Guide du Programmeur

La Casio Graph100

© Olivier COUPELON
www.gprog.tk
Olivier.COUPELON@wanadoo.fr

2nd Edition
Mai 2002

GENERALITES

Ce guide s'applique aux Casio Graph100, Graph100+, Fx2.0 et Fx2.0+. Sont contenu évolue à chaque mises à jours, veillez donc à surveiller son évolution, par le biais de mon site, www.gprog.tk.

La programmation de la Graph100 est un travail complexe car il requiert une parfaite maîtrise du matériel dont elle dispose.
Je vais au fur et à mesure de ce tutorial vous présenter point par point les différents éléments la constituant, ainsi que leurs programmations.

Table des matières :

1° La connaissance du système

Plusieurs éléments de la graph100 doivent vous être expliqués avant de pouvoir débuter la programmation. La graph100 possède notamment son propre système d'exploitation, Rom-dos 6.2 de Datalight (www.datalight.com). Il est compatible à 100% avec MS-DOS de Microsoft, ce qui rend la programmation aisée. Cependant, nous le verrons par la suite, mieux vaut essayer de s'en passer pour accélérer la vitesse d'exécution des programmes. Car la graph100 n'est autre qu'un ordinateur ayant eu son heure de gloire dans les années 1980.

Elle est composée de:

- Un Microprocesseur 16bit **Nec V30Mx** cadencé à environ 8 Mips (Million d'instructions par secondes).
- 256Ko de mémoire **Sram**
- Mémoire **Rom** de 4Mo contenant la configuration originale de la graph100
- 1Mo de mémoire **Flash**, celle-ci ayant la capacité d'être réécrite plusieurs fois tout en conservant les informations qu'elle contient
- Un clavier (peu pratique pour saisir des textes, et non standard quand aux codes des touches)
- Un écran LCD de 128 pixels de large et de 64 pixels de hauteur. Il est capable d'afficher jusqu'à quatre niveaux de gris (gris foncé, gris, gris clair et blanc), bien que seulement trois ne soient réellement accessibles, c'est-à-dire visibles. Il peut également n'afficher qu'un noir prononcé et du blanc, c'est le mode que vous connaissez et que vous pouvez observer quand vous utilisez normalement la calculatrice.
- Un port de communication pouvant communiquer à des vitesses allant jusqu'à 115200 bauds par seconde (double d'un modem classique) mais dans un sens uniquement à la fois.

2° Outils nécessaires à la programmation

Il s'agit de trouver le meilleur compilateur pour le langage choisi. Or ici, on a l'embarras du choix, car la programmation pour PC ne date pas d'hier, et les compilateurs en sont d'autant plus nombreux. Voici donc ce que je considère comme être les meilleurs compilateurs pour le graph100, pour les langages assembleur et C, langages eux-mêmes les mieux adaptés à la graph100 en raison de la rapidité du premier, et de la souplesse du second.

Pour l'Assembleur je conseille l'utilisation de **Nasm** (Netwide Assembler), compilateur assembleur gratuit et très performant, n'ayant rien à envier à Masm de Microsoft ou Tasm de Borland. De plus, il existe un environnement graphique pour ce compilateur, ce qui simplifie grandement son utilisation.

Pour le langage C, mon choix se tourne vers deux compilateurs, ayant chacun leurs avantages. **Turbo C** de Borland en version 2.01 (gratuite, www.borland.com) ou 3.0 (payante mais plus puissante). N'envisagez pas l'acquisition de versions ultérieures, car la 3.1 et la 4.5 compilent uniquement des exécutables pour Windows. La 4.0J (en Japonais) quand à elle, est moins performante pour notre processeur.

Un autre compilateur gratuit et performant est aussi disponible, **Pacific C**. Bien que complexe à utiliser, il contient des fonctions avancées comme la compilation spécifique pour les processeurs Nec V25 (proche du notre) ainsi que la possibilité de créer des binaires (qui permettent de se passer du système d'exploitation, mais cette fonction est payante).

L'outsider ici s'appelle **Paradigm C++**. Ce compilateur à la faculté de faire tout ce que font les compilateurs précédents, compile spécifiquement pour les Nec V30 (entre autre), et fonctionne intégralement sous Windows. Ce petit bijou a malheureusement un défaut rédhibitoire, celui de coûter la bagatelle de 2495$ US, environ 2846€ (Si quelqu'un le possède, j'aimerais qu'il m'informe sur sa compatibilité avec la graph100, car je n'ai jamais pu l'utiliser en version complète, la version de démonstration étant bien trop limitée).

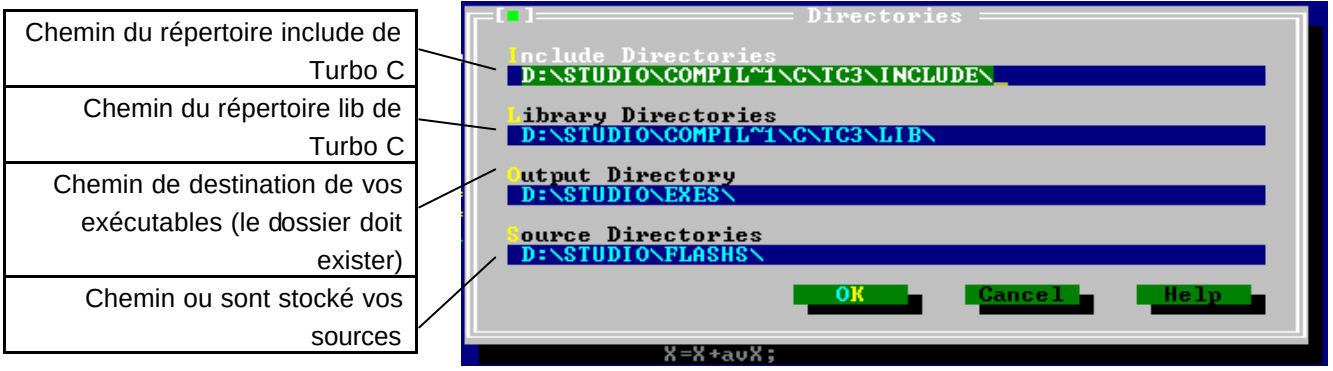
Le système d'exploitation sur lequel vous développez à aussi son importance, car comme vous l'avez peut-être déjà remarqué, tous les programmes cités ci-dessus fonctionnent sous MS-DOS et tendent à ne plus être compatibles avec les dernières versions de Windows.

2.1 Turbo C 3.0

Plusieurs paramètres de Turbo C 3.0 doivent être correctement réglé si l'on veut pouvoir compiler des programmes fonctionnels et de taille minimale. Pour lancer turbo C, il vous suffit d'exécuter le fichier TC.EXE dans le répertoire bin. L'éditeur de Turbo C s'ouvre alors. C'est depuis son menu que nous allons opérer.

Les voici :

- Sous l'onglet *Option -> Compilers -> Code Generation*, sélectionnez Tiny.
- Sous *Option -> Compilers ? Advanced Code Generation* choisissez dans Floating Point soit **none** si vous n'utilisez pas ne nombres a virgule, soit **Emulation**. Sélectionner également 80286 sous Instruction Set. Enfin sous Options, décochez **Debug info in Objs** et sélectionnez **Fast floating point** si vous utilisez les nombres a virgule.
- Sous l'onglet *Option -> Directories*, veillez à remplir correctement les 4 champs suivants :



- Sous *Environnement -> Préférences* choisissez 43/50 lines, vous aurez plus de lignes de code à l'écran ce qui améliore la lisibilité.

Si vous décidez d'utiliser des fichiers annexes de votre création pour alléger vos sources, vous avez deux possibilité : Les réunir dans un projet (pour créer un projet faire *Project -> Open Project* puis taper le nom du fichier projet que vous désirez créer), ou plus simplement, avant de compiler, faire *File -> Change Dir* et sélectionner l'endroit ou sont stocké vos fichier, mais alors vous devrez réaliser cette opération à chaque fois que vous redémarrez Turbo C.

Pour compiler un nouveau programme, il vous suffit de l'ouvrir ou de le créer avec le menu File, puis de cliquer sur *Compile ->Build All*. Votre programme, s'il ne comporte pas d'erreur, est alors disponible dans sa version exécutable dans le répertoire que vous avez sélectionné grâce au menu Directories.

Ces paramètres sont en parties valables pour les autres versions du compilateur Turbo C de Borland.

2.2 NASM (Netwide Assembler)

Nasm se pilote a peu près de la même façon que Turbo C. En fait toutes les interfaces Dos se ressemblent un peu. Pour accéder à l'éditeur graphique, il vous suffit de télécharger (si il n'est pas présent) le menu Nasm-ide. Ensuite, rendez-vous dans le répertoire de Nasm, et lancez NASMIDE.exe. L'interface graphique apparaît.

- Sous *Options -> Assembler* : dans l'encadré **Nasm Location**, entrez le chemin complet et le nom de Nasm. Par exemple c:\Nasm\nasm.exe. Sous Target, choisissez également **Com executable binary file**
- Sous *Options ->Directories* : remplissez les deux champs de la même manière que pour Turbo C ci-dessus
- Sous *Options ->Environment* : choisissez **43/50 lines**.

Ensuite, pour créer et exécuter un programme, faites *File ->New*. Entrez dans l'encadrer bleu votre code, puis sauvegardez le. Pour le compiler, faites *Assembler ->Assemble* puis *Build*. Votre programme est compilé et prêt à être exécuter (si aucune erreur n'a été détectée).

3° Affichage d'un texte

Afficher un texte sous graph100, c'est exactement la même chose que d'afficher un texte sous MS-DOS.

Pour afficher un texte à l'écran, le bios de la graph100 ainsi que Rom-dos mettent à notre disposition une série d'interruptions, que nous allons utiliser. Cependant, en C, leur utilisation est transparente, mais vous devez savoir que vos programmes y font appel, ça peut servir si vous souhaitez développer sans Rom-dos. Comme rien ne vaut un bon exemple, en voici deux, un en C, l'autre en Assembleur.

Code C affichant le texte Bonjour a l'écran :

```
#include <stdio.h>      // Contient le prototype de la fonction printf();

void main(void)         // Début de la fonction principale, c'est le corps du
programme
{
printf("Bonjour");      // Syntaxe permettant d'afficher le texte bonjour a
l'écran
}
```

La même chose en Assembleur pour Nasm :

```
[BITS 16]                   ; Nous compilons pour un processeur 16 bit
[ORG 0x0100]                ; Nous créons un .com
[SEGMENT .text]             ; Début du segment text, qui contient le code
exécutable de
                            ; Notre programme
mov si,Message              ; Met l'adresse mémoire de la chaîne de caractères
message
call AfficheText            ; dans le registre Si, puis appelle la routine
AfficheText

mov ax,0x4C00               ; Quitte le programme.
int 0x21

AfficheText:                ; Début de la routine AfficheText
mov ah,0eh                  ; La fonction, de l'interruption à utiliser
                            ; est la 0x0e
xor bl,bl                   ; Elle requiert ici que bl vaut 0
.Chsuite mov al,[ds:si]     ; al doit contenir le caractère à afficher
         cmp al,'$'         ; Si al contient le caractère $,
                            ; on a atteint la fin de la chaîne
         je .Chfin          ; Dans ce cas on saute à la fin de la routine
         int 10h            ; Sinon, on affiche le caractère dans al,
         inc si             ; On passe au caractère suivant
         jmp .Chsuite       ; Et on revient à la capture du caractère dans al
.Chfin   ret                ; Ici on quitte la routine pour revenir au
                            ; programme principal

[SEGMENT .data]             ; Zone de définition des données

Message db 'Bonjour$'       ; On défini la chaîne de caractère Message.
```

Un bref regard sur ces listings nous permet de voir que le code Assembleur est bien plus long que le C. En fait, quand on programme en Assembleur, on écrit quasiment ce que le processeur comprend, les mots étant remplacé par un code binaire.

En compilant ces deux exemples, on assiste à autre phénomène découlant du langage utilisé, qui influe sur la taille du fichier. Selon le compilateur, le premier exemple prendra entre 4000 et 8000 octets, alors que le programme assembleur en prendra 50. Car le C utilise des en-têtes, des fonctions prédéfinies, qui sont nécessaires au bon déroulement du programme, mais qui en contrepartie le ralentissent, le grossissent. Le C reste cependant le langage le plus rapide de sa catégorie.

4° Le Clavier

Le Clavier est l'élément à partir duquel l'utilisateur va utiliser votre programme. Il est donc nécessaire d'en connaître parfaitement son utilisation et son fonctionnement. 3 types d'accès sont possibles, avec leurs avantages et leurs inconvénients.

4.1 Le Clavier – Accès par interruptions

C'est la méthode la plus simple d'accès au clavier. Elle consiste simplement à appeler une interruption du dos ou du bios, et d'interpréter leur réponse.

LISTE NON EXHAUSTIVE DES PRINCIPALES INTERRUPTIONS D'ACCES AU CLAVIER			
Fonctions	**Code**	**Sortie**	**Commentaires**
Attend qu'une touche soit pressée	`mov ah,08h` `int 21h`	valeur de la touche dans AL	
Teste si une touche est pressée met sa valeur dans le buffer (mais n'en attend pas)	`mov ah,01h` `int 16h`	Met la valeur de la touche dans le buffer	Si aucune touche n'est pressée, on sort par un jz Fonction rapide, mais bloque l'accès au menu
	`mov ah,0bh` `int 21h`		Plus lent que précèdent
Lit une touche dans le buffer (si il y en a une)	`mov ah,07h` `int 21h`	Al prend la valeur de la touche pressée	Vide le buffer
	`mov ah,08h` `int 21h`		
	`mov ah,10h` `int 16h`		Cette fonction est plus rapide que les autres.

Remarques : L'interruption la plus rapide est toujours 16h, car c'est le bios qui gère l'accès au clavier, tandis que l'interruption 21h est une interruption du dos.

4.2 Le Clavier – Accès direct en mémoire

Entrons plus en détail dans le fonctionnement du clavier : Toute pression d' une touche du clavier génère l'interruption 9. Cette action va mettre en mémoire la valeur de la touche qui a été activée. Cette zone est située en 0000h : 041Ch. Pour y accéder, il suffit donc de faire :

En C

```
peekb(0x41, 0xC) ;  // cette fonction renvoie la valeur de la touche pressée
```

En asm

```
mov ax,0x41
mov es,ax
mov si,0xC
mov al,[es :si] ;  al prend la valeur de la touche initialement pressée.
```

Grâce à ceci, vos programmes fonctionneront bien plus vite sans prendre plus de place qu'un simple `getch() ;` . Mais ce type d'accès peut ne pas être utile, car la fonction `getch() ;` ou les interruptions offrent des délais de répétitions très appréciables et bien dosés, ce qui peut être utile dans une application.

4.3 Le Clavier – Accès par les ports

L'accès au clavier directement à travers les ports de communication offre des avantages exceptionnels en terme de vitesse, et surtout de gestion multiple de touches. C'est le moyen le plus direct et rapide d'accéder à une touche. Malheureusement, ce n'est pas aussi simple que pour les types d'accès précédents.

Pour savoir si une touche a été activée, la recherche va s'effectuer par ligne de clavier, la réponse sera ultérieurement communiquée par colonne de clavier.

Organisation du clavier par ligne de touches :

0 : On
1 : 0 a EXE
2 : 1 a -
3 : 4 a /
4 : 7 a DEL
5 : a+b/c à ?
6 : X,?,T à tan
7 : ALPHA a ESC
8 : SHIFT a MENU
9 : touches directionnels
10 : F1 à F6

Nous devons prendre une variable de 16bit initialisée a 0 et positionner le nième bit a 1, en fonction du numéro de ligne. Il s'agit d'un simple décalage d'au maximum 10bit puisqu'il y a 10 lignes. Par exemple, si on souhaite tester les touches directionnelles, notre variable aura cette forme : 0000000001000000

Pour le décalage, en C on écrira :

```
unsigned short Var ; //crée une variable de 16bit
…
Var=0 ;
Var<<n ; //ou n est le numéro de la ligne
```

En Assembleur :

```
mov ax,0
shl ax,n ; ou n est le numéro de la ligne
```

Une fois notre variable prête, on l'écrit dans le port numéro 13h. Mais la taille d'un port étant de seulement 8bit, écrire 16bit en 13h aura pour effet d'écrire également dans le port 14h, mais ne vous inquiétez pas ceci est tout a fait normal. De cette manière, le contrôleur clavier a pris connaissance de la touche que nous souhaitons tester, ou plutôt de la ligne car il s'agit pour l'instant d'une ligne. Le contrôleur clavier renvoi alors la valeur de cette ligne dans le port 13h, sur 8bit seulement.

Pour les opérations d'entrée sortie en C, on utilisera `inportb` et `outport`, en assembleur `in` et `out`.

Nous devons à présent lire la valeur envoyée par ce port et l'interpréter.
Si elle vaut 0, aucune touche n'a été pressée. Sinon, il faut examiner sa valeur binaire.
Cette dernière est donnée cette fois ci en fonction du numéro de colonne de la touche.

Le tableau ci-dessous représente la touche correspondant au croisement final des lignes et des colonnes :

Correspondance des touches clavier avec la lecture binaire

N° ligne	N° des colonnes							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0								On
1		0	.	x10	(-)	Exe		
2		1	2	3	+	-		
3		4	5	6	x	/		
4		7	8	9	Del			
5		A+b/c	xy	(	)	‘	?	
6		X,?,T	log	ln	sin	cos	tan	
7		Alpha	Vars	^	Esc			
8		Shift	Ctrl	Optn	Menu			
9		Gauche	Haut	Bas	Droite			
10		F1	F2	F3	F4	F5	F6	

Par exemple, si vous testez la ligne 10, et que la valeur colonne 01010000 est renvoyée, cela signifie que les touches F1 et F3 avaient été pressées.

Les algorithmes utilisés peuvent avoir plusieurs formes selon que l'on souhaite simplement savoir si une touche quelconque est pressée ou si l'on cible une touche précise.

Exemple de test en C (Cette fonction teste si une touche spécifique est pressée et renvoie 1 le cas échéant. On envoie comme paramètre le numéro de la ligne et la valeur renvoyée par le clavier si la touche était pressée) :

```
int keyport(int ligne,int valeur)        // Vérifie si la touche choisie
                                         // est pressée.
{
outport (0x13,(1<<ligne));
if (inportb(0x13)==valeur) return 1;     // La touche est bien pressée,
                                         // on renvoie 1
else return 0;                           // La touche n'est pas pressée,
                                         // on renvoie 0
}
```

Cependant, en utilisant ce système d'accès au clavier, vous allez rencontrer un problème.
Les touches pressées continueront à être mises dans le buffer touche (zone contenant une liste des touches pressées) de la graph100. Ceci est gênant si vous souhaitez par la suite utiliser un autre type d'accès clavier, il vous faudra d'abord vider ce buffer, puis seulement après capter une touche.
Pour résoudre ce problème, plusieurs solutions s'offrent à vous, ma préférée et qui donne les meilleurs résultats, consiste à désactiver l'interruption 9. Ceci offre de multiples avantages exposés ci-dessous :

- Le buffer touche n'est plus alimenté, car c'est le rôle de cette interruption.
- Le retour au menu par la pression de la touche Menu, et la modification du contraste par les touches Shit+Droite ou Shift+Gauche sont désactivés.
- Votre programme s'en trouve légèrement accéléré.

Il est à noter que cette interruption peut être restaurée à tout moment si on prend la précaution de sauvegarder son adresse initiale, en utilisant en C la fonction `dos_getvect()`
Pour la substituer par une fonction de votre choix (une fonction vide fera évidement l'affaire), utiliser `_dos_setvect()`.

En assembleur, il vous suffit de sauvegarder l'adresse de cette interruption lue dans la table d'interruption, puis de remplacer cette adresse par celle de l'interruption 0xFF. (Voir chapitre sur les interruptions).

5° La Mémoire

Comme nous l'avons vu précédemment, il existe plusieurs types de mémoires internes dans la graph100, dont voici les noms et caractéristiques techniques :

Type	Nom	Temps d'accès	Taille	Opérations
Sram	Nec PD442000GU-B85X-9JH	85ns max	256Ko	Lecture / écriture
Flash	Fujitsu MBM29LV800BA-90	90ns max	8Mbit soit 16x64Ko soit 1Mo	Lecture/ écriture
Rom Graph100	Nec PD23C32000L	140ns max	32Mbit soit 4Mo	Lecture
Rom Graph100+	Oki MR53V3202K-24	120ns max	32Mbit soit 4Mo	Lecture

Par la suite, nous ne ferons pas de différence entre la rom de la graph100 et celle de la graph100+ car elles s'utilisent de la même manière.

5.1 La mémoire Sram

C'est la mémoire vive de la graph100. Elle se situe entre les adresses 0000h : 0000h et 4000h : 0000h. Elle ne mesure que 256 Ko, alors que le plupart des ordinateurs équivalents possédaient 640 Ko. Ceci va grandement nous pénaliser dans la réalisation de nos programmes, car Rom-dos, pour compenser cette petite taille, nous astreint à utiliser des programmes ne mesurant pas plus de 64ko, ce qui ne simplifie pas les choses. Cependant, le format de fichier Rxe de datalight permet d'outrepasser cet inconvénient en modifiant l'entête des exécutables. Les outils permettant cette transformation sont inclus dans les kits de développements de Datalight, kits qui coûtent plusieurs milliers de dollar, c'est pourquoi nous allons nous restreindre financièrement et utiliser des programmes ne mesurant pas plus de 64Ko.
Cette mémoire à la particularité d'être persistante, il est donc possible d'éteindre et de rallumer la calculette, sans rien perdre du travail en cours.
De plus, il est a noter que le buffer vidéo et les programmes basics sont logés dans cette mémoire, aux adresses 1A20h :0000h et 1C20h :0000 respectivement. Elle contient également la table d'interruption, et des données du bios, comme dans un ordinateur en mode réel. Car ici, vous l'aurez compris, pas question de mode protégé.

Adresse	Contenu
0000h : 0000h - 0000h : 03FFh	Table des vecteurs d'interruptions
0000h : 0400h - 0000h : 04FFh	Variables du bios
0000h : 0500h - 1000h : A1FFh	Zone dos, pour charger les programmes, les variables, l'environnement…
1000h : A200h - 1000h : C1FFh	Buffer mémoire vidéo
1000h : C200h - 3000h : FFFFh	Programmes basics, matrices, listes, tout les éléments de calculs de la graph100

5.2 La rom

La rom de la graph100 mesure 4 Mo. Elle contient les programmes de calcul de la graph100, une sauvegarde de la zone système et du lecteur A :, le menu constructeur, et les langues de la calculette. Voici la manière dont elle est organisée par segments:

Segment :	Contenu :
0000h - 0FFFh	Zone système de base
1000h - 2FFFh	Menu constructeur
3000h - 3FFFh	Sauvegarde Lecteur A :

4000h - 5FFFh	Langues pour le système
6000h - 6FFFh	Vide
7000h - 7FFFh	? - Bout de programmes
8000h - 3FFFFh	Lecteurs B : a K :

Mais ces adresses sont les adresses physiques internes de la rom, et en aucun cas celles auxquelles on trouve ces données en mémoire.

Pour y accéder depuis la mémoire, il faut « mapper » une zone de ce disque (rom) dans la mémoire. Sous ce terme barbare, j'entends en fait qu'on a besoin de dire au contrôleur de la rom quelle est la zone mémoire de 128 Ko à laquelle on veut accéder, mais également l'endroit où il doit la placer en mémoire.

Il est impossible d'accéder à la totalité des 4Mo directement pour la simple raison que l'adresse maximale qu'on peut atteindre avec le système actuel de ciblage d'adresse est F000h : FFFFh, c'est-à-dire 1Mo, alors que la Rom en fais 4.

Pour ce faire, il va falloir écrire dans un certain port une certaine valeur suivant l'endroit où nous souhaitons placer nos 128 ko et quelle partie de la rom nous souhaitons charger a cet endroit.

Le tableau suivant montre dans quel port écrire pour « mapper » une zone de rom dans la zone mémoire de la graph100 correspondante.

Port d'accueil	Zone mémoire correspondante	Commentaires
54h	0000h - 1FFFh	**Ne pas utiliser** ces ports, car **zone mémoire vive**
55h	2000h - 3FFFh	
56h	4000h - 5FFFh	
57h	6000h - 7FFFh	
58h	8000h - 9FFFh	
59h	A000h - BFFFh	
5Ah	C000h - DFFFh	

Les valeurs à écrire dans ces ports s'étalent de C0h à DFh, C0h ciblant les 128 premiers Ko de la rom, et DFh les derniers (ce qui cible bien 4Mo, 32x128Ko).

5.3 La flash

C'est elle qui va contenir nos programmes. Elle mesure 1Mo.

Segment :	Contenu :
0000h - 0FFFh	Zone système
1000h - 1FFFh	Lecteur A :
2000h - 2FFFh	Langue en cours d'utilisation
3000h - 3FFFh	Presque vide
4000h - FFFFh	Contient les flashs envoyées par l'utilisateur, c'est-à-dire normalement les lecteurs L : a Q :

On y accède exactement de la même manière qu'on accède à la rom, seule la valeur à envoyer au port change. On y envoie les valeurs de A0h à A7h.
A0h mappe les 128 premiers Ko en mémoire, A7h les 128 derniers.

La disposition mémoire présentée précédemment est celle de base, à l'acquisition de la calculette. Mais la flash étant réinscriptible, il est possible que des programmes modifient son contenu. Quoi qu'il en soit, si une telle opération est réalisée et que Rom-dos s'en trouve altéré (notamment à la suppression du lecteur A:), le système va de lui-même restaurer la flash dans cet état initial, l'écriture sur la flash est donc sans danger.

Mais y écrire n'est pas aussi simple qu'y lire. Car pour y lire, il suffit de faire peek() en C pour accéder aux données qu'on vient de mapper , alors que pour y écrire, un simple poke() ne suffit pas. En fait,

les données sur la flash sont protégées en écriture, il faut donc passer par plusieurs étapes avant de pouvoir modifier quoi que ce soit.
Il est impératif de savoir que sur la flash, lors d'une écriture, la seule modification possible est de remplacer (en binaire) des 1 par des 0, l'inverse étant physiquement impossible à réaliser sur un bit isolé. La seule possibilité de contourner cet obstacle, est de formater le secteur entier que l'on souhaite modifier, ce qui positionne tous les bit de ce secteur à 1. Une fois cette opération réalisée, on peut écrire ce que l'on veut, les 1 resteront des 1, ou se transformeront en 0, si nécessaire.

Cette opération est très délicate, car formater détruit tous les fichiers et données présents dans le secteur. Il faut donc dans une première étape sauvegarder les données en mémoire, puis formater, ensuite modifier les données en mémoire, et enfin réécrire l'intégralité des 64 Ko modifiés.

Pour sauvegarder ces données, on peut utiliser l'espace de la ram entre 2000h : 0000h et 4000h : 0000h, espace qui peut cependant être occupé par des fichiers basic, ce qui entraîne leur perte et le cas échéant des messages d'erreurs lors du prochain arrêt de la calculatrice (au pire la perte totale des fichiers basic).

5.4 Ecriture sur la flash

Considérons donc que le fait de perdre les données importe peu pour le moment, et concentrons nous maintenant sur la manière d'effectuer les formatages et les écritures sur la flash. Pour ce faire, vous devez tout d'abord mapper la zone à écrire en mémoire.
Considérons que c'est le lecteur L :, que nous avons mappé en 4000h : 0000 (code à exécuter : `outportb(0x56,0xA2) ;`).
Nous souhaitons maintenant formater le lecteur L:
Il nous suffit simplement de le demander au contrôleur de la flash. Pour cela, nous lui donnerons des instructions, par écrit, à des endroits précis du secteur
Code à écrire en asm :

```
      mov ax,0x4000                    ;nous plaçons es au début de notre
      mov es,ax                        ;secteur
      mov [es:0xAAA],byte 0xAA         ;puis nous écrivons 6 valeurs
      mov [es:0x554],byte 0x55         ;à des endroits bien précis
      mov [es:0xAAA],byte 0x80         ;le contrôleur comprendra alors
      mov [es:0xAAA],byte 0xAA         ;que l'on souhaite formater
      mov [es:0x554],byte 0x55         ;le secteur en cours
      mov [es:420 ],byte 0x30          ;420 ou une autre valeur,
                                       ;peu importe ici
            ; Début de formatage de la  flash
```

A ce stade il reste encore deux choses à faire. La première est de tester la fin du formatage qui prend quelques secondes. Pour ce faire, on va lire une zone du secteur en format. Ceci ne renverra pas la valeur présente à l'endroit lu, mais l'état d'avancement de l'opération. Si le 6eme bit de l'octet renvoyé est nul, c'est que le formatage est terminé.

Voici un exemple en assembleur du test de fin d'opération :

```
.Attend     mov bl,[es:si]          ; Demande à la flash deux octets
                                    ; de vérification (on lit n'importe
      mov bh,[es:si]                ; où dans le secteur, pas à un endroit
      xor bl,bh                     ; spécifique.)
      and bl,0x40                   ; Le 6eme bit est-il mis ?
      jz .fin                       ; si oui, la flash est formatée
      and bl,0x20                   ; Sinon on regarde le 5eme bit
      jz .Attend                    ; s'il est mis c'est que le formatage
                                    ; n'est pas terminé,
      code en cas d'erreur          ; sinon il y a eu une erreur
.fin  ret
```

Les exemples précédents sont tous en asm car ces opérations demandent une exécution rapide. Vous pouvez les inclure dans vos programmes en C en prenant soin de modifier leurs syntaxe si besoin, et de rajouter une des directives proposées ci-dessous, selon votre compilateur :

```
asm {lignes de code assembleur}     // Pour Turbo C
ou
#asm                                // Pour Pacific C
lignes de code assembleur
#endasm
```

Vous pouvez également les re-écrire directement en C, avec des peekb et des pokeb.

Mais souvenez vous que chaque secteur ne fait que 64ko et qu'un lecteur (le lecteur L en l'occurrence) mesure 128 Ko. Il nous faut donc par la suite mettre es à 5000h et recommencer les opérations précédentes.

Maintenant il serait souhaitable d'écrire sur notre flash vierge. C'est le même type d'opération que le formatage, voici une routine écrivant un octet sur la flash en Assembleur :

```
mov [es:0xaaa],byte 0xaa      ; 3 commandes pour prévenir la flash qu'on
mov [es:0x554],byte 0x55      ; souhaite écrire un octet.
mov [es:0xaaa],byte 0xa0
mov [es:si],al                ; al contient la valeur à écrire, et si pointe
                              ; vers l'offset du secteur où on souhaite écrire
```

Après ceci, il nous faut encore appeler la routine de vérification, celle présentée plus haut fonctionnant encore dans ce cas.

Voici un tableau regroupant les opérations possible à effectuer sur la flash :

	1er Cycle		2ème Cycle		3ème Cycle		4ème Cycle		5ème Cycle		6ème Cycle	
	Offset	Valeur	Offset	Valeur	Offset	Valeur	Offset	Valeur	Offset	Valeur	Offset	Valeur
Ecrire	AAAh	AAh	554h	55h	AAAh	A0h	?	?	-	-	-	-
Formater secteur	AAAh	AAh	554h	55h	AAAh	80h	AAAh	AAh	554h	55h	xxx	30h
Formater chip	AAAh	AAh	554h	55h	AAAh	80h	AAAh	AAh	554h	55h	AAAh	10h

Notes :

? : C'est vous qui choisissez l'offset et la valeur à écrire sur le segment en cours.
xxx : N'importe quel offset du segment en cours (donc n'importe quelle valeur).

Sachez enfin que tous les secteurs de la flash mesurent 64 Ko, sauf les 64 premiers Ko qui sont organisés en 4 secteurs de 16Ko, 8Ko, 8Ko, et 32Ko dans l'ordre.

Pour plus d'informations sur les fonctionnalités avancées de la flashs, reportez-vous à son manuel (en anglais).

5.5 Autre méthode d'accès à la flash/rom

Il est également possible d'accéder à la mémoire par l'intermédiaire de l'interruption 48h. Mais les valeurs à utiliser sont alors différentes. Voici comment fonctionne cette interruption. :

Interruption 48h	**Accès aux disques**	
Fonction 0		Ah = 0

Bl = numéro du segment mémoire dans lequel on veut mapper (0 = 0 ; 1 = 2000h : 0000h ; 2 = 4000h : 0000h…).

Bh ? 6.
Al = numéro de la zone à mapper selon la table suivant :

Zone mémoire flash	Numéro dans al
0000h - 1FFFh	0x40
2000h - 3FFFh	0x42
4000h - 5FFFh	0x44
6000h - 7FFFh	0x46
8000h - 9FFFh	0x48
A000h - BFFFh	0x50
C000h - DFFFh	0x52
Zone mémoire rom	**Numéro dans al**
0000h - 1FFFh	0x80
…	
3E000h - 3FFFh	0xBE

Fonction 1	Ah = 1

Bl = numéro du segment mémoire (0 = 0 ; 1 = 2000h : 0000h ; 2 = 4000h : 0000h…).
Bh ? 6.
Retour : Al prend la valeur de la zone mémoire mappée à l'endroit ciblé par Bl, valeur correspondant à celle vue ci-dessus par Ah=0.

Fonction 2	Ah = 2

Renvoie dans ax le mot écrit en 0040h : 00C6

6° Utilisation de l'écran

L'écran de la graph100 se compose de 8192 pixels, 128x64. Il est capable d'afficher soit en noir et blanc, soit en niveaux de gris. Pour pouvoir stocker des pixels, l'écran possède un buffer vidéo, situé habituellement à l'adresse 1A20h :0000h, dans la ram. Si vous écrivez dans cette zone, l'écran en sera directement affecté.

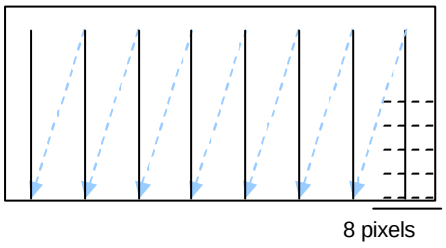
Pour sélectionner un mode, vous devez entrez sa valeur dans le port 2 de la calculatrice. Par exemple pour passer en mode DB, vous devrez faire :

En C :
```
outportb(2,0xDB);
```

En Assembleur :
```
mov al, 0xDB
out 2, al
```

6.1 Le mode normal C3

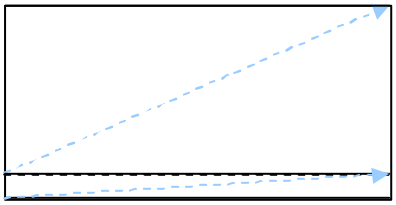
Le mode C3 est le mode utilisé par défaut par la graph100. C'est donc un mode noir et blanc, un bit représentant un pixel à afficher à l'écran. Comme le buffer contient toujours 8192 pixels, il mesure 8192 bit soit 1024 octet, 1 Ko. Les bits sont repartis par colonnes de huit, le début du buffer pointant vers le coin bas droite de l'écran, la fin vers le haut gauche.



Le schema ci-contre illustre la façon dont sont disposées les données de la mémoire video à l'écran.
Le probleme rencontré lorsque l'on souhaite allumer un pixel seul, c'est qu'on ne peut pas manipuler les données bit à bit, mais seulement octet par octet (1 octet = 8 bits). La méthode va donc consister à créer un algorithme permettant de se positionner sur le bon octet à traiter dans le buffer video, puis à utiliser une commande specifique au processeur Nec qui va permettre de ne modifier qu'un bit de cet octet, celui que l'on souhaite allumer (le cas echeant l'éteindre).

6.2 Le mode noir et blanc D3

Ce mode video est à l'heure actuelle encore très (trop) peu utilisé par les programmeurs. Il diffère du mode C3 uniquement par sa répartition du buffer video à l'écran, bien plus simple à utiliser. Le début du buffer pointe toujours vers le coin bas droite de l'écran, et la fin vers le haut gauche.



La seule différence est qu'il affiche le contenu du buffer lignes par lignes comme le montre le schéma ci-contre. Il est alors bien plus aisé et intuitif de dessiner quelque chose dans ce mode, et aussi légèrement plus rapide. Pour ce faire, on utilisera par exemple l'algorithme suivant.
L'affichage nécessitant une grande rapidité d'exécution, l'assembleur est encore une fois de rigueur :

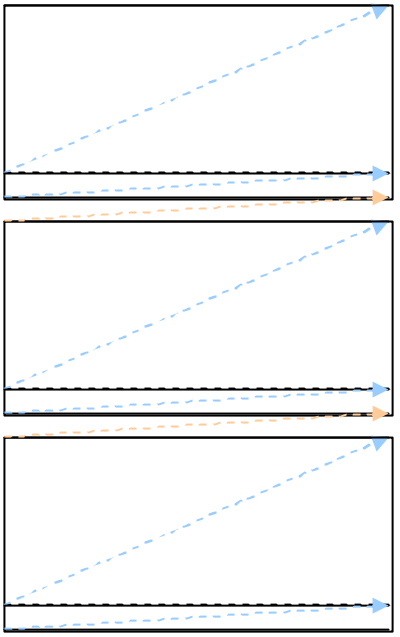
```
mov ax,0x1A20     ;On met dans es le segment du buffer vidéo
mov es,ax         ;on ne peut affecter directement es, on passe par ax

mov bx,x          ; X représente l'abscisse
mov si,y          ; Y l'ordonnée
shl si,4          ; Une ligne mesure 16 octet (128bit). si pointe vers la bonne
ligne
mov cl,bl         ; 0<=bx<128 Donc on peut manipuler bl au lieu de bx.
                  ; On place alors bl dans cl.
shr bl,3          ; Divise bl par 8. Le résultat est dans bl
and cl,7          ; On garde les 3 derniers bit de cl, c'est le reste de la
division
add si,bx         ; si pointe vers le bon octet (16*y+x)
db 0x0F,0x14,0x0C7  ; set1 bh,cl
```

```
                    ; Met à 1 le cl^ieme bit de bh (qui était nul jusque la)
                    ; Cette commande est expliquée plus précisément dans
                    ; le chapitre consacré au processeur
or es:[si],bh       ; On place bh en mémoire vidéo en prenant soin de ne
                    ; pas effacer les autres
```

Cette routine a été optimisée au maximum pour bénéficier du plus rapide des affichages point par point.

6.3 Le mode niveau de gris DB



Ce mode permet l'utilisation de trois niveaux de gris : le blanc, le gris et le gris foncé.
Son utilisation reste tout de fois assez simple, car il utilise un système de couches. Une couche est en fait une superposition de plusieurs pages dans le but de n'en afficher qu'une seule nuancée. Ces couches ont exactement la même structure que la couche unique utilisée dans le mode D3, sauf qu'il faut en utiliser trois consécutives. La taille du buffer vidéo passe alors de 1024 octets à 3072 octets. Mais seules deux couches sont réellement utiles ici, la première et la troisième. La seconde apporte une nuance quasiment imperceptible, donc inutile. Ensuite, le calcul est simple, plus on superpose de bits allumés, plus le résultat à l'écran est foncé. En considérant seulement les deux couches utiles, il en ressort la palette suivant :

0 bit allumés : blanc
1 bit allumé : gris clair
2 bits allumés : gris foncé

Ces couches sont contiguës, sans marque de distinction dans le buffer. Donc si la première commence (c'est le cas habituel) en 1A20h :0000, la seconde commencera en 1A60h :0000 et la troisième en 1A80h :0000. On y applique ensuite les règles de calcul présentées plus haut avec la possibilité de gagner quelques cycles d'horloges si on souhaite dessiner un pixel gris foncé en ajoutant simplement à la fin de la routine précédente le code suivant :

```
mov ax,0x1A80           ;On se déplace vers la couche suivante,
mov es,ax
or es:[si],bh           ; et on affiche le bit au même endroit
```

Il est bien évidemment possible dans toutes ces routines d'effacer un pixel en remplaçant simplement les lignes suivantes :

```
db 0x0F,0x14,0x0C7
or es:[si],bh
```

Que l'on remplacera par :

```
db 0x0F,0x12,0x0C7
and es:[si],bh
```

6.4 Le mode niveau de gris CB

Ce mode n'est pas très utile car son homologue vu ci-dessus permet d'effectuer les même opérations plus simplement. Sachez simplement qu'il fonctionne comme le mode DB, mais en prenant exemple sur le mode C3 (et non le mode D3).

6.5 L'interruption 7Ch

L'interruption 7Ch a pour but de contrôler l'écran. En voici l'action détaillée. Cette Interruption possède les fonctions 2, 3,0Ch, 0Dh, 0Fh, 20h, 21h, 22h, 23h, 24h, mais on ne connaît pas encore intégralement leurs fonctionnements.

Interruption 7Ch	**Contrôle du contraste**	**BIOS**
Fonction 22h	Ah=22h	

C'est cette fonction qui agit sur le contraste.

bl = 0 : Augmente le contraste

bl = 1 : Diminue le contraste

Pour l'utiliser, il suffit de mettre les valeur adéquates dans ah et bl et d'appeler l'interruption par une commande telle que `int 0x7C`.

La valeur renvoyée dans al est le niveau de contraste actuel.

Pour fonctionner, elle fait une demande au contrôleur de l'écran, sans passer par les ports de communication, simplement en écrivant une séquence précise de valeurs à un endroit donné que le contrôleur va recevoir et interpréter. C'est la même méthode que pour l'écriture sur la flash qui est utilisée ici. La fiche technique pourrait expliquer plus précisément son fonctionnement, mais on ne connaît pas le type d'écran de la graph100. (Probablement de marque Nec).

La séquence est écrite aux adresses contenues en E000h :0010h et E000h :0020h.

Séquence utilisée pour communiquer avec l'écran									
FFh	FFh	0	0	0	0	0	0	0	3Fh
C0h	80h	x	66h	0	FFh	FFh	0	0	0
0	0	0	0	3Fh	0	0	FFh	FFh	0
0	0	0	0	0	0	3Fh	C0h	A0h	x
66h	0	FFh	FFh	0	0	0	0	0	0
3Fh	4	0	0	FFh	FFh	0	0	0	0
0	0	0	3Fh	C0h	E0h	x	66h	0	FFh
FFh	0	0	0	0	0	0	0	3Fh	4
0									

Légende :

	Ecrire ces valeurs à l'offset contenue en E000h :0010h, au segment E000
	Ecrire ces valeurs à l'offset contenue en E000h :0020h, au segment E000
x :	Octet à écrire pour effectuer l'opération souhaitée.

Ce tableau se lit dans le sens de lecture classique, de gauche à droite puis de haut en bas. Il faut écrire ces valeurs aux adresses données séquentiellement commençant par la première valeur du tableau.

La valeur de x ici n'est pas encore clairement identifiée.

On peut constater que la longueur de la séquence à écrire n'est pas négligeable et prend beaucoup de place. Aussi, si la vitesse de changement de contraste importe peu dans vos applications, vous avez tout intérêt à utiliser l'interruption dédiée.

Fonction 24h	Ah=24h

Cette fonction a pour effet de rafraîchir l'écran. Tout comme la fonction 22h, elle utilise le système de séquence pour communiquer avec l'écran. Toutefois, si vous souhaitez intercepter cette fonction, préférez plutôt agir sur l'interruption 53h, interruption matérielle ayant entre autre comme effet d'appeler cette fonction. Ici, la valeur de x doit être préalablement lue en E000h :00E5h. La séquence à utiliser est la même que pour la fonction 22h.

7° Le port de communication

La graph100 possède un port de communication de type asynchrone, capable de transférer des données à des vitesses allant jusqu'à 115200bps. On y accède par l'intermédiaire des ports (attention tout de fois à ne pas confondre le port de communication et les port internes de la graph100).
Mais un problème majeur se pose car quand on l'utilise, il rend impossible toute utilisation de touches.

7.1 Configuration du port

Pour pouvoir fonctionner le port de communication de la graph100 a besoin d'être initialisé. Pour ce faire, il faut allumé les bits 5 et 6 du port interne 11h. Ce port renvoyant les données écrites sur lui-même, il suffit de lire sa valeur, de mettre les bits 5 et 6 a 1, et d'écrire la nouvelle valeur.
Assembleur :

```
in al,0x11  ;On lit la valeur du port 0x11
or al,0x60  ;0x60 = 01100000b cette instruction ne modifie
            ;que les bits 5 et 6 de al
out 0x11,al ;On réécris la nouvelle valeur.
```

Après ceci, le port de communication est a 4,92V DC, et donc prêt a communiquer.
Par la suite, il faut écrire dans le port interne 0x44 la valeur 0, ceci permet d'initialisé la communication.
Enfin, en fonction de la vitesse de transmission que vous souhaitez obtenir, vous devez envoyé dans le port 0x47 une des valeurs suivantes, la valeur du port 0x45 vous sera utile plus tard. Notez que plusieurs combinaisons peuvent être possibles, selon la vitesse choisie.

Port 0x47 / Port 0x44	0x0B	0x11	0x17	0x21	0x2B	0x41	0x5B	0x7F
0x70	14400 bps	9600 bps	7200 bps	4800 bps	3600 bps	2400 bps	1800 bps	1200 bps
0x74	28800 bps	19200 bps	14400 bps	9600 bps	7200 bps	4800 bps	3600 bps	2400 bps
0x78	57800 bps	38400 bps	28800 bps	19200 bps	14400 bps	9600 bps	7200 bps	4800 bps
0x7C	115200 bps	76800 bps	57800 bps	38400 bps	28800 bps	19200 bps	14400 bps	9600 bps

Une fois ces opérations terminées, le port de communication se trouve prêt à recevoir ou a envoyer des informations.

7.2 Envoi de données

- ✓ Procédez tout d'abord à la configuration du port (voir ci-dessus).
- ✓ Ensuite, vous devez mettre l'octet à envoyer dans le port 0x46.
- ✓ Mettez maintenant la valeur 0x41 dans le port 0x45.
- ✓ Nous allons maintenant utilisé la valeur restante du tableau vu pendant l'initialisation (0x70, 0x74, 0x78, ou 0x7C). Envoyez la valeur correspondant à la vitesse que vous avez choisi dans le port 0x44.

A partir de cet instant, la communication a débuté et l'octet est entrain d'être envoyé. Il faut maintenant attendre la fin de l'envoie de cet octet pour continué a utilisé le port de communication. Pour vérifié que l'octet a bien été envoyé, vous devez lire la valeur contenue dans le port 0x45 et attendre que le bit n°0 de ce port soit allumé. Pour cela, vous devez rédigez une boucle de test, comme celle présentée ci-dessous.

Assembleur :

```
Attendre :
in al,0x45  ;prend la valeur du port 0x45, qui n'est pas celle
            ;que l'ont y a écrite précédemment
and al,1    ;ceci préserve le premier bit de al (n°0),
            ;en mettant les autres a zéro.
```

```
Jz Attendre ;le bit n°0 vaut 0, donc l'envoie de l'octet n'est
            ;pas encore terminé, donc on recommence le teste.
```

Maintenant que l'octet est envoyé, on peut renvoyé d'autres octets. Pour cela, vous devez :

- ✓ mettre l'octet a envoyé dans le port 0x46.
- ✓ testé la fin d'envoi de l'octet.

Quand tout vos octets on été envoyés, vous devez fermer le port (voir 7.4).

7.3 Réception de données

- ✓ Procédez tout d'abord à la configuration du port (voir ci-dessus).
- ✓ Mettez la valeur 0x41 dans le port 0x45.
- ✓ Nous allons maintenant utilisé la valeur restante du tableau vu pendant l'initialisation (0x70, 0x74, 0x78, ou 0x7C). Envoyez la valeur correspondant à la vitesse que vous avez choisi dans le port 0x44.

Ici, vous devez tester l'arrivée d'un octet. Pour cela, vous devez lire la valeur du port 0x45, puis testé ses bits 1, 3 et 4. Si le bit 1 est mis et pas les deux autres, c'est qu'un octet est arrivé. Sinon, vous recommencé jusqu'à ce que le test soit concluant.
Assembleur :
Attendre :

```
in al,0x45      ;lis la valeur du port 0x45
and al,0x1A     ;conserve les bits 1, 3 et 4. Les autres sont mis a 0.
cmp al,0x2      ;test si seul le premier bit est mis ( 0x2 = 10b)
jne Attendre    ;si ce n'est pas le cas, on recommence le test.
```

Une fois ce test conclu, l'octet reçu se trouve dans le port 0x44. Il suffit donc de lire ce port et de le traité, ou de le stocké. Si vous attendez d'autres octets, recommencez le test d'arrivée, et faites ceci jusqu'à ce que tous les octets attendus soient reçus.
Quand c'est le cas, vous devez fermer la communication.

7.4 Fermeture du port

La fermeture du port est très rapide, il vous suffit de

- ✓ mettre 0 dans le port 0x44
- ✓ désactivé les bits 5 et 6 du port 0x11

Ainsi le port est clos, ce sui empêche toute opération de réception ou d'envoie sans une initialisation.

7.5 Caractéristiques

Grâce au modes présentés ci-dessus, vous pouvez recevoir et envoyé des informations très rapidement. Mais la graph100 ne possède que 8Mhz, ce qui impose dans les communications à grandes vitesses, et surtout en réception, de ne pas permettre au programme d'exécuter de longues taches sous peine de rater la réception de certains octets.

Pour accroître la vitesse de votre code, vous pouvez également utilisé les instruction CLI (Clear interrupts) et STI (Set interrupts) afin d'empêcher toute interruptions de perturber votre code. Attention toutefois a ne pas utilisé l'instruction HLT après l'instruction CLI, ceci bloquerais la calculatrice jusqu'au prochain reset.

De plus afin de protéger la communication, il est préférable lors de la réception de données ne nécessitant pas l'intervention immédiate de l'utilisateur de désactiver le clavier.
Pour cela, il suffit de mettre le bit 3 du port 0x0Ah à 0. Pour le réactiver, il faut y mettre la valeur 1.

Il est également possible de passer du mode de réception au mode d'envoie et inversement sans repasser par la procédure d'initialisation, tant que le port n'as pas été clos.

8° Le microprocesseur Nec V30Mx

Caractéristiques techniques :

- 14 registres
- Jeu de 101 instructions
- Instructions de manipulations de bits (set1, not1, clr1 et test1)
- Registres spéciaux supportant le standard LIM EMS 4.0
- Complètement compatible 8086
- 8 Mips (Million d'instructions par secondes), dont 5 exploitables en utilisation normale (environs)

8.1 Instructions de bits

Un des atouts intéressant du Nec V30Mx est sa capacité a manipulé les données bit à bit, ou du moins d'effectuer des opérations sur ceux-ci directement et rapidement. Pour pouvoir utilisé ces instructions, il vous faut les inclure vous-même dans votre code, c'est-à-dire écrire leur code binaire. Car ces instructions ne sont présentes que chez le constructeur Nec, qui ne fourni pas de compilateur, ce qui ne rend pas leurs utilisations simples. Le code binaire à utilisé pour compiler ces instructions se trouve dans le manuel « Nec User's Manual 16-bit V séries » au pages ci-contre.

Nec User's Manual 16-bit V séries	
Description	**Pages**
Code binaire des registres	P.22
Clr1	P.65
Not1	P.121
Set1	P.155
Test1	P.174

Clr1 dst, src

Cette instruction éteint le bit numéro src de l'opérande dst. Par conséquent, avec bh=12 et cl = 3, l'instruction

```
Clr1 bh,cl
```

donneras bh=4 et cl=3.
En effet : bh=12=1100b. Donc en éteignant le 3eme bit, on obtient bh=4=0100b.
Ceci est très utile, notamment avec l'affichage graphique de notre graph100 qui fait correspondre un bit a un pixel.

Not1 dst,src

Cette instruction inverse la valeur du bit numéro scr de l'opérande dst.

Avec bh=4 et cl=2,

```
Not1 bh,cl
```

Donneras bh=0 et cl=2 car bh=4=0100b donc en inversant le second bit on obtient bh=0000b.

On trouve ici aussi un intérêt pratique dans le mode vidéo.

Set1 dst,src

Le bit numéro scr de dst est mis a 1.

Test1 dst,src

Met l'indicateur de retenu a 1 si le bit numéro scr de dst vaut 0, et met cet indicateur a 0 dans le cas contraire. Cette instruction est utile si on souhaite simplement faire un branchement en fonction du bit testé, bien que l'instruction Test puisse également remplir cette fonction.

8.2 Lim EMS 4.0

Le standard EMS, également connu sous le nom de mémoire paginée, permet d'accéder à plus d'1Mo de mémoire maximale théoriquement possible. Pour ce faire, les fabricants et développeurs Lotus, Intel et Microsoft (Lim) on crée un protocole et une carte d'extension permettant l'ouverture d'une fenêtre en mémoire dans laquelle défilerait la ram supplémentaire, sans pour autant franchir le seuil des 1Mo. Voici comment cela a été conçu :

Mémoire Principale d'un PC		Mémoire EMS
Adresse	Contenu	
0000h : 0000h - 0000h : FFFFh	Taille maximale de la ram en mode réel, 640Ko. Mais le graph100 n'en possède que 256Ko.	*Zone affichée*
1000h : 0000h - 1000h : FFFFh		
2000h : 0000h - 2000h : FFFFh		
3000h : 0000h - 3000h : FFFFh		
4000h : 0000h - 4000h : FFFFh		
5000h : 0000h - 5000h : FFFFh		
6000h : 0000h - 6000h : FFFFh		
7000h : 0000h - 7000h : FFFFh		
8000h : 0000h - 8000h : FFFFh		
9000h : 0000h - 9000h : FFFFh		
A000h : 0000h - A000h : FFFFh	Emplacement réservé aux cartes d'extensions, au bios, et a la mémoire vidéo. C'est ici que se situe la fenêtre.	
B000h : 0000h - B000h : FFFFh		
C000h : 0000h - C000h : FFFFh		
D000h : 0000h - D000h : FFFFh		
E000h : 0000h - E000h : FFFFh		
F000h : 0000h - F000h : FFFFh		

En fait, seule une partie de toute la mémoire paginée est affichée a un instant donné, ce qui permet, en changeant la zone affichée d'utilisé une grande quantité de mémoire. A l'époque, on utilisait une fenêtre de 64Ko à l'intérieur de laquelle s'affichaient 4 zones de mémoires EMS. C'était la version 3.2 de l'EMS, la 4.0 ne s'étant jamais imposée.
Mais cette méthode devrait vous rappeler la méthode d'accès à la rom et à la flash, car elle fonctionne exactement sur le même principe, sauf que la fenêtre a une taille de 128Ko et que l'on peut en ouvrir plusieurs. C'est justement la nouveauté qu'ajoute la version 4.0 de l'EMS. Nous accédons donc à nos mémoires (Rom et flash) en tant que mémoire paginée, grâce a ce système.

8.3 Le mode d'émulation

http://www.cpu-museum.com/

Dans le jeu d'instruction du Nec V30, deux d'entre elles sont tout a fais spéciales et peut communes, BRKEM et RETEM. Grâce à elles, le V30 se voit doter d'un nouveau jeu d'instruction, celui du 8080.
Ce processeur à été développer en 1974 par Intel. Il possédait un système d'adressage sur 8bit, et tournait à 2Mhz. Il s'est vu cloner par d'autres sociétés comme Amd ou encore Nec, ce dernier l'ayant rebaptisé 8080D. Après avoir subit quelques améliorations, il a finalement porté le nom de 8080AF.
En pratique, celui-ci permet de substituer totalement le jeu d'instruction du V30 pour pourvoir utilisé celui du 8080AF, seule l'instruction RETEM ayant été ajoutée, celle-ci permettant de sortir du mode d'émulation.
En pratique, ce mode n'est pas très utile, car il n'y a pas un grand intérêt à faire tourner des programmes pour 8080 sur graph100.

Note : Certain document de NEC se contredisent au sujet de sa présence au sein du Nec V30Mx. Cependant l'exécution de l'instruction BRKEM n'est pas sans effet. Aucun test n'a pu prouver que celle-ci activait le mode d'émulation. La graph100 n'est peut-être pas concernée pas ce mode.

BRKEM

Cette instruction sur 3 octets permet de passer en mode d'émulation. Les deux premiers valent 0FFFh. Le dernier octet contient un numéro d'interruption. Le processeur sauvegarde tout d'abord les informations lui permettant de sortir du mode émulation dans la ram, puis saute a l'adresse contenue par le vecteur d'interruption du troisième octet, les données étant alors interprétées comme du code 8080.

RETEM

Cette fonction sort du mode d'émulation en chargeant de la pile les informations stockées par BRKEM.

9° Les Timers

Il existe deux horloges sur graph100. La première est tenu à jour par les interruption 8 et 1Ch . Elle fonctionne 2,7 fois plus vite qu'une horloge normale et est remise à 0 à chaque extinction de la calculatrice. On peut l'utilisé grâce a l'interruption 1Ch :

Interruption 1Ch	**Timer**	

Interruption appelée après chaque Int 8 mettant a jour le timer bios.
On incrémente d'abord le mot 0040h :006Ch puis s'il devient nul le mot 0040h :006Eh et enfin l'octet 0040h :0070h.

La graph100 possède également une horloge temps réel (Real Time Clock). La théorie voudrait que notre calculatrice soit un PC trop ancien pour posséder une RTC. Mais il n'en est rien. En effet, la graph100 possède bien une horloge temps réel fonctionnant même quand la calculatrice est arrêtée, et ceci à vitesse normale. On y accède grâce aux ports 1Dh à 22h. Elle transmet les secondes, minutes, heures et date. Elle est réglable et fonctionne en mode 24 heures.
Son réglage peut s'effectuer soit directement par les ports, soit grâce à l'interruption 4Ah :

Interruption 4Ah	**RTC**	
Fonction 0…3	Ah < 3	

Renvoi dans dh le nombre de seconde de la RTC, dans cl le nombre de minutes, dans ch le nombre d'heures.

Fonction 3	Ah = 3

Cl = nombre de minutes
Ch = nombre d'heures
La fonction met alors à jour l'heure avec celle précisée dans cx, les secondes sont remisent a 0.

Fonction 4	Ah = 4

Renvoie le nombre de jours comptabilisés par la RTC dans cx.

Fonction 4…FFh	Ah > 4

Cx = Nombre de jours.

La fonction met à jour le nombre de jours de la RTC avec celui inscrit dans cx.

10° Le format Romdisk

C'est le format d'allocation de fichier utilisé par Rom-dos pour stocker les fichiers dans la flash et dans la rom. Veuillez dans le tableau suivant ne pas oublier que les valeurs hexadécimales contenues dans l'image romdisk sont au format Intel, donc sous un éditeur hexadécimal, une valeurs de 100h sera écrite 0001 car les octets des mots sont inversé deux a deux.

Adresse dans l'image	Taille	Valeur	Commetaires
00000h – 0000Ah	10 octets	EB28 9044 4C52 4449 534B	Entête de l'image Romdisk
0000Bh	Mot	Taille des secteurs : 80h : 128 octets 100h : 256 octets 200h : 512 octets	On utilisera toujours la valeurs 200h, seule valeur que la graph100 reconnais.
00011h	1 octet	Selon la taille des secteurs, cet octet vaut 4,8 ou 10h. Il faut ensuite multiplier cette valeur par le nombre de secteur nécessaire pour la zone des noms de fichiers en ?00h.	Donc 10h*nombre_secteur_en_ ?00h.
00013h	Mot	Nombre de secteurs constituants le lecteur.	
00016h	Mot	Nombre de secteurs utilisé par la zone de validité du lecteur (offset 200h)	
00024h – 001FDh	474 octets	Tous les octets valent FFh	
001Feh	Mot	AA55h	
00200h	A cette adresse commence la zone de validité du lecteur qui permet de vérifier le bon format des données de l'image, des fichiers qu'elle contient		
00200h	3 octets	F8 FF FF	Header de cette zone
	Ensuite, on trouve des groupes contigus de la forme suivante. Ces groupes correspondent chacun à un secteur. Le premier de ces groupes représentant le premier secteur du premier fichier du disque (voir fin tableau), les autres les suivants (s'il y en a).		
00203	1 octet	Pour le premier groupe, cet octet vaut 3. Ensuite, pour chacun des groupes suivants, ajoute 2 a cet octet.	Groupes pontants vers les secteurs des fichiers de l'image.
00204	1 mot	Ce mot correspond au type de donnée rencontrée sur le secteur ciblé. Deux cas de figures se présentent : - secteur « normal » Pour le premier groupe, ce mot vaut 40h, puis pour chaque groupe supplémentaire on rajoute 10h. - secteur correspondant a une fin de fichier. Dans ce cas, on prend la valeur normale en lui faisant faire un OR soit avec FF0Fh si le nombre de secteur composant le fichier est impaire, soit avec F0FFh sinon.	
00 ?00h	Cette zone commence au secteur suivant le dernier de la zone de validité. Elle va contenir des informations sur les fichiers inclus dans l'image. Ces informations sont stockées par groupe de 32octets, chacun de ces groupes donnant des informations sur un fichier a la fois. Le premier groupe est encore un header, contenant simplement le texte ROM-DISK, suivi de 20 20 20 08. Le premier groupe utile se situe donc en 00 ?20h. Il y a autant de groupes utiles que de fichiers, +1 header.		
00 ?20h	8 octets	Nom du fichier sans extension, en majuscule. S'il n'est pas assez long, on complète par des espaces (20h).	Groupe utile

00 ?28h	3 octets	Extension en majuscule	
00 ?2Bh	1 octet	20h : Séparation	
00 ?2Ch	10 octets	Les 10 octets de cette zone sont a 0	
00 ?36h	4 octets	Date et heures de modification du fichier, selon le format suivant, nombre de 32bits pris de droite a gauche : 5bits : jour 4bits : mois 7bits : année 5bits : secondes divisée par 2 6bits : minutes 5bits : heures	
00 ?3Ah	1 mot	Indique le numéro de secteur contenant le début du fichier. Pour le premier groupe cette valeur est toujours à 2. Les autres en découlent en ajoutant la taille du fichier précèdent en secteur avec son secteur de départ.	
00 ?3Ch	2 mots	Taille du fichier en octets.	
	Tant que tous les fichiers du lecteur n'ont pas leurs groupes utiles correspondant, on en rajoute sous la même forme que celui vu précédemment. Quand tout les fichiers ont leurs groupes respectifs, on va au début du secteur suivant, et on commence a écrire les fichiers. Quand on atteint la fin d'un fichier, au passe au secteur suivant et on commence a y écrire le prochain si bien que 2 fichiers ne peuvent pas se retrouver sur le même secteur et que chaque fichier commence au début d'un secteur.		

11° Le format RXE

Ce format de fichiers a été développé par Datalight. Il permet d'exécuter des programmes directement depuis une rom, en ne chargeant en ram que les variables. Ceci est utile pour plusieurs raisons :

- Gain significatif de place dans la ram, car on ne charge que ce qui va etre modifier.
- Permet d'exécuter des programmes de plus de 64Ko.

Mais il possède également un défaut important, celui de ralentir la vitesse d'exécution des programmes.

Car les fichiers Rxe sont des exécutables de type .exe modifiés pour répondre aux attentes présentées ci-dessus. Pour cette raison, et pour rester compatible avec MS-DOS, ce format garderas également l'extension .exe .
Il est utilisé sur la graph100 par tous les programmes initialement présents sur la graph100 de plus de 64Ko, c'est-à-dire tout les programmes du lecteur B : a K :.

11.1 Analyse de fonctionnement

Habituellement, MS-DOS, pour exécuter des programmes, commence à les copier du disque sur lequel ils sont stockés vers la mémoire. Des informations concernant l'adresse de chargement du programme ainsi que sa taille sont alors écrites dans l'entête de l'image copiée.

Les modifications se font au niveau de zones contenant l'adresse de chaque partie du programme, les Fix-Ups. En effet, en modifiant leurs valeur, on peut prévenir MS-DOS que les données a traité ou a exécuté sont situé a un endroit précis, sur un disque hors de la ram.
Ces Fix-Ups sont contenus dans l'entête de l'exécutable, c'est donc a cet endroit que vont avoir lieu les modifications.

Il existe trois types de Fix-Ups. Le premier est celui qui se refere aux segments de code. Un segment de code ne contient théoriquement aucune données destinées a être modifiée, on peut donc l'exécuté directement depuis le disque. Toutes les références aux segments de code du programme sont donc modifiées pour avoir une valeur fixe pointant vers le disque contenant ces segments. Cette méthode empêche donc d'écrire un code capable de se modifier lui-même, le code étant exécuté sans avoir été chargé en mémoire (XIP : eXecute In Place).

Au moment de l'exécution du programme, MS-DOS effectue lui aussi ce genre de modifications, dans l'entête de l'exécutable pour que celui-ci sache ou se trouve les données qu'il va devoir manipuler, car un exécutable n'est jamais chargée a la même adresse dans la ram. C'est ces manipulations qui vont devoir être prévues a l'avance par le convertisseur Rxe pour ne laisser MS-DOS manipuler que les données à charger en mémoire.

Le troisième type de modification est a appliqué au segment de code. En effet, ceux-ci on besoin d'accéder aux données. Hors, les données ne sont plus du tout au même endroit que le code, l'un étant dans la ram, l'autre sur un disque. Mais les données étant situées aléatoirement en ram, on ne peut en prévoir les adresses à l'avance. Le convertisseur Rxe vas donc remplacer les appels de données du code par une interruption de sa création chargée d'accédée au données requise. C'est cette partie qui va ralentir significativement l'exécution du programme. De plus ces appels peuvent être ambiguë, ce qui ne facilite pas la programmation.

Vous trouverez de plus amples informations sur la fiche technique de datalight « RXE Theory of Operation ».

11.2 En pratique

Le format Rxe convient tout à fait aux applications de calculs tels qu'elles sont présentes sur la graph100, car ces programmes sont essentiellement constitués de code.
Il n'est va pas de même pour les jeux qui eux souffrirais de ces accès trop fréquents au données notamment pour l'affichage des graphismes stockés en tant que données dans les programmes. La solutions consisterais alors à les rendre statiques (elles serait ainsi stocké dans les segments de code), mais il ne faut pas non plus oublier que la flash et la rom on des temps d'accès au données supérieurs à ceux de la ram.

12° Le Bios

Le bios est un programme confiné dans un circuit en lecture seule. Sur la graph100, c'est un T-Note BIOS v0.60 (Révision 1.10). Comme son nom l'indique (Basic Input/Output System), c'est grâce à lui que les programmes vont pouvoir communiquer avec les périphériques de la machine. Mais c'est également lui qui est charger de démarrer et de vérifier le bon fonctionnement de la machine. Il installe au démarrage toute une liste d'interruptions qui vont permettre d'utiliser les périphériques tel le clavier. Ces interruptions s'appellent toutes de la même manière quelque soit le PC, ce qui augmente leurs compatibilité les uns avec les autres.

Mais le bios, pour qu'il soit accessible possède sont propre emplacement mémoire, en général dans le segment F000h. Sur Graph100, il commence à l'adresse F000h :F000h.

A l'allumage de la calculatrice, le bios est dans ce segment. Sur tout les PC, y compris sur la graph100, la première instruction exécutée par le système est située a l'adresse F000h :FFF0h. Cette partie de la mémoire bios contient un saut inconditionnel vers une autre zone qui va elle testée le bon fonctionnement du système. Cette zone est appelée Power-on Self Test, POST. Après son exécution, si tout c'est bien passé, le bios passe alors la main au système d'exploitation, Rom-dos.

13° Les Interruptions

Le chapitre suivant vous propose des explications sur les interruptions.
Une interruption est une fonction de la machine, résidente en mémoire, qui quand on l'appel effectue une opération a laquelle elle est dédiée. Les interruptions peuvent être de deux types, logiciels et matériels. Les premières ne sont exécutées que par un appel a elles de la par d'un programme. Les interruptions matériels quand à elles sont générées par les périphériques qui demande eux même leurs exécutions quand ils en ont besoin, comme le rafraîchissement de l'écran avec l'interruption 53h par exemple. Pour que le processeur puisse exécuter les interruptions directement grâce a leurs numéro, il a été décidé de placer une table de vecteurs d'interruptions au tout début de la ram a l'adresse 0000h : 0000 jusqu'à l'adresse 0000h : 03FFh qui contient l'adresse mémoire de toute les interruptions dans l'ordre de leurs numéros. Ainsi l'interruption 0h a son adresse en 0000h : 0000h jusqu'à 000h : 0003h. Par la suite, nous parlerons de l'adresse réelle, nous dirons simplement que l'adresse de l'interruption 0 est comprise entre 000h et 003h, celle de l'interruption 1 entre 004h et 007h…
L'adresse inscrite à cet endroit est écrite l'offset en premier et le segment en second.
Le processeur va alors recevoir le numéro de fonction, lire dans cette table son adresse et aller exécuter les instructions s'y trouvant. Cette table n'est que peut protégée, et on peut à loisir la modifier pour ajouter des interruptions ou pour en modifier. Car toutes les interruptions ne sont pas utilisées.
Voici une liste des interruptions présentent sur la graph100+.

N°	Adresse	Description
0	000 – 003	CPU: Division par zéro
9	024 – 027	IRQ1: Clavier
10	040 – 043	BIOS: Fonctions vidéo
11	044 – 047	BIOS: Détermination configuration
12	048 – 04B	BIOS: Détermination taille RAM
13	04C – 04F	
14	054 – 057	
16	058 – 05B	BIOS: Interrogation du clavier
19	064 – 067	BIOS : Démarrage à chaud (ALT+CTRL+DEL)
1A	068 – 06B	BIOS : Lecture date et heure
1B	06C – 06F	Touche Break actionnée
1C	070 – 073	Controle le timer
20	080 – 083	DOS: Terminaison du programme
21	084 – 087	DOS: Fonction de DOS
22	088 – 08B	Adresse de routine DOS fin du programme
23	08C – 08F	Adresse de routine CTRL-BREAK du DOS
24	090 – 093	Adresse de routine d'erreur du DOS
25	094 – 097	DOS : Lecture disquette/disque dur
26	098 – 09B	DOS : Ecriture sur disquette/disque dur
27	09C – 09F	DOS : Fin programme, laisser résident
28	0A0 – 0A3	
29	0A4 – 0A7	
2B	0AC – 0AF	
2C	0B0 – 0B3	
2F	0BC – 0BF	
41	104 – 107	
44	110 – 113	
45	114 – 117	
47	11C – 11F	
48	120 – 123	Mémoire EMS
4A	128 – 12B	RTC

4B	12C – 12F	Mémoire EMS
4C	130 – 133	
4D	134 – 137	
4^E	138 – 13B	
4F	13C – 13F	
51	144 – 147	
53	14C – 14F	Ecran : Mise à jour de l'affichage
58	160 – 163	
5C	170 – 173	Général (Ecran, flash…)
5^E	178 – 17B	
5F	17C – 17F	
7C	1F0 – 1F3	Contrôle de l'écran
7D	1F4 – 1F7	

On remarquera que l'interruption 2 contrôlant NMI ne semble pas être présente sur la graph100+, bien qu'elle le soit sur les autres modèles (Elle doit se trouver a un autre endroit). C'est valable pour d'autres interruptions, mais il n'existe pas à l'heure actuelle de liste les recensant.

De plus, quelques test permettent de constater que des interruptions tel 16h existent mais on été modifié de sorte que certaines fonction ai été ajoutées ou retirées, par rapport à une interruption 16h normale.

Un point important très utile, est que l'appel d'une des fonctions non listée n'entraînera pas de bug, mais simplement un retour direct vers le programme appelant. Ceci peut être utilisé si on souhaite rediriger une interruption pour la rendre inactive temporairement, car il suffit de faire pointé son adresse vers celle d'une des interruption non listée ici.

14° Police de caractère

Une police de caractère est un groupement de caractère ayant un style particulier, ces caractères ayant toujours un ordre précis et ne pouvant être plus de 256. La graph100, par défaut, en possède 3. Le premier est notre alphabet classique, il contient également des caractères spéciaux. Le second regroupe des caractères accentué ou grecs. Le troisième est un alphabet japonais, le kata kana.

Les caractères, en mémoire, sont représentés sur 8 octets, avec 1bit = 1 pixel, ce qui donne à chaque caractères 64 pixels :

0	0	**1**	**1**	**1**	0	0	0
0	**1**	0	0	0	**1**	0	0
0	**1**	0	0	**1**	**1**	0	0
0	**1**	0	**1**	0	**1**	0	0
0	**1**	**1**	0	0	**1**	0	0
0	**1**	0	0	0	**1**	0	0
0	0	**1**	**1**	**1**	0	0	0
0	0	0	0	0	0	0	0

8 lignes, 1 ligne = 1 octet

8 bits = 1 octets.

Ici, c'est le caractere 48 de l'aphabet classique qui est representé. On remarque que sur sur droite, deux colonnes de bits ne sont pas utilisé. Ceci vient du fait qu'il est bien plus simple de manipuler 1 octet complet que 6 bits seules. A l'écran, chaque caracteres mesure 6 pixels de large. Quand on crée une police pour casio, il faut donc laissé ces deux colonnes vide, sans s'en occuper.

Pour acceder au autre polices de caractere, ou a une police personnel, il est necessaire d'agir sur le mot situé en 0040h : 00C4h, ou bien d'écrire le caractère 0xF6 ou 0xF7 a l'ecran. En effet, le fait de vouloir afficher ces valeur entrainera la modification du mot précité, sans pour autant affiché les caractères correspondant au code ascii 0xF6 et 0xF7 (qui de toute façon sont nuls).

Voici les valeurs que doit contenir ce mot pour modifier l'alphabet utilisé :

Type d'alphabet	**Valeur**
Alphabet Normal	0xCE50
1er alphabet spécial (0xF6)	0xCED0
2nd alphabet spécial (0xF7)	0xCF50

Attention : à chaque fois qu'un caractère est affiché à l'écran cette valeur est remise à 0xCE50.
Ces valeurs sont les adresses de segment contenant la police de caractère à utilisées. Pour utiliser sa propre police, il suffit de charger son modèle en mémoire, au début d'un segment, et d'écrire l'adresse de ce segment dans ce mot à chaque fois que l'on souhaite l'utiliser.

15° Le système d'exploitation Rom-Dos

La graph100, comme tout ordinateur personnel, possède un système d'exploitation, Rom-Dos. Il est compatible MS-DOS et a été conçu par Datalight. Il se charge au démarrage de la calculatrice, juste après le bios qui lui passe la main.
Tout comme MS-DOS, il est capable de lancer des fichiers exécutables de types .com ou .exe . Cependant, du fait de la taille de la mémoire vive, il ne permet pas l'utilisation de programme compiler dans un autre mode que tiny, ce qui revient à n'exécuter que des programmes au format .com, les .exe ayant alors la même structure.
Il se compose d'une partie résidente, notamment de son jeu d'interruption de 20h a 27h, ainsi qu'une copie de command.com. Celui-ci est présent dans sa version « mini », Datalight l'ayant crée pour des systèmes embarqué ne nécessitent pas que l'utilisateur accède a la ligne de commande. C'est pourquoi il ne comprend qu'un jeu de commandes limité. Il n'est d'ailleurs pas possible d'utiliser ce fichier simplement en faisant appel à lui depuis un explorateur de fichier, car les ingénieurs de Casio n'ont pas juger utile de remplacer l'attende de la touche « entrée » par la touche « exe », command.com n'étant pas censé être exécuté. Ce phénomène est d'autant plus prononcé que depuis la graph100+, il n'est plus possible d'accéder au lecteur A : depuis Rom-dos, lecteur qui renferme entre autre command.com, car ce lecteur se protège en lecture immédiatement après que le premier programme de la session en cours est été terminer. De plus, en raison du fait que l'utilisateur ne doit pas pourvoir accéder à Rom-dos en utilisation classique, l'accès au paramètres du système été totalement bloqué, ce qui en fait un système n'apportant pratiquement rien au programmeur, ses interruptions étant très souvent disponible directement au niveau de bios.

15.1 Le Psp

Avant de lancer un programme, Rom-dos, tout comme Ms-Dos crée une zone d'entête appelée Program Segment Prefix (PSP). Cette zone va contenir des informations concernant le chemin d'accès au fichier en cours, la sauvegarde de certains vecteurs d'interruptions, et une multitude d'informations plus ou moins utiles pour le programmeur. Cette zone se situe juste avant la copie du programme en mémoire, et mesure 256 octets. Elle n'a pas le même format sous Rom-dos que sous Ms-Dos. En voici la structure tel qu'elle est connue à l'heure actuelle.

Structure du PSP		
Adresse	**Contenu**	**Taille**
00h	Appel de l'interruption 20h	1 mot
02h	Adresse de bp	1 mot
50h	Appel de l'interruption 21h	1 mot
80h	Taille du premier argument	1 octet
81h	1er argument	Voir ci-dessus
C6h	Chemin du programme	

Ces données bien que fragmentaires aiderons certainement les programmeurs sui souhaitent accéder au paramètres de la ligne de commande en assembleur. Il faut savoir que le chargeur d'exécutables fait pointer les segment cs, ds et es vers le début du PSP. En effet, les premières instructions du programme en cour doivent sauvegardé la valeur d'un de ces segment afin de pouvoir accédé au PSP ultérieurement. Il est alors simple d'utiliser les informations que celui-ci contient.

Annexe A - Codes Ascii Casio

Dec.	*Hexa*	Ascii 1	Ascii 0xF6	Ascii 0xF7
0	*00*			
1	*01*			
2	*02*			
3	*03*			
4	*04*			
5	*05*			
6	*06*			
7	*07*			
8	*08*			
9	*09*			
10	*0A*			
11	*0B*			
12	*0C*			
13	*0D*			
14	*0E*			
15	*0F*			
16	*10*			
17	*11*			
18	*12*			
19	*13*			
20	*14*			
21	*15*			
22	*16*			
23	*17*			
24	*18*			
25	*19*			
26	*1A*			
27	*1B*			
28	*1C*			
29	*1D*			
30	*1E*			
31	*1F*			
32	*20*			
33	*21*			
34	*22*			
35	*23*			
36	*24*			
37	*25*			
38	*26*			
39	*27*			
40	*28*			
41	*29*			
42	*2A*			
43	*2B*			
44	*2C*			
45	*2D*			
46	*2E*			
47	*2F*			
48	*30*			

Dec.	*Hexa*	Ascii 1	Ascii 0xF6	Ascii 0xF7
49	*31*			
50	*32*			
51	*33*			
52	*34*			
53	*35*			
54	*36*			
55	*37*			
56	*38*			
57	*39*			
58	*3A*			
59	*3B*			
60	*3C*			
61	*3D*			
62	*3E*			
63	*3F*			
64	*40*			
65	*41*			
66	*42*			
67	*43*			
68	*44*			
69	*45*			
70	*46*			
71	*47*			
72	*48*			
73	*49*			
74	*4A*			
75	*4B*			
76	*4C*			
77	*4D*			
78	*4E*			
79	*4F*			
80	*50*			
81	*51*			
82	*52*			
83	*53*			
84	*54*			
85	*55*			
86	*56*			
87	*57*			
88	*58*			
89	*59*			
90	*5A*			
91	*5B*			
92	*5C*			
93	*5D*			
94	*5E*			
95	*5F*			
96	*60*			
97	*61*			

98	*62*			
99	*63*			
100	*64*			
101	*65*			
102	*66*			
103	*67*			
104	*68*			
105	*69*			
106	*6A*			
107	*6B*			
108	*6C*			
109	*6D*			
110	*6E*			
111	*6F*			
112	*70*			
113	*71*			
114	*72*			
115	*73*			
116	*74*			
117	*75*			
118	*76*			
119	*77*			
120	*78*			
121	*79*			
122	*7A*			
123	*7B*			
124	*7C*			
125	*7D*			
126	*7E*			
127	*7F*			
128	*80*			
129	*81*			
130	*82*			
131	*83*			
132	*84*			
133	*85*			
134	*86*			
135	*87*			
136	*88*			
137	*89*			
138	*8A*			
139	*8B*			
140	*8C*			
141	*8D*			
142	*8E*			
143	*8F*			
144	*90*			
145	*91*			
146	*92*			
147	*93*			
148	*94*			
149	*95*			

150	*96*			
151	*97*			
152	*98*			
153	*99*			
154	*9A*			
155	*9B*			
156	*9C*			
157	*9D*			
158	*9E*			
159	*9F*			
160	*A0*			
161	*A1*			
162	*A2*			
163	*A3*			
164	*A4*			
165	*A5*			
166	*A6*			
167	*A7*			
168	*A8*			
169	*A9*			
170	*AA*			
171	*AB*			
172	*AC*			
173	*AD*			
174	*AE*			
175	*AF*			
176	*B0*			
177	*B1*			
178	*B2*			
179	*B3*			
180	*B4*			
181	*B5*			
182	*B6*			
183	*B7*			
184	*B8*			
185	*B9*			
186	*BA*			
187	*BB*			
188	*BC*			
189	*BD*			
190	*BE*			
191	*BF*			
192	*C0*			
193	*C1*			
194	*C2*			
195	*C3*			
196	*C4*			
197	*C5*			
198	*C6*			
199	*C7*			
200	*C8*			
201	*C9*			

202	*CA*		
203	*CB*		
204	*CC*		
205	*CD*		
206	*CE*		
207	*CF*		
208	*D0*		
209	*D1*		
210	*D2*		
211	*D3*		
212	*D4*		
213	*D5*		
214	*D6*		
215	*D7*		
216	*D8*		
217	*D9*		
218	*DA*		
219	*DB*		
220	*DC*		
221	*DD*		
222	*DE*		
223	*DF*		
224	*E0*		
225	*E1*		
226	*E2*		
227	*E3*		
228	*E4*		

229	*E5*	
230	*E6*	
231	*E7*	
232	*E8*	
233	*E9*	
234	*EA*	
235	*EB*	
236	*EC*	
237	*ED*	
238	*EE*	
239	*EF*	
240	*F0*	
241	*F1*	
242	*F2*	
243	*F3*	
244	*F4*	
245	*F5*	
246	*F6*	
247	*F7*	
248	*F8*	
249	*F9*	
250	*FA*	
251	*FB*	
252	*FC*	
253	*FD*	
254	*FE*	
255	*FF*	

Annexe B - Interruptions

Les informations sur les interruptions présentées ici on été étudiées dans le détail, mais peuvent cependant être incomplètes ou erronée. La remarque « complet » signifie que toute les fonctions de l'interruptions son présentées ici. Vous trouverez d'autres études d'interruptions tout le long de ce document.

Interruption 10h	Graphismes	BIOS - Complet
Fonction 0	Ah = 0	

Choix du mode vidéo

Al = mode vidéo a mettre en place. Seule les 3 premiers bits de al sont gardé. Cette valeur est alors écrite en 0040h:0049h
Recadre l'affichage en 1A20h et met les variables du bios à jour.

Fonction 2	Ah = 2

Positionne le curseur

Bh = numéro de page écran
Dh = ligne de l'écran
Dl = colonne de l'écran

Fonction 5	Ah = 5

Change de page/re-initialise l'affichage.

bh = numéro de page
Modifie l'adresse du buffer vidéo en lisant la valeur contenue en 0040h : 00E0h

Fonction 6	Ah = 6

Fais défiler l'écran vers le haut.

Al : nombre de ligne de décalage
Ch : ligne de l'écran du coin supérieur gauche de la fenêtre
Cl : colonne de l'écran du coin supérieur gauche de la fenêtre
Dh : ligne de l'écran du coin supérieur droit de la fenêtre
Dl : colonne de l'écran du coin supérieur droit de la fenêtre

Fonction 7	Ah = 7

Rien

Fonction 9	Ah = 9

Ecrire un caractère à l'écran

Al = code ascii du caractère. Si Al = FF, aucun caractère ne sera écrit. Si Al = 0xF6, le prochain caractère écrit sera lu dans le premier alphabet spécial, si Al=0xF7, le prochain caractère écrit sera lu dans le second alphabet spécial.
Bh = numéro de page à afficher. La position du curseur sera choisie en fonction.
Cx = nombre de répétition d'affichage du caractère. (1 : 1 fois, 2 : 2 fois…).
Bl = attribut du caractère. Si son 6ème bit vaut 0, le caractère sera écrit en noir sur blanc, sinon en blanc sur noir.

Fonction 0Ch	Ah = 0x0C

Dessine un point à l'écran.

Bh = page écran
Dx = ligne de l'écran
Cx = colonne de l'écran
Al = couleur. Si le 7eme bit vaut 1, la pixel sera blanc, sinon il est noir.

Fonction 0Dh	Ah = 0x0D

Lit un point à l'écran.

Bh = page écran
Dx = ligne de l'écran
Cx = colonne de l'écran
Retourne dans Al la couleur du pixel ciblé.

Fonction 0Eh	Ah = 0x0E

Affiche un caractère.

Différences avec la fonction 9 non encore déterminée.

Interruption 11h		**Complet**

Retourne dans ax le mot en 0040h : 0010h (config de la caltos)

Interruption 12h		**Complet**

Retourne dans ax le mot en 0040h : 0013h (taille de la ram)

Interruption 13h		**Complet**

Met le premier bit de l'octet situé en sp+6 à 1.

Interruption 14h		
Fonction 88h	Ah=88h	

Retourne ax =0

Fonction C0h	Ah=C0h

Modifie la pile, retourne bx = 264Ah, es = cs (C000)

Fonction autre	Ah= autre

Modifie la pile.

Interruption 16h	**Clavier**	**BIOS - Complet**

Classique interruption 16, mais seule les fonctions 0, 1, 5, 10h, 11h, 20h et 30h sont disponibles.

Fonction 20h	Ah= 20h

Met les mots des adresse 0040h:001Bh et 0040h:001Dh à 1Fh.

Fonction 30h	Ah= 30h

Capte une valeur directement sans attendre l'interruption 9.

Interruption 4Bh	**Mémoire EMS**	**Complet**
Fonction 0	Ah = 0	

Fais correspondre les zones de données bios B2, B3, B4, B5 avec la zone 65, 66, 67,63.

Fonction 1	Ah = 1

Fais l'inverse puis quitte le programme avec l'interruption 21h, ah = 4Ch.

Fonction 2	Ah = 2

Action indéterminée, puis quitte le programme avec l'interruption 21h, ah = 4Ch.

Fonction 3	Ah = 3

Met la zone mémoire bios 6Bh a la valeur contenue dans bl a l'appel.

Interruption 4Eh		**Complet**
Fonction 0	Ah=0	

Lit la valeur des ports 8 et 9, la sauve dans bx, puis met 0 dans ces ports

Fonction 1	Ah=1

Ah = 1 : Lit la valeur des ports 8 et 9, la sauve dans bx, puis met 1 dans le port 8, 0 dans le port 9

Fonction autre	Ah = autre

La valeur contenue dans bx est écrite dans les ports 8 et 9

Annexe C - Les variables du bios

Le bios, comme tout programme, a besoin de variables. Mais puisqu'il ne peut les stocker directement sur sa rom, il utilise de la ram, à partir du segment 40h, sur un peu plus de 256 octets. Voici un descriptif de ces données qui peuvent se révélées très utiles dans certains cas. Cette liste n'est pas exhaustive et se complétera au fur et a mesure du développement de ce tutorial. Certaines peuvent se révélée erronée, mais les informations sur ce type de bios sont introuvables, il faut donc procédé par déduction.

00h Adresses des interfaces séries	4 mots

Le bios y stocke les adresses des 4 ports séries du PC. Il semble les trouver également sur graph100. Ces valeurs sont sans doutes utilisées a d'autres fins.

08h Adresses des interfaces parallèles	4 mots

Le bios y stocke les adresses des 4 ports parallèles du PC. La aussi il semble les trouver sur graph100.

1Ch Clavier	1 octet

Le bios stocke ici la dernière touche pressée au clavier par l'utilisateur.

49h Affichage : mode vidéo actuel	1 octet

Cet octet contient la valeur du mode vidéo en cours.

4Ah Affichage : nombre de colonnes actuelles	1 mot

Ce mot contient le nombre de colonnes par lignes du mode courant. (défaut : 21)

4Ch Affichage : taille page écran	1 mot

Ce mot contient la taille d'une page de l'écran dans le mode actuel. (défaut : 1024).

4Eh Affichage : page	1 mot

Ce mot contient la taille d'une page écran multiplié par le numéro de la page courante.

50h Position du curseur dans les huit pages d'écran	8 mots

Le bios stocke ici les coordonnées du curseur dans chaque page d'écran. Mais jusqu'à preuve du contraire, nous ne pouvons pas changé cette page et somme contraint de resté à la première, seul le premier mot est donc utiles. Changé la valeur depuis cette adresse affecte directement la position du curseur à l'écran.

62h Page vidéo	1 octet

Cet octet contient le numéro de la page courante.

6Ch Compteur système	5 octets

Compteur système, environ incrémenté 2,7 fois par secondes. Il n'est pas alimenté quand la calculatrice est éteinte, et bouge au rythme de l'interruption 8.

85h Taille d'un caractère en octet	1 octets

Cet octet contient la taille d'un caractère texte en mémoire. Par défaut, il vaut 8 octets.

B0h Relatif aux zones mémoires a mapper	7 octets

Les octets inscrits représentent les valeurs actuelles inscrites dans les port 54h a 5Ah.
Pour une compatibilité optimale avec le système, il est nécessaire de mettre a jours manuellement ces valeurs si on change la disposition des zones mémoires (voir La mémoire), a la manière du système. Aucun conflit ne semble cependant apparaître avec le système actuel si l'on omet cette étape.

C4h Police de caractère	1 mot

Ce mot permet de sélectionner la police de caractère en cours d'utilisation. Il contient l'adresse de segment de cette police.

E0h Adresse buffer vidéo	1 mots

A cette zone est stocké l'adresse de segment du buffer vidéo. Changé cette valeur n'affecte pas l'adresse. Deux solutions s'offrent alors à vous :

- Manipuler les ports 5 à 7. Cette méthode n'est conseiller qu'a ceux qui recherche une optimisation maximale, la seconde méthode étant elle bien plus simple.
- Appelé l'interruption 10h, avec ah = 5. La valeur entrée dans le mot sera alors écrite dans les ports, mais vous devez choisir une valeur dont les quatre premiers bits sont nuls, c'est à dire qui commence par un 0h tel 1A20h.

E5h Menu constructeur	1 octet

Contient une valeur à écrire dans la séquence écran.

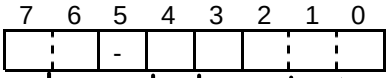
Annexe D – Les ports de communications

Les ports sont les chemins par lesquels le processeur communique avec les périphériques. Pour y accédé, vous devez utilisé les fonctions `inportb()` et `outportb()` en C, et `in` et `out` en Assembleur, respectivement pour y lire et y écrire des données. Certains de ces ports ne fonctionnent que dans un seul sens, soit en lecture soit en écriture. Cela signifie qu'une action non permise par le port ne sera pas interprété, une lecture sur un port en écriture seule renverra une valeur quelconque. Les ports notés « Non déterminé » sont actifs et utilisé, mais on ne connaît pas encore leurs actions.

Port 00h	**Non déterminé**	**Ecriture**

Port 02h	**LCD**	**Ecriture**

Ce port permet de changer le mode d'affichage vidéo de l'écran. Les valeurs écrites ont ce format :



Hauteur de l'écran :
- 000b = ¼ de l'écran, contraste maximum (bleu)
- …
- 011b = écran total (normal)
- …
- 111b = écran grandement élargit, contraste minimum (clair)

Couleur des pixels :
- 0b = Noir et Blanc
- 1b = Niveau de gris

Format :
- 0b = Affichage en colonne
- 1b = Affichage linéaire

Paramètres de fonctionnement :
- 00b, 01b = Ecran blanc
- 10b = Ligne bleu au travers de l'écran
- 11b = Affichage normal

Port 03h	**LCD**	**Ecriture**

Modifie la longueur d'affichage de l'écran. 7 est la valeur écrite par défaut.

Port 04h	**LCD**	**Ecriture**

Modifie la vitesse de balayage de l'écran.

1 : Stoppe le balayage
2 : Balayage le plus rapide
4 : Valeur standard

Les valeurs supérieures le ralentiront sans l'arrêter.

Port 05h à 07h	**LCD**	**Ecriture**

Modifient l'adresse de départ du buffer vidéo. Le format des adresses étant toujours sur 20bit, nous devons utiliser ce format pour écrire dans ces ports. Prenons la valeur par défaut, 1A20h :0000h. Cette adresse, sous sa forme condensée s'écrit 1A20h*16+0000h = 1A200h.
C'est ce dernier type de valeur qu'il nous faut écrire dans ces ports, les ports étant mis bout a bout pour pourvoir contenir cette adresse. Ne pouvant pas l'écrire directement a cause de sa taille, on peut l'écrire comme suit :

Port 7h				Port 6h								Port 5h							
3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Segment compris entre 0h et FFFFh, donc sur 16bits.																Offset compris entre 0h et Fh			

Ports 08h et 09h	**Non déterminé**	**Ecriture**

Port 0Ah	**Port de communication**	**Lecture / Ecriture**

Etat du port de communication (Voir section port de communication)

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

Contrôle de la communication (1 = établie)
Accès au clavier (1 = possible)
1b = Allume le bit 6 du port 0eh

Port 0Bh	**Port de communication**	**Lecture / Ecriture**

Exécute la réception ou l'envoi de données (Voir section port de communication)

Port 0Ch	**Général**	**Lecture / Ecriture**

Arrête la calculatrice, comme une pression sur la touche Off.

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

1b = Arrête la calculatrice

Port 0Dh	**Non déterminé**	**Ecriture**

Port 0Eh	**Port de communication**	**Lecture / Ecriture**

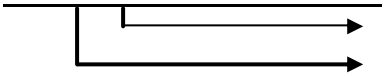
Contrôle la communication série de la graph100

Port 10h	**Non déterminé**	**Lecture / Ecriture**

Port 11h	**Port de communication**	**Lecture / Ecriture**

Mise en place du bus de transfert de donnée (Voir section port de communication).

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



Applique une tension de 4,92V à la partie entrante du port.
Applique une tension de 4,92V à la partie sortante du port.

Port 12h	Non déterminé	Ecriture / Ecriture

Ports 13h et 14h	Clavier	Lecture / Ecriture

Ces ports permettent de communiquer avec le clavier.
Avant de lire dans le port 13h, vous devez écrire dans les ports 13h et 14h la ligne de touches que vous souhaitez testé, puis lire sa valeur dans le port 13h. Pour plus d'information, reportez vous au chapitre consacré au clavier.

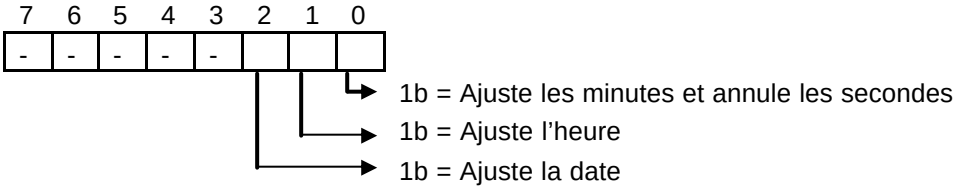
Port 16h	Non déterminé	Lecture / Ecriture

Ports 17h et 18h	Non déterminé	Lecture / Ecriture

Port 1Ch	Non déterminé	Lecture / Ecriture

Port 1Dh	Horloge temps réel (RTC)	Lecture / Ecriture

En lecture, ce port renvoie les secondes actuelles.
En écriture, vous devez tout d'abord mettre la bonne valeur dans les ports 1Eh à 22h, puis écrire comme suit :



Port 1Eh	Horloge temps réel (RTC)	Lecture / Ecriture

En lecture, ce port renvoie les minutes actuelles.
Pour les modifier, vous devez écrire dans ce port la valeur souhaiter, puis ajuster le port 1Dh

Port 1Fh	Horloge temps réel (RTC)	Lecture / Ecriture

En lecture, ce port renvoie les heures actuelles.
Pour les modifier, vous devez écrire dans ce port la valeur souhaiter, puis ajuster le port 1Dh

Ports 20h à 22h	Horloge temps réel (RTC)	Lecture / Ecriture

Ces ports renvoient le nombre de jours écouler depuis le p1er janvier 1970 (date arbitraire sur laquelle se base certaines fonctions de Turbo C concernant la date). A noter que le port 22h n'as que sont premier bit fonctionnel.

Port 24h	Non déterminé	Ecriture

Port 25h	Non déterminé	Ecriture

Port 27h	Police de caractère	Lecture / Ecriture

Ecrire dans ce port change la police de caractère, passant de nos lettres a des pictogrammes dont je ne connais pas l'origine. Les valeurs à y écrire sont 1xh et 3xh (x = n'importe quel chiffre). Les valeurs 5xh et 7xh bloque la calculatrice. Toutes les autres valeurs rendent la police normale. Le but réel de ce port n'est pas encore connu.

Port 28h	**Non déterminé**	**Ecriture**

Port 2Ch	**Non déterminé**	**Ecriture**

Ports 2Eh et 2Fh	**Non déterminé**	**Ecriture**

Port 31h	**Non déterminé**	**Ecriture**

Port 32h	**Non déterminé**	**Ecriture**

Port 33h	**Non déterminé**	**Ecriture**

Port 34h	**Vitesse interne du CPU**	**Ecriture**

Ceci permet de ralentir la vitesse de fonctionnement de la calculette. La valeur par défaut est 0, plus la valeur est élevée, plus la calculatrice ralentie. La valeur maximale est 0Fh. La combinaison de cette valeur avec la valeur 2 dans le port 4h bloque la calculatrice, celle-ci ne pouvant plus assuré le rafraîchissement de l'écran.

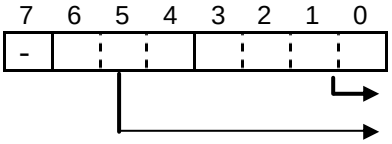
Ports 35h à 39h	**Non déterminé**	**Ecriture**

Ports 3Ah et 3Bh	**Non déterminé**	**Lecture / Ecriture**

Ports 3Ch et 3Dh	**Non déterminé**	**Lecture**

Port 44h	**Port de communication**	**Lecture / Ecriture**

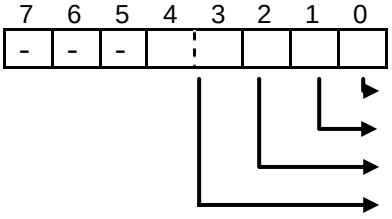
Ecrire sur ce port modifie les paramètres de communications. Y lire renvoie si elle est disponible une valeur envoyée.



Contrôle la vitesse de transfert.
Mode de transfert :
011b : Mode réception
101b : Mode envoi
111b : Mode bidirectionnel

Port 45h	**Port de communication**	**Lecture / Ecriture**

Type de transfert.
En écriture, ce port autorise la communication.
En lecture, il permet de vérifié l'état de celle-ci, selon le schéma suivant :



1b : Un octet à été envoyé
1b : Un octet à été reçu
1b : Un octet n'a pas été lu alors qu'il avait été reçu, un autre a pris sa place
1b : Erreur de reception

Port 46h	Port de communication	Ecriture

Sert de buffer pour l'envoi de données.

Port 47h	Port de communication	Ecriture

Vitesse de transfert (Voir section port de communication).

Port 4Eh	Non déterminé	Ecriture

Port 50h	Non déterminé	Ecriture

Port 52h	Non déterminé	Lecture

Port 54h à 5Ah	Mémoire EMS	Ecriture

Ces ports servent a mettre en mémoire une partie définie de la rom et de la flash. Tout ces ports on chacun cette capacité, mais chacun d'eux va mapper la mémoire à un endroit bien précis, en fonction de leurs numéros. De plus, les valeurs sont écrites dans ces ports en fonction du block de donnée de la rom ou de la flash a mapper. Les tableaux suivant indiquent les valeurs à écrire et la zone ou mappent les ports.

Valeur à écrire dans les ports	Type	Zone de Flash/Rom correspondante	informations
A0h	Flash	0000h : 0000h - 1000h : FFFFh	Zone système Lecteur A :
A1h	Flash	2000h : 0000h - 3000h : FFFFh	Langue en cours d'utilisation
A2h	Flash	4000h : 0000h - 5000h : FFFFh	L:\
A3h	Flash	6000h : 0000h - 7000h : FFFFh	M:\
A4h	Flash	8000h : 0000h - 9000h : FFFFh	N:\
A5h	Flash	A000h : 0000h - B000h : FFFFh	O:\
A6h	Flash	C000h : 0000h - D000h : FFFFh	P:\
A7h	Flash	E000h : 0000h - F000h : FFFFh	Q:\
C0h	Rom	00000h : 0000h - 01000h : FFFFh	Zone système de base Menu constructeur
C1h	Rom	02000h : 0000h - 03000h : FFFFh	Menu constructeur Sauvegarde A:\
C2h	Rom	04000h : 0000h - 05000h : FFFFh	Langues pour le système
C3h	Rom	06000h : 0000h - 07000h : FFFFh	
C4h	Rom	08000h : 0000h - 09000h : FFFFh	B:\
C5h	Rom	0A000h : 0000h - 0B000h : FFFFh	
C6h	Rom	0C000h : 0000h - 0D000h : FFFFh	
C7h	Rom	0E000h : 0000h - 0F000h : FFFFh	C:\
C8h	Rom	10000h : 0000h - 11000h : FFFFh	
C9h	Rom	12000h : 0000h - 13000h : FFFFh	
CAh	Rom	14000h : 0000h - 15000h : FFFFh	D:\
CBh	Rom	16000h : 0000h - 17000h : FFFFh	
CCh	Rom	18000h : 0000h - 19000h : FFFFh	E:\
CDh	Rom	1A000h : 0000h - 1B000h : FFFFh	
CEh	Rom	1C000h : 0000h - 1D000h : FFFFh	
CFh	Rom	1E000h : 0000h - 1F000h : FFFFh	
D0h	Rom	20000h : 0000h - 21000h : FFFFh	F:\
D1h	Rom	22000h : 0000h - 23000h : FFFFh	
D2h	Rom	24000h : 0000h - 25000h : FFFFh	G:\
D3h	Rom	26000h : 0000h - 27000h : FFFFh	
D4h	Rom	28000h : 0000h - 29000h : FFFFh	
D5h	Rom	2A000h : 0000h - 2B000h : FFFFh	
D6h	Rom	2C000h : 0000h - 2D000h : FFFFh	H:\
D7h	Rom	2E000h : 0000h - 2F000h : FFFFh	
D8h	Rom	30000h : 0000h - 31000h : FFFFh	I:\
D9h	Rom	32000h : 0000h - 33000h : FFFFh	

DAh	Rom	34000h : 0000h - 35000h : FFFFh	
DBh	Rom	36000h : 0000h - 37000h : FFFFh	J:\
DCh	Rom	38000h : 0000h - 39000h : FFFFh	
DDh	Rom	3A000h : 0000h - 3B000h : FFFFh	
DEh	Rom	3C000h : 0000h - 3D000h : FFFFh	K:\
DFh	Rom	3E000h : 0000h - 3F000h : FFFFh	

Port 6Ch	**Non déterminé**	**Ecriture**

Ports 6Eh et 6Fh	**Non déterminé**	**Ecriture**