La Casio Graph100

Olivier Coupelon

13 février 2005

Table des matières

1 Introduction

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.

2 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 composéede :

- 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 bps.

3 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, plusieus compilateurs existent, 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. **Digital Mars** est un compilateur plus rescent que **Turbo C**, gratuit pour un usage non commercial et optimise mieux le code. C'est le compilateur adopt par la communaute.

Citons enfin **Dev86**. C'est un packetage sous Unix qui permettant de crer des executables Dos, grâce au compilateur **Bruce Evans's C**.

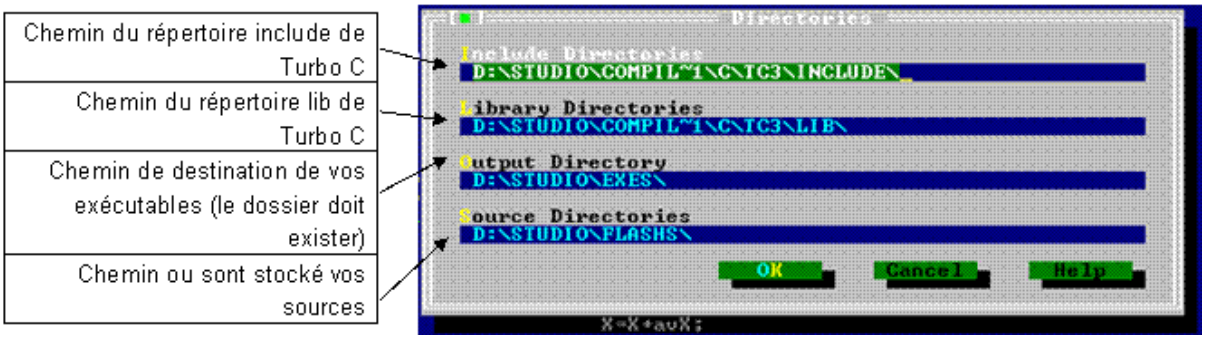
Le système d'exploitation sur lequel vous développez à aussi son importance, car comme vous l'avez peut-être déjà remarqué, tous les programmes cités ci-dessus fonctionnent sous Linux ou MS-DOS et ces derniers tendent à ne plus être compatibles avec les dernières versions de Windows.

3.1 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'editeur 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'ecran 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.

3.2 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'editeur 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).

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.

TAB. 1 – LISTE DES PRINCIPALES INTERRUPTIONS D'ACCES AU CLAVIER

4 Le Clavier

Le Clavier est l'elément 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 à 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.

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 à 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 assembleur

```
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 à 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.

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

TAB. 2 – Organisation du clavier par ligne de touches

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'ecrit dans le port numéro 13h. Mais la taille d'un port étant de seulement 8bit, écrire 16bit en 13h aura pour effet d'ecrire 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 :

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	

TAB. 3 – Correspondance des touches clavier avec la lecture binaire

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

Tab. 4 – Coposants mmoire de la graph100

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'eteindre 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

Tab. 5 – Contenu de la Sram

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 :

Tab. 6 – Contenu de la Rom

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 (zone mémoire vive)
55h	2000h - 3FFFh	
56h	4000h - 5FFFh	
57h	6000h - 7FFFh	
58h	8000h - 9FFFh	
59h	A000h - BFFFh	
5Ah	C000h - DFFFh	

Tab. 7 – Zones de mapping

Les valeurs à écrire dans ces ports s'etalent 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.

On y accède exactement de la même manière qu'on accède à la rom, seule la valeur à

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 :

Tab. 8 – Contenu de la mémoire Flash

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'ecriture 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 demapper, 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 plaons 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
; D\'ebut 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'etat 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 termin\'e,
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'ecrire 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 :

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).

	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	1	1	-	-	-	-
Formater secteur	AAAh	AAh	554h	55h	AAAh	80h	AAAh	AAh	554h	55h	2	30h
Formater chip	AAAh	AAh	554h	55h	AAAh	80h	AAAh	AAh	554h	55h	AAAh	10h

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

TAB. 9 – Opérations possibles sur la Flash

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'ecran possède un buffer vidéo, situé habituellement à l'adresse 1A20h :0000h, dans la ram. Si vous écrivez dans cette zone, l'ecran 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'ecran. 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'ecran, la fin vers le haut gauche.

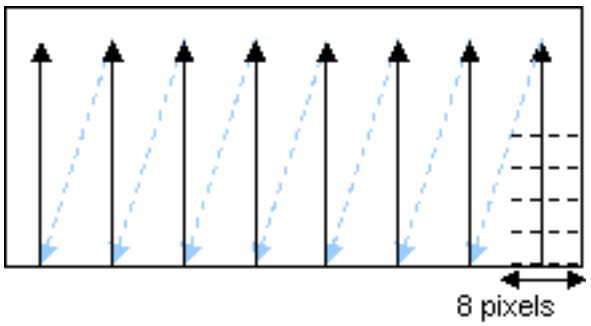


FIG. 1 – Diposition de la memoire video

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'eteindre).

6.2 Le mode noir et blanc D3

Il diffère du mode C3 uniquement par sa répartition du buffer video à l'ecran, bien plus simple à utiliser. Le début du buffer pointe toujours vers le coin bas droite de l'ecran, 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 :

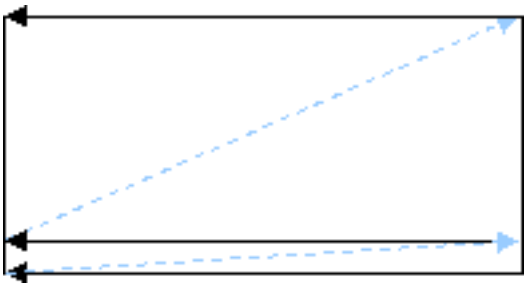


FIG. 2 – Mode D3

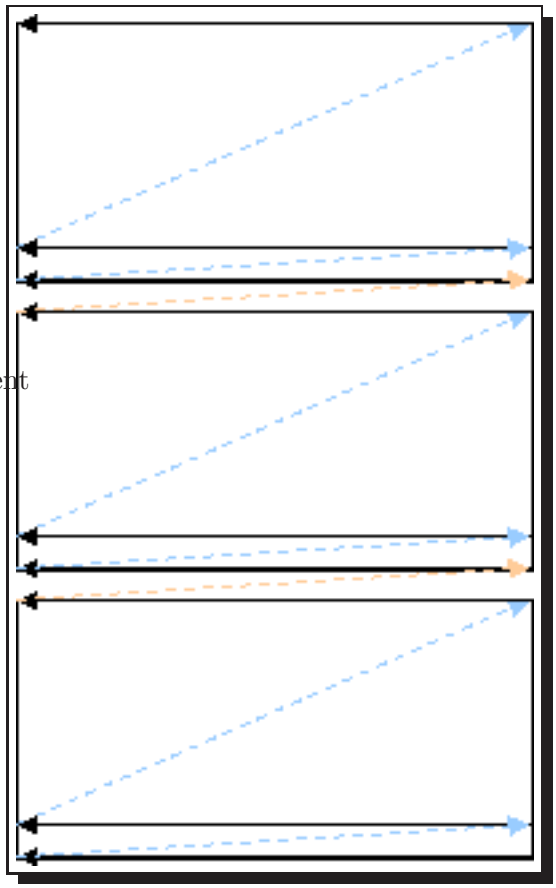
```
mov ax,0x1A20 //On met dans es le segment du buffer vidéo
mov es,ax //on ne peut affecter directement es, on passe par ax

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 clieme 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'ecran 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 1AA0h :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 deplace vers la couche suivante,
mov es,ax
or es:[si],bh // et on affiche le bit au meme endroit
```

Il est bien évidemment possible dans toutes ces routines d'effacer un pixel en remplaant 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'ecran. 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)**
Fonction22h	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'ecran, 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'ecriture sur la flash qui est utilisée ici. La fiche technique pourrait expliquer plus précisément son fonctionnement, mais on ne connaît pas le type d'ecran de la graph100. (Probablement de marque Nec). La séquence est écrite aux adresses contenues en E000h :0010h et E000h :0020h.

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									

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.

Tab. 10 – Séquence utilisée pour communiquer avec l'ecran

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.

Interruption 7Ch	**Rafraîchissement de l'écran (BIOS)**
Fonction 24h	Ah=24h

Cette fonction a pour effet de rafraîchir l'ecran. Tout comme la fonction 22h, elle utilise le système de séquence pour communiquer avec l'ecran. 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).

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'ecrire 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	9600	7200	4800	3600	2400	1800	1200
0x74	28800	19200	14400	9600	7200	4800	3600	2400
0x78	57800	38400	28800	19200	14400	9600	7200	4800
0x7C	115200	76800	57800	38400	28800	19200	14400	9600

TAB. 11 – Vitesse de communication (en Bauds par secondes)

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.

Attendre (Assembleur) :

```
in al,0x45 ;prend la valeur du port 0x45, qui n'est pas celle
;que l'ont y a \'ecrite pr\'ec\'edemment
and al,1 ;ceci pr\'eserve le premier bit de al (n$^{\circ}$0),
;en mettant les autres a z\'ero.
Jz Attendre ;le bit n$^{\circ}$0 vaut 0, donc l'envoie de l'octet
n'est
;pas encore termin\'e, 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 (7.4, p.22).

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.

Attendre (Assembleur) :

```
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

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

Tab. 12 – Instructions supplementaires du Nec V30Mx

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.

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 :

Adresse	Contenu
0000h : 0000h - 0000h : FFFFh	Taille maximale de la ram en mode réel, 640Ko. Mais le graph100 n'en possède que 256Ko.
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	

Tab. 13 – Mémoire Principale d'un PC

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'epoque, on utilisait une fenêtre de 64Ko à l'intérieur de laquelle s'affichaient 4 zones de mémoires EMS. C'etait la version 3.2 de l'EMS, la 4.0 ne s'etant 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'emulation

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'emulation.

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 prsence au sein du Nec V30Mx. Cependant l'xcution de l'instruction BRKEM n'est pas sans effet. Aucun test n'a pu prouver que celle-ci activait le mode d'mulation. La graph100 n'est peut-être pas concerne par ce mode.

8.3.1 BRKEM

Cette instruction sur 3 octets permet de passer en mode d'emulation. 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.

8.3.2 RETEM

Cette fonction sort du mode d'emulation 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.	
Fonction3	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.	
Fonction4	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.

TODO : Correction de cette partie!!

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'ecrire 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'ecran 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 à 003	CPU : Division par zéro
9	024 à 027	IRQ1 : Clavier
10	040 à 043	BIOS : Fonctions vidéo
11	044 à 047	BIOS : Détermination configuration
12	048 à 04B	BIOS : Détermination taille RAM
13	04C à 04F	
14	054 à 057	
16	058 à 05B	BIOS : Interrogation du clavier
19	064 à 067	BIOS : Démarrage à chaud (ALT+CTRL+DEL)
1A	068 à 06B	BIOS : Lecture date et heure
1B	06C à 06F	Touche Break actionnée
1C	070 à 073	Controle le timer
20	080 à 083	DOS : Terminaison du programme
21	084 à 087	DOS : Fonction de DOS
22	088 à 08B	Adresse de routine DOS fin du programme
23	08C à 08F	Adresse de routine CTRL-BREAK du DOS
24	090 à 093	Adresse de routine d'erreur du DOS
25	094 à 097	DOS : Lecture disquette/disque dur
26	098 à 09B	DOS : Ecriture sur disquette/disque dur
27	09C à 09F	DOS : Fin programme, laisser résident
28	0A0 à 0A3	
29	0A4 à 0A7	
2B	0AC à 0AF	
2C	0B0 à 0B3	
2F	0BC à 0BF	
41	104 à 107	
44	110 à 113	
45	114 à 117	
47	11C à 11F	
48	120 à 123	Mémoire EMS
4A	128 à 12B	RTC
4B	12C à 12F	Mémoire EMS
4C	130 à 133	
4D	134 à 137	
4E	138 à 13B	
4F	13C à 13F	
51	144 à 147	
53	14C à 14F	Ecran : Mise à jour de l'affichage
58	160 à 163	
5C	170 à 173	Général (Ecran, flash...)
5E	178 à 17B	
5F	17C à 17F	
7C	1F0 à 1F3	Contrôle de l'ecran
7D	1F4 à 1F7	

Tab. 14 – Vecteur d'interruption de la graph100

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 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'etant 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.

14.1 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.

Adresse	**Contenu**	**Taille**
00h	Appel de l'interruption 20h	1 mot
02h	Adresse de bp	1 mot
0Ah	Copie du vecteur d'interruption 22h	2 mots
0Ch	Copie du vecteur d'interruption 23h	2 mots
2Ch	segment du block d'environnement	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	

Tab. 15 – Structure du PSP

Ces données bien que fragmentaires aiderons certainement les programmeurs qui 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.