

TRANSLATED INTO ENGLISH BY [Mohamed Ibrahim](#) USING SYSTRAN PREMIUM 4

<http://www.geocities.com/algebrafx2>
algebrafx2@hotmail.com

Programming Guide

The ALGEBRA FX

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

2nd Edition
May 2002
Translation finished 3rd of
Sept 2003

GENERALITES

This guide applies to CASIO Graph100, Graph100+, Fx2.0 and Fx2.0+. Sont contained evolves/moves with each updates, thus take care to supervise its evolution, by the means of my site, www.gprog.tk.

The programming of Graph100 is a complex work because it requires a perfect control of the hardware it has.

I progressively with this tutorial will present the various elements point by point to you constituting it, like their programmings.

Contents:

1°	<i>The knowledge of the system</i>	4
2°	<i>Tools necessary for programming</i>	5
2.1	Turbo C 3.0	5
2.2	NASM (Netwide To assemble)	6
3°	<i>Text Display</i>	7
4°	<i>The Keyboard</i>	8
4.1	The Keyboard – Access by interrupts	8
4.2	The Keyboard – Access direct in memory	8
4.3	The Keyboard – Access by the ports	8
5°	<i>Memory</i>	11
5.1	Memory SRAM	11
5.2	the ROM	11
5.3	the flash	12
5.4	Writing on the flash	13
5.5	Another method access with the flash/ROM	14
6°	<i>Use of the screen</i>	16
6.1	The normal mode C3	16
6.2	Mode black and white D3	16
6.3	The level mode of gray dB	17
6.4	The level mode of gray CB	17
6.5	The interrupt 7Ch	17
7°	<i>The communication port</i>	19
7.1	Configuration of the port	19
7.2	Sending of data	19
7.3	Reception data	20
7.4	Closing of the port	20
7.5	Characteristics	20
8°	<i>The microprocessor Nec V30Mx</i>	21
8.1	Instructions of bits	21
8.2	Lim EMS 4.0	21
8.3	Mode of emulation	22
9°	<i>Timers</i>	24
10°	<i>The ROMdisk format</i>	25
11°	<i>RXE format</i>	27
11.1	Operation Analysis	27
11.2	In practice	27
12°	<i>BIOS</i>	28
13°	<i>Interrupts</i>	29
14°	<i>Character font</i>	31
15°	<i>The system of exploitation ROM-DOS</i>	32
15.1	Psp	32
	Appendix A - CASIO ASCII Codes	33
	Appendix B - Interrupts	36
	Appendix C - The BIOS Variables	39
	Appendix D - Communications Ports	41

1° The knowledge of the system

Several elements of the graph100 must be explained to you before being able to begin the programming. The graph100 has in particular its own operating system, ROM-back 6.2 of Datalight (www.datalight.com). It is compatible at 100% with MS-DOS from Microsoft, which makes the programming easy. However, we will see it thereafter, better is worth to try to do some to accelerate the speed of execution of the programs. Because the graph100 is not other than a computer having had its hour of glory in the years 1980.

It is made up of:

- A Microprocessor 16bit **Nec V30Mx** given rhythm to approximately 8 Mips (Million of instructions per seconds).
- 256Kb of **SRAM** memory
- **ROM** memory of 4Mb containing the original configuration of the graph100
- 1Mb of **Flash** memory, this one having the capacity to be rewritten several times while storing the information which it contains
- A keyboard (little practises to access texts, and nonstandard when with the codes of the keys)
- A screen LCD of 128 pixels broad and of 64 pixels height. It is able to display up to four levels of gray (gray dark, gray, gray clearly and white), although only three are not really accessible, i.e. visible. It can also display only one marked black and of the white, it is the mode which you know and which you can observe when you use the computer normally.
- A communication port which can communicate at speeds going up to 115200 bps

2° Tools necessary to the programming

It is a question of finding the best compiler for the selected language. However here, there are the embarrassment of the choice, because the programming for PC does not go back to yesterday, and the compilers are all the more numerous. Here thus what I regard as being the best compilers for the graph100, for the assembly languages and C, languages themselves best adapted to the graph100 because of the speed of the first, and the flexibility of the second.

For Assembleur I advise the use of **Nasm** (Netwide Assembler), compiler free and very powerful assembler, not having anything to envy Masm of Microsoft or Tasm from Borland. Moreover, there is a graphic environment for this compiler, which simplifies its use largely.

For the language C, my choice turns to two compilers, having each one their advantages. **Turbo C** of Borland in version 2.01 (free, www.borland.com) or 3.0 (paying but more powerful). Do not consider the acquisition of later versions, because the 3.1 and the 4.5 compile only the executable ones for Windows. The 4.0J (in Japanese) when with it, is less powerful for our processor.

Another free and powerful compiler is also available, **Pacific C**. Although complex to use, it contains advanced functions as specific compilation for the processors Nec V25 (near to our) as well as the possibility of creating the binary ones (which make it possible to do without the operating system, but this function is paying).

The outsider here is called **Paradigm C++**. This compiler with faculty to do all that make the preceding compilers, compiles specifically for Nec V30 (amongst other things), and functions completely under Windows. This small jewel has a crippling defect unfortunately, that to cost the trifle of US 2495\$, approximately 2846€ (If somebody has it, I would like that it informs me on his compatibility with the graph100, because I never could use it in complete version, the version of demonstration well too being limited).

The operating system on which you develop with also his importance, because as you perhaps already noticed it, all the programs quoted above function under MS-DOS and tend not to be compatible more with the last versions of Windows.

2.1 Turbo C 3.0

Several parameters of Turbo C 3.0 must be correctly regulated if one wants to be able to compile functional programs and of minimal size. For launching turbo C, it is enough for you to carry out file TC.EXE in the repertory bin. The editor of Turbo C opens then. It is since its menu that we will operate.

Here :

- Under *the Option* mitre - > *Compilers - > Code Generation*, select Tiny.
- Under *Option - > Compilers? Advanced Codes Generation* choose in Floating Point is **none** if you do not use numbers does not have comma, that is to say **Emulation**. To also select 80286 pennies Instruction Set. Enfin under Options, strip **Debug information in Objs** and select **Fast floating not** if you use the numbers has comma.
- Under *the Option* mitre - > *Directories*, take care to fill the 4 following fields correctly :





- Under *Environment - > Preferences* choose 43/50 lines, you will have more lines of code to the screen what improves legibility.

If you decide to use additional files of your creation to reduce your sources, you have two possibility: To join together in a project (to create a project to make *Project - > Open Project* then to type the name of the project file that you wish to create), or more simply, before compiling, to make *File - > Change Dir* and to select the place or are stored your file, but then you will have to carry out this operation with each time you start again Turbo C.

To compile a new program, it is enough for you to open it or to create it with small File, then to click on *Compile ->Build All*. Your program, if it does not comprise an error, is then available in its executable version in the repertory which you selected thanks to small Directories.

These parameters are in parts valid for the other versions of the compiler Turbo C of Borland.

2.2 NASM (Netwide Assembler)

Nasm is controlled has little close in the same way that Turbo C In fact all interfaces DOS resemble each other a little. To reach to the graphic editor, it is enough for you to download (if it is not present) small Nasm-ide. Then, go in the repertory of Nasm, and launch NASMIDE.exe. The graphic interface appears.

- Under *Options - > To assemble*: framed **Nasm Location**, enter the complete path and the name of Nasm. For example c:\Nasm\nasm.exe. Under Target, also choose **Com executable binary file**
- Under *Options ->Directories*: fill the two fields of the same manner as for Turbo C above
- Under *Options ->Environment*: choose **43/50 lines**.

Then, to create and carry out a program, made *File ->New*. Enter framing it blue your code, then back up it. To compile it, made *Assembler ->Assemble* then *Build*. Your program is compiled and ready to be to carry out (if no error were detected).

3° Display of a text

To display a text under graph100, it is exactly the same thing as to display a text under MS-DOS. To display a text with the screen, the BIOS of the graph100 as ROM-back place at our disposal a series of interrupts, than we will use. However, out of C, their use is transparent, but you must know that your programs there appeal, that can be useful if you wish to develop without ROM-back. As nothing is worth a good example, here are two, one out of C, the other in Assembleur.

Code C displaying the Bonjour text has the screen:

```
#include <stdio.h>    // The prototype of the function contains printf();

void main(void)      // Beginning of the principal function, it is the body of the program
{
printf("Bonjour");   // Syntax allowing to display the text hello the screen has
}
}
```

The same thing in Assembleur for Nasm :

```
[BITS 16]                ; Nous let us compile for a processor 16 bit
[ORG 0x0100]             ; We create one .com
[SEGMENT .text]         ; Début segment text, which contains the executable code of
                        ; Our program
mov si,Message          ; Met the address memory of the character string message
call AfficheText        ; in the register If, then calls the AfficheText routine

mov ax,0x4C00           ; Leave the program.
int 0x21

AfficheText:           ; Beginning of the AfficheText routine
mov ah,0eh              ; The function, of the interrupt to be used
                        ; est la 0x0e
xor bl,bl               ; It requires here that bl is worth 0
.Chsuite mov al,[ds:si] ; al must contain the character to be displayed
        cmp al,'$'      ; If Al contains the character $,
                        ; the end of the chain was reached
        je .Chfin       ; In this case one jumps at the end of the routine
        int 10h         ; If not, one displays the character in Al,
        inc si          ; One passes to the following character
        jmp .Chsuite    ; Et one returns to the capture of the character in Al
.Chfin  ret             ; Here one leaves the routine to return to
                        ; main program

[SEGMENT .data]         ; Zone of definition of the data

Message db 'Bonjour$'   ; One definite the character string Message.
```

A short glance on these listings enables us to see that the Assembleur code is much longer than C In fact, when one programs in Assembleur, one writes almost what the processor includes/understands, words being replaced by a binary code.

By compiling these two examples, one attends another phenomenon rising from the language used, which influences the size of the file. According to the compiler, the first example will take between 4000 and 8000 bytes, whereas the program assembler takes 50 of them. Because C uses headings, preset functions, which are necessary to the correct running of the program, but which n the other hand slows down it, make it look fatter. C remains however the fastest language of its category.

4° The Keyboard

Keyboard is the element from which the user will use your program. It is thus necessary to know of it perfectly its use and its operation. 3 types of access are possible, with their advantages and their disadvantages.

4.1 The Keyboard – Access by interrupts

It is the simplest method of access to the keyboard. It simply consists in calling an interrupt of the back or BIOS, and to interpret their answer.

NONEXHAUSTIVE LIST OF The PRINCIPAL INTERRUPTS OF ACCESS TO The KEYBOARD			
Functions	Code	Output	Comments
Waits until a key is pressed	<code>mov ah,08h</code> <code>int 21h</code>	value of the key in AL	
Test if a key is pressed puts its value in the buffer (but does not expect any)	<code>mov ah,01h</code> <code>int 16h</code>	The value of the key in the buffer puts	If no key is pressed, one leaves by a <code>jz</code> Fast function, but blocks the access to the menu
	<code>mov ah,0bh</code> <code>int 21h</code>		Slower than precede
Reads a key in the buffer (if there is one of them)	<code>mov ah,07h</code> <code>int 21h</code>	Al takes the value of the key pressed	Vacuum the buffer
	<code>mov ah,08h</code> <code>int 21h</code>		
	<code>mov ah,10h</code> <code>int 16h</code>		This function is faster than the others.

Note: The fastest interrupt is always 16h, because it is the BIOS which manages the access to the keyboard, while the interrupt 21h is an interrupt of the back.

4.2 The Keyboard – Random access in memory

Let us enter more in detail operation of the keyboard: Any pressure of a key of the keyboard generates interrupt 9. This action will put in memory the value of the key which was activated. This zone is located in 0000h: 04Ch. Pour to reach it, it is thus enough to make :

In C

```
peekb(0x41, 0xC) ; // this function returns the value of the key pressed
```

In asm

```
mov ax,0x41
mov es,ax
mov si,0xC
mov al,[es :si] ; Al takes the value of the key initially pressed.
```

Thanks to this, your programs will function much more quickly without taking more place than simple `getch();` . But this type of access can not be useful, because the function `getch();` or the interrupts offer very appreciable and well proportioned times of repetitions, which can be useful in an application.

4.3 The Keyboard – Access by the ports

The access to the keyboard directly through the communication port offers exceptional advantages in term speed, and especially multiple management of keys. It is the most direct means and rapid to reach a key. Unfortunately, it is not as simple as for the types of preceding accesses.

To know if a key were activated, search will be carried out by line of keyboard, the answer will be in due course notified by column of keyboard.

Organization of the keyboard by touchline :

```
0 : On                2 : 1 a -
1 : 0 a EXE          3 : 4 a /
```

4 : 7 a DEL
 5 : a+b/c à →
 6 : X,θ,T à tan
 7 : ALPHA a ESC

8 : SHIFT a MENU
 9 : direction keys
 10 : F1 à F6

We must take a variable of 16bit initialized has 0 and to position the nth bit has 1, according to the number of line. It is about a simple shift of to the 10bit maximum since there are 10 lines. For example, if one wishes to test the directional keys, our variable will have this form : 0000000001000000

For the shift, out of C one will write:

```
unsigned short Var ; // create a variable of 16bit
```

...

```
Var=0 ;
```

```
Var<<n ; // or N is the number of the line
```

In Assembler:

```
mov ax,0
```

```
shl ax,n ; or N is the number of the line
```

Once our variable ready, one writes it in the port number 13h. Mais the size of a port being of only 8bit, to write 16bit in 13h will cause to also write in the port 14h, but you do not worry this is completely normal. In this manner, the controller keyboard took note of the key which we wish to test, or rather line because it acts for the moment of a line. The controller keyboard reference then the value of this line in the port 13h, on 8bit only.

For the operations of input left out of C, one will use `inportb` and `outport`, out of assembler `in` and `out`. We must now read the value sent by this port and interpret it.

If it is worth 0, no key was pressed. If not, its binary value should be examined.

The latter is given this time C_i according to the number of column of the key.

The table below represents the key corresponding to the final crossing of the lines and the columns:

Correspondance des touches clavier avec la lecture binaire								
N° line	N° of the columns							
	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,θ,T	log	ln	sin	cos	tan	
7		Alpha	Vars	^	Esc			
8		Shift	Ctrl	Optn	Menu			
9		Left	Up	Down	Right			
10		F1	F2	F3	F4	F5	F6	

For example, if you test line 10, and that the value column 01010000 is returned, that means that the keys F1 and F3 had been pressed.

The algorithms used can have several forms according to whether one simply wishes to know if an unspecified key is pressed or if a precise key is targeted.

Example of test out of C (Cette fonction tests if a specific key is pressed and returns 1 if necessary. One sends like parameter the number of the line and the value returned by the keyboard if the key were pressed) :

```
int keyport(int ligne,int valeur) // Check if the selected key
// is pressed.
{
outport (0x13,(1<<ligne));
if (inportb(0x13)==valeur) return 1; // The key is quite pressed,
```

```
else return 0;           // 1 is returned
                        // The key is not pressed,
                        // 0 are returned
}
```

However, while using this system of access to the keyboard, you will encounter a problem.

The keys pressed will continue to be put in the buffer touches (zone containing a list of the keys pressed) graph100. This is awkward if you wish thereafter to use another type of access keyboard, it will initially be necessary for you to empty this buffer, then only after collecting a key.

To solve this problem, several solutions are offered to you, my preferred and who gives the best results, consists in decontaminating interrupt 9. This offers multiple advantages exposed below:

- The buffer key is not supplied any more, because it is the role of this interrupt.
- The return to the menu by the pressure of the Menu key, and the modification of contrast by the keys Shift+Droite or Shift+Left are decontaminated.
- Your program is some slightly accelerated.

It should be noted that this interrupt can be restored constantly if one takes the precaution to back up his initial address, by using out of C the function `dos_getvect()`

To substitute it by a function of your choice (an empty function will make cavity the affaire), to use `_dos_setvect()`.

Out of assembler, it is enough for you to back up the address of this interrupt read in the table of interrupt, then to replace this address by that of the interrupt 0xFF. (see chapter on the interrupts).

5° The Memory

As we saw previously, there are several types of storages internal in the graph100, of which here names and design features:

Type	Nom	Temps d'accès	Taille	Opérations
SRAM	Nec PD442000GU-B85X-9JH	85ns max	256Kb	Read/write
Flash	Fujitsu MBM29LV800BA-90	90ns max	8Mbit soit 16x64Kb soit 1Mb	Read/write
ROM Graph100	Nec PD23C32000L	140ns max	32Mbit soit 4Mb	Read
ROM Graph100+	Oki MR53V3202K-24	120ns max	32Mbit soit 4Mb	Read

Thereafter, we will not make a difference between the ROM of the graph100 and that of the graph100+ because they are used same manner.

5.1 The SRAM Memory

It is the read-write memory of the graph100. It is located between the addresses 0000h: 0000h and 4000h: 0000h. Elle measures only 256 KB, whereas the majority of the equivalent computers had 640 Kb. Ceci largely will penalize us in the realization of our programs, because ROM-back, to compensate for this small size, compels us to use programs not measuring more 64ko, which does not simplify the things. However, the format of Rxe file of datalight makes it possible to exceed this disadvantage by modifying the heading of the executable ones. The tools allowing this transformation are included in the kits of developments of Datalight, kits which cost several thousands of dollar, this is why us will restrict themselves financially and use programs not measuring more 64Kb.

This memory with the characteristic to be persistent, it is thus possible to extinguish and relight the pocket calculator, without anything to lose work in progress.

Moreover, it is it should be noted that the video buffer and the programs basics are placed in this memory, with the addresses 1A20h:0000h and Ç20h:0000 respectively. It also contains the table of interrupt, and the data of the BIOS, as in a computer in real mode. Because here, you will have included/understood it, not question of protected mode.

Adress	Contents
0000h : 0000h - 0000h : 03FFh	Count of the vectors of interrupts
0000h : 0400h - 0000h : 04FFh	Variables of the BIOS
0000h : 0500h - 1000h : A1FFh	Zone back, to charge the programs, the variables, the environment ...
1000h : A200h - 1000h : C1FFh	Buffer video report
1000h : C200h - 3000h : FFFFh	Programs basics, matrices, lists, all elements of calculations of the graph100

5.2 The ROM

The ROM of the graph100 measurement 4 Mb. Elle contains the calculation programmes of the graph100, a backup of the zone system and drive a:, the small manufacturer, and languages of the pocket calculator. Here the way in which it is organized by segments:

Segment :	Contents :
0000h - 0FFFh	Zone basic system
1000h - 2FFFh	Menu construction
3000h - 3FFFh	Back up drive A :
4000h - 5FFFh	System Languages
6000h - 6FFFh	Vacuum
7000h - 7FFFh	? – Programs Boots
8000h - 3FFFFh	Drives B : to K :

But these addresses are the physical addresses internal ROM, and to in no case those to which one finds these data in memory.

To reach it since the memory, one needs "mapper" a zone of this disc (ROM) in the memory. Under this barbarian term, I hear in fact that one needs to say to the controller ROM which is the zone memory of 128 KB to which one wants to reach, but also the place where it must place it in memory. It is impossible to reach the totality of the 4Mb directly for the simple reason that the maximum address that one can reach with the current system of ciblage of address is F000h: FFFFh, i.e. 1Mb, whereas ROM make 4 of them.

With this intention, it will be necessary to write in a certain port a certain value according to the place where we wish to place our 128 KB and which part of the ROM we wish to charge has this place.

The following table shows in which port to write for "mapper" a zone of ROM in the zone memory of the corresponding graph100.

Port	Zone corresponding memory	Comments
54h	0000h - 1FFFh	Not to use these ports, because zone read-write memory
55h	2000h - 3FFFh	
56h	4000h - 5FFFh	
57h	6000h - 7FFFh	
58h	8000h - 9FFFh	
59h	A000h - BFFFh	
5Ah	C000h - DFFFh	

The values to be written in these ports are spread out of C0h with DFh, C0h targeting the first 128 KB of the ROM, and DFh the last (what targets well 4Mb, 32x128Kb).

5.3 The flash

It is it which will contain our programs. It measures 1Mb.

Segment :	Contents :
0000h - 0FFFh	System zone
1000h - 1FFFh	Drive A :
2000h - 2FFFh	Used language
3000h - 3FFFh	Almost vacuum
4000h - FFFFh	Contains the flashes sent by the user, i.e. normally them drives L : a Q :

One reaches it exactly in the same way that one reaches the ROM, only the value to be sent to the port changes. One sends the values of A0h to it to A7h. A0h map the first 128 KB in memory, A7h the 128 last.

The provision memory presented previously is that basic, with the acquisition of the pocket calculator. But the flash being rewriteable, it is possible that programs modify its contents. At all events, if such an operation is carried out and that ROM-back is some faded (in particular with the removal of drive a:), the system itself will restore the flash in this initial state, the writing on the flash is thus without danger.

But to write there is not as simple as to read there. Because to read there, it is enough to make peek() out of C to reach the data which one comes from mapper, whereas to write there, simple a poke() is not enough. In fact, the data on the flash are protected in writing, it is thus necessary to pass by several stages before being able to modify anything.

It is imperative of knowing that on the flash, at the time of a writing, the only possible modification is to replace (into binary) of the 1 by 0, the reverse being physically impossible to realize on an insulated bit. The only possibility of circumventing this obstacle, is to format the whole sector which one wishes to modify, which positions all the bit of this sector with 1. Once this operation carried out, one can write what one wants, the 1 will remain of the 1, or will be transformed into 0, if necessary.

This operation is very delicate, because to format destroyed all the files and data present in the sector. It is thus necessary in a first stage to back up the data in memory, then to format, then modify the data in memory, and finally to rewrite the entirety of the 64 KB modified.

To back up these data, one can use the space of the RAM between 2000h: 0000h and 4000h: 0000h, space which can however be occupied by files BASIC, which involves their loss and if necessary error messages at the time of the next stopping of the computer (in the worst case the total loss of the files BASIC).

5.4 Writing on the flash

Thus let us consider that the fact of losing the data imports little for the moment, and concentrate being maintained on the manner of carrying out the formattings and the writings on the flash. With this intention, you must first of all mapper the zone be written in memory.

Let us consider that it is the drive L:, that we mappé in 4000h: 0000 (code to be carried out: `outportb(0x56,0xA2);`).

We now wish to format the drive L:

It is enough for us simply to ask it to the controller of the flash. For that, we will give him instructions, in writing, at precise places of the sector

Code to write in asm :

```
mov ax,0x4000          ; we place are at the beginning of our
mov es,ax              ; sector
mov [es:0xAAA],byte 0xAA ; then we write 6 values
mov [es:0x554],byte 0x55 ; at quite precise places
mov [es:0xAAA],byte 0x80 ; the controller will include/understand then
mov [es:0xAAA],byte 0xAA ; that one wishes to format
mov [es:0x554],byte 0x55 ; the sector in progress
mov [es:420 ],byte 0x30  ; 420 or another value,
                        ; it does not matter here
                        ; Beginning of formatting of the flash
```

This stage it remains still two things to be made. First is to test the end of the formatting which takes a few seconds. With this intention, one will read a zone of the sector in format. This will not return the value present at the place read, but the status report of the operation. If the 6eme bit of the returned byte is null, it is that the formatting is finished.

Here an example out of assembler of the test of end of operation:

```
.Attend mov bl,[es:si] ; Ask the flash two bytes
                ; of checking (one reads does not import
                ; where in the sector, not at a place
                mov bh,[es:si] ; specific.)
                xor bl,bh      ; The 6th bit it is put ?
                and bl,0x40    ; if so, the flash is formatted
                jz .fin        ; If not the 5th bit is looked at
                and bl,0x20    ; if it is put it is that the formatting
                jz .Attend     ; is not finished,
                ; if not there was an error
                code en cas d'erreur
.fin          ret
```

The preceding examples all are in asm because these operations require a fast execution. You can include them in your programs out of C by taking care to modify their syntax if need be, and to add one of the directives suggested below, according to your compiler:

```
asm { lines of code assembler } // For Turbo C
or
#asm // For Pacific C
lines of code assembler
#endasm
```

You can also Re-write them directly out of C, with `peekb` and `pokeb`.

But remember that each sector makes only 64ko and which a drive (the drive L in fact) measurement 128 KB. It is necessary for us thus thereafter to put are with 5000h and to start again the preceding operations.

Now it would be desirable to write on our virgin flash. It is the same type of operation as the formatting, here a routine writing a byte on the flash in Assembleur :

```
mov [es:0xaa],byte 0xaa ; 3 commands to prevent the flash that one
mov [es:0x554],byte 0x55 ; wish to write a byte.
mov [es:0xaa],byte 0xa0
```

mov [es:si],al ; Al contains the value to write, and if point
; towards the offset of the sector where one wishes to write

After this, it is necessary still to call us the routine of checking, that presented higher still functioning in this case.

Here a table gathering the operations possible to carry out on the flash:

	1st Cycle		2nd Cycle		3rd Cycle		4th Cycle		5th Cycle		6th Cycle	
	Offset	Value	Offset	Value	Offset	Value	Offset	Value	Offset	Value	Offset	Value
To write	AAAh	AAh	554h	55h	AAAh	A0h	?	?	-	-	-	-
To format sector	AAAh	AAh	554h	55h	AAAh	80h	AAAh	AAh	554h	55h	xxx	30h
To format chip	AAAh	AAh	554h	55h	AAAh	80h	AAAh	AAh	554h	55h	AAAh	10h

Notes :

? : It is you who choose the offset and the value to be written on the segment in progress.
xxx : Any offset of the segment in progress (thus any value).

Will know finally that all the sectors of the flash measure 64 KB, except the first 64 KB which are organized in 4 sectors of 16Kb, 8Kb, 8Kb, and 32Kb in the order.

For more information on the advanced functionalities of the flashes, you defer to his English handbook.

5.5 Another access method to the flash/ROM

It is also possible to reach the memory via the interrupt 48h. Mais the values to be used are then different. Here how this interrupt functions. :

Interrupt 48h	Access to the disks	
Function 0	Ah = 0	

Bl = number of the segment report in which one wants mapper (0 = 0 ; 1 = 2000h : 0000h ; 2 = 4000h : 0000h...).

Bh ≠ 6.

Al = number of the zone with mapper according to the table according to:

Zone memory flash	Number in 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

Function 1	Ah = 1
------------	--------

Bl = number of the segment report (0 = 0 ; 1 = 2000h : 0000h ; 2 = 4000h : 0000h...).

Bh ≠ 6.

Return: Al takes the value of the zone memory mappée at the place targeted by Bl, value corresponding to that seen above by Ah=0.

Function 2	Ah = 2
------------	--------

Return in ax the word written in 0040h : 00C6

6° Use of the screen

The screen of the graph100 is composed of 8192 pixels, 128x64. It is able to display either in black and white, or in levels of gray. To be able to store pixels, the screen has a video buffer, usually located at the address 1A20h:0000h, in the RAM. If you write in this zone, the screen will be directly affected.

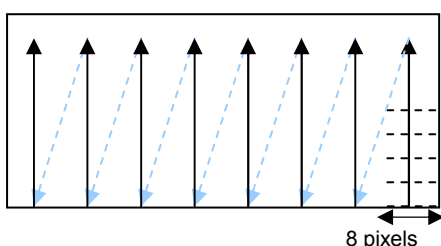
To select a mode, you must enter his value port 2 of the computer. For example to pass in mode dB, you will have to make :

In C :
outportb(2,0xDB);

In Assembler:
mov al, 0xDB
out 2, al

6.1 Le mode normal C3

The C3 mode is the mode used by defect by the graph100. It is thus a black and white mode, a bit representing a pixel to be displayed with the screen. As the buffer always contains 8192 pixels, it measures 8192 bit is 1024 byte, 1 KB. The bits set out again by columns of eight, the beginning of the buffer pointing towards the corner low right of the screen, the end to the top left.



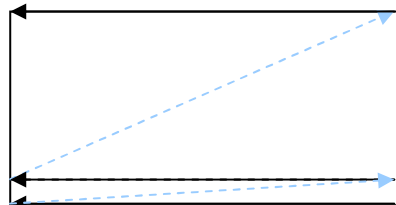
The diagram opposite illustrates the way in which the data of the video memory are laid out with the screen.

The encountered problem when one wishes to light a pixel alone, it is that one not be able to handle the data bit with bit, but only byte by byte (1 byte = 8 bits). The method thus will consist in creating an algorithm making it possible to position on the good byte to treat in the video buffer, then to use a command specific to the Nec processor which will make it possible to modify only one bit of this byte, that which one wishes to light (the case echeant to extinguish

it)

6.2 Black and white mode D3

This video mode (too much) is at the present time still very used little by the programmers. It differs from the C3 mode only by its distribution of the video buffer to the screen, much simpler to use. The beginning of the buffer always points towards the corner low right of the screen, and the end to the top left.



The only difference is that it displays the contents of the buffer lines by lines as shows it opposite the diagram. It is then much easier and intuitive to draw something in this mode, and also slightly faster. With this intention, one will use for example the following algorithm. The display requiring a great speed of execution, the assembler is once again required :

```
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 represent the X-coordinate
mov si,y           ; Y the ordinate
shl si,4           ; A line measures 16 byte (128bit). if point towards the good line
mov cl,bl          ; 0<=bx<128 Thus one can handle bl instead of bx.
                  ; One places bl then in cl.
shr bl,3           ; Divide bl by 8.The result is in bl
and cl,7           ; One keeps the 3 last Cl bit, it is the remainder of division
add si,bx          ; if point towards the good byte (16*y+x)
db 0x0F,0x14,0x0C7 ; set1 bh,cl
                  ; At 1 Cl puts ith bit of bh (which was null until)
                  ; This command is explained more precisely in
                  ; the chapter devoted to the processor
or es:[si],bh      ; One places bh in video memory by taking care of
                  ; not to erase the others
```

This routine was optimized to the maximum to profit from fastest of the displays point by point.

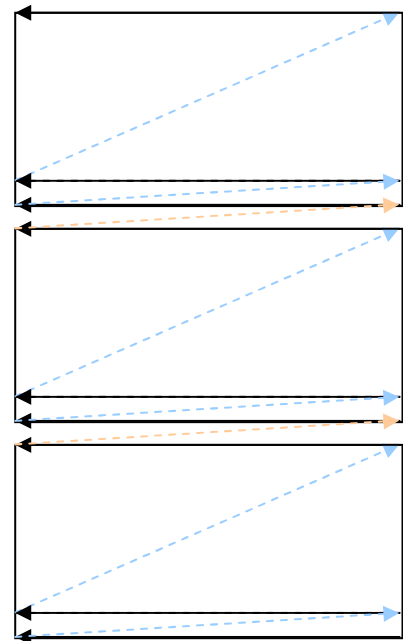
6.3 The level mode of gray dB

This mode allows the use of three levels of gray: the white, gray and dark gray.

Its use remains very of rather simple time, because it uses a system of layers. A layer is in fact a superposition of several pages with an aim of displaying only one moderate. These layers have the same structure exactly as the single layer used in the D3 mode, except that it is necessary to use three consecutive. The size of the video buffer passes then from 1024 bytes to 3072 bytes. But only two layers are really useful here, the first and the third. The second brings an almost unperceivable nuance, therefore useless.

Then, calculation is simple, more one superimposes lit bits, more the result with the screen is dark. By considering only the two useful layers, it brings out the pallet from it according to:

0 bit on : white
 1 bit on : clear gray
 2 bits on : dark gray



These layers are contiguous, without mark of distinction in the buffer.

Thus if the first starts (it is the usual case) in 1A20h:0000, the second will start in 1A60h:0000 and the third in 1AA0h:0000. One applies to it then the rules of calculation presented higher with the possibility of gaining some cycles of clock if one wishes to draw a dark gray pixel by simply adding at the end of the preceding routine the following code:

```
mov ax,0x1A80 ; One moves towards the following layer,
mov es,ax
or es:[si],bh ; and one displays the bit at the same place
```

It is obviously possible in all these routines to erase a pixel by replacing the following lines simply:

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

That one will replace by:

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

6.4 The level mode of gray CB

This mode is not very useful because its counterpart considering above makes it possible to carry out same the operations more simply. Will know simply that it functions like mode dB, but by taking example on the C3 mode (and not the D3 mode).

6.5 The interrupt 7Ch

The purpose of the interrupt 7Ch is to control the screen. Here is the detailed action. This Interrupt has functions 2, 3, 0Ch, 0Dh, 0Fh, 20h, 21h, 22h, 23h, 24h, but their operations yet completely are not known.

Interrupt 7Ch	Contrast Control	BIOS
Function 22h		Ah=22h

It is a this function which acts on contrast.

bl = 0 : Increase contrast
 bl = 1 : Decrease contrast

To use it, it is enough to put the value adequate in ah and bl and to call the interrupt by a command such as `int 0x7C`.

The value returned in AI is the level of current contrast.

To function, it makes a request with the controller of the screen, without passing by the communication port, simply by writing a precise sequence of values at a given place that the controller will receive and interpret. It is the same method as for the writing on the flash which is used here.

The chart could explain its operation more precisely, but the type of screen of the graph100 is not known. (Probably of Nec mark).

The sequence is written with the addresses contained in E000h :0010h et E000h :0020h.

Sequence used to communicate with the screen									
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									

Legend :

	To write these values with the offset contained in E000h:0010h, the E000 segment
	To write these values with the offset contained in E000h:0020h, the E000 segment
x :	Byte to write to carry out the desired operation.

This table is read in the traditional direction of reading, of from top to bottom left on the right then. It is necessary to write these values with the addresses given sequentially starting with the first value of the table.

The value of X here is not yet clearly identified.

One can note that the length of the sequence to be written is not negligible and takes much place. Also, if the speed of change of contrast imports little in your applications, you may find it very beneficial to use the dedicated interrupt.

Function 24h	Ah=24h
--------------	--------

This function causes to refresh the screen. Just like the function 22h, it uses the system of sequence to communicate with the screen. However, if you wish to intercept this function, rather prefer to act on the interrupt 53h, interrupt material having amongst other things like effect to call this function.

Here, the value of X must be read beforehand in E000h:00E5h. The sequence to be used is the same one as for the function 22h.

7° The communication port

The graph100 has a communication port of the asynchronous type, able to transfer from the data to speeds going until 115200bps. One reaches it by the intermediary of the ports (attention very of time not to confuse the communication port and the port intern graph100).

7.1 Port Configuration

To be able to function the communication port of the graph100 need has to be initialized. With this intention, it is necessary lit bits 5 and 6 of the port interns 11h. This port returning the data written on itself, it is enough to read its value, to put bits 5 and 6 A 1, and to write the new value.

Assembler:

```
in al,0x11      ; The value of the port is read 0x11
or al,0x60      ;0x60 = 01100000b this instruction does not modify
                ; that bits 5 and 6 of Al
out 0x11,al     ; One rewrite the new value.
```

After this, the communication port is has 4,92V cd., and thus ready A to communicate.

Thereafter, it is necessary to write in the port interns 0x44 value 0, this allows initialized communication.

Lastly, according to the transmission speed that you wish to obtain, you must sent in the port 0x47 one of the following values, the value of the port 0x45 will be useful later for you. Note that several combinations can be possible, according to selected speed.

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

Once these operations finished, the communication port is ready to receive or has to send information.

7.2 Data Sending

- ✓ Proceed first of all to the configuration of the port (see above).
- ✓ Then, you must put the byte to be sent in the port 0x46.
- ✓ Put now the value 0x41 in the port 0x45.
- ✓ We go now used the remaining value of the table seen during initialization (0x70, 0x74, 0x78, or 0x7C). Send the value corresponding at the speed which you chose in the port 0x44.

From this moment, the communication began and the byte is spirit to be sent. It is now necessary to await the end of sends this byte for continued used the communication port. For checked that the byte was indeed sent, you must read the value contained in the port 0x45 and wait until the bit n°0 of this port is lit. For that, you must write a loop of test, like that presented below.

Assembler :

To wait :

```
in al,0x45      ; takes the value of the port 0x45, which is not that
                ; that have it there wrote previously
and al,1        ; this preserve the first bit of al (number 0),
                ; by putting other A zero.
Jz Attendre    ; the bit number 0 is worth 0, therefore byte sends it is not
                ; not finished yet, therefore one starts again tests it.
```

Now that the byte is sent, one can returned other bytes. For that, you must:

- ✓ to put the byte sent in the port 0x46.
- ✓ tested the end of sending of the byte.

When all your bytes one sent, you must close the port (look 7.4).

7.3 Receiving data

- ✓ Proceed first of all to the configuration of the port (see above).
- ✓ Put the value 0x41 in the port 0x45.
- ✓ We go now used the remaining value of the table seen during initialization (0x70, 0x74, 0x78, ou 0x7C). Send the value corresponding at the speed which you chose in the port 0x44.

Here, you must test the arrival of a byte. For that, you must read the value of the port 0x45, then tested its bits 1, 3 and 4. If bit 1 is put and not the two others, it is that a byte arrived. If not, you started again until the test is conclusive.

Assembler :

To wait :

```
in al,0x45      ; read the value of the port 0x45
and al,0x1A     ; preserve bits 1, 3 and 4. The others are put a 0.
cmp al,0x2      ;test if only the first bit is put ( 0x2 = 10b)
jne Attendre    ; if it is not the case, the test is started again.
```

Once this test concluded, the received byte is in the port 0x44. It is thus enough to read this port and of the treaty, or of stored. If you await other bytes, start again the test of arrival, and made this until all the awaited bytes are received.

When it is the case, you must close the communication.

7.4 Closing the port

The closing of the port is very fast, it is enough for you to

- ✓ to put 0 in the port 0x44
- ✓ decontaminated bits 5 and 6 of the port 0x11

Thus the port is closed, this sui prevents any operation of reception or of sends without an initialization.

7.5 Characteristics

Thanks to the modes presented above, you can receive and sent information very quickly. But the graph100 has only 8Mhz, which imposes in the high speed data communications, and especially in reception, not to make it possible the program to carry out long spots under penalty of missing the reception by certain bytes.

To increase the speed of your code, you can also used instruction CLI (Clear interrupts) and STI (Set interrupts) in order to prevent any interrupts from disturbing your code. Attention however did not use instruction HLT after the instruction CLI, this would block the computer to the next reset.

Moreover in order to protecting the communication, it is preferable at the time of the reception of data not requiring the immediate intervention of the user to decontaminate the keyboard.

For that, it is enough to put bit 3 of the port 0x0Ah at 0. To reactivate it, it is necessary to put value 1 at it.

It is also possible to pass from the mode of reception to the mode of sends and conversely without passing by again by the procedure of initialization, as long as the port were not closed.

8° The microprocessor Nec V30Mx

Design features:

- 14 registers
- Set of 101 instructions
- Instructions of bit bangings (set1, not1, clr1 and test1)
- Special registers supporting standard LIM EMS 4.0
- Completely compatible 8086
- 8 Mips (Million of instructions per seconds), including 5 exploitable of normal use (surroundings)

8.1 Bit Instructions

One of the assets interesting of Nec V30Mx is its capacity handled the data bit with bit, or at least to quickly carry out operations on those directly and. To be able used these instructions, it is necessary for you to include them yourself in your code, i.e. to write their binary code. Because these instructions are present only in the Nec manufacturer, who provided not compiler, which does not make their uses simple. The binary code with used to compile these instructions is in the handbook "Nec User' S Manual 16-bit V series" in the pages opposite.

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

This instruction extinguishes the bit number src operand dst. Par.conséquent, with bh=12 and Cl = 3, the instruction

Clr1 bh,cl

will give bh=4 et cl=3.

Indeed: bh=12=1100b. Thus by extinguishing the 3rd bit, one obtains bh=4=0100b.

This is very useful, in particular with the graphic display of our graph100 which makes correspond a bit has a pixel.

Not1 dst,src

This instruction reverses the value of the bit number scr of the operand dst.

With bh=4 et cl=2,

Not1 bh,cl

Will give bh=0 and cl=2 bus bh=4=0100b thus by reversing the second bit one obtains bh=0000b.

One finds here also an interest practical in the video mode.

Set1 dst,src

The bit number scr of dst is put 1.

Test1 dst,src

Puts the indicator of selected has 1 if the bit number scr of dst is worth 0, and puts this indicator has 0 in the contrary case. This instruction is useful if one simply wishes to make a branch according to the bit tested, although the Test instruction can also fulfill this function.

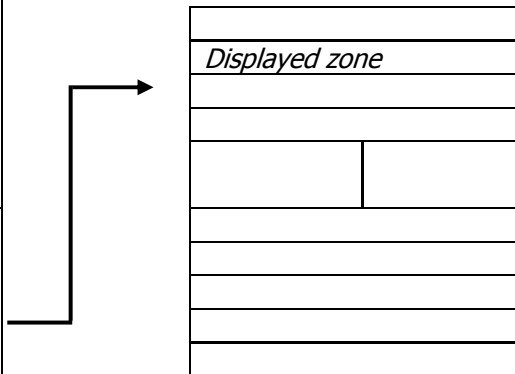
8.2 Lim EMS 4.0

Standard EMS, also known under the name of paginated memory, gives access more 1Mb of theoretically possible maximum memory. With this intention, the manufacturers and Lotus developers, INTEL and Microsoft (Lim) one creates a protocol and an expander card allowing the opening of a window in memory in which would ravel the additional RAM, without crossing the threshold of the 1Mb. Here how that was conceived:

Principale memory of a PC

Adress	Contents
0000h : 0000h - 0000h : FFFFh	Maximum size of the RAM in real mode, 640Kb. But the graph100 has only 256Kb of it.
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	Site reserved for the expander cards, the BIOS, and has the video memory. It is here that the window is located.
B000h : 0000h - B000h : FFFFh	
C000h : 0000h - C000h : FFFFh	
D000h : 0000h - D000h : FFFFh	
E000h : 0000h - E000h : FFFFh	
F000h : 0000h - F000h : FFFFh	

Memory EMS



In fact, only part of all the paginated memory is displayed has one moment given, which allows, by changing the zone displayed of used a great quantity of memory. At the time, one used a window of 64Kb inside which were displayed 4 storage areas EMS. It was version 3.2 of the EMS, 4.0 being never imposed. But this method should point out to you the access method to the ROM and the flash, because it functions exactly on the same principle, except that the window has a size of 128Kb and that one can open several of them. It is precisely the innovation which adds version 4.0 of the EMS. We thus reach our reports (ROM and flash) as a paginated memory, grace has this system.

8.3 Mode of emulation

In the instruction set of Nec V30, two of them are all A make special and can common, BRKEM and RETEM. Thanks to them, V30 is seen equipping with a new instruction set, that of the 8080. This processor at summer to develop in 1974 by INTEL. It had an addressing system on 8bit, and turned to 2Mhz. It saw cloner by other companies like Amd or Nec, this last having renamed it 8080D. Après to have undergoes some improvements, it finally bore the name of 8080AF.

In practice, this one makes it possible to completely substitute the instruction set of V30 to provide used that with the 8080AF, only instruction RETEM having been added, this one making it possible to leave the mode of emulation.

In practice, this mode is not very useful, because there is not great to make turn of the programs for 8080 on graph100.



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

Note : Certain document of NEC are contradicted about its presence within Nec V30Mx. Cependant the execution of instruction BRKEM is not without effect. No test could prove that this one activated the mode of emulation. The graph100 is perhaps not concerned step this mode.

BRKEM

This instruction on 3 bytes makes it possible to pass in mode of emulation. The two first are worth 0FFFh. The last byte contains a number of interrupt. The processor backs up information first of all enabling him to leave the emulation mode in the RAM, then change has the address contained by the vector of interrupt of the third byte, the data being then interpreted like code 8080.

RETEM

This function leaves the mode of emulation by charging of the pile the information stored by BRKEM.

9° Timers

There is two clocks on graph100. First is held up to date by interrupt 8 and Çh. It functions 2,7 times more quickly than a normal clock and is given to 0 with each extinction of the computer. One can used the grace has the interrupt 1Ch :

Interrupt 1Ch	Timer	
----------------------	--------------	--

Interrupt called after each Int 8 updating the timer BIOS.

One increments initially the word 0040h:006Ch then if it becomes null the word 0040h:006Eh and finally the byte 0040h :0070h.

The graph100 also has a clock real time (Real Time Clock). _The theory would like that our computer is a too old PC to have a STN Mais it of it is not nothing. Indeed, the graph100 has a clock real time well functioning even when the computer is stopped, and this at normal speed. One reaches it thanks to the ports 1Dh with 22h. It transmits the seconds, minutes, hours and date. It is adjustable and functions in mode 24 hours. Its tuning can be carried out either directly by the ports, or thanks to the interrupt 4Ah :

Interrupt 4Ah	RTC	
Function 0...3	Ah < 3	

Reference in dh the number of second of the STN, in Cl the number of minutes, CH the number of hours.

Function 3	Ah = 3
------------	--------

Cl = number of minutes

Ch = number of hours

The function then spends up to date the hour with that specified in cx, the seconds are garage has 0.

Function 4	Ah = 4
------------	--------

Return the number of days entered by the STN in cx.

Function 4...FFh	Ah > 4
------------------	--------

Cx = number of days.

The function updates the number of days of the STN with that registered in cx.

1° The ROM disk format

It is the format of allowance of file used by ROM-back to store the files in the flash and the ROM. Please in the following table not forget that the hexadecimal values contained in the image ROMdisk are with format INTEL, therefore under a hexadecimal editor, values of 100h will be written 0001 bus the bytes of the words are reversed two A two.

Address in the image	Length	Value	Comments
00000h – 0000Ah	10 bytes	EB28 9044 4C52 4449 534B	Heading of the ROMdisk image
0000Bh	Word	Cut sectors: 80h : 128 octets 100h : 256 octets 200h : 512 octets	One will always use the values 200h, only value that the graph100 recognize.
00011h	1 byte	According to the size of the sectors, this byte is worth 4,8 or 10h. It is then necessary to multiply this value by the number of sector necessary for the zone of the file names in ?00h.	Thus 10h*sector_number_in_ ?00h.
00013h	Word	A number of constituent sectors the drive.	
00016h	Word	A number of sectors used by the zone of validity of drive (offset 200h)	
00024h – 001FDh	474 bytes	All the bytes are worth FFh	
001Feh	Word	AA55h	
00200h	At this address starts the zone of validity of the drive who allows to check the good format of the data of the image, of the files which it contains		
00200h	3 bytes	F8 FF FF	Header of this zone
	Then, one finds groups contiguous of the form following. These groups correspond each one to a sector. The first of these groups representing the first sector of the first file of the disc (see fine table), the others the following (if it there of a).		
00203	1 byte	For the first group, this byte is worth 3. Then, for each following group, add 2 A this byte.	Groups bridging towards the sectors of the files of the image.
00204	1 word	This word corresponds to the type of data met on the targeted sector. Two cases of figures arise: - "normal" sector Pour the first group, this word is worth 40h, then for each additional group one adds 10h. - sector corresponding has an end of file. In this case, one takes the normal value while making him make a GOLD is with FF0Fh if the number of sector composing the file is odd, that is to say with F0FFh if not.	
00 ?00h	This zone starts with the sector according to the last of the zone of validity. It will contain information on the files included in the image. This information is stored by group of 3 octets, each one of these groups giving of information on a file has the time. The first group is still a header, simply containing the ROM-disk text, follow-up of 20 20 20 08. The first useful group is thus in 00?20h. There are as many useful groups of files, +1 header.		
00 ?20h	8 bytes	Name of the file without extension, in capital letter. If it is not long enough, one supplements by spaces (20h).	Useful group
00 ?28h	3 bytes	Extension in capital letter	
00 ?2Bh	1 byte	20h : Separation	
00 ?2Ch	10 bytes	The 10 bytes of this zone are has 0	
00 ?36h	4 bytes	Date and hours of modification of the file, according to the following format, 32bits taken from right to left numbers: 5bits: day	

		4bits: month 7bits: year 5bits: seconds divided by 2 6bits: minutes 5bits: hours	
00 ?3Ah	1 word	Indicate the number of sector containing the beginning of the file. For the first this value groups is always to 2. The others result from this by adding the size from the file precede in sector with its sector by departure.	
00 ?3Ch	2 words	Cut file in bytes.	
		As long as all the files of the drive do not have their useful groups corresponding, one adds some in the same form as that considering previously. When all the files have their respective groups, one goes at the beginning of the following sector, and one starts has to write the files. When one reaches the end of a file, with the master key with the following sector and one starts has to write the next one there so that 2 files cannot find itself on the same sector and than each file starts at the beginning of a sector.	

11° The RXE format

This format of files was developed by Datalight. It makes it possible to carry out programs directly since a ROM, by charging in RAM only the variables. This is useful for several reasons:

- Significant gain of place in the RAM, because one charges only what will be to modify.
- Allows to carry out programs of more than 64Kb.

But it also has a significant defect, that to slow down the speed of execution of the programs.

Because the Rxe files of executable of exe type are modified to answer waitings presented above. For this reason, and to remain compatible with MS-DOS, this format will also keep the exe extension.

It is used on the graph100 by all the programs initially present on the graph100 of more than 64Kb, i.e. all the programs of drive B: to K:

11.1 Operation Analysis

Usually, MS-DOS, to carry out programs, starts to copy them disc from which they are stored towards the memory. Information concerning the load address of the program as its size is then written in the heading of the image copied.

The modifications are made at the level zones containing the address of each part of the program, Fix-Ups. Indeed, by modifying their value, one can warn MS-DOS that the data treated or carried out are located has a precise place, on a disc out of the RAM.

These Fix-Ups are contained in the heading of executable, it is thus has this place which will have place the modifications.

There are three types of Fix-Ups. The first is that which refere with the segments of code. A segment of code does not contain any intended data theoretically has to be modified, one can thus carried out directly since the disc. All the references to the segments of code of the program are thus modified to have a fixed value pointing towards the disc containing these segments. This method thus prevents from writing a code able to change itself, the code being carried out without to have been charged in memory (XIP: eXecute In Place).

At the object time of the program, MS-DOS carries out to him also this kind of modifications, in the heading of the executable one so that this one knows or is the data which it will have to handle, because executable is never charged A the same address in the RAM. These is the handling which will have to be envisaged has the advance by the Rxe converter to let MS-DOS handle only the data to be charged in memory.

The third type of modification is applied to the segment of code. Indeed, those one need to reach the data. Out, the data are not any more at the same place as the code, one being in the RAM, the other on a disc. But the data being located by chance in RAM, one cannot envisage the addresses in advance of them. The Rxe converter thus replace the calls of data of the code by an interrupt of its creation charged with reached the data necessary. It is this part which will slow down the execution of the program significantly. Moreover these calls can be ambiguous, which does not facilitate the programming.

You will find further information on the chart of datalight « RXE Theory of Operation ».

11.2 In practice

The Rxe format is appropriate completely for the applications of calculations such as they are present on the graph100, because these programs primarily consist of code.

It east does not go in the same way for the plays which them would suffer from these too frequent accesses to the data in particular for the display of the graphics stored as data in the programs. The solutions would then consist in them making static (they would be thus stored in the segments of code), but one should not either forget only the flash and the ROM one of the access times to the data higher than those of the RAM.

12° The BIOS

The BIOS is a program confined in a read-out circuit alone. On the graph100, it is T-Note BIOS v0.60 (Révision 1.10). As its name indicates it (Basic Input/Output System), it is thanks to him that the programs will be able to communicate with the peripherals of the machine. But it is also him which is to charge with starting and to check the correct operation of the machine. It installs with starting a whole list of interrupts which will make it possible to use the peripherals the such keyboard. These interrupts all are called in the same way some is the PC, which increases their compatibility the ones with the others.

But the BIOS, so that it is accessible has are proper site report, in general in the segment F000h. Sur Graph100, it starts with the address F000h:F000h.

With the lighting of the computer, the BIOS is in this segment. On all the PC, including the graph100, the first instruction carried out by the system is located has the address F000h:FFF0h. Cette left the memory BIOS contains an unconditional jump towards another zone which goes it tested the correct operation of the system. This zone is called Power one Self Test, POST. After its execution, if all it passed well, the BIOS then passes the hand to the operating system, ROM-dos.

13° Interrupts

The following chapter proposes explanations to you on the interrupts.

An interrupt is a function of the machine, resident in memory, which when one it call carries out an operation has which it is dedicated. The interrupts can be of two types, softwares and hardware. The first are not carried out that by a call has they by program. The interrupts hardware when with them are generated by the peripherals which requires them even their executions when they need some, like the cooling of the screen with the interrupt 53h for example. So that the processor can carry out the interrupts directly grace has their number, it was decided to place a table of vectors of interrupts at the whole beginning of the RAM has the address 0000h: 0000 to the address 0000h: 03FFh which contains the address memory of all the interrupts in the order of their numbers. Thus the interrupt 0h has its address in 0000h: 0000h until 000h: 0003h. Par.la.suite, we will speak about the actual address, we will simply say that the address of interrupt 0 lies between 000h and 003h, that of interrupt 1 between 004h and 007h...

The address registered at this place is written the offset in first and the segment as a second.

The processor then will receive the number of function, will read in this table its address and will carry out the instructions being there. This table is not which can protected, and one can with leisure modify it to add interrupts or to modify some. Because all the interrupts are not used.

Here a list of the interrupts present on the graph100+.

N°	Adress	Description
0	000 – 003	CPU: Division by Zero
9	024 – 027	IRQ1: Keyboard
10	040 – 043	BIOS: Video Functions
11	044 – 047	BIOS: Configuration Determination
12	048 – 04B	BIOS: RAM length Determination
13	04C – 04F	
14	054 – 057	
16	058 – 05B	BIOS: Keyboard Interrogation
19	064 – 067	BIOS : Hot start (ALT+CTRL+DEL)
1A	068 – 06B	BIOS : Drive dates and hour
1B	06C – 06F	Touch Break Actuated
1C	070 – 073	Control the timer
20	080 – 083	DOS: Termination of the program
21	084 – 087	DOS: Function of DOS
22	088 – 08B	Address fine routine DOS of the program
23	08C – 08F	Address routine CTRL-BREAK of DOS
24	090 – 093	Address routine of error of DOS
25	094 – 097	DOS : Drive disquette/disque hard
26	098 – 09B	DOS : Writing on disquette/disque hard
27	09C – 09F	DOS : Fine program, to leave resident
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	EMS Memory
4A	128 – 12B	RTC
4B	12C – 12F	EMS Memory
4C	130 – 133	
4D	134 – 137	
4E	138 – 13B	
4F	13C – 13F	
51	144 – 147	

53	14C – 14F	Screen: Update of the display
58	160 – 163	
5C	170 – 173	General (Screen, flash ...)
5E	178 – 17B	
5F	17C – 17F	
7C	1F0 – 1F3	Control screen
7D	1F4 – 1F7	

It will be noticed that interrupt 2 controlling NMI does not seem to be present on the graph100+, although it is on the other models (It must be has another place). It is valid for other interrupts, but there does not exist at the present time of list counting them.

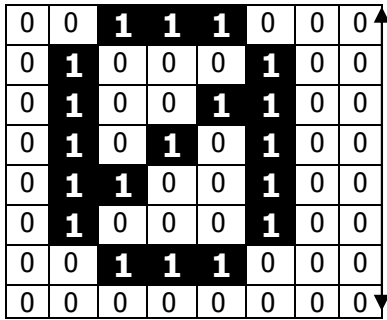
Moreover, some test make it possible to note that interrupts such 16h exist but one modified so that certain function was added or withdrawn, compared to a normal interrupt 16h.

A very useful significant point, is that the call of one of the functions not listed will not involve a bug, but simply a direct return towards the appealing program. This can be used if one wishes to redirect an interrupt to make temporarily it inactive, because it is enough to make pointed its address towards that of one of the interrupt not listed here.

14° Character font

A character font is a grouping of character having a particular style, these characters having always a precise command and not being able to be more than 256. The graph100, by defect, has 3 of them. First is our traditional alphabet, it also contains special characters. The second gathers characters accentuated or Greeks. The third is a Japanese alphabet, the kata kana.

The characters, in memory, are represented on 8 bytes, with 1bit = 1 pixel, which gives to each characters 64 pixels:



8 lines, 1 line = 1 byte

8 bits = 1 bytes.

Here, it is character 48 of the traditional alphabet which is représenté. It is noticed that on on straight line, two columns of bits are not used. This comes owing to the fact that it is much simpler to handle 1 complete byte than 6 bits only. With the screen, each characters measures 6 pixels broad. When one creates a font for CASIO, it is thus necessary left these two columns empties, without occupying itself some.

For acceder with the other character fonts, or has a font personnel, it is necessary to act on the word located in 0040h: 00C4h, or to write the character 0xF6 or 0xF7 has the screen. Indeed, the fact of wanting to display these value will entrainera the modification of the above mentioned word, without displayed the characters corresponding to the ASCII code 0xF6 and 0xF7 (which in any event is null).

Here the values which this word must contain to modify the alphabet used:

Alphabet Type	Value
Normal Alphabet	0xCE50
1 st special alphabet (0xF6)	0xCED0
2 nd special alphabet (0xF7)	0xCF50

Caution: each time that a character is displayed with the screen this value is given to 0xCE50. These values are the addresses of segment containing the character font with used. To use its own font, it is enough to charge its model in memory, at the beginning of a segment, and to write the address of this segment in this word with each time one wishes to use it.

15° Le système d'exploitation ROM-Dos

The graph100, like any micro-computer, has an operating system, ROM-DOS. It is compatible MS-DOS and was conceived by Datalight. It takes care with starting of the computer, just after the BIOS which passes the hand to him.

Just like MS-DOS, it is able to launch executable files of types com or exe. However, because of the sharp memory size, it does not allow the use program to compile in another mode only tiny, which amounts carrying out only programs with the com format, the exe having then the same structure.

It is composed of a part resident, in particular of its play of interrupt of 20h has 27h, as well as a copy of command.com. This one is present in its "mini" version, Datalight having creates it for systems embarked do not require that the user reaches has the line of command. This is why it includes/understands only one command set limited. It is not possible besides to simply use this file by calling upon him since a browser of file, because the engineers of CASIO do not have to consider it useful to replace awaits it key "entered" by the key "exe", command.com not being supposed being carried out. This phenomenon is all the more marked as since the graph100+, it is not possible any more to reach to drive a: since ROM-back, drive who contains command.com amongst other things, because this drive protects himself in reading immediately after the first program of the session in progress is been to finish.

Moreover, because owing to the fact that the user should not provide to reach ROM-back of traditional use, the access to the parameters of the completely blocked system, which makes of it a system practically not bringing anything to the programmer, its interrupts being very often available directly to the level of BIOS.

15.1 The Psp

Before launching a program, ROM-back, just like MSDOS creates a zone of heading called Program Segment Prefix (PSP). This zone will contain information concerning the access path to the file in progress, the backup of certain vectors of interrupts, and a multitude of more or less useful information for the programmer. This zone is right before the copy of the program in memory, and measures 256 bytes. It does not have the same format under ROM-back as under MSDOS. Here is the structure such as it is known at the present time.

PSP structure		
Address	Contents	Length
00h	Call of the interrupt 20h	1 word
02h	Bp Address	1 word
0Ah	Copy vector of interrupt 22h	2 words
0Ch	Copy vector of interrupt 23h	2 words
2Ch	segment of the block of environment	1 word
50h	Call of the interrupt 21h	1 word
80h	Length of the first argument	1 byte
81h	1 st argument	See above
C6h	Path of the program	

These data although fragmentary will help certainly the programmers who wish to reach the parameters of the line of assembler instruction. It is necessary to know that the charger of executable makes point the segment Cs, ds and are towards the beginning of the PSP. En.effet, the first instructions of the program in court must backed up the value of one of these segment in order to be able reached the PSP later on. It is then simple to use information that this one contains.

Appendix A – CASIO ASCII codes

Dec.	Hexa	Ascii 1	Ascii 0xF6	Ascii 0xF7	Dec.	Hexa	Ascii 1	Ascii 0xF6	Ascii 0xF7
0	00				49	31	!	!	!
1	01				50	32	"	"	"
2	02				51	33	#	#	#
3	03				52	34	\$	\$	\$
4	04				53	35	%	%	%
5	05				54	36	&	&	&
6	06				55	37	'	'	'
7	07				56	38	(((
8	08				57	39)))
9	09				58	3A	:	:	:
10	0A				59	3B	;	;	;
11	0B				60	3C	<	<	<
12	0C				61	3D	=	=	=
13	0D				62	3E	>	>	>
14	0E				63	3F	?	?	?
15	0F				64	40	@	@	@
16	10				65	41	A	A	A
17	11				66	42	B	B	B
18	12				67	43	C	C	C
19	13				68	44	D	D	D
20	14				69	45	E	E	E
21	15				70	46	F	F	F
22	16				71	47	G	G	G
23	17				72	48	H	H	H
24	18				73	49	I	I	I
25	19				74	4A	J	J	J
26	1A				75	4B	K	K	K
27	1B				76	4C	L	L	L
28	1C				77	4D	M	M	M
29	1D				78	4E	N	N	N
30	1E				79	4F	O	O	O
31	1F				80	50	P	P	P
32	20				81	51	Q	Q	Q
33	21	!	!	!	82	52	R	R	R
34	22	"	"	"	83	53	S	S	S
35	23	#	#	#	84	54	T	T	T
36	24	\$	\$	\$	85	55	U	U	U
37	25	%	%	%	86	56	V	V	V
38	26	&	&	&	87	57	W	W	W
39	27	'	'	'	88	58	X	X	X
40	28	(((89	59	Y	Y	Y
41	29)))	90	5A	Z	Z	Z
42	2A	*	*	*	91	5B	[[[
43	2B	+	+	+	92	5C	\	\	\
44	2C	,	,	,	93	5D]]]
45	2D	-	-	-	94	5E	^	^	^
46	2E	.	.	.	95	5F	_	_	_
47	2F	/	/	/	96	60	`	`	`
48	30	0	0	0	97	61	a	a	a

98	62	b	00000000	U
99	63	c	00000000	U
100	64	d	00000000	U
101	65	e	00000000	U
102	66	f	00000000	U
103	67	g	00000000	U
104	68	h	00000000	U
105	69	i	00000000	U
106	6A	j	00000000	U
107	6B	k	00000000	U
108	6C	l	00000000	U
109	6D	m	00000000	U
110	6E	n	00000000	U
111	6F	o	00000000	U
112	70	p	00000000	U
113	71	q	00000000	U
114	72	r	00000000	U
115	73	s	00000000	U
116	74	t	00000000	U
117	75	u	00000000	U
118	76	v	00000000	U
119	77	w	00000000	U
120	78	x	00000000	U
121	79	y	00000000	U
122	7A	z	00000000	U
123	7B	{	00000000	U
124	7C		00000000	U
125	7D	}	00000000	U
126	7E	~	00000000	U
127	7F		00000000	U
128	80	A	00000000	U
129	81	B	00000000	U
130	82	C	00000000	U
131	83	D	00000000	U
132	84	E	00000000	U
133	85	F	00000000	U
134	86	P	00000000	U
135	87		00000000	U
136	88	Q	00000000	U
137	89	R	00000000	U
138	8A	S	00000000	U
139	8B	T	00000000	U
140	8C	U	00000000	U
141	8D	V	00000000	U
142	8E	W	00000000	U
143	8F	X	00000000	U
144	90	Y	00000000	U
145	91	Z	00000000	U
146	92	1	00000000	U
147	93	2	00000000	U
148	94	3	00000000	U
149	95	4	00000000	U

150	96	6	X	カ
151	97	7	X	キ
152	98	8	X	ク
153	99	9	X	ケ
154	9A	0	X	コ
155	9B	1	X	サ
156	9C	2	X	シ
157	9D	3	X	ス
158	9E	4	X	セ
159	9F	5	X	ソ
160	A0	6	X	タ
161	A1	7	X	チ
162	A2	8	X	リ
163	A3	9	X	ト
164	A4	0	X	ナ
165	A5	1	X	ニ
166	A6	2	X	ノ
167	A7	3	X	ハ
168	A8	4	X	ヒ
169	A9	5	X	フ
170	AA	6	X	ユ
171	AB	7	X	リ
172	AC	8	X	ロ
173	AD	9	X	ハ
174	AE	0	X	ニ
175	AF	1	X	ノ
176	B0	2	X	ハ
177	B1	3	X	ヒ
178	B2	4	X	フ
179	B3	5	X	ユ
180	B4	6	X	リ
181	B5	7	X	ロ
182	B6	8	X	ハ
183	B7	9	X	ニ
184	B8	0	X	ノ
185	B9	1	X	ハ
186	BA	2	X	ヒ
187	BB	3	X	フ
188	BC	4	X	ユ
189	BD	5	X	リ
190	BE	6	X	ロ
191	BF	7	X	ハ
192	C0	8	X	ニ
193	C1	9	X	ノ
194	C2	0	X	ハ
195	C3	1	X	ヒ
196	C4	2	X	フ
197	C5	3	X	ユ
198	C6	4	X	リ
199	C7	5	X	ロ
200	C8	6	X	ハ
201	C9	7	X	ニ

202	CA	√	√
203	CB	√	√
204	CC	√	√
205	CD	√	√
206	CE	√	√
207	CF	√	√
208	DO	√	√
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	√	√

Appendix B - Interrupts

Information on the interrupts presented here one studied in detail, but can however be incomplete or erroneous. The remark "complete" means that all the functions of the interrupts its presented here. You will find other studies of interrupts all along this document.

Interrupt 10h	Graphics	BIOS - Complete
Function 0		Ah = 0

Choice of the video mode

Al = video mode has to set up. Only the first 3 Al bits are kept. This value is then written in 0040h:0049h
Recadre the display in 1A20h and puts the variables of the up to date BIOS.

Function 2		Ah = 2
------------	--	--------

Position the cursor

Bh = number of screen page

Dh = line of the screen

DI = column of the screen

Function 5		Ah = 5
------------	--	--------

Change page/re-initializes the display.

bh = number of page

Modify the address of the video buffer by reading the value contained in 0040h : 00E0h

Function 6		Ah = 6
------------	--	--------

Make ravel the screen upwards.

Al : line of shift numbers

Ch : line of the screen of the left higher corner of the window

Cl : column of the screen of the left higher corner of the window

Dh : line of the screen of the corner higher right of the window

DI : column of the screen of the corner higher right of the window

Function 7		Ah = 7
------------	--	--------

Nothing

Function 9		Ah = 9
------------	--	--------

To write a character with the screen

Al = code ASCII character. If Al = FF, no character will be written. If Al = 0xF6, the next character written will be read in the first special alphabet, if Al=0xF7, the next character written will be read in the second special alphabet.

Bh = number of page to be displayed. The position of the cursor will be selected in function.

Cx = repetition of display of the character numbers. (1: 1 time, 2: 2 times...).

Bl = attribute of the character. If its 6th bit is worth 0, the character will be written in black on white, if not white on black.

Function 0Ch		Ah = 0x0C
--------------	--	-----------

Draw a point with the screen.

Bh = screen page

Dx = line of the screen

Cx = column of the screen

Al = color. If the 7eme bit is worth 1, the pixel will be white, if not it is black.

Function 0Dh		Ah = 0x0D
--------------	--	-----------

Reads a point with the screen.

Bh = screen page

Dx = line of the screen

Cx = column of the screen

Turn over in Al the color of the targeted pixel.

Function 0Eh

Ah = 0x0E

Display a character.

Differences with not yet given function 9.

Interrupt 11h

Complete

Turn over in ax the word in 0040h: 0010h (config of the caltos)

Interrupt 12h

Complete

Returns in ax the word in 0040h: 0013h (size of the RAM)

Interrupt 13h

Complete

Puts the first bit of the byte located in sp+6 at 1.

Interrupt 14h

Function 88h

Ah=88h

Turn over ax =0

Function C0h

Ah=C0h

Modify the pile, returns bx = 264Ah, es = cs (C000)

Function autre

Ah= autre

Modify the pile.

Interrupt 16h

Keyboard

BIOS - Complete

Traditional interrupt 16, but only functions 0, 1, 5, 10h, 11h, 20h and 30h are available.

Function 20h

Ah= 20h

Puts the words of the address 0040h:001Bh and 0040h:001Dh at 1Fh.

Function 30h

Ah= 30h

Collect a value directly without awaiting interrupt 9.

Interrupt 41h

Zone 1A20h+1C0h

Function 7Dh..FFh

Ah >= 7Dh

Special Function 3Bh..FFh

Al >= 3Bh

ah = 1BE0h :0000 , al = 1BE0h :0000, bh = 1BE0h :0000, bl=1BE0h :0000, bl = FFh

Interrupt 4Bh

EMS Memory

Complete

Function 0

Ah = 0

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

Function 1

Ah = 1

Make the reverse then leaves the program with the interrupt 21h, ah = 4Ch.

Function 2	Ah = 2
------------	--------

Unspecified action, then leaves the program with the interrupt 21h, ah = 4Ch.

Function 3	Ah = 3
------------	--------

The zone memory BIOS 6Bh puts has the value contained in bl has the call.

Interrupt 4Eh		Complete
Function 0	Ah=0	

Reads the value of ports 8 and 9, saves it in bx, then puts 0 in these ports

Function 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

Different function	Ah = other
--------------------	------------

The value contained in bx is written in ports 8 and 9

Appendix C – The BIOS variables

The BIOS, like any program, needs variables. But since it cannot store them directly on its ROM, it uses RAM, starting from the segment 40h, on a little more than 256 bytes. Here a description of these data which can be revealed very useful in certain cases. This list is not exhaustive and will progressively be supplemented with the development of this tutorial. Some can be erroneous, but information on this type of BIOS is untraceable, it is thus necessary to proceed by deduction.

00h Addresses of the series interfaces	4 words
---	---------

The BIOS stores there the addresses of the 4 ports series of the PC. It seems to also find them on graph100. These values without doubt are used for other ends.

08h Addresses of the parallel interfaces	4 words
---	---------

The BIOS stores there the addresses of the 4 parallel ports of the PC. There too it seems to find them on graph100.

13h Memory	1 word
---------------	--------

This cell contains the size of the RAM

1Ch Keyboard	1 byte
-----------------	--------

The BIOS stores here the last key pressed with the keyboard by the user.

49h Display: current video mode	1 byte
------------------------------------	--------

This byte contains the value of the video mode in progress.

4Ah Display: a number of current columns	1 word
---	--------

This word contains the number of columns by lines of the mode running (default: 21)

4Ch Display: screen page size	1 word
----------------------------------	--------

This word contains the size of a page of the screen in the current mode. (default : 1024).

4Eh Display: page	1 word
----------------------	--------

This word contains the size of a screenful multiplied by the number of the current page.

50h Position of the cursor in the eight pages of screen	8 words
--	---------

The BIOS stores here the co-ordinates of the cursor in each page of screen. But until proof of the opposite, we cannot not change this page and summons constrained remained with the first, only the first word is thus useful. Changed the value since this address directly assigns the position of the cursor to the screen.

62h	1 byte
-----	--------

Video page	
------------	--

This byte contains the number of the current page.

6Ch Meter system	5 bytes
---------------------	---------

Meter system, approximately incremented 2,7 times per seconds. It is not fed when the computer is extinct, and moves at the rate/rhythm of interrupt 8.

85h Size of a character in byte	1 bytes
------------------------------------	---------

This byte contains the size of a character text in memory. By defect, it is worth 8 bytes.

B0h Relating to the zones memories has mapped	7 bytes
--	---------

The registered bytes represent the current values registered in the port 54h has Äh.

For an optimal compatibility with the system, it is necessary to put has days these values manually if one changes the provision of the zones memories (see the memory), has the manner of the system. No conflict however seems to appear with the current system if this stage is omitted.

C4h Character font	1 word
-----------------------	--------

This word makes it possible to select the character font in the course of use. It contains the address of segment of this font.

E0h Video buffer adress	1 words
----------------------------	---------

At this zone the address of segment of the video buffer is stored. Changed this value does not affect the address. Two solutions are offered then to you:

- To handle ports 5 to 7. This method is not to advise which has those which seeks a maximum optimization, the second method being it much simpler.
- Called the interrupt 10h, with ah = 5. The value entered the word will then be written in the ports, but you must choose a value whose four bits are null, i.e. who begins with a 0h such 1A20h.

E5h Menu construction	1 byte
--------------------------	--------

Contains a value to write in the sequence screen.

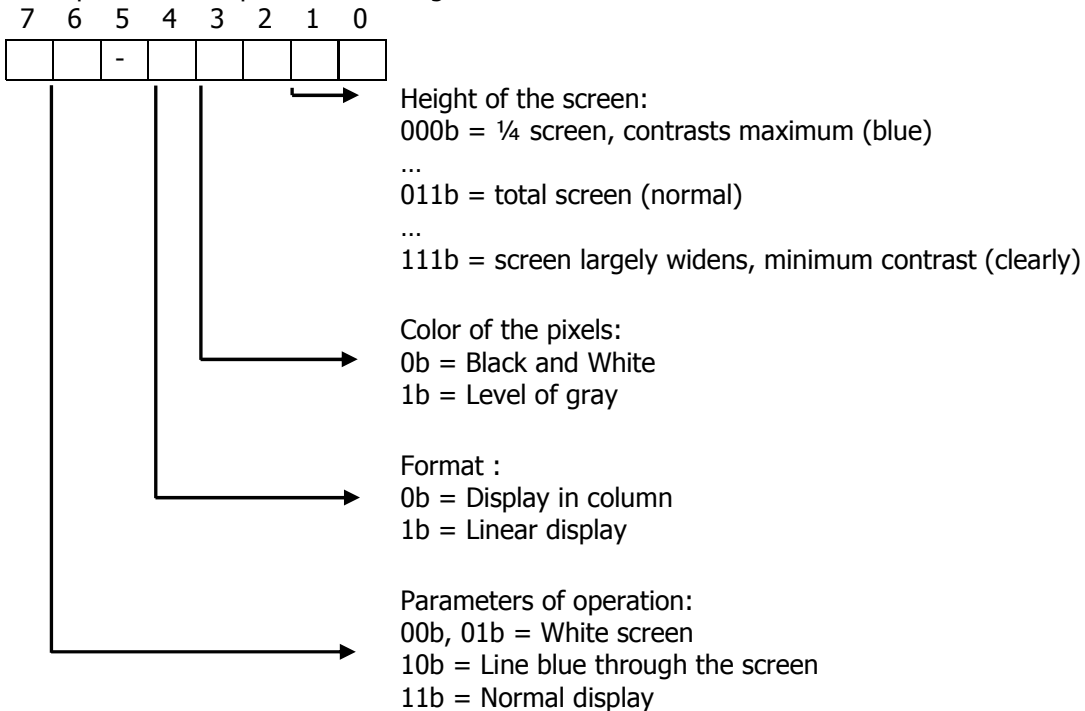
Appendix D – Communication Ports

The ports are the paths by which the processor communicates with the peripherals. For y reached, you must used the functions `inportb()` and `outportb()` out of C, and `in` and `out` in Assembleur, respectively there to read and y to write data. Some of these ports function only in one direction, either in reading or in writing. That means that action not permitted by the port will not be interpreted, a reading on a port in writing alone will return an unspecified value. The noted ports "given Non" active and are used, but their actions yet are not known.

Port 00h	Undetermined	Write
-----------------	---------------------	--------------

Port 02h	LCD	Write
-----------------	------------	--------------

This port makes it possible to change the video view of the screen. The written values have this format:



Port 03h	LCD	Write
-----------------	------------	--------------

Modify the length of display of the screen. 7 is the value written by defect.

Port 04h	LCD	Write
-----------------	------------	--------------

Modify the scanning rate of the screen.

- 1: Stop sweeping
- 2: The fastest sweeping
- 4: Standard value

The higher values will slow down it without stopping it.

Port 05h à 07h	LCD	Write
-----------------------	------------	--------------

Modify starting l` adress of the video buffer. The format of the addresses being always on 20bit, we must use this format to write in these ports. Let us take the default value, 1A20h:0000h. This address, in its condensed form is written $1A20h * 16 + 0000h = 1A200h$.

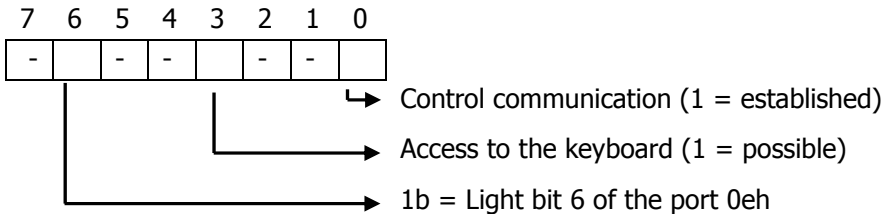
It is the latter type of value which it is necessary for us to write in these ports, ports being put end to end to provide to contain this address. Not being able to write it directly because of his size, one can write it as follows:

Port 7h 3 2 1 0	Port 6h 7 6 5 4 3 2 1 0	Port 5h 7 6 5 4 3 2 1 0
Segment ranging between 0h and FFFFh, therefore on 16bits.		Offset ranging between 0h and Fh

Ports 08h et 09h	Undetermined	Write
-------------------------	---------------------	--------------

Port 0Ah	Communication Port	Read / Write
-----------------	---------------------------	---------------------

Port state of communication (see section communication ports)

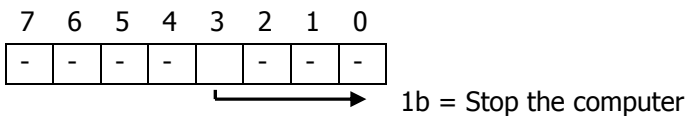


Port 0Bh	Communication Port	Read / Write
-----------------	---------------------------	---------------------

Carry out the reception or the sending of data (see section communication ports)

Port 0Ch	General	Read / Write
-----------------	----------------	---------------------

Stop the computer, as a pressure on the Off key.



Port 0Dh	Undetermined	Write
-----------------	---------------------	--------------

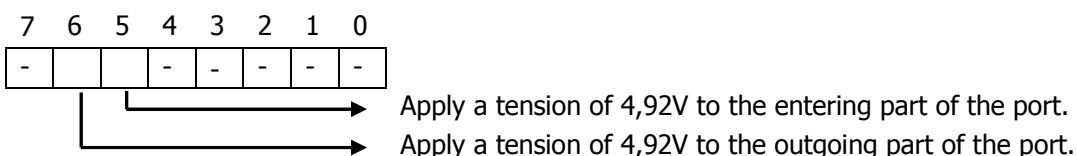
Port 0Eh	Communication Port	Read / Write
-----------------	---------------------------	---------------------

Control the communication series of the graph100

Port 10h	Undetermined	Read / Write
-----------------	---------------------	---------------------

Port 11h	Communication Port	Read / Write
-----------------	---------------------------	---------------------

Installation of the bus of transfer of data (see section communication port).



Port 12h	Undetermined	Write / Write
-----------------	---------------------	----------------------

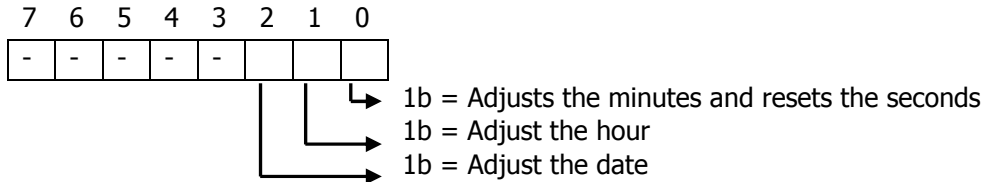
Ports 13h et 14h	Keyboard	Read / Write
-------------------------	-----------------	---------------------

These ports make it possible to communicate with the keyboard. Before reading in the port 13h, you must write in the ports 13h and 14h the touchline which you wish tested, then to read its value in the port 13h. Pour more information, defer to the chapter devoted to the keyboard.

Port 16h	Undetermined	Read / Write
Ports 17h et 18h	Undetermined	Read / Write
Port 1Ch	Undetermined	Read / Write
Port 1Dh	Real Time Clock (RTC)	Read / Write

In reading, this port returns the second current ones.

In writing, you must first of all put the good value in the ports 1Eh at 22h, then to write as follows:



Port 1Eh	Real Time Clock (RTC)	Read / Write
-----------------	------------------------------	---------------------

In reading, this port returns the current minutes.

To modify them, you must write in this port the value to wish, then to adjust the port 1Dh

Port 1Fh	Real Time Clock (RTC)	Read / Write
-----------------	------------------------------	---------------------

In reading, this port returns the current hours.

To modify them, you must write in this port the value to wish, then to adjust the port 1Dh

Ports 20h à 22h	Real Time Clock (RTC)	Read / Write
------------------------	------------------------------	---------------------

These ports return the number days to run out since the 1er January 1970 (arbitrary date on which bases certain functions of Turbo C concerning the date). It should be noted that the port 22h do not have which are first functional bit.

Port 24h	Undetermined	Write
-----------------	---------------------	--------------

Port 25h	Undetermined	Write
-----------------	---------------------	--------------

Port 27h	Character font	Read / Write
-----------------	-----------------------	---------------------

To write in this port changes the character font, passing from our letters has pictograms of which I do not know the origin. The values to be written there are 1xh and 3xh (X = any figure). The values 5xh and 7xh blocks the computer. All the other values make the font normal. The real goal of this port is not yet known.

Port 28h	Undetermined	Write
-----------------	---------------------	--------------

Port 2Ch	Undetermined	Write
-----------------	---------------------	--------------

Ports 2Eh et 2Fh	Undetermined	Write
-------------------------	---------------------	--------------

Port 31h	Undetermined	Write
-----------------	---------------------	--------------

Port 32h	Undetermined	Write
-----------------	---------------------	--------------

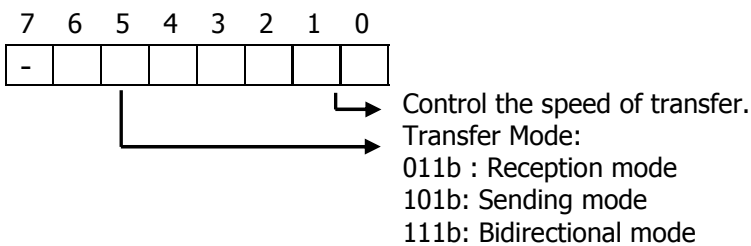
Port 33h	Undetermined	Write
-----------------	---------------------	--------------

Port 34h	CPU internal speed	Write
-----------------	---------------------------	--------------

This makes it possible to slow down the speed of operation of the pocket calculator. The default value is 0, plus the value is raised, the slowed down computer. The maximum value is 0Fh. The combination of this value with value 2 in the port 4h blocks the computer, this one not being able more assured cooling the screen.

Ports 35h à 39h	Undetermined	Write
Ports 3Ah et 3Bh	Undetermined	Read / Write
Ports 3Ch et 3Dh	Undetermined	Read
Port 44h	Communication Port	Read / Write

To write on this port modifies the parameters of communications. Y lira returns if it is available a value sent.

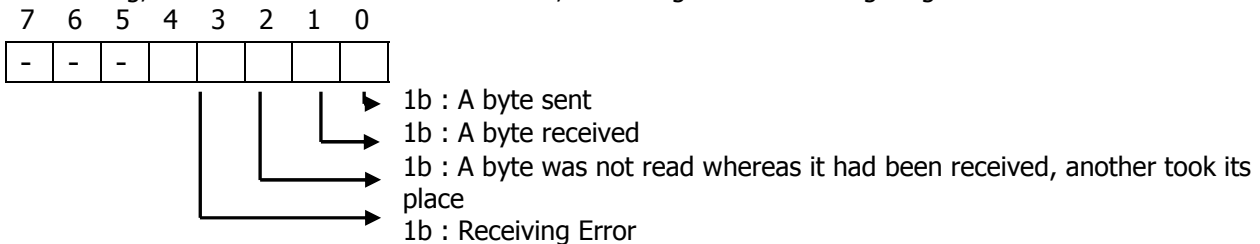


Port 45h	Communication Port	Read / Write
-----------------	---------------------------	---------------------

Transfer type.

In writing, this port authorizes the communication.

In reading, it allows checked state of this one, according to the following diagram:



Port 46h	Communication Port	Write
-----------------	---------------------------	--------------

Sert de buffer pour l'envoi de données.

Port 47h	Communication Port	Write
-----------------	---------------------------	--------------

Transfer speed (See section communication port).

Port 4Eh	Undetermined	Write
-----------------	---------------------	--------------

Port 50h	Undetermined	Write
-----------------	---------------------	--------------

Port 52h	Undetermined	Read
-----------------	---------------------	-------------

Port 54h to 5Ah	EMS memory	Write
------------------------	-------------------	--------------

These ports are useful has to put in memory a definite part of the ROM and flash. All these ports one each one this capacity, but each one of them goes mapper the memory to a quite precise place, according to their numbers. Moreover, the values are written in these ports according to the block of data of the ROM or of the flash mapper has. The tables according to indicate the values to be written and the zone or mappent the ports.

Value to be written in the ports	Type	Corresponding zone of Flash/ROM	Information
A0h	Flash	0000h : 0000h - 1000h : FFFFh	Zone system Drive A :
A1h	Flash	2000h : 0000h - 3000h : FFFFh	Language in the course of use
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 basic system Menu construction
C1h	ROM	02000h : 0000h - 03000h : FFFFh	Menu construction Back up A:\
C2h	ROM	04000h : 0000h - 05000h : FFFFh	Languages in the system
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	K:\
DEh	ROM	3C000h : 0000h - 3D000h : FFFFh	
DFh	ROM	3E000h : 0000h - 3F000h : FFFFh	

Port 6Ch	Undetermined	Write
-----------------	---------------------	--------------

Ports 6Eh et 6Fh	Undetermined	Write
-------------------------	---------------------	--------------