-dessous : `ùLIGNE, x1, y1, x2, y2 [, couleur]`

ETIQUETTE	INSTRUCTION	COMMENTAIRE
FIXE_RSX	ORG #A000	; ADRESSE POUR FIXER LES RSX EN MEMOIRE
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE_RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD BC, ADR_NOM	; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
	LD HL, OCTET_X4	; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR_NOM	DW LIST_NOM	; LE REGISTRE 'BC' POINTE ICI SUIVI
	JP RSX_LIG	; DU SAUT VERS LA ROUTINE DU RSX 'LIGNE'
LIST NOM	DEFB "LIGN","E" + #80	; ICI COMMENCE LA LISTE DES NOMS DES RSX
	DB 00	; LA LISTE DES NOMS DOIT FINIR PAR UN OCTET A 0
OCTET_X4	DB 00, 00, 00, 00	; 'HL'= 4 OCTETS DE RESERVES
		; ###### LES ROUTINES DES RSX COMMENCENT ICI
RSX_LIG	SUB A, 4	; REGARDER SI LE REGISTRE 'A' CONTIENT 4 PARAMETRES
	JR Z, PARAM X4	; OUI ALORS NE PAS FAIRE LA COULEUR
	DEC A	; VERIFIER QU'IL Y A BIEN 5 PARAMETRES
	RET NZ	; SI LE NOMBRE DE PARAMETRE NE CORRESPOND PAS
	LD A, E	; 'A'= COULEUR
	CALL #BBDE	; FIXER LA COULEUR (GRAPHIC PEN)
	INC IX	; REPLACER LE REGISTRE SUR LE PARAMETRE Y2
	INC IX	; COMME SI IL N'Y AVAIT QUE 4 PARAMETRES
PARAM_X4	LD D, (IX + 7)	; CHARGER DANS 'DE' LA VALEUR DU PARAMETRE X1
	LD E, (IX + 6)	; 'DE'= X1 (200)
	LD H, (IX + 5)	; CHARGER DANS 'HL' LA VALEUR DU PARAMETRE Y1
	LD L, (IX + 4)	; 'HL'= Y1 (80)
	CALL #BBC0	; POSITIONNER LE CURSEUR GRAPHIQUE (MOVE)
	LD D, (IX + 3)	; CHARGER DANS 'DE' LA VALEUR DU PARAMETRE X2
	LD E, (IX + 2)	; 'DE'= X2 (480)
	LD H, (IX + 1)	; CHARGER DANS 'HL' LA VALEUR DU PARAMETRE Y2
	LD L, (IX + 0)	; 'HL'= Y2 (260)
	JP #BBF6	; TRACER LA LIGNE ET RETOURNER AU BASIC (DRAW)

Le programme a été modifié de sorte qu'il ne soit adapté que pour 4 paramètres, mais si un cinquième paramètre est transmis, alors au traitement de ce cinquième paramètre, il ne faudra pas oublier de faire pointer le registre 'IX' comme si il n'y avait plus que 4 paramètres.

En incrémentant 2 fois le registre IX (Ce registre occupe 2 octets par paramètre) il est simple de passer du cinquième paramètre au quatrième paramètre.
Ainsi, la routine continuera comme s'il n'y avait eu que 4 paramètres envoyées.

La remontée des paramètres :

Il est parfois nécessaire de remonter au premier paramètre et je vais vous en donner l'exemple ci-après.

Regardons la nouvelle commande RSX suivante :
`ùSPOKE, adresse, valeur1 [, valeur2] ... [, valeur29] [, valeur30]`

Cette RSX permettra d'inscrire jusqu'à 30 valeurs depuis l'adresse indiquée. Un super POKE des temps anciens !

Etalons le problème :
1 - Regarder si le nombre da paramètre est compris entre 2 au minimum et 31 au maximum.
31= (30 pour les valeurs + 1 pour l'adresse)
2 - Soustraire 1 au nombre de paramètre et utiliser le registre 'B' pour en faire une boucle.

3 - Multiplier le chiffre par deux (2 octets par paramètre).

4 - Additionner le registre 'IX' avec ce nombre. (IX pointe sur le paramètre adresse).

5 - Récupérer l'adresse dans 'DE' (IX + 0 et IX + 1)

6 - Décrémenter 2 fois le registre 'IX' pour revenir au paramètre après l'adresse.

7 - Récupérer la valeur du paramètre suivant et l'inscrire dans 'DE' puis incrémenter 'DE'

8 - Puiser la boucle du registre 'B' en revenant en ligne 5.

9 - Ne pas oublier le retour au BASIC.

Le programme assembleur en lui-même n'est pas trop compliqué une fois que l'on a bien compris comment revenir sur le paramètre qui suit le nom de la commande RSX.
En voici le listing complet :

ETIQUETTE	INSTRUCTION	COMMENTAIRE
FIXE_RSX	ORG #A000	; ADRESSE POUR FIXER LES RSX EN MEMOIRE
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE_RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD BC, ADR NOM	; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
	LD HL, OCTET X4	; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR_NOM	DW LIST_NOM	; LE REGISTRE 'BC' POINTE ICI SUIVI DES SAUTS
	JP RSX_SPOK	; SAUT VERS LA ROUTINE DU RSX 'SPOKE'
LIST_NOM	DEFB "SPOK","E" + #80	; ICI COMMENCE LA LISTE DES NOMS DES RSX
		; POUR SIGNALER LA FIN DU NOM D'UNE RSX, IL FAUT
		; QUE LE BIT 7 DU DERNIER CARACTERE SOIT A 1
	DB 00	; LA LISTE DES NOMS DOIT FINIR PAR UN OCTET A 0
OCTET_X4	DB 00, 00, 00, 00	; LE REGISTRE 'HL' POINTE ICI SUR
		; L'EMPLACEMENT DES 4 OCTETS RESERVES
		; ###### LES ROUTINES DES RSX COMMENCENT ICI
RSX_SPOK	CP A, 2	; VERIFIER LES PARAMETRES
	RET C	; SI MOINS DE 2 PARAMETRES
	CP A, 32	; ET SI PLUS DE 31 PARAMETRES
	RET NC	; ANNULER ET RETOURNER AU BASIC
	DEC A	; ENLEVER 1 PARAMETRE
	LD B, A	; 'B'= COMPTEUR DU NOMBRE DE PARAMETRES
	ADD A, A	; MULTIPLIER PAR 2 OCTETS PAR PARAMETRES
	LD D, 0	; ADDITIONNER IX EN PASSANT PAR LE REGISTRE 'DE'
	LD E, A	; 'DE'=(NOMBRE DE PARAMETRE X 2 OCTETS)
	ADD IX, DE	; 'IX' POINTE SUR LE PARAMETRE 'ADRESSE'
	LD D, (IX + 1)	; CHARGER L' ADRESSE DANS 'DE'

```
            LD E, (IX + 0)          ; 'DE'= ADRESSE DE DEPART
BOUCLE_B    DEC IX                  ; ALLER AU PARAMETRE SUIVANT
            DEC IX                  ; EN REDESCENDANT LE POINTEUR IX
            LD A, (IX + 0)          ; CHARGER DANS 'A' LA VALEUR DES POKE
            LD (DE), A              ; ET L'INSCRIRE A L'ADRESSE
            INC DE                  ; PASSER A L'ADRESSE SUIVANTE
            DJNZ, BOUCLE_B          ; FAIRE TOUS LES PARAMETRES
            RET                     ; RENDRE LA MAIN AU BASIC
```

S'il y a une chose à retenir ici, c'est qu'une fois que le registre 'IX' pointe sur le paramètre le plus à gauche, il faut décrémenter ce registre et non pas l'inverse, pour récupérer les autres paramètres.

Des paramètres par la PILE :

Il existe une autre méthode pour lire la valeur des paramètres en utilisant le pointeur de PILE (registre SP). Cependant, il ne faut surtout pas oublier de remettre dans le registre 'SP' sa valeur initiale avant le retour au BASIC. Regardons ensemble comment y parvenir en créant le/la RSX suivante.

```
ùLDIR, adresse source, adresse destination, longueur
```

Vous l'avez compris, cette nouvelle commande copiera le nombre d'octets indiqué en troisième paramètre de l'adresse source vers l'adresse destination à une vitesse XXL.

```
ETIQUETTE   INSTRUCTION             COMMENTAIRE
FIXE_RSX    ORG #A000               ; ADRESSE POUR FIXER LES RSX EN MEMOIRE
            LD A, #C9               ; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
            LD (FIXE_RSX), A        ; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
            LD BC, ADR_NOM          ; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
            LD HL, OCTET_X4         ; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
            JP #BCD1                ; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR_NOM     DW LIST_NOM             ; LE REGISTRE 'BC' POINTE ICI SUIVI
            JP RSX_LDIR             ; DU SAUT VERS LA ROUTINE DU RSX 'LDIR'
LIST_NOM    DEFB "LDI","R" + #80    ; ICI COMMENCE LA LISTE DES NOMS DES RSX
            DB 00                   ; LA LISTE DES NOMS DOIT FINIR PAR UN OCTET A 0
OCTET X4    DB 00, 00, 00, 00       ; 'HL'= 4 OCTETS DE RESERVES

RSX_LDIR    CP A, 3                 ; APPEL BASIC > |LDIR, SOURCE, DESTINATION, NOMBRE
            RET NZ                  ; LE NOMBRE DE PARAMETRE EST INCORRECT
            LD (ADR_SP + 1), SP     ; MZMORISER LA VALEUR DU REGISTRE 'SP'
            LD SP, IX               ; 'SP' POINTE SUR LE DERNIER PARAMETRE
            POP BC                  ; 'BC'= NOMBRE D'OCTETS A TRANSFERER (IX+0 IX+1)
            POP DE                  ; 'DE'= ADRESSE DE DESTINATION (IX+2 IX+3)
            POP HL                  ; 'HL'= ADRESSE SOURCE (IX+4 IX+5)
ADR_SP      LD SP, #0000            ; 'SP'=VALEUR D'ORIGINE (AUTO-MODIFIE)
            LDIR                    ; EFFECTUER LE TRANSFERT
            RET                     ; ET RENDRE LA MAIN AU BASIC
```

Le programme parle de lui-même. A l'adresse `ADR_SP` se trouve l'op-code de la commande `LD SP, #xxxx`. Il faut donc décaler de 1 octet l'écriture du contenu du registre 'SP' d'où l'inscription `LD (ADR_XP + 1), SP`. (écriture en vert clair).

Dans la prochaine étape, nous parlerons des types de variables dans les paramètres.

Les différents types de variables :

Il existe 3 types de variables que l'on peut traiter avec des commandes RSX. Pour les différencier, il faut placer l'indicateur à la suite du nom de la variable, comme dans les exemples du tableau ci-dessous.

TYPE	INDICATEUR	VALEUR	EXEMPLE
Entiers	%	de (-32768) à (+32767)	var%=100
Avec virgule	!	de (-1.7E+38) à (+1.7E+38)	var!=26.02
Chaines de caractères	$	longueur : de (0 à 255)	var$="AMSTRAD"

Type de variables dans les paramètres :

Jusqu'à maintenant, on n'a utilisé que des chiffres sans virgule dans les paramètres. Il faut se rappeler que le registre 'IX' pointe sur les valeurs des paramètres. Ce registre ne dispose que de 2 octets par paramètre pour y stocker les valeurs.
Ce qui signifie que pour des variables à virgule qui demandent 5 octets, ce ne sera donc pas la valeur réelle qui sera envoyée, mais son entier.

Si on reprend comme exemple la commande : `CALL &BB06,32`
Et que l'on remplace la valeur 32 par un chiffre à virgule comme dans cet exemple :
`CALL &BB06, 32.56` alors l'AMSTRAD arrondira la valeur en un entier (soit 33) avant de l'envoyer dans le registre 'IX'.

Pour que la valeur soit retrouvée, il faudra d'abord déclarer une variable de type réel (à virgule) en plaçant la valeur dans cette variable comme dans cet exemple :

```
variable! = 32.56
```

Cependant, ce type de variable occupe 5 octets et comme le registre 'IX' ne dispose que de deux par paramètre, il faudra envoyer l'adresse de la variable pour retrouver sa valeur.

Les adresses des variables dans la mémoire des CPC occupent 2 octets. Pour envoyer l'adresse de la variable et non pas sa valeur, il faut placer le caractère '@' devant le nom de cette variable. Ainsi, ce sera l'adresse mémoire de la variable que l'on retrouvera dans le registre 'IX'. Le caractère '@' est appelé 'POINTEUR DE VARIABLE'.

Maintenant détaillons les différents type de variables.

Variables de type « ENTIER » :

Reprenons la commande RSX ùCLIC et regardons comment sont inscrites les valeurs directes et les variables de type ENTIER.

COMMANDE	IX + 1	IX + 0	DE
ùCLIC, 32	00	32	32
var%=32: ùCLIC, var%	00	32	32
var%=32: ùCLIC, @var%	adresse		adresse

Sans le pointeur de variables, c'est la valeur directe que contiendra le registre 'IX' et dans ces conditions, il ne sera pas possible de savoir de quel type de variables appartient ce paramètre depuis la routine de la RSX.

Ce n'est pas trop gênant si on travaille qu'avec des entiers, sauf si on veut obtenir une valeur en retour de paramètre ou il faudra obligatoirement passer par le pointeur de variables.

Regardons maintenant comment récupérer un entier depuis une adresse de variable et reprenons pour cela, l'exemple du RSX ùCLIC : `var%=128: ùCLIC, @var%`

Après appel, l'adresse de la variable se trouve dans le registre 'DE'. Mais avant de récupérer la valeur de la variable, vérifions qu'il s'agit bien d'une variable de type 'ENTIER'. Cet indication se trouve dans l'octet (adresse - 1) et en décrémentant 'DE' comme dans l'exemple qui suit, on pourra récupérer l'indicateur.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
...	...	; ùCLIC, @var%
	DEC DE	; 'DE'= ADRESSE - 1
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE DE LA VARIABLE
	CP A, 1	; SI LA VARIABLE EST DE TYPE 'ENTIER' ALORS 'A'=1
	JR NZ, ERREUR	; CETTE VARIABLE N'EST PAS UN 'ENTIER'
...	...	; ...

Une variable de type entier a le chiffre '1' pour indicateur et occupe 2 octets de mémoire.

Maintenant, récupérons la valeur de la variable dans le registre 'DE' en passant l'adresse dans le registre 'HL'.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
...	...	; ùCLIC, @var%
	DEC DE	; 'DE'= ADRESSE - 1
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE DE LA VARIABLE
	CP A, 1	; SI LA VARIABLE EST DE TYPE 'ENTIER' ALORS 'A'=1
	JR NZ, ERREUR	; CETTE VARIABLE N'EST PAS UN 'ENTIER'
	EX HL, DE	; PASSER L'ADRESSE DANS 'HL' POUR MOINS DE CODE
	LD E, (HL)	; 'E'= OCTET SU POIDS FAIBLE DE LA VARIABLE
	INC HL	; 'HL' POINTE SUR ADRESSE + 1
	LD D, (HL)	; 'D'= OCTET DU POIDS FORT DE LA VARIABLE 'DE'=VALEUR
...	...	; ...

Ce n'est pas plus sorcier que ça !

Découvrons maintenant comment renvoyer une valeur de type entier à travers un paramètre. En fait, c'est assez simple puisque l'on connaît déjà l'adresse ou cette variable est stockée.

Mais un exemple est mieux parlant. Nous allons diviser par deux la valeur de la variable envoyée et la retourner depuis le même paramètre pour que l'on puisse la lire depuis le BASIC.

Pour cela, nous ne prendrons plus la commande ùCLIC, mais nous programmerons directement à l'adresse &A000. Exemple : `var%=128: CALL &A000, @var%: PRINT var%`

Le résultat de cet exemple sera '64' mais vous pourrez bien sur tester d'autres valeurs à la condition que la variable soit de type entier sinon un BIP sera émis.

Programme en assembleur :

ETIQUETTE	INSTRUCTION	COMMENTAIRE
DIV_PAR2	ORG #A000	;; var%=128: CALL &A000, @var%
	CP A, 1	; REGARDER LE NOMBRE DE PARAMETRE (DEC A)
	JR NZ, ERREUR	; LE NOMBRE DE PARAMETRE NE CORRESPOND PAS
	DEC DE	; 'DE'= ADRESSE - 1
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE DE LA VARIABLE
	CP A, 1	; SI LA VARIABLE EST DE TYPE 'ENTIER' ALORS 'A'=1
	JR NZ, ERREUR	; CETTE VARIABLE N'EST PAS UN 'ENTIER'
	EX HL, DE	; PASSER L'ADRESSE DANS 'HL' POUR MOINS DE CODE
	LD E, (HL)	; 'E'= OCTET FAIBLE DE LA VARIABLE
	INC HL	; 'HL' POINTE SUR ADRESSE + 1
	LD D, (HL)	; 'D'= OCTET FORT DE LA VARIABLE 'DE'=VALEUR
		; DIVISER 'DE' PAR DEUX
	SRL D	; ROTATION VERS LA GAUCHE AVEC RETENUE DE 'D'
	RR E	; 'E'=ROTATION GAUCHE + LA RETENUE
		; RETOURNER LA VALEUR OBTENUE 'HL'=ADRESSE + 1
	LD (HL), D	; INSCRIRE LE POIDS FORT DANS (ADRESSE + 1)
	DEC HL	; REPLACER 'HL' SUR ADRESSE
	LD (HL), E	; INSCRIRE LE POIDS FAIBLE DANS (ADRESSE + 0)
	RET	; ET RETOURNER AU BASIC
ERREUR	LD A, 7	; EMETTRE UN SIMPLE BEEP
	JP #BB5A	; LA GESTION D'ERREURS SERA TRAITEE PLUS LOIN

Ce programme n'est volontairement pas optimisé.

Après avoir contrôlé le nombre de paramètres et le type de variable, le registre ‘DE ‘ est échangé avec le registre ‘HL’. Ensuite, on charge dans le registre ‘DE’ la valeur du paramètre et son contenu est divisé par deux. Pour finir, on inscrit la valeur obtenue dans le registre ‘HL’ qui contient l’adresse du paramètre.

Maintenant, vous savez à peu près tout sur les variables de type entier.

Variables de type « A VIRGULE » :

Regardons comment à travers la commande RSX ùCLIC, sont inscrites les différentes valeurs des variables de type A VIRGULE.

COMMANDE	IX + 1	IX + 0	DE
ùCLIC, 32.56	00	33	33
var!=32.56: ùCLIC, var!	00	33	33
var!=32.56: ùCLIC, @var!	adresse		adresse

Comme pour les entiers, si le pointeur de variables est absent, on n'obtiendra qu'une valeur arrondie de la variable et on ne pourra pas savoir de quel type elle appartient.
Il ne sera pas non plus possible de retourner une valeur depuis les paramètres.

Pour savoir si la variable est bien du type à virgule, il faut que l'octet situé à (adresse - 1) de la variable contienne la valeur '4'. On procédera donc comme pour les entiers.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
...	...	; ùCLIC, @var!
	DEC DE	; 'DE'= ADRESSE - 1
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE DE LA VARIABLE
	CP A, 4	; 'A'=4 SI C'EST UNE VARIABLE 'A VIRGULE'
	JR NZ, ERREUR	; CETTE VARIABLE N'EST PAS DE TYPE 'A VIRGULE'
...	...	; ...

Une variable de type à virgule a le chiffre '4' pour indicateur et occupe 5 octets de mémoire.

Maintenant, que vous êtes en mesure de créer des commandes RSX, le prochain exemple ne concernera que la partie essentielle sur les variables des paramètres.
Nous allons créer une routine qui permettra d'échanger deux variables à virgule entre elles, comme dans l'exemple suivant :

```
var1!=26.02
var2!=28.03
CALL &A000, @var1!, @var2!

PRINT var1!
  28.03

PRINT var2!
  26.02
```

Les étapes :
1 - Le registre 'DE' pointe déjà sur l'adresse du dernier paramètre (var2!).
2 - Comparer si la variable est bien du type à virgule.
3 - Faire pointer le registre 'HL' sur l'adresse du premier paramètre (var1!).
4 - Vérifier que la variable (var1!) et du même type que (var2!)
5 - Faire une boucle d'échange de 5 octets avec le registre 'B'
6 - Échanger 5 octets de 'HL' et de 'DE' et terminer le programme.

Routine assembleur :

ETIQUETTE	INSTRUCTION	COMMENTAIRE
SWAP	ORG #A000	;;var1!=26.02: var2!=28.03!: CALL &A000, @var1!, @var2!
	CP A, 2	; REGARDER LE NOMBRE DE PARAMETRE
	JR NZ, ERREUR	; LE NOMBRE DE PARAMETRE NE CORRESPOND PAS
	DEC DE	; 'DE'= ADRESSE var2! - 1
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE var2!
	CP A, 4	; SI 'A'=4 ALORS C'EST UNE VARIABLE A VIRGULE
	JR NZ, ERREUR	; NON CE N'EST PAS LE BON TYPE DE VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE var2!
	LD H, (IX + 3)	; CHARGER DANS 'HL' L'ADRESSE DE var1!
	LD L, (IX + 2)	; 'HL'= ADRESSE var1!

```
        DEC HL                  ; 'HL'= ADRESSE adr1! - 1
        LD B, (HL)              ; 'B'= INDICATEUR DU TYPE DE var1!
        INC HL                  ; REPLACER 'HL' SUR ADRESSE adr1!
        CP A, B                 ; SI LES DEUX VARIABLES SONT DE MEME TYPE
        JR NZ, ERREUR           ; NON var1! N'EST PAS UNE VARIABLE A VIRGULE

        INC B                   ; 'B'=4 (INDICATEUR) +1 SOIT 5 OCTETS A TRANSFERER
OCTET_X5 LD A, (DE)             ; 'A'= VALEUR DE L'OCTET var2!
        LD C, (HL)              ; 'C'= VALEUR DE L'OCTET var1!
        LD (HL), A              ; 'HL'= VALEUR DE L'OCTET (DE)
        LD A, C                 ; 'A'= VALEUR DE L'OCTET var1!
        LD (DE), A              ; 'DE'= VALEUR DE L'OCTET (HL)
        INC HL                  ; PASSER A L'OCTET var1! SUIVANT
        INC DE                  ; FAIRE PAREIL POUR var2!
        DJNZ, OCTET_X5          ; ECHANGER LES 5 OCTETS
        RET                     ; ET RETOURNER AU BASIC

ERREUR  LD A, 7                 ; EMETTRE UN SIMPLE BEEP
        JP #BB5A                ; LA GESTION D'ERREURS SERA TRAITEE PLUS LOIN
```

S'il y a une chose à retenir, c'est que si on ajoute 1 à l'indicateur de variable, on obtient alors le nombre d'octets qu'occupe la variable. Et c'est valable aussi pour les entiers et les chaînes de caractères.

Variables de type « CHAÎNE DE CARACTÈRES » :

Voyons voir comment sont inscrites les variables de type chaîne de caractères.

COMMANDE	IX + 1	IX + 0	DE
ùCLIC, "Amstrad"	??	??	??
var$="Amstrad": ùCLIC, var$	??	??	??
var$="Amstrad": ùCLIC, @var$	adresse	adresse	adresse

Bien que le pointeur de variables soit automatique sur les CPC 6128, il vaut toujours mieux le prévoir pour que vos commandes soient entièrement compatibles sur tous les CPC.

Ici, on ne pourra travailler qu'avec l'adresse de la chaîne de caractères, car elle ne peut pas tenir que sur les deux octets du registre 'IX'.

Ce n'est pas plus mal et de cette manière, on peut modifier la variable avant son retour au BASIC.
Il y a juste un souci avec ce genre de variable, c'est que la longueur de la chaîne de caractère ne devra jamais être augmentée à partir du langage machine.

Regardons maintenant, comment sont stockées les données d'une chaîne de caractères par cet exemple : `chaine$="Amstrad": CALL &A000, @chaine$`
Le registre 'IX' contiendra l'adresse du descripteur de chaîne. Cette adresse est composée de 3 octets :
Le 1er octet contient la longueur de la chaîne de caractères.
Le 2ème et le 3ème octet contiennent l'adresse du premier caractère de la chaîne.

Voici un aperçu rapide du code pour faire pointer le registre 'DE' sur le premier caractère et charger le registre 'A' avec le nombre de caractères de la chaîne.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
	EX HL, DE	; 'HL'= ADRESSE DESCRIPTEUR DE CHAINE
	LD A, (HL)	; 'A'= LONGUEUR DE LA CHAINE
	INC HL	; SE DEPLACER SUR L'ADRESSE DU PREMIER CARACTERE
	LD E, (HL)	; FAIRE POINTER 'DE' SUR LE PREMIEUR CARACTERE
	INC HL	;
	LD D, (HL)	; 'DE' POINTE SUR LE PREMIER CARACTERE

Maintenant, on va vérifier que la variable est bien une chaîne de caractères. Comme pour les autres variables, cette information se trouve dand (adresse - 1) et son contenu est '2'.

ETIQUETTE	INSTRUCTION	COMMENTAIRE
ROT G	ORG #A000	;; chaine$="Amstrad": CALL &A000, @chaine$
	DEC A	; REGARDER LE NOMBRE DE PARAMETRE
	JR NZ, ERREUR	; LE NOMBRE DE PARAMETRE NE CORRESPOND PAS
	DEC DE	; 'DE'= ADRESSE (chaine$ - 1)
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE chaine$!
	CP A, 2	; SI 'A'=2 ALORS C'EST UNE CHAINE DE CARACTERE
	JR NZ, ERREUR	; NON CE N'EST PAS LE BON TYPE DE VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE chaine$!
	EX HL, DE	; 'HL'= ADRESSE DESCRIPTEUR DE CHAINE
	LD A, (HL)	; 'A'= LONGUEUR DE LA CHAINE
	OR A	; SI 'A' EST VIDE
	RET Z	; LA CHAINE ET VIDE
	INC HL	; PLACER 'HL' SUR L'ADRESSE DES CARACTERES
	LD E, (HL)	; FAIRE POINTER 'DE' SUR LE PREMIER CARACTERE
	INC HL	;
	LD D, (HL)	; 'DE'= ADRESSE DU PREMIER CARACTERE
	RET	; ET RETOURNER AU BASIC
ERREUR	LD A, 7	; EMETTRE UN SIMPLE BIP
	JP #BB5A	; LA GESTION D'ERREURS SERA TRAITEE PLUS LOIN

Une variable de type chaîne de caractères a le chiffre '2' pour indicateur et occupe 3 octets.

À présent que l'on sait où récupérer les caractères, on peut créer des routines telles que l'exemple ci-dessous qui produira une rotation vers la gauche de toute la chaîne.

Résultat :

```
a$="0123456789"
CALL &A000, @a$ : PRINT A$
1234567890

CALL &A000, @a$ : PRINT A$
2345678901
```

Le principe :
Mémoriser le premier caractère de la chaîne dans le registre 'A'. Déplacer tous les autres caractères vers la gauche en partant du deuxième caractère. Inscrire le caractère contenu dans le registre 'A' en dernière position.

Routine assembleur :

ETIQUETTE	INSTRUCTION	COMMENTAIRE
ROT_G	ORG #A000	;; chaine$="Amstrad": CALL &A000, @chaine$
	DEC A	; REGARDER LE NOMBRE DE PARAMETRE
	JR NZ, ERREUR	; LE NOMBRE DE PARAMETRE NE CORRESPOND PAS
	DEC DE	; 'DE'= ADRESSE (chaine$ - 1)
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE chaine$!
	CP A, 2	; SI 'A'=2 ALORS C'EST UNE CHAINE DE CARACTERE
	JR NZ, ERREUR	; NON CE N'EST PAS LE BON TYPE DE VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE chaine$!
	EX HL, DE	; 'HL'= ADRESSE DESCRIPTEUR DE CHAINE
	LD A, (HL)	; 'A'= LONGEUR DE LA CHAINE
	OR A	; SI 'A' EST VIDE
	RET Z	; LA CHAINE ET VIDE
	INC HL	; PLACER 'HL' SUR L'ADRESSE DES CARACTERES
	LD E, (HL)	; FAIRE POINTER 'DE' SUR LE PREMIER CARACTERE
	INC HL	;
	LD D, (HL)	; 'DE'= ADRESSE DU PREMIER CARACTERE
	DEC A	; INITIALISER LE COMPTEUR POUR LDIR
	LD C, A	; DANS LE REGISTRE 'BC'
	LD B, 0	; 'BC'=NOMBRE DE CARACTERE - 1
	LD A, (DE)	; 'A'= 1er CARACTERE DE LA CHAINE
	LD H, D	; FAIRE POINTER 'HL' SUR LE 2eme CARACTERE
	LD L, E	; 'HL' POINTE SUR LE 1er CARACTERE
	INC HL	; 'HL' POINTE SUR LE 2eme CARACTERE
	LDIR	; DEPLACER TOUS LES (CARACTERE-1) VERS LA GAUCHE
	LD (DE), A	; ET PLACER LE 1er CARACTERE EN FIN DE CHAINE
	RET	; ET RETOURNER AU BASIC
ERREUR	LD A, 7	; EMETTRE UN SIMPLE BEEP
	JP #BB5A	; LA GESTION D'ERREURS SERA TRAITEE PLUS LOIN

Cette routine pause cependant un problème. Il faut impérativement que le nombre de caractères de la chaîne soit supérieur à 1, sinon l'AMSTRAD risque de se paralyser dans l'infini car on décrémente le registre 'BC' avant la boucle ‘LDIR’.
Comme cette routine contrôle déjà si la chaîne n'est pas vide, ce sera facile pour vous de corriger ce petit problème.

Pour résumer : Si le pointeur de variable est placé devant le nom de la variable, alors le registre ‘IX’ contiendra l’adresse de la variable, sinon ce sera la valeur arrondie de la variable sauf pour les chaînes de caractères ou une erreur sera produite sur CPC 464 si le pointeur est absent et sur les autres CPC, le registre ‘IX’ contiendra l’adresse du descripteur de chaîne de caractères.

Vous devez savoir maintenant manipuler les principales variables des RSX. La prochaine étape sera de gérer les erreurs des paramètres transmis dans une commande RSX.

Les erreurs dans les paramètres des RSX :

Pouvoir créer de nouvelles fonctions, c'est super bien. Savoir gérer les variables dans les paramètres, c'est extra formidable. Mais, si en plus, on pouvait afficher les erreurs adéquates. Ce serait le Graal des RSX sur l'AMSTRAD !

Bon, assez parlé. Gérer les erreurs n'est pas trop difficile à faire et ça ne demande pas trop de place mémoire. Cependant, avec une dizaine d'octets supplémentaires, on pourra rendre les RSX 100% compatibles sur les AMSTRAD 464, 664, 6128, 464+, 6128+.

Pour ne pas reprendre tout le code, la routine débutera au label [ERREUR] des précédents programmes.
Routine assembleur :

ETIQUETTE	INSTRUCTION	COMMENTAIRE
ERREUR		; AFFICHE L'ERREUR DONT LE NUMERO EST DANS 'B'
	CALL #B915	; CE VECTEUR RETOURNE LA VERSION CPC DANS 'A'
	LD HL, #CB55	; 'HL'=VECTEUR DES ERREURS (CPC 664 ET 6128)
	OR A	; SI 'A' >0 ALORS CPC 664 OU 6128
	JR NZ, CPC_OK	; C'EST UN CPC 664 OU 6128
	LD HL, #CA94	; 'HL'=VECTEUR DES ERREURS (CPC 464)
CPC OK	LD (VECT ERR + 1), HL	; INSCRIRE L'ADRESSE DU VECTEUR ERREUR
	CALL #B900	; VECTEUR D'ACCES DE LA ROM SUPERIEURE
	LD A, C	; RECUPERER LE NUMERO D'ERREUR QUI EST DANS 'C'
	LD E, A	; LE NUMERO D'ERREUR DOIT ETRE DANS 'A' ET 'E'
VECT_ERR	CALL #CB55	; SAUT AUTOMODIFIE DU VECTEUR QUI AFFICHE LES ERREURS
	JP #B903	; VECTEUR FERMETURE ROM SUPERIEURE

Ce code n'est pas optimisé et on peut gagner de la place si on considère que de retour au BASIC, la mémoire ROM supérieure se ferme automatiquement.

D'abord on inspecte le vecteur #CA94 qui nous retourne la version du BASIC. Si la valeur retournée est égal à zéro, alors on est sur un CPC 464 et il faut adapter le vecteur d'erreur dans le registre 'HL'. Sinon le registre 'HL' contient la valeur pour le CPC 664 et 6128.

On auto-modifie le vecteur d'erreur en inscrivant la valeur du registre 'HL' juste après la commande 'CALL' (VECT_ERR + 1). Si on n'incrémente pas de 1 alors la commande 'CALL' sera écrasée par le contenu du registre 'HL' et l'ordinateur plantera des choux au lieu d'afficher l'erreur.

Le numéro d'erreur avant appel de la routine doit être dans le registre 'C'. Vous pouvez changer les registres, mais pour le vecteur d'erreur, il faut que le numéro d'erreur soit dans le registre 'A' et dansle registre 'E'.

Voici quelques numéros d'erreurs utiles pour les RSX :

00 Unknown error (erreur inconnue)

02 Syntax error (erreur de syntaxe)

05 Improper argument (argument inapproprié)

```
06 Overflow (overdose)
07 Memory full (Mémoire pleine)
11 Division by zero (Division interdite par zéro)
13 Type mismatch (Type incompatible)
14 String space full (espace chaîne pleine)
15 String too long (prend une taille en dessous)
16 String expression too complex (chaîne trop complexée)
18 Unknown user function (fonction inconnue)
22 Operand missing (il manque un truc)
32 Broken in (c'est foutu)
```

Pour tester les autres valeurs, lancez cette routine dans une adresse de la mémoire (par exemple en &A000) et tapez : `CALL &A000, (numéro d'erreur de 0 à 32)`

Pour terminer, on va créer le ou la ou l' RSX "SWAP" avec contrôle des erreurs. On a déjà vu comment échanger deux variables à virgule. Cette fois, l' RSX sera capable d'échanger toutes les sortes de variables de même type entres elles.

```
ùSWAP, @var1%, @var2%
ùSWAP, @var1!, @var2!
ùSWAP, @var1$, @var2$
```

ETIQUETTE	INSTRUCTION	COMMENTAIRE
FIXE_RSX	ORG #A000	;; ùSWAP, var1, var2 (VARIABLES TOUS TYPES)
	LD A, #C9	; EMPECHER DE RELANCER LE PROCESSUS POUR EVITER
	LD (FIXE_RSX), A	; QUE LES RSX INTERNES (ùDIR, ùA, ...) PLANTENT
	LD BC, ADR_NOM	; 'BC' POINTE SUR L'ADRESSE DES NOMS DES RSX
	LD HL, OCTET_X4	; 'HL' POINTE SUR 4 OCTETS RESERVES POUR LES RSX
	JP #BCD1	; VECTEUR QUI FIXE LES RSX EN MEMOIRE
ADR_NOM	DW LIST_NOM	; 'BC'=ADRESSE DES NOMS DES RSX
	JP RSX_SWAP	; SAUT VERS LA ROUTINE RSX_SWAP
LIST_NOM	DEFB "SWA","P" + #80	; LISTE DES NOMS DES RSX
	DB 00	; FINR DE LA LISTE DES NOMS PAR UN OCTET A 0
OCTET X4	DB 00,00, 00, 00	; 'HL'= ADRESSE DE 4 OCTETS DE LIBRE
RSX SWAP	LD C, 5	; 'C'=ERREUR 5 (Improper argument)
	CP A, 2	; VERIFIER LE NOMBRE DE PARAMETRE
	JR NZ, ERREUR	; LE NOMBRE DE PARAMETRE NE CORRESPOND PAS
	DEC DE	; 'DE'= ADRESSE - 1'
	LD A, (DE)	; 'A'= INDICATEUR DU TYPE DE LA VARIABLE
	INC DE	; REPLACER 'DE' SUR L'ADRESSE
	LD C, 13	;'C'=ERREUR 13 (Type mismatch)
	OR A	; CONTROLER LE TYPE DE LA VARIABLE
	JR Z, ERREUR	; TYPE 0 N'EST PAS UNE VARIABLE VALIDE
	CP A, 8	; CONTINUER LE CONTROLE DU TYPE
	JR NC, ERREUR	; SI TYPE>7 CE N'EST PAS UNE VARIABLE VALIDE
	LD H, (IX + 3)	; CHARGER DANS 'HL' L'ADRESSE DE LA PREMIERE VARIABLE
	LD L, (IX + 2)	; 'HL'= ADRESSE VARIABLE_1
	DEC HL	; 'HL'= ADRESSE VARIABLE_1 - 1
	LD B, (HL)	; 'B'= INDICATEUR DU TYPE DE LA VARIABLE 1

```
            INC HL                  ; REPLACER 'HL' SUR ADRESSE_1
            CP A, B                 ; SI LES DEUX VARIABLES SONT DE MEME TYPE
            JR NZ, ERREUR           ; NON LES VARIABLES NE SONT PAS DU MEME TYPE
            INC B                   ; 'B'= (INDICATEUR +1) SUIVANT LE TYPE DE VARIABLE
   ECHANGE  LD A, (DE)              ; 'A'= VALEUR DE L'OCTET VARIABLE 2
            LD C, (HL)              ; 'C'= VALEUR DE L'OCTET VARIABLE_1
            LD (HL), A              ; 'HL'= VALEUR DE L'OCTET (DE)
            LD A, C                 ; 'A'= VALEUR DE L'OCTET VARIABLE_1
            LD (DE), A              ; 'DE'= VALEUR DE L'OCTET (HL)
            INC HL                  ; PASSER A L'OCTET SUIVANT
            INC DE                  ; FAIRE PAREIL POUR VARIABLE_2
            DJNZ, ECHANGE           ; ECHANGER LES OCTETS
            RET                     ; ET RETOURNER AU BASIC

ERREUR                              ; AFFICHE L'ERREUR DONT LE NUMERO EST DANS 'B'
            CALL #B915              ; CE VECTEUR RETOURNE LA VERSION CPC DANS 'A'
            LD HL, #CB55            ; 'HL'=VECTEUR DES ERREURS (CPC 664 ET 6128)
            OR A                    ; SI 'A' >0 ALORS CPC 664 OU 6128
            JR NZ, CPC_OK           ; C'EST UN CPC 664 OU 6128
            LD HL, #CA94            ; 'HL'=VECTEUR DES ERREURS (CPC 464)
   CPC_OK   LD (VECT_ERR + 1), HL   ; INSCRIRE L'ADRESSE DU VECTEUR ERREUR
            CALL #B900              ; VECTEUR D'ACCES DE LA ROM SUPERIEURE
            LD A, C                 ; RECUPERER LE NUMERO D'ERREUR QUI EST DANS 'C'
            LD E, A                 ; LE NUMERO D'ERREUR DOIT ETRE DANS 'A' ET 'E'
 VECT_ERR   CALL #CB55              ; SAUT AUTOMODIFIE DU VECTEUR QUI AFFICHE LES ERREURS
            JP #B903                ; VECTEUR FERMETURE ROM SUPERIEURE
```

Ce programme en lui-même ne fait que 94 octets avec la gestion d'erreurs et la création des RSX inclus. Soit moins que la moitié d'une ligne BASIC.
Vous pouvez bien sur l'optimiser et le prendre pour base pour vos prochains programmes.

N O T E

Astuce : Connaître l'adresse de lancement d'une routine RSX.

Dans le cas où on désirerait faire une commande RSX déplaçable, il est parfois utile de connaître l'adresse ou la routine de cette commande sera installée. Il faut pour cela, remonter à travers le registre 'IX' au paramètre zéro et charger la valeur dans un des registres.

Prenons l'exemple de la commande RSX ùCLIC sans paramètre est chargeons l'adresse d'exécution de la routine dans le registre 'HL'.

```
RSX_CLIC
LD H, (IX + 1)
LD L, (IX + 0)
```

Et mieux encore, le registre 'DE' contient lui aussi l'adresse d'exécution de la routine.

Et s'il y a un ou des paramètre(s), alors il faudra remonter au paramètre zéro comme dans l'exemple suivant : `ùCLIC, 32`

```
RSX_CLIC
LD H, (IX + 3)
LD L, (IX + 2)
```

Dans le cas où il peut y avoir des paramètres optionnels :

`ùSPOKE, adresse, var1[,var2] [,var3]...`

Puisque l'on sait que le registre 'A' contient le nombre de paramètres.

```
RSX_SPOK
ADD A, A
LD D, 0
LD E, A
ADD IX, DE
LD H, (IX + 1)
LD L, (IX + 0)
```

Ce dernier exemple modifie la valeur du registre 'IX'. il faudra penser à lire la valeur des paramètres dans le sens inverse (IX - 1) (IX - 2)..., ou plus simplement avec un 'PUSH/POP IX" pour revenir au début.

N O T E

A savoir : Stockage du nom d'une variable dans la mémoire.
On sait que le type de la variable peut-être récupéré à l'adresse – 1 quand le pointeur de variable est présent. Mais que ce cache t'il avant l'indicateur ?

Pour l'exemple, nous allons créer une variable de type entier : `compte%=500`

Regardons le tableau ci-dessous :

NOM DE LA VARIABLE						TYPE	REGISTRE	
c	o	m	p	t	e	1	IX + 1	IX + 0
#43	#4F	#4D	#50	#54	#C5	1 = ENTIER	E	D
-7	-6	-5	-4	-3	-2	ADRESSE -1	ADRESSE	

En partant de la droite, nous avons l'adresse dans le registre 'DE' et 'IX' et à cette adresse – 1 octet, on a le type de la variable dont sa valeur ici, est égal à '1' (type entier).
A l'adresse – 2 octets commence le nom de la variable par la fin:

```
Adresse-2 = #C5 soit le caractère 'e' converti en majuscule + #80
Adresse-3 = #54 soit le caractère 't' converti en majuscule
Adresse-4 = #50 soit le caractère 'p' converti en majuscule
Adresse-5 = #50 soit le caractère 'm' converti en majuscule
Adresse-6 = #50 soit le caractère 'o' converti en majuscule
Adresse-7 = #50 soit le caractère 'c' converti en majuscule
```

Le nom de la variable a été converti en majuscule et la valeur du dernier caractère a été augmentée de + #80 pour indiquer la fin du nom de la variable.

Il est alors possible de modifier les noms des variables même à partir du BASIC comme le montre l'exemple suivant :

EXEMPLE :

```
taxe%=26
PRINT taxe%
  26
POKE @taxe% - 2, ASC(''I'')+ &80
PRINT taxi%
  26
```

Vous trouverez peut-être une utilité à cette possibilité.

La lecture de ce guide est maintenant terminée. J'espère que vous aurez trouvé les réponses que vous cherchiez.

Quelques sites qui m'ont beaucoup aidé :
https://www.cpc-power.com/
https://amstrad.eu/
https://acpc.me/
https://asmtradcpc.z80-8bits.fr/

Et d'autres que j'apprécie aussi :
https://jerres12.net/border0/
http://crazypiri.eu/
https://cpcrulez.fr/
https://fr.wikibooks.org/wiki/Programmation_Assembleur_Z80
http://quasar.cpcscene.net/
http://tj.gpa.free.fr/
…
Liste non exclusive.

Au sujet de l'auteur :
Blog **:** http://retropoke.canalblog.com/
E-mail **:** philippe.moulin.fr@laposte.net

PS :
Si vous avez des informations supplémentaires à apporter sur les commandes RSX de l'Amstrad CPC et que vous aussi vous désirez les partager, vous pouvez me contacter à l'adresse e-mail ci-dessus.

N O T E

Philippe Moulin - 06/2022