GDOS

188

grifo® Disk Operating System
188 Family

USER MANUAL

grifo®
ITALIAN TECHNOLOGY
Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it
http://www.grifo.it http://www.grifo.com
Tel. +39 051 892.052 (a.r.) FAX: +39 051 893.661

GDOS 188 Edizione 5.10 *Rel. 11 October 1999*

GDOS

188

grifo® Disk Operating System
188 Family

USER MANUAL

GDOS 188 is a complete software package, developed by **grifo**®, to run on the **80C188** microprocessor-based boards and derivates, belonging to the wide **Intel 86** family, from the **ABACO**® industrial standard.
This manual constitutes a complete user guide to the correct employment of **GDOS 188** (**Grifo**® **D**isk **O**perating **S**ystem **188** family).
GDOS 188 is made of many subprograms, both independent and not, which make up a comfortable develpment evironment. In detail, in addition to the **Operating System**, the package includes **Monitor/Debugger 188**, the compiler **PASCAL 188**, capable to generate ROMable code, and the utility **TOOLS 188**, to easily manage some on-board resources.

grifo®
ITALIAN TECHNOLOGY
Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it
http://www.grifo.it http://www.grifo.com
Tel. +39 051 892.052 (a.r.) FAX: +39 051 893.661

GDOS 188 Edizione 5.10 *Rel. 11 October 1999*

DOCUMENTATION COPYRIGHT BY **grifo**®, ALL RIGHTS RESERVED

IMPORTANT

SYMBOLS DESCRIPTION

In the manual could appear the following symbols:

⚠ Attention: Generic danger

⚡ Attention: High voltage

Trade Marks

GENERAL INDEX

INDICE DELLE FIGURE

INTRODUCTION

The use of these devices has turned - IN EXCLUSIVE WAY - to specialized personnel.

The purpose of this handbook is to give the necessary information to the cognizant and sure use of the products. They are the result of a continual and systematic elaboration of data and technical tests saved and validated from the manufacturer, related to the inside modes of certainty and quality of the information.

The reported data are destined- IN EXCLUSIVE WAY- to specialized users, that can interact with the devices in safety conditions for the persons, for the machine and for the enviroment, impersonating an elementary diagnostic of breakdowns and of malfunction conditions by performing simple functional verify operations , in the height respect of the actual safety and health norms.

The informations for the installation, the assemblage, the dismantlement, the handling, the adjustment, the reparation and the contingent accessories, devices etc. installation are destined - and then executable - always and in exclusive way from specialized warned and educated personnel, or directly from the TECHNICAL AUTHORIZED ASSISTANCE, in the height respect of the manufacturer recommendations and the actual safety and health norms.

The devices can't be used outside a box. The user must always insert the cards in a container that rispect the actual safety normative. The protection of this container is not threshold to the only atmospheric agents, but specially to mechanic, electric, magnetic, etc. ones.

To be on good terms with the products, is necessary guarantee legibility and conservation of the manual, also for future references. In case of deterioration or more easily for technical updates, consult the AUTHORIZED TECHNICAL ASSISTANCE directly.

To prevent problems during card utilization, it is a good practice to read carefully all the informations of this manual. After this reading, the user can use the general index and the alphabetical index, respectly at the begining and at the end of the manual, to find information in a faster and more easy way.

VERSIONS

The present handbook is reported to the **GDOS 188** version **3.2** and later. The validity of the bring informations is subordinate to the number of the firmware version. The user must always verify the correct correspondence among the two denotations. The version number is printed on the label attached on the EPROM or FLASH EPROM included in the **GDOS 188** package, or it can be obtained executing **GDOS 188** itself on the target board.
In addition, this manual reports information about **GET188** version **3.2** and later; this version number is printed in the Information window, shown at the program start and recallable by the proper command.

GENERAL FEATURES

GDOS 188 (**grifo**® **D**isk **O**perating **S**ystem **188** family) is a complete software package, developed by **grifo**®, to run on the **80C188** microprocessor-based boards and derivates, belonging to the wide **Intel 86** family, from the **ABACO**® industrial standard.
This package allows the User to take advantage of the on-board resources from a well-developed working environment, avoiding so the need of a deep knowledge about the hardware details of the board being used.
GDOS 188 is made of many subprograms, both independent and not, which make up a comfortable develpment evironment. In detail, in addition to the **Operating System**, the package includes **Monitor/Debugger 188**, the compiler **PASCAL 188**, capable to generate ROMable code, and the utility **TOOLS 188**, to easily manage some on-board resources, such as RAM Disk, Real Time Clock, etc.
This manual gives a complete description of the whole package, except for the **PASCAL 188**, following a suddivision based on the meaning of the single components.
GDOS 188 allows to take advantage of the **PASCAL 188** features, provided with the package, that make the User able to access many resources, normally not available with other programming environments. It is possible to develop, to test and to install any kind of program that the User may want to make, obtaining quickly, very comfortably and very efficently specific applications.
GDOS 188 is a simple software package that can be compared to an elementary, but efficent, industrial specific operating system. In fact, like all operating systems, also **GDOS 188** features an high level interface between the User and the hardware to employ. So this package gives the possibility to take advantage of all the hardware resources of the system, using directly high level instructions and commands, without the need to develop specific management firmware.The remarkable power and flexibility of **GDOS 188** allow also to profit by an external computer, to greatly simplify the application's development and set up.

As the software packages is under continuous development, please take care of the presence of an eventual file called "READ.ME" in the work disk. This file reports additions, modifications and improvements applied to the package and not yet reported in the manual; if the file is present it must be read, printed and attached to this manual.

ESSENTIAL HARDWARE AND SOFTWARE

Here follows a short description of the essential hardware and software needed to topertate with **GDOS 188**.

80C188 BASED CONTROL BOARD

Is is a card belonging to the **grifo**® industrial standard, based on the microprocessor **80C188** or derivates, like: **GPC® 188F**, **GPC® 188D**, **GPC® 884**, etc.

NOTE
This manual uses the definition **target board** to refer to one of above reported **grifo**® hardware structures.

The target board, indipendently from the tasks that must perform, must be provided with:

EPROM or FLASH EPROM sized at least 128 KByte with **GDOS 188** packageinstalled.
At least 32KByte RAM.
One RS 232 asynchronous serial line.

The above specifications are to be inntended as minimun requirements to work, in fact the system can be expanded increasing its features. **GDOS 188** always manages an high level interface to the following sections of the target board:

1) *One consolle serial line*:	Indispensable whenever **GDOS 188** is used also for developing and debugging the application.
2) *One auxiliary serial line*:	The high level reception and transmission of characters is managed employing a User protocol.
3) *16 TTL I/O lines*:	High level managed to drive a parallel printer. (For example interfaced to the **IAC 01** or **DEB 01**).
4) *Unused RAM*:	Managed as RAM DISK by high level procedures, data is written and read as files.
5) *Real Time Clock*:	Time and date setting and acquisition are managed by high level procedures.

The configuration of the target board must be chosen in respect to the specific needs of the application that the User wants to develop

EPROM OR FLASH EPROM FOR GDOS 188

The code of the operating system is always delivered in EPROM or FLASH EPROM ready to be installed on the chosen target board. If the User purchases both the board and the **GDOS**, the memory device is delivered already installed on the board. The label on the memory device reports the informations about the type of target board and the version of the operating system.

PERSONAL COMPUTER

An IBM (or compatible) personal computer provided with a floppy disk drive, at least 640 KBytes of RAM, one RS 232 serial line compliant to the V24 normatives, MS-DOS operating system version 3.3 or later. The presence of an hard disk drive is suggested, but not essential, to speed up all the file access operations.
This configuration is not indispensable but is normally used because makes possible and very easied to develop and to debug any program the User wants to write.
In detail, the P.C., connected by the consol serial line to the target board, works as terminal that the User may use to interact to the **GDOS 188** running on the target board and lets the target board take advantage of its resources (mass storage devices and printer) by high level procedures.

SERIAL COMMUNICATION CABLE

If the User wants to employ the consolle serial line of the target board to develop and debug management software, the he/she must connect opportunely the consolle device.
The connection must be made according to the C.C.I.T.T. V24 normatives, remarking that consolle serial line of the target board is always serial line A and that **GDOS 188** needs only the receive and transmit (RxD and TxD) signals.
In case the consolle device is a Personal Computer, the connection to make must be DTE<->DCE, as shown in the following figures.

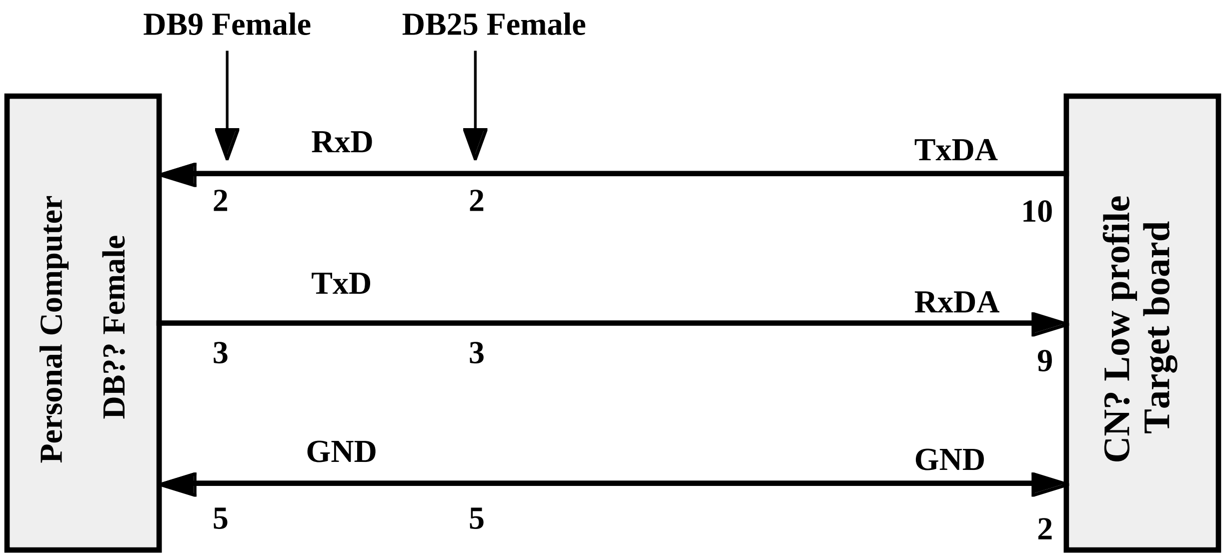

FIGURE 1: SERIAL CONNECTION P.C. - TARGET BOARD BY 16 PINS LOW PROFILE CONNECTOR

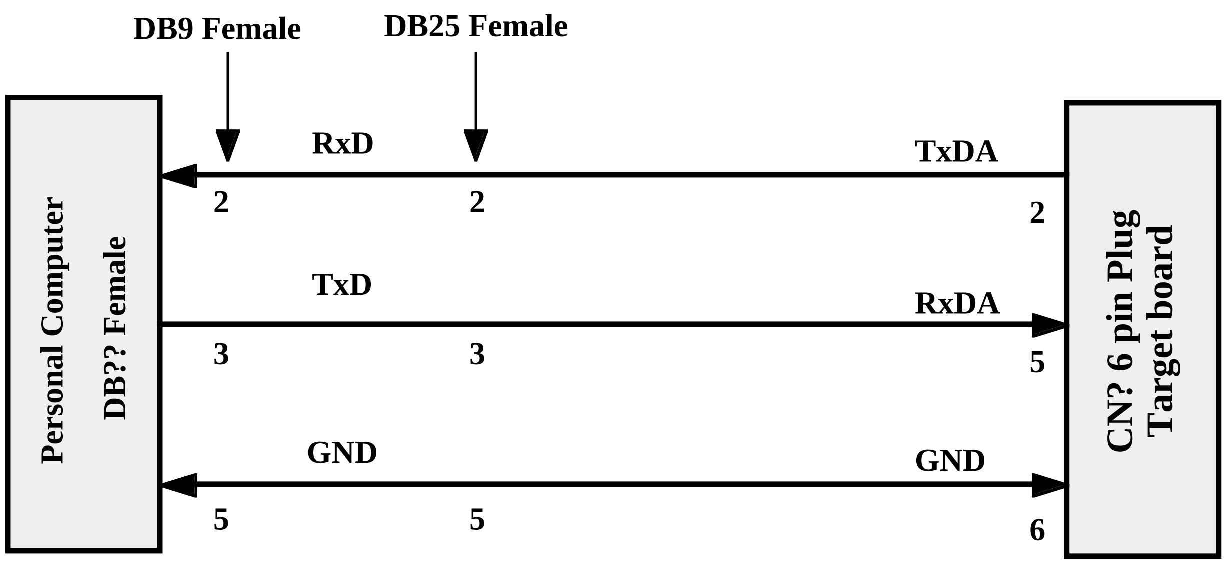

FIGURE 2: SERIAL CONNECTION P.C. - TARGET BOARD BY 16 PINS PLUG CONNECTOR

The indications CN? low profile target board and CN? 6 pins plug target board refer to serial communication standard connectors made by **grifo**®, and are described in the technical manual of the target board. The figure below shows the names of these connectors and the codes of the accessories (cables, boards, etc.) that **grifo**® can provide to easy and speed the connection phase.

TARGET BOARD	CONNECTOR	CODES OF ACCESSORIES FOR SERIAL COMMUNICATION
GPC® 188F	CN1	FLT 16+16; NCS 01; CCR 25+25 or CCR 25+9
GPC® 188D	CN1	FLT 16+16; NCS 01; CCR 25+25 or CCR 25+9
GPC® 884	CN3A	CCR.PLUG25 or CCR.PLUG9

FIGURE 3: CONNECTORS AND ACCESSORIES FOR SERIAL COMMUNICATION

SOFTWARE TO WORK

If the Personal Computer is used to develop and set up the management program, it is essential to have got a set of programs and files, stored in the **GDOS 188** distribution disk. Some of these programs are: **GET188.EXE**, intelligent terminal emulation program; **WFLASH.HEX**, the utility that writes the user program in the FLASH EPROM; the file, both binary and Intel HEX, of the operating system **GDOS 188** itself.

GDOS 188 USER MANUAL

It's the present manual and reports any technical information about **GDOS 188**.
In detail this manual reports funcions, procedures and utility programs descriptions, commands syntax, hardware connections, memory use, etc.

FIRMWARE FOR PROGRAMMING

It is made by a set of working structures that the User must employ to create and test the management program under development. In detail the firmware for programming delivered in the **GDOS 188** package, stored in the EPROM or FLASH EPROM installed on the target board, is made by:

- ***GDOS 188*** *Operating system*:	Software that provides an high level interface between the User and the hardware to use. In fact, by means of the several **GDOS 188** function calls, the resources of the target board and the P.C., used as terminal, are available.
- *Monitor Debugger 188*:	Software that allows to perform operations that involve the target board's hardware resources and to debug, at the Assembly level, the user program. Some of the possible operations are: input/output, verify or overwrite of the memory content, CPU registers management, step-by-step execution of a program, break-points setting, etc.
- *PASCAL 188 Compiler*:	Development and testing environment for any kind of program the User wants to make, working directly on the target board. The user program can be compiled to generate code executable on an EPROM.
- *TOOLS 188*:	Software to easy the management of some resources of the target board; it performs RAM-Disk formatting, file copy between P.C. and RAM-Disk, file renaming and deleting, Real Time Clock setting, etc.

DISTRIBUTION DISK

The **GDOS 188** software package, in addition to the EPROM or FLASH EPROM, is made by a disk that contains the software structure needen to work.
The content of the disk is organized in directories, whose structure recalls the software package structure; as suggested in the "HOW TO START" chapter it is convenient to copy all the files into an unique work directory.Here follows a list of directories and files stored in the distribution disk; for further informations about their meaning, please refer to the paragraphs of the following chapters:

Root:
Contains all the most frequently used programs when operating with **GDOS 188**, which constitutes the main work structure:

GET188.EXE	->	Editor and intelligent terminal emumalation program for **GDOS 188** that runs on the P.C.
G188HELP.HLP	->	Help on line file for **GET188.EXE**.

EPROM directory:
Contains the set of programs and files that must be employed to store into EPROM or FLASH EPROM the user program; according to the kind of (EPROM or FLASH EPROM) of **GDOS 188** purchased, some of the following files will be present.

MON?????.BIN	->	Binary image of the **GDOS 188** operating system madefor the **GPC® ?????** board. This file contains also the **Monitor/Debugger 188**. It must be used when creating the user EPROM, while they are not needed when using a FLASH EPROM.
GHEX2.COM	->	Utility to convert a binary file into the equivalent Intel Hex format file. It must be used if the EPROM programmer employed doesn't support the binary file format.
WFLASH.HEX	->	Program to execute on the target board, used to manage the FLASH EPROM features.
TOOLS188.COM	->	Binary image of the **TOOLS 188** utility program.
PAS188_L.BIN	->	Binary image of the **PASCAL 188** compiler, to use if needed to insert that file in EPROM or FLASH EPROM, starting from address **E000:0000 Hex**.
PAS188_H.BIN	->	Partial binary image of the **PASCAL 188** compiler, to use if needed to insert that file in FLASH EPROM, starting from the address **F000:0000 Hex**. Not utilizzable if using an EPROM.

EXAMPLES directory:
Contains a set of demostration programs that show how to use the resources of the target board; this directory may contain subdirectories containing demonstration programs for general purpose applications.

HOW TO START

This chapter describes the operations to perform for a first, elementary use of the **GDOS 188** software package. In detail, the proper sequence of operations to employ the Personal Computer as development system is reported. This chapter refers to the informations reported in the previous chapter “ESSENTIAL HARDWARE AND SOFTWARE”.

1) Read completely the documentation delivered.

2) Prepare the target board to work (board supplied, configuration verified, etc.), assure that the EPROM or FLASH EPROM is installed in the proper socket, as indicated in the target board's technical manual.

3) Make sure the the jumper Run/Debug on the target board is not connected and that the Baud Rate of the board is 115.2 KBaud.
Normally the board is delivered already configured this way, however to check the correct configuration please refer to the chapter “DESCRIPTION OF **GDOS 188**”.

4) Perform the serial connection following the indications in the paragraph “SERIAL COMMUNICATION CABLE”.

5) Turn on the Personal Computer.

6) Create a directory on the hard disk of the Personal Computer. If the P.C. is not provided with hard disk, make a copy of the work disk and jump to point **9**.

7) Copy into the directory all the work software, the programming software and the examples considered intresting (see “DISTRIBUTION DISK”).

8) Enter into the work directory.

9) Install **GET188.EXE**, this meas the installation routine, typing at the MS-DOS prompt the command **GET188 /I** <**Enter**>. Provide all the informations asked by the window: the serial port number (COM) used to connect to the target board, the Baud Rate used to communicate to the target board, the kind of monitor used (coloror black and white), the name of the User and of the Firm that will use the program **GET188**. For further informations please refer to the paragraph “INSTALLATION”.

10) Run the editor and terminal emulation program GET188.EXE, (please see the paragraph “DESCRIPTION OF GET188”) and wait for the presentation window.

11) Close the presentation window pressing <**Enter**> and select the option "Terminal" (shortcut ALT+T), now a clear screen is shown, with the cursor located in the top left corner of and the following status line in the last line visible:

```
F10 Menu | TERMINAL for Mon./Pasc. 188 - GRIFO° Tel. +39-051-892052
```

12) Supply the target board: immediatly after this on the monitor of the P.C. an indication reporting the identification message of the PASCAL 188 compiler and the operating system prompt must appear:

```
------------------------------------------------------------------------------------------
PASCAL 188 Editor-Compiler  Version x.x  grifo° Italian Technology
------------------------------------------------------------------------------------------

<Menu and several informations>

P188>
```

13) Work using the PASCAL 188 compiler according to themodalities shown in its manual.
To return to the MS-DOS operating system, press the F10 function key then the combination ALT+X.
The PASCAL 188 may accept, for example, the following instructions:

```
P188>E    <name of the program file, for example TEST.PAS>
```
this gives access to the editor of the PASCAL188.

```
Program TEST;

Var IND:Integer;

Begin
        For IND:= 1 To 10 Do
                Writeln(IND);
End.
```

this example program performs a count from 1 to 10.

Now press the combintion CTRL+K, the press the D key, to return to the main menu.

```
P188>R
```
to compile and execute the program in memory.

In this chapter, when showing messages output on the P.C. monitor, **X** letters have been used to replace generic symbols, a letter or a number, that indicate version numbers or release kinds, and so may vary in time and according to the target board used.

USING THE GET188 PROGRAM

Here follow all the instruction about how to use the editor and intelligent terminal emulator program for P.C. and its installation program.

GENERAL FEATURES

The **GET188** (**Grifo**® **E**ditor **T**erminal **188** family) program is used in conjunction with **GDOS 188** software package, that runs on the target board, and gives the User the possibility to edit a program, transfer it from P.C. to target board and vice versa, execute it and test it.
GET188 performs two main tasks: editing the application programs developed by the User and intelligent terminal emulation, that features all the common terminal emulation function, like any dumb terminal, but in addition it provides the ability to employ the mass storage devices on the P.C. to transfer programs to the target board or to store on disk a program from the target board local memory. **GET188** also includes two utilities; the first of them allows the User to run an external compiler, without any need to return to the operating system. This way it is possible, for example, to edit an Assembly source file using the editor, to compile it by the appropiate program, without returning to DOS, then to send the compiled code to the Monitor/Debugger 188 running on the target board.
The second utility allows the User to create the image of the user program in Auto-Run mode and to store into an EPROM; all this immediatly and easily, without any need to know the physical addresses where to write the files in the EPROM.
The program is completely based on MS-DOS function calls, so it can run on any computer that uses this operating system version 3.3 and later.

ESSENTIAL HARDWARE AND SOFTWARE

Here follows a list of the minimum P.C. requirements to run properly the **GET188.EXE** program:

Personal Computer:	IBM or compatible.
RAM Memory:	640 KBytes minimum.
MS-DOS:	Version 3.3 or later.
Monitor:	Color or Black and White.
Mass storage devices:	Floppy Disk Drive and optionally Hard Disk Drive of any format driven by MS-DOS.
Serial port:	COM 1, 2, 3, 4 V 24 specifications compliant.
Mouse:	Microsoft or compatible with its driver installed.

DESCRIPTION OF GET188

GET188 is an easy to use program that features an high level interface providing on line help, pull down menus, colors, listboxes, keybord shortcuts, mouse management, etc.
Please remark that the P.C. running GET188 is a valid support tool only during the development and the debugging of the programs; in fact the target board can work anyway without the Personal Computer, or in conjunction with a common serial based terminal emulator.
The syntax used to run the program is:

GET188 <Enter>

to give directly to the MS-DOS prompt.
GET188, as first, initializes the Personal Computer and shows an information window at the center of the monitor that reports all the version informations and all the particular informations about **grifo**® and the User, as specified during the installation phase. Pressing the ENTER key or clicking OK by the mouse the information window disappears and the main window of the program becomes accessible; this window carries six pull down menus that make available a set of options described in the following paragraphs.

INSTALLATION

Before using the GET188.EXE program it must be installed. To do this a specific utility program has been created. It can be executed typing, from the MS-DOS prompt, the following line:

GET188 /I <Invio>

The installation utility shows a requester to input five installation parameters: the number of P.C. serial port (COM), the Baud Rate used, the type of P.C. monitor (color or black and white), the name of the User and of the Firm that are going to use the program. Please remark that these five informations are requested only if **GET188.EXE** hasn't already been installed, otherwise only the first three informations are requested; this means that the name of the User and the Firm cannot be changed after the installation.
These informations constitute the default configuration of **GET188**, the first three can be changed in any moment.
The installation can be aborted by pressing the “Abort” key, or it can be confirmed pressing the “Install” key; if the name of the User and the Firm have not been typed, pressing the “Install” key will abort the installation, which will not be completed.
If the User tries to tun **GET188.EXE** before installing it, the program will not run and a proper error message will be displayed, otherwise the program will run displaying all the informations input during the installation phase.

EDITOR

The **GET188.EXE** program features a powerful and versatile Editor capable to create ASCII files directly usable by **GDOS 188**. The Editor performs all the common functions proper of and editor, allowing the User to take advantage of a wide set of options that make the diting easier, in addition it can operate on more than one file at the same time displaying them in several windows, one for each file.
The only limit for the Editor consists in the size of each single file, that cannot be greater than 64 KBytes: this limit can be however easily overcome editing more than one file in more than one window.

Here follows the list of the commands available in the menu bar (activable pressing **F10**) when **GET188** is in Editor mode.

Option	*Key*	*Function*
Menu	F10	Activates the menu bar to select the desired function

File Menu

Option	*Key*	*Function*
New	-	Opens a new file for editing called "Untitled"
Open...	F3	Opens an existing file into a new Editor window
Save	F2	Saves the content of the current editing window in a file whose name is displayed on top of the window itself
S**a**ve as...	-	Saves the content of the current editing window in a file whose name is specified by the User
Change dir...	Alt+F5	Changes the current directory of MS-DOS
Dos shell	-	Exits to MS-DOS temporarily, to execute other applications; GET188 remains resident in memory and must not be run again
E**x**it to DOS	Alt+X	Exits to MS-DOS permanently

Edit Menu

Option	*Key*	*Function*
Undo	-	Undo last action, if possible
Cu**t**	Shift+Del	Deletes from the current editing window the text selection, saving it in the clipboard
Copy	Ctrl+Ins	Copies from the current editing window the text selection, saving it in the clipboard
Paste	Shift+Ins	Deletes into the current editing window the content of the clipboard, starting from the cursor position
Cl**e**ar	Ctrl+Del	Deletes from the current editing window the text selection, without saving it
Show clipbard	-	Displays the clipboard content

Search Menu

Option	*Key*	*Function*
Find...	-	Searchs for a specified string inside the current editing window
Replace...	-	Searchs for a specified string inside the current editing window and repalces it with another specified string
Serach **A**gain...	Ctrl+L	Repeats the last “Find” or “Replace” operation

Window Menu

Option	*Key*	*Function*
Tile	-	Tiles the monitor with all the open windows
C**a**scade	-	Shows all the open windows in cascade, letting see only their frame and name
Size/Move	Ctrl+F5	Mover and/or resizes the current editing window
Zoom	F5	Explodes the current editing window to the maximum size
Next	F6	The next open window becomes the current editing window
Previous	Shift+F6	The previous open window becomes the current editing window
Close	Alt+F3	Closes the current editing window

Options Menu

Option	*Key*	*Function*
Terminal	Alt+T	Activates the intelligent terminal emulation mode. For further informations about this option, please refer to the paragraph “TERMINAL EMULATION ”
Serial port...	-	Selects the Baude Rate and the serial port number on the P.C.
Video	-	Selects the color or black and white representation mode
Help	F1	Activates the on line Help window
Help **L**anguage...	-	Selects the language of the on line Help
Information...	-	Shows the information window

Utility Menu

Option	*Key*	*Function*
Compiler...	-	Allows to execute a compiler without exiting to the MS-DOS
Compile **a**gain	Alt+F9	Allows to execute another compilation with the previously set parameters
Make EPROM...	-	Allows to create the binary image to store in EPROM, containing the user program in Auto-Run mode

In the previous description of the options, the **bold** letter indicated the key to press to select quicly the relative option from inside the pull down menu without having to use the arrow keys, while under the coloumn *Key* the shortcut combination to activate the option without even having to activate the pull down menu is indicated.
The "..." indication, after some options, indicates that the User, after activating the option, will have to input more informations that will be requested by the program itself (for example, a file name, a string to search for, the directory to select, etc.).
These selections are easied by the use of the mouse, in fact it's enough to click the desired option to activate it, without having to touch the keboard.

A deeper description of the **GET188** options is not reported in this manual; please refer directly to the on line Help, available from inside the program, by simply pressing the **F1** key.

TERMINAL EMULATION

The intelligent terminal emulation mode manages all the P.C. resources, making them available to the target board. This program allows to use Floppy Disk Drives, Hard Disk Drives, printer, keyboard and monitor of the P.C. directly from programs and software running on the target board through the **GDOS 188** operating system.
The communication to the target board is managed through one of the serial lines of the P.C. (COM ports), employing the connection and the protocol indicated in the paragraphs " SERIAL COMMUNICATION CABLE " and "TECHNICAL FEATURES OF **GDOS 188**".
When the "Terminal" option of the menu "Options" is activated, the selected serial line is configured with the selected Baud Rate and the terminal emulation window is displayed.
The serial line and Baud Rate can be selected by the User, through the "Serial port" option of the menu "Options", these selections, at the program start, have the default values set by the User during the installation phase. The User should instal or reinstall the **GET188.EXE** program to speed up the use of the intelligent terminal emulation.

All the characters sent by the target board are displayed on the P.C. monitor, while all the keys pressed on the P.C. keyboard are sent to the target board through a specific logic protocol that is used also to manage the files transmission. This logic protocol is completely transparent to the User; it emploies software handshakes to control fast communications.

NOTE
The software handshake management is performed through the value stored in the **CONSOLE Flag** (please refer to the paragraph "WORK RAM ORGANIZATION") as here explained:

CONSOLE Flag (Address 00402H) = 55AA H	->	*software handshake disabled*
CONSOLE Flag (Address 00402H) ≠ 55AA H	->	*software handshake enabled*

During the communication with **GET188**, software handshake must be enabled, this is the reason why **GDOS 188** sets by default the CONSOLE Flag to 55AA Hex.
If the user program cannot use the software handshake (for example, it uses serial line A to communicate to other systems like modem, terminals, etc.), it will have the responsability to set the CONSOLE Flag to the opportune value.

All the programs developed using **GDOS 188** can take advantage of the intelligent terminal emulation described in the following paragraphs, remarkably easing the user interface development, while most of the programming languages is provided already configured to take advantage of this feature.
The intelligent terminal emulation program is completely asynchronous, so the supply of the remote board running **GDOS 188** isn't time-dependent, it can be turned ON and OFF independently from the terminal emulator.

NOTE
It is possible to run **GET188** from the MS-DOS prompt directly in intelligent terminal emulation mode typing the following line at the MS-DOS prompt:

GET188 /T <Enter>

Terminal Emulation commands:

In the intelligent terminal emulation mode a set of options is available, through the menu bar (activablr pressing F10 key), these options help the User in the firmware utilization; here follows a list of these options and a short description:

Option	*Key*	*Function*
Menu	F10	Activates the menu bar to select the desired function

File Menu

Option	*Key*	*Function*
Change dir...	Alt+F5	Changes the MS-DOS current directory
Dos shell	-	Exits to MS-DOS temporarily, to execute other applications; GET188 remains resident in memory and must not be run again
E**x**it to DOS	Alt+X	Exits to MS-DOS permanently

Options Menu

Option	*Key*	*Function*
Edi**t**or	Alt+T	Activates the intelligent terminal emulator
Reset Terminal	Ctrl+Home	Clears the terminal window and resets the serial communication to the target board
Serial Port...	-	Selects the Baud Rate and the serial port on the P.C.
Video	-	Selects the color or black and white representation mode
Help	F1	Activates the on lin Help window
Help **L**anguage...	-	Selects the language of the on line Help
Information...	-	Shows the information window

Utility Menu

Option	*Key*	*Function*
Compiler...	-	Allows to execute a compiler without exiting to the MS-DOS
Compile **a**gain	Alt+F9	Allows to execute another compilation with the previously set parameters
Make EPROM...	-	Allows to create the binary image to store in EPROM, containing the user program in Auto-Run mode

In the “Utility” menu these two sub-menus are present:

Transmit file to Monitor sub-menu

Option	*Key*	*Function*
HEX Intel file...	F8	Allows to transfer ti the Monitor/Debugger, a filE in the Intel HEX format, stored in the mass storage device of the P.C. This function uses the **X** command of the Monitor/Debugger, for furthrt informations please refer to the chapter “MONITOR/DEBUGGER DESCRIPTION”
BINARY file...	F8	Allows to transfer ti the Monitor/Debugger, a fil in the binary format, stored in the mass storage device of the P.C. The program will be stored in the RAM of the target board starting from the address specified by the **CS:IP** registers. This function uses the **X** command of the Monitor/Debugger, for furthrt informations please refer to the chapter “MONITOR/DEBUGGER DESCRIPTION”

Send command to Pascal sub-menu

Option	*Key*	*Function*
Save (^K D S)	F7	Sends to the **PASCAL 188** compiler the code sequence equivalent to the keys: CTRL+K D S Using this feature it is possible, pressing only the **F7** key, to exit from the editor (sequence CTRL+K, D) and save the current file (equivalent to the S key from the main menu)
Compiler (^K D C)	F9	Sends to the PASCAL 188 compiler the code sequence equivalent to the keys: CTRL+K D C Using this feature it is possible, pressing only the **F9** key, to exit from the editor (sequence CTRL+K, D) and compile the current file (equivalent to the C key from the main menu)
Run (^K D R)	Ctrl+F9	Sends to the PASCAL 188 compiler the code sequence equivalent to the keys: CTRL+K D R Using this feature it is possible, pressing only the **Ctrl+F9** keys, to exit from the editor (sequence CTRL+K, D) and run the current file (equivalent to the S key from the main menu)

In the previous description of the options, the **bold** letter indicated the key to press to select quicly the relative option from inside the pull down menu without having to use the arrow keys, while under the coloumn *Key* the shortcut combination to activate the option without even having to activate the pull down menu is indicated.

The “...” indication, after some options, indicates that the User, after activating the option, will have to input more informations that will be requested by the program itself (for example, a file name, a string to search for, the directory to select, etc.).
These selections are easied by the use of the mouse, in fact it's enough to click the desired option to activate it, without having to touch the keboard.

A deeper description of the **GET188** options is not reported in this manual; please refer directly to the on line Help, available from inside the program, by simply pressing the **F1** key.

Terminal emulation control sequences:

GET188 intelligent terminal emulation board recognizes some byte sequences received from the serial line. These sequences are reported in the following table:

COMMAND	*BYTE CODE*	*ASCII CODE*
HOME	01	SOH
CURSOR LEFT	08	BS
CURSOR RIGHT	06	ACK
CURSOR DOWN	10	LF
CURSOR UP	26	SUB
CARRIAGE RETURN	13	CR
CARRIAGE RETURN + LINE FEED	29	GS
cursor absolut positioning	126 17 column row	~ DC1 ASCII(column) ASCII(row)
CLEAR PAGE	126 28	~ FS
CLEAR LINE	126 14	~ S0
CLEAR END OF LINE	126 15	~ SI
CLEAR END OF PAGE	126 16	~ DLE

FIGURE 4: GET188 CONTROL SEQUENCES TABLE 1

COMMAND	*BYTE CODE*	*ASCII CODE*
One Line Clear With SCROLL-UP	126 19	~ DC3
New Line Insertion	126 26	~ SUB
Cursor Disable	126 22	~ SYN
“Line” Shaped Cursor	126 20	~ SUB
“Block” Shaped Cursor	126 21	~ NACK
Set Normal Luminosity	126 25	~ EM
Set High Luminosity	126 31	~ US
Reverse Attribute Setting	126 27	~ ESC
BELL	07	BEL

FIGURE 5: GET188 CONTROL SEQUENCES TABLE 2

The values of row and column may range respectively 0÷23 and 0÷79.For example, if the User wants to move the cursor to row 10, column 20 then he/she will have to send the sequence:

126, 17, 20, 10

All the programs may take advantage of the above described commands, making so very easier the development of the User interface.

Terminal emulation special keys:

The intelligent terminal emulation codes the special keys of the P.C. keyboard (arrow keys, Ins, Del, etc.) that are sent to the target board as **GDOS 188** standard codes, as shown in the following table:

KEY	*CODES*	*HEX CODES*
UP Arrow	0, 200	0, C8
DOWN Arrow	0, 208	0, D0
LEFT Arrow	0, 203	0, CB
RIGHT Arrow	0, 205	0, CD
Page UP	0, 201	0, C9
Page DOWN	0, 209	0, D1
Home	0, 199	0, C7
End	0, 207	0, CF
Insert	0, 210	0, D2
Delete	0, 211	0, D3

FIGURE 6: GET188 SPECIAL KEY CODES TABLE

This feature easies remarkably the development when using programming languages with an integrated editor (for example PASCAL 188), in fact it is possible to move inside the application program in an intuitive, comfortable and quick way.

EXECUTING A COMPILER

This feature allows the compilation of any file, using the desired compiler, without having to return to the operating system.
It is possible, for example, to generete an Assembly source using the editor, compile it using the proper program, transfer it to the Monitor/Debugger 188 running on the target board and test it. All this without ever exiting from the **GET188** program, which allows to perform the above mentioned operations from inside its integrated environment.

To run an external compiler through the dedicated utility, the User must follow the below described steps:

1) Assure that the compiler, the source file and the eventual linker or conversion program are stored in the drive of the P.C.

2) Execute the utility through the “Compiler...” option of the “Utility” menu, in order to open the compilation window

3) Select the compiler to use

4) Select the source file to compile

5) Type the eventual compiler parameters

6) Specify whether the eventual parameters must be put before or after the source file name. This detail depends on the compiler being used; some compilers request the parameters to be put before the source file name, other act viceversa

7) Type the eventual command line to be executed after the end of the compilation. This feature can be useful when, for example, after the compiler it is essential to run a linker or another conversion program

8) Confirm the input and execute the compilation

These operations can be executed through the options of the compile window, as follows:

Compile**r**	Allows to select the file name of the compiler to use. The complete pathname will be displayed near the option; if no selection has been made the indication “**Not Selected**” will be displayed
Source File	Allows to select the source file name. The complete pathname will be displayed near the option; if no selection has been made the indication “**Not Selected**” will be displayed
Parameters:	Allows to type a string of, at most, 80 characters, where to specify the set of parameters to pass to the compiler. If no parameter is to be passed then the line must be left blank

Parameters position: Indicates whether the parameters must be put before (*Before surce file name*) or after (*After source file name*) the source file name

Second command line: Allows to type a string, at most, 70 characters long, containing the MS-DOS command line that must be executed after the compilation.
If this string contains characters, after the compilation it will be passed to the operating system, just like typed; otherwise there will be a return to the **GET188** integrated environment

After setting these values the compilatin can be executed using the "**R**un" option. In this phase an MS-DOS screen will be displayed, showing the messages output by the executed programs. Such operation may be executed again, employing the same parameters, using the option "Compile **a**gain" (shortcut keys **Alt+F9**) of the "Utility" menu.

NOTE
Whenever a compilation is executed, the souce file, if changed, must be saved using the Editor of **GET188**. This must be done because the external compiler reads the source file from disk, so it must ve updated with the latest changes.

Example

The compilation feature can be used also to execute the **GHEX2.COM** utility, delivered in the distribution disk.
In fact, for example, if the User wants to convert the binary file **PROVA.BIN** in Intel HEX format starting from the address **1000H**, then he/she will have to follow these steps:

1) Select the program **GHEX2.COM** through the "Compiler" option

2) Select the file **PROVA.BIN** through the "Source File" option

3) Type the parameter: **1000**

4) Set the parameter position after the source file name

5) Execute the program confirming the input values pressing the "Run" button

6) The program PROVA.BIN has been converterted in Intel HEX format generating the file **PROVA.HEX**

CREATING A BINARY IMAGE OF THE USER EPROM

This function allows to create very easily and quickly the binary image to store in EPROM, containing the user program in Auto-Run mode, without no need to know about the physical addresses in EPROM at which the files must be written.
To build the EPROM image through this utility the USer must follow these steps:

1) Assure that the user program, the eventual data file and the binary image of **GDOS 188** (delivered in the distribution disk) are stored in the P.C. drive

2) Execute the utility selecting the option “Make EPROM” from the “Utility” menu, this will open the window of the utility

3) Select the file containing the binary image of the **GDOS 188** + Monitor/Debugger 188 for the target board. This file, called **MON?????.BIN** is stored in the directory EPROM of the distrubution disk

4) Select the executable file containing the user program. This file can be obtained using the PASCAL 188 compiler; to do this the User must compile the source file, after selecting the option to compile on disk from the “compiler Options” menu; this will create a file with the “**.COM**” extension, that can be used directly

Select the eventual file containing data, messages or other informations used by the user program; this operation is optional. This file is stored in the EPROM starting from the address **11000H**, that the **GDOS 188** operating system allocates from the address:

F1000H = F100:0000

So data coming from this file will be available to the user program by memory read operations from that address.
For further informations please refer to the paragraph “EPROM OR FLASH EPROM ORGANIZATION”

6) Type the name of the binary file containing the EPROM image to be generated by the utility; the eventual unused area is filled with 0FF Hex bytes.

7) Confirm all the input parameters and begin the building of the binary image file

8) Exit from **GET188** to MS-DOS and verify the effective presence of the file just generated

9) Burn a 27c010 (1 Mbit x 8) EPROM with the binary file just got. This file must be written starting from the address **0000 Hex**; if the EPROM programmed requires the **Intel Hex** format, provide to convert the binary file into this format, specifing this address as offset. To perform this operation, the **GHEX2.COM** program, provided with the distribution disk, can be used.

The programmed EPROM can be installed on the target board; this executes in Auto Run mode, on a reset or a Power-ON, the program written.

The steps **3÷7** previously described are executable through the options of the utility's window, here follows their description:

Monitor **F**ile	Allows to select the name of the binary image file containing the operating system. The complete pathname will be displayed near the option; if no file has been selected yet then a "**Not Selected**" indication will be displayed
Program File	Allows to select the name of the executable file containing the user program. The complete pathname will be displayed near the option; if no file has been selected yet then a "**Not Selected**" indication will be displayed
Data File	Allows to select the name of an eventual data file containing messages or other informations. The complete pathname will be displayed near the option; if no file has been selected yet then a "**Not Selected**" indication will be displayed
BIN EPROM File	Allows to select the name of the binary image file to be generated. The complete pathname will be displayed near the option; if no file has been selected yet then a "**Not Selected**" indication will be displayed

Once the several values have been given, the option "**M**ake" can be selected. During this phase a window will be displayed; the window contains a percentual indication of the progress status and a line where informative messages will be reported.

NOTE
The data file selection is optional; if it has been performed but the User wants to cancel it, then it will be enough to select the option "**C**lear" in the window that appears when selecting the "**D**ata File" option.

TECHNICAL FEATURES OF GDOS 188

Here follow all the GDOS 188 package technical features; these informations will be explained more in the details in the rest of this manual, these pages only summarize them.

1) **Mass storage devices:** P.C. drives from **A** up to **L**
On board RAM Disk as drive **M**

These drives can be formatted in different formats, for the RAM Disk there is also a write protection circuitry management

2) **Communication:**

Consolle line	->	A serial line
Auxiliary line	->	B serial line

These lines are initialized and managed by GDOS 188 with these communication protocols:

Consolle line:

Baud Rate	= **GPC® 188F** e **GPC® 188D** boards: selectable amongst: 19200, 38400, 57600 and 115200 Baud **GPC® 884** board: 115200 Baud
Parity	= None
Stop Bit	= 1
Bits / Char	= 8
Handshake	= None

Auxiliary line:

Baud Rate	= 9600 Baud
Parity	= None
Stop Bit	= 1
Bits / Char	= 8
Handshake	= None

3) Work RAM:	**GPC® 188F** and **GPC® 188D** boards: 32 KBytes minimum **GPC® 884** boards: 128 KBytes minimum
4) Printer:	Parallel printer connected to the I/O connector of the target board
5) Watch Dog:	Retriggered during RAM Disk formatting and reading and during the printer Time-Out
6) Real Time Clock:	High level management through the **GDOS 188** function calls
7) EEPROM:	High level management through the **GDOS 188** function calls

DESCRIPTION OF GDOS 188

GDOS 188 is an operating system that performs an high level interfacement between the User and the hardware to be used. In fact, through simple function calls, both the resources of the target board and the resources of the P.C., acting as terminal emulator, are available to the user program. Here follows a description about how to use the **GDOS 188** operating system on the target board. In detail are reported all the indications regarding the high level management of the several peripherials, the memory management, the registers mapping, the configuration jumers management and the Autorun programs.

WATCH DOG

GDOS 188 provides to retrigger the watch dog, on all the target boards provided with a watch dog, during an operation whose duration may exceed the watch dog intervent time.
These operations are: GDOS 188 boot phase, RAM Disk formatting, RAM Disk read and printer Time-Out.
This feature allows to develop applications capable to take advantage of chain or overlay techniques (the running program loads and runs other programs) and use the watch dog circuitry at the same time.
This means that seeing OFF the watch dog activity LED when one of the above mentioned operations arein progress is a normal situation.

SERIAL EEPROM READ AND WRITE

GDOS 188 make available two functions, reachable through INT 3B Hex, that allow the user program to read and/or write a byte to a selectable address of the serial EEPROM installed on the target board.
These functions, whose description is reported in the chapter “FUNCTIONS OF **GDOS 188** OPERATING SYSTEM”, can be called through high level instructions of the PASCL 188 compiler and allow to simplify the device management, reducing the amount of generated code.
GDOS 188 can manage six serial EEPROM sizes: 2 KBit (256 Bytes), 4 KBit (512 Bytes), 8 KBit (1024 Bytes), 16 KBit (2048 Bytes), 32 KBit (4096 Bytes) e 64 KBit (8192 Bytes); please remark that no check is effectuated to detect whether a certain read or write operation will produce an address out of range, the User has the complete responsability to avoid these situations.

REAL TIME CLOCK MANAGEMENT

GDOS 188 makes availabe several functins, reachable through the **INT 21 Hex**, that allow the user program to read and set date and time of the Real Tile Clock installed on the target board.
These functions, whose description is reported in the chapter “FUNCTIONS OF **GDOS 188** OPERATING SYSTEM”, can be called through high level instructions of the PASCL 188 compiler and allow to simplify the device management, reducing the amount of generated code.

DIGITAL I/O INTERFACES

All the industrial control board made by **grifo®** are provided with at least 16 I/0 TTL lines, available on a standard connector. There are also available a wide range of interface boards provided with the same standard I/O connector, that can be connected directly to the target board through a simple flat cable. Amongst these several interfaces we would want to remark didactic boards, I/O power interfaces to connect signals on the field, boards provided with mass storage devices, operator interfaces, etc. GDOS 188 performs the high level management of some of these boards, to let the User take advantage of their features without having to learn how to use their hardware; there are also specific high level instructions.
The following table reports which connector to use to connect to the target board the interfaces described further in this chapter and the proper connection cable.

TARGET BOARD	*CONNECTOR*	*CONNECTION CABLE*
GPC® 188F	**CN2**	**FLT 20+20**
GPC® 188D	**CN2**	**FLT 20+20**
GPC® 884	**CN5**	**FLT 26+20**

FIGURE 7: I/O INTERFACES CONNECTORS TABLE

Of course only one of the further described interfaces can be connected and so used by **GDOS 188** and in case of use the sixteen I/O lines of the proper connector reported in the above table will not be available to the User any more. **GDOS 188** perfroms no operations on these lines unless the user program calls a function to activate the signals; this is made to keep the system safe when the lines are managed for the application program, only the User can decide to activate the signals through the proper modalities described in the following paragraphs.

LOCAL PRINTER

Amongst the several **grifo®** interface boards, we would want to remark **IAC 01** and **DEB 01** which allow to manage a local printer connected directly to the target board. The printer to empmoy is a generic printer provided with CENTRONICS parallel interface (office printer, panel printer, etc.) that must be connected to the two digital I/O ports whose signals are available on the connector reported in figure 7. These boards are provided with standard pin out connectors, so a generic and cheaper standard connection cable can be used. In software, to communicate to the printer the PASCAL 188 command Write(LST, ...) can be used, or the **GDOS 188** functions can be called directly, as explained further in this manual.
The digital I/O lines used to manage the printer are set only when printing the first character after the last Power On or reset of the target board, so if the user program doesn't employ the printer, no setting is made by **GDOS 188** on the printer management digital I/O lines. The **GDOS 188** print function doesn't risk to hang the system: if the printer is not ready after a **2 seconds**, a Time Out exits the function itself; in addition it is not possible to have informations about the printer status.

JUMPERS FOR GDOS 188 SETTINGS

The **GDOS 188** operating system, when a Power On or a reset occour, cheks the status of a set of jumpers on the target board, to set several working modalities.
This way the User can comfortably select the desired modalities, under any operating condition, without changing the user program.
In fact through these jumpers it is possible to select the Run/Debug execution modality, set the Baud Rate for the consolle line and configure the RAM Disk. Here follows the meaning of each jumper, the possible connections and the obtained configurations. To easily locate the jumpers on the target board please refer to the figures reported in its TECHNICAL MANUAL.

RUN / DEBUG MODALITY SELECTION

Through a jumper located on the target board, it is possible to select the **GDOS 188** operating system working modality between **Run** or **Debug**; this happens when a Power On or a reset occour, the status of the proper jumper is checked, in detail:

RUN Modality:	The user program or the PASCAL 188 compiler (in case the User is employing the working structure) is automatically executed.
DEBUG Modality:	Even if an user program or the PASCAL 188 compiler is present, the **Monitor/Debugger 188** program is executed.

The working modality selection is useful especially when the User must intervent on a system already installed. In fact, selecting the Debug mode, the User can make changes to the program, even when on the field, then restart the system without any further problem or time waste. Or it is possible to observe the effects of the user program after its execution, perform statistic data acquisition, etc.
Here follows a table reporting the name of the RUN/DEBUG jumper on some **grifo**® boards:

TARGET BOARD	*RUN/DEBUG JUMPER*
GPC® 188F	**J18**
GPC® 188D	**J18**
GPC® 884	**J1**

FIGURE 8: RUN/DEBUG SELECTION JUMPERS TABLE

The jumper connections have the following meaning:

Jumper J??	*Connected*	->	*DEBUG Modality*
	Not connected	->	*RUN Modality*

For further informations please refer to the paragraph "AUTORUN PROGRAMMING" and to the chapter "MONITOR/DEBUGGER 188 DESCRIPTION".

CONSOLLE LINE BAUD RATE SELECTION

Through two jumpers installed only on **GPC® 188F** and **GPC® 188D** boards called **J16** and **J17**, it is possible to select the Baud Rate for the consolle line if the **GDOS 188** (serial line A on the target board); this happens when a reset or a Power On occour and the status of the jumpers is checked. This way the User can comfortably select one of the several Baud Rate available forthe consolle seial line, according to the terminal or P.C. being used, without changing the user software.
Possible jumper configurations and Baud Rate are reported in the table below.

Jumper J16	Jumper J17	BAUD RATE
Not connected	Not connected	**115200 Baud**
Connected	Not connected	**57600 Baud**
Not connected	Connected	**38400 Baud**
Connected	Connected	**19200 Baud**

FIGURE 9: CONSOLLE LINE BAUD RATE SELECTION ON GPC® 188F AND GPC® 188D TABLE

NOTE
This particular working modality is not available when using **GDOS 188** and **GPC® 884** as target board, becuse the consolle line of this board is always initialized with a Baud Rate of 115200 Baud.

For further informations about the communication protocol of the consolle line please refer to the chapter "**GDOS 188** TECHNICAL FEATURES".

GPC® 188F E GPC® 188D RAM CONFIGUATION

Through one jumpers installed only on **GPC® 188F** and **GPC® 188D** boards called **J2**, it is possible to enable or disable the RAM write protection circuitry for the RAM device installed in IC9 socket. The device is configured as RAM Disk in case this circuityu is enabled, otherwise it is managed as normal work RAM by **GDOS 188** operating system.
In detail, the following configurations are possible:

Jumper J2	*Connected*	*->*	*IC9 = RAM Disk*
	Not Connected	*->*	*IC9 = Normal Work RAM*

For further informations please refer to the next paragraphs "MEMORY ORGANIZATION" and "RAM DISK MANAGEMENT".

MEMORY ORGANIZATION

GDOS 188 manages all the memory installed on the target board, making it available to the User through a set of high level commands. Because of this the User doesn't have to worry about the hardware configuration of the target bord; for the User it is enough to follow the indications given in this paragraph and employ the available memory areas.
If the PASCAL 188 is employed, the compiler itself performs the necessary checks.
The following paragraphs report the memory organization for each of the several boards that may execute the **GDOS 188** operating system.

MEMORY ORGANIZATION ON GPC® 188F E GPC® 188D BOARDS

When a resetor a Power On occour , **GDOS 188** checks the amount of RAM available and the abilitation to use the RAM Disk (jumper **J2**); then mempry is organized as follows:

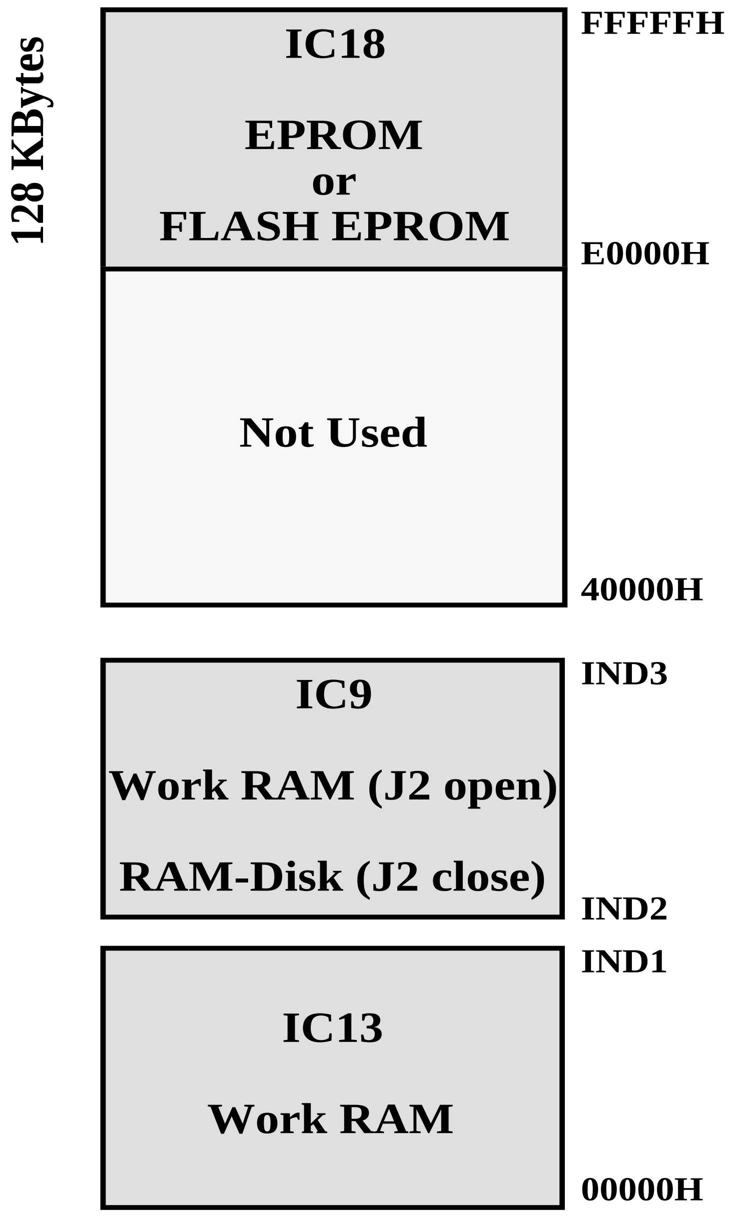

FIGURE 10: MEMORY ORGANIZATION ON GPC® 188F AND GPC® 188D TABLE

The memory map shown in the figure reports some addresses as generical indications; in fact these vary according to the device installed on the respective socket and the abilitation or disabilitation of the RAM Disk (jumper **J2** as previously stated).
Here follows the meaning of these generic indications.

***IND1**: End address of work RAM installed on IC13*

Indicates the last address available, according to the amount of RAM installed on IC13, for **GPC® 188F** and **GPC® 188D** memory organization; it can assume one of the following values:

32 KBytes of RAM installed on IC13	->	IND1 = 07FFFH (0700:0FFF)
128 KBytes of RAM installed on IC13	->	IND1 = 1FFFFH (1F00:0FFF)

***IND2** and **IND3**: Start and End address of work RAM or RAM Disk installed on IC9*

These addresses vary according to the amount of RAM installed on sockets IC9 and IC13 and the abilitation or disabilitation of RAM Disk (jumper **J2** as previously stated). To determine the values assumed please refer to the following table.

RAM on IC13	RAM on IC9	Jumper J2	IND2	IND3
32 KByte	32 KByte	Not connected	08000H 0800:0000	0FFFFH 0F00:0FFF
32 Kbyte	32 KByte	Connected	20000H 2000:0000	27FFFH 2700:0FFF
32 Kbyte	128 Kbyte	Not connected	08000H 0800:0000	27FFFH 2700:0FFF
32 Kbyte	128 Kbyte	Connected	20000H 2000:0000	3FFFFH 3F00:0FFF
128 Kbyte	32 KByte	Not connected	20000H 2000:0000	27FFFH 2700:0FFF
128 Kbyte	32 KByte	Connected	20000H 2000:0000	27FFFH 2700:0FFF
128 Kbyte	128 Kbyte	Not connected	20000H 2000:0000	3FFFFH 3F00:0FFF
128 Kbyte	128 Kbyte	Connected	20000H 2000:0000	3FFFFH 3F00:0FFF

FIGURE 11: START AND END ADDRESSES FOR IC9 RAM ON GPC® 188F AND GPC® 188D TABLE

MEMORY ORGANIZATION ON GPC® 884

When a Power On or a reset occour, **GDOS 188** checks the amount of RAM installed (128 or 512 KBytes), then the available memory is organized as follows:

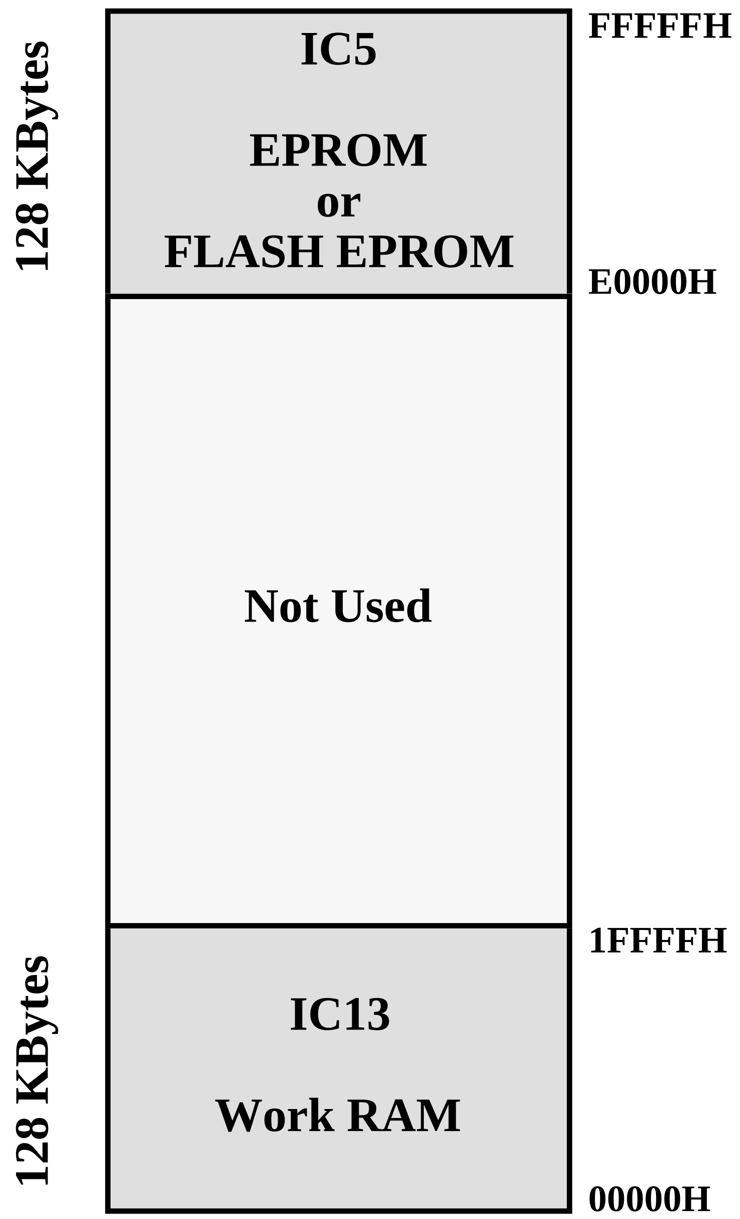

FIGURE 12: MEMORY ORGANIZATION ON GPC® 884 MOUNTING 128 KBYTES OF RAM TABLE

As shown in the figure, in this memory configuration RAM Disk is not available, because the amount of RAM installed on **GPC® 884** is completely managed by **GDOS 188** as work RAM.

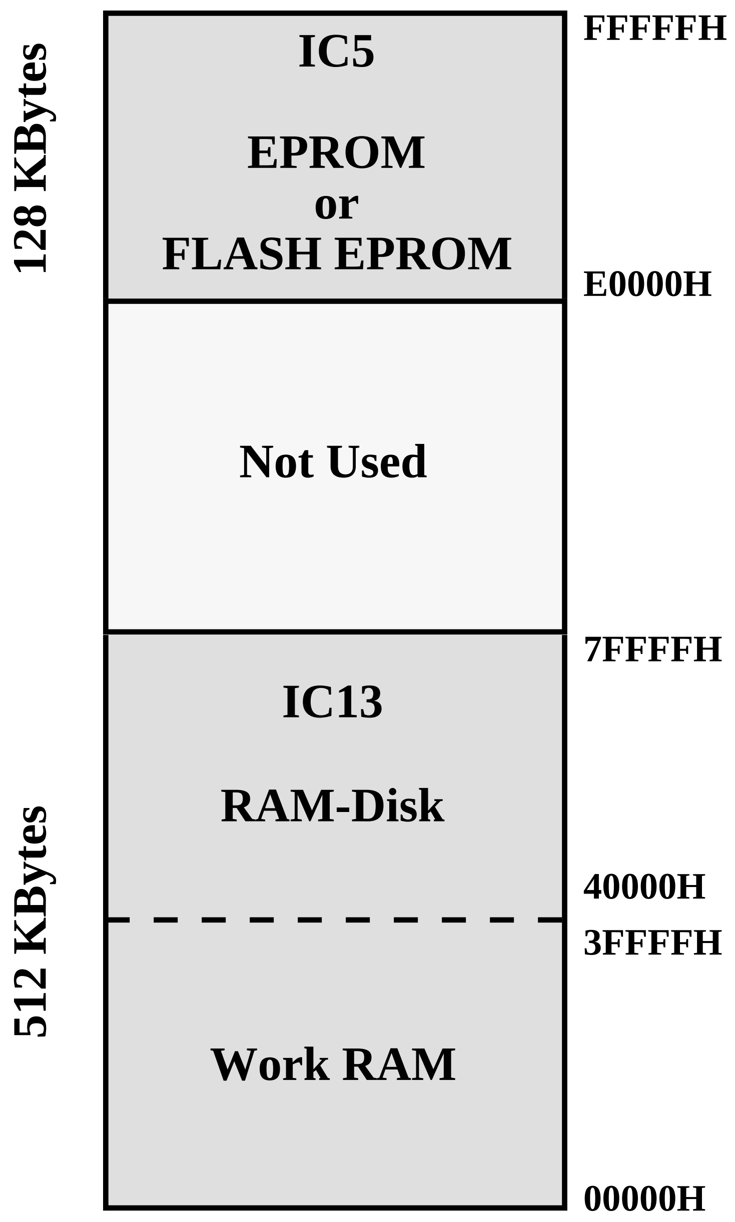

FIGURE 13: MEMORY ORGANIZATION ON GPC® 884 MOUNTING 512 KBYTES OF RAM TABLE

As shown in the figure, in this memory configuration the amount of RAM installed on **GPC® 884** is managed by **GDOS 188** as work RAM for the first 256 KBytes, while the remaining memory is available as RAM Disk.

WORK RAM ORGANIZATION

GDOS 188 operating system divides the amount of work RAM available into three parts: the first part (**4 KByte**) is used to store system variables; the second part, **1 KByte**, is completely available to the User to store data and parameters; the last part, the third, (called **TPA**: **T**ransient **P**rogram **A**rea) is available to the user program.
PASCAL 188 compiler uses directly this part of the memory to store its own several structures: data area, code area, Heap, Stack, etc.; if the User is employing PASCAL 188 doesn't have to manage the **TPA**, he/she will be able to access it through PASCAL style system structure, for example variables, vectors, pointers, sets, lists, etc.
The work RAM organization is reported in the figure below:

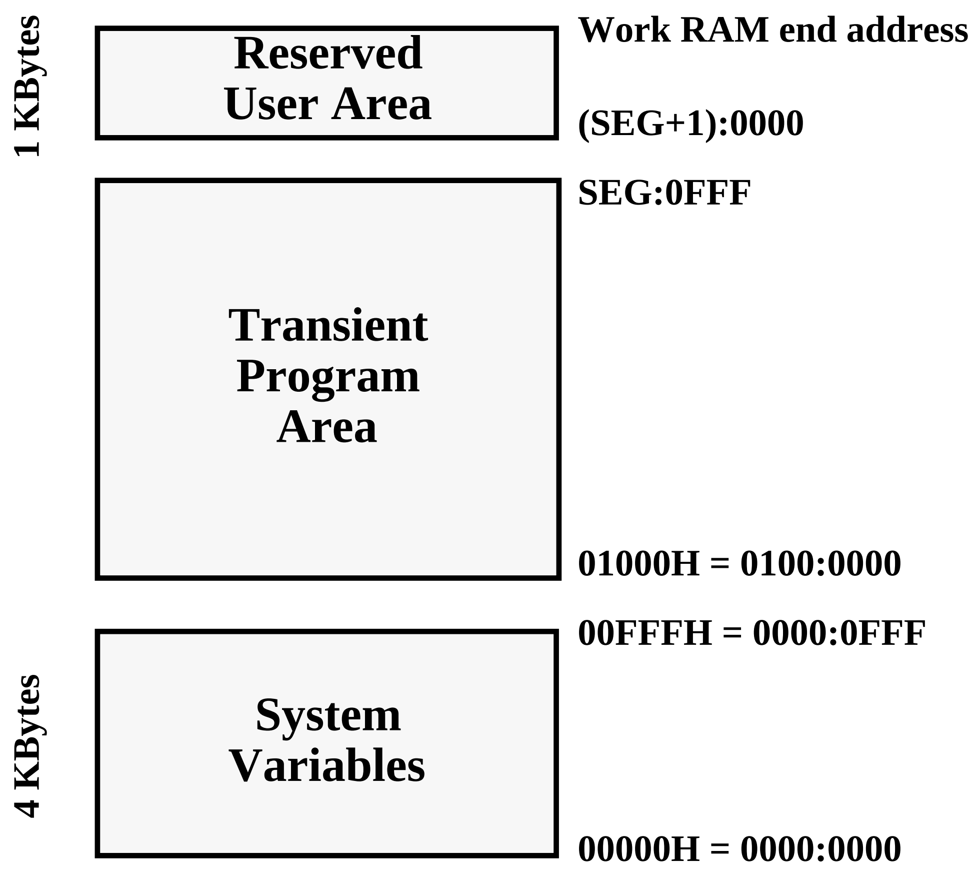

FIGURE 14: WORK RAM ORGANIZATION

The End address of the User reserved RAM (**1 KByte**), called *Work RAM end address*, can be calculated by the indications reported in the previous paragraphs and assumes one of these values:

GPC® 188F and **GPC® 188D**:	IND1 (J2 connected) IND3 (J2 not connected)
GPC® 884:	1FFFFH = 1F00:0FFF (IC13 = 128 KBytes) 3FFFFH = 3F00:0FFF (IC13 = 512 KBytes)

The End address of **TPA** and Start address of User reserved RAM can be drawn by **SEG**: a User readable word set by **GDOS 188** during a reset or Power On, after checking the available RAM.

The 4 KByte RAM area dedicated to store system variables contains three locations where informations obtained by **GDOS 188** during the initialization phase are stored; these variables are User available, they can be read or, under particular conditions, written. Here follows a list of them and their meaning:

Address 00400H (0040:0000) *- Last TPA segment available*

Contains the word **SEG**, previously described, that indicates the last work RAM segment destined by **GDOS 188** to be **TPA**.
If the User wants to read the values of this variable using PASCAL 188 compiler, then he/she will have to insert the following program line:

```
<variable>:=MemW[$0040:$0000];
```

If then the User wants to write a byte into the first address available of the User reserved RAM area, then he/she will just have to type:

```
Mem[<variable>+1:$0000] := <byte>;
```

NOTE
This memory location is read only and must not be altered by the User for no motivation.

Address 00402H (0040:0002) *- CONSOLE Flag*

It allows to specify to **GDOS 188** operating system whether the consolle line (serial line A on the target board) is used to communicate to a generic device or to the intelligent terminal emulation of **GET188**, which uses a specific logic protocol. If the first configuration is the desired configuration, then the CONSOLE Flag will have to contain the word **55AA Hex**, otherwise any other value will set the communication modality to communicate to **GET188** intelligent terminal emulator.
If, for example, working under PASCAL 188 compiler, the User wants to use the consolle line to communicate to a terminal belonging to th **QTP** serie, made by **grifo**®, then he/she will have to insert the following line as first program line:

```
MemW[$0040:$0002]:=$55AA;
```

Address 00404H (0040:0004) *- Value of CPU80c188 PBA register*

It contains the setting of the **Peripheral Base Address** register, programmed by **GDOS 188** operating system during the initalization phase. This value corresponds to the target board own peripherals start mapping address and will be explained with a greater amount of details in the paragraph " TARGET BOARD PERIPHERAL MAPPING".
If, under the PASCAL 188 compiler, the USer wants to read this variable, then he/she will have to insert the following line in the program:

```
<variable>:=MemW[$0040:0004];
```

NOTE
This memory location is read only and must not be altered by the User for no motivation.

EPROM OR FLASH EPROM ORGANIZATION

GDOS 188 package can be purchased in the version including an EPROM or a FLASH EPROM. This must be installed on the proper socket of the target board and contains teh operating system, the PASCAL 188 compiler and the TOOLS 188 utility.
These programs and the other utility programs stored in EPROM or FLASH EPROM are executed directly from this device, leaving free all the work RAM which is, in fact, used only to store variables and data structures and, when a program is developed, to store the program itself.
Executing the code in EPROM gives an intrinsic safety to the system under development against noise, etc.
Here follows the organization of EPROM and FLASH EPROM in the several possible configurations. Please remark that the figures report the memory areas allocation addresses inside the microprocessor addressing space; these addresses are shifted by E0000 Hex compared with the physical addresses of the device itself.

EPROM ORGANIZATION

The EPROM delivered with the GDOS 188 package, called work EPROM, is organized as shown in the following figure:

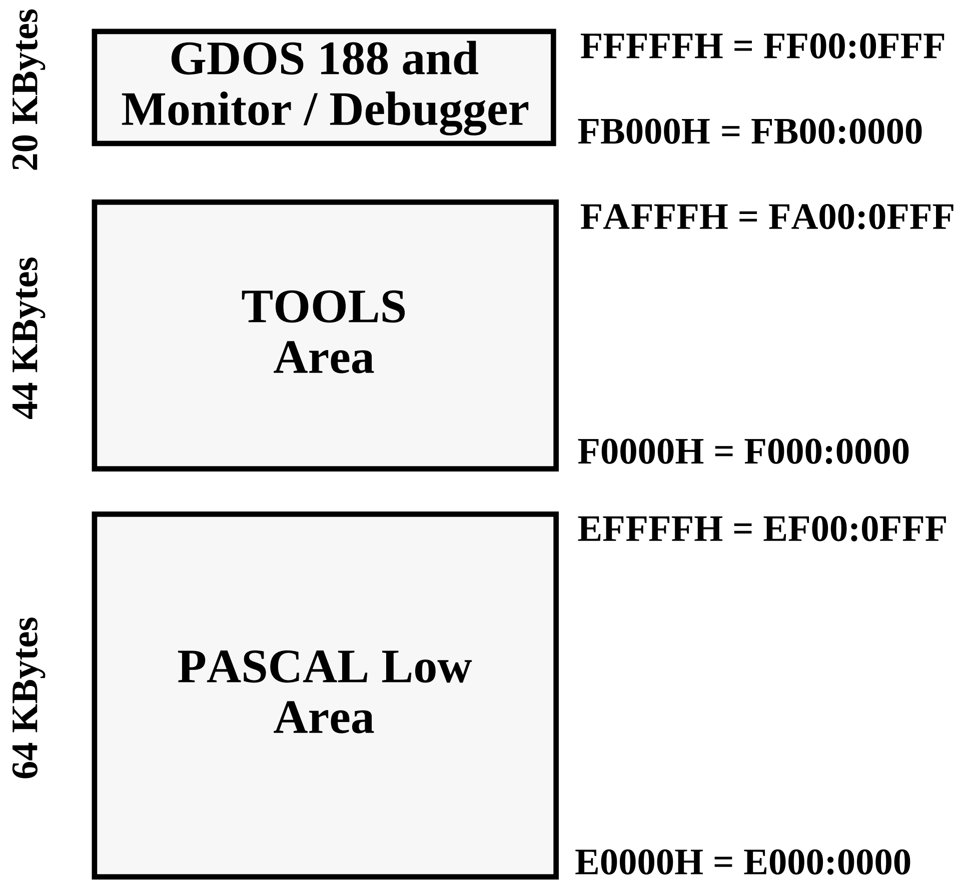

FIGURE 15: WORK EPROM ORGANIZATION

For information purpose only, it is here reported the organization ot the EPROM containing the User program, generated by the specific utility of **GET188** program.
In fact this utility, as previously explained, organizes automatically the several files, without asking the User the allocation addresses.

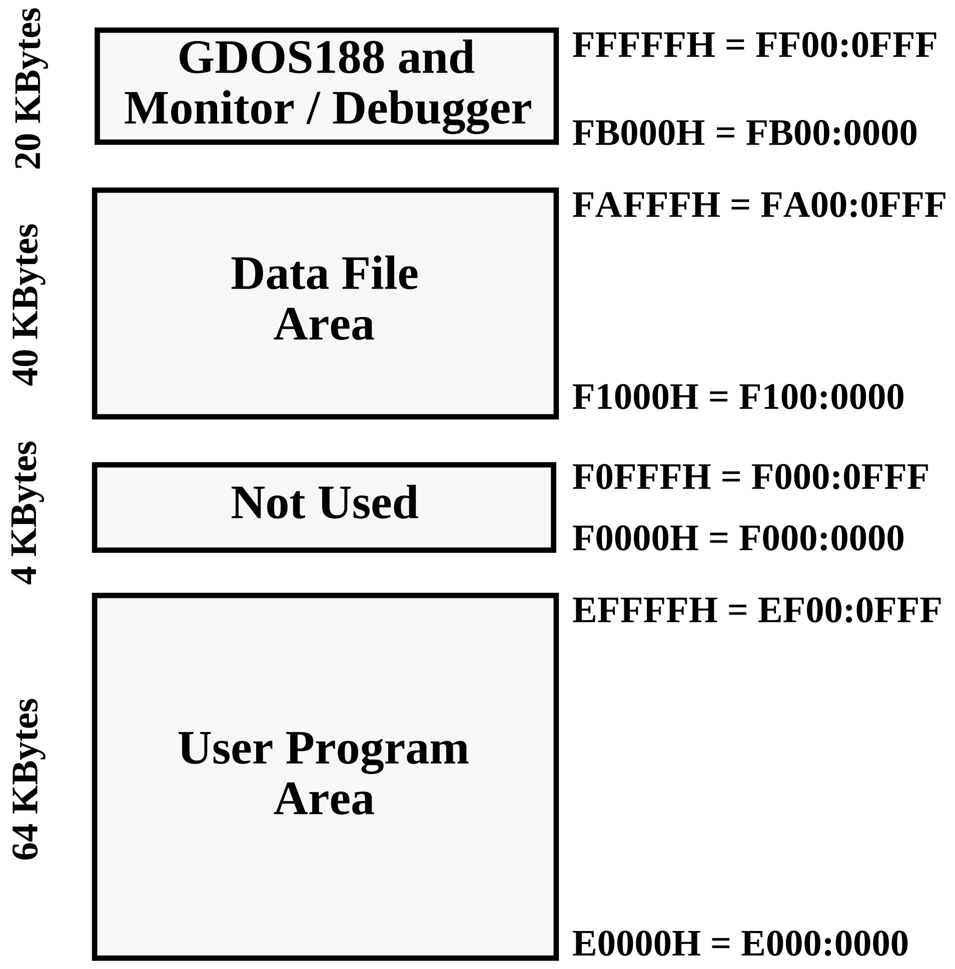

FIGURE 16: USER EPROM ORGANIZATION

GDOS 188 and Monitor / Debugger
It takes the last 20 KByte of EPROM and containes the code of the board basic firmware. This code is the one always executed when a Power On or a reset occour and allows to perforl all the possible execution and / or development phases for the application program.
The content of this area corresponds to the file **MON?????.BIN** delivered in the distribution disk.

TOOLS Area
Low AutoRun priority area, it takes 44 KByte of the work EPROM and contains the TOOLS 188 utility program, corresponding to the file **TOOLS188.BIN** delivered in the distribution disk.

PASCAL Low area
High AutoRun priority area, it takes the first 64 KByte of work EPROM and contains the PASCAL 188 compiler, corresponding to the file PAS188_L.BIN delivered in the distribution disk.

User Program Area
High AutoRun priority area, it takes the first 64 KByte of user EPROM, containing the user application program.

Data File Area
It takes the first 40 KByte of user EPROM and can contain any kind of data (for example configuration data, messages, table, etc.). This area will be managed directly by the application program by memoty read operations starting from the address **F1000H**.
If, under the PASCAL 188 compiler, the User wants to read the first byte of this area, then he/she will have to type the following program line:

```
<byte>:=Mem[$F100:$0000];
```

All the programs stored in the EPROM areas are executable under the Monitor/Debugger 188 through proper commands (please see the chapter "MONITOR/DEBUGGER 188 DESCRIPTION"), one only amongst them can be executed automatically, according to the priorities described in the paragraphs "AUTORUN PROGRAM".

FLASH EPROM ORGANIZATION

The FLASH EPROM delivered in the GDOS 188 package, called work FLASH EPROM, is organized as shown in the following figure:

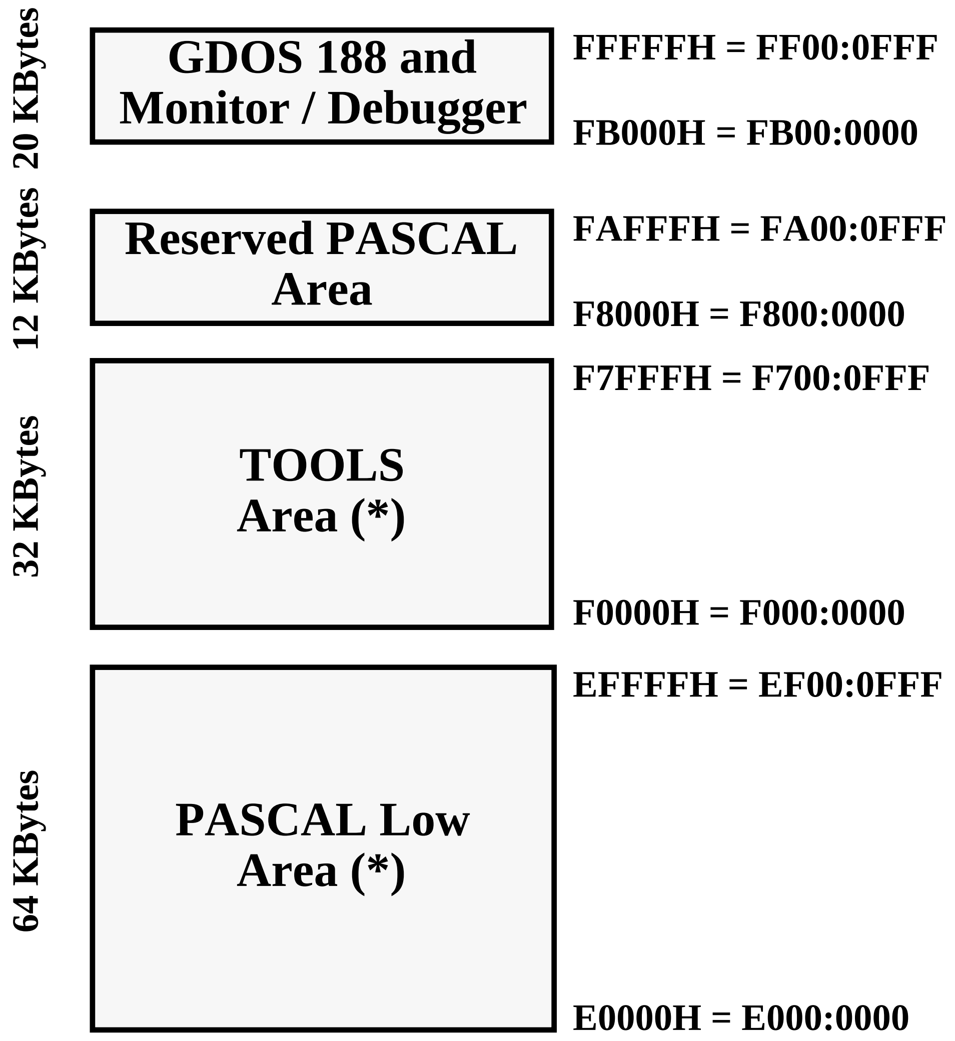

FIGURE 17: WORK FLASH EPROM ORGANIZATION

The areas marked with (*) in the previous and following figures may be overwritten by the FLASH EPROM management program, called, **WFLASH.HEX**, under the modalities described furhter. So the work FLASH EPROM can be rebuilt in any moment using the files deliveredin the distribution disk.
The remaining areas take completely the last two available 16 KByte sectors of the FLASH EPROM that, for safety reasons, are never accessed, providing an extremly high warranty for the integrity of the **GDOS 188** and Monitor/Debugger code. This is prejudicial to the amount of the Data File area that decreases from 40 to 28 KByte, certainly an acceptable value.

The two following figures report the FLASH EPROM organization in two possible configurations, obtainable through the WFLASH.HEX program; the first case is called “debug” (application program + PASCAL 188 compiler), the second case is called “User” (application program + data files).

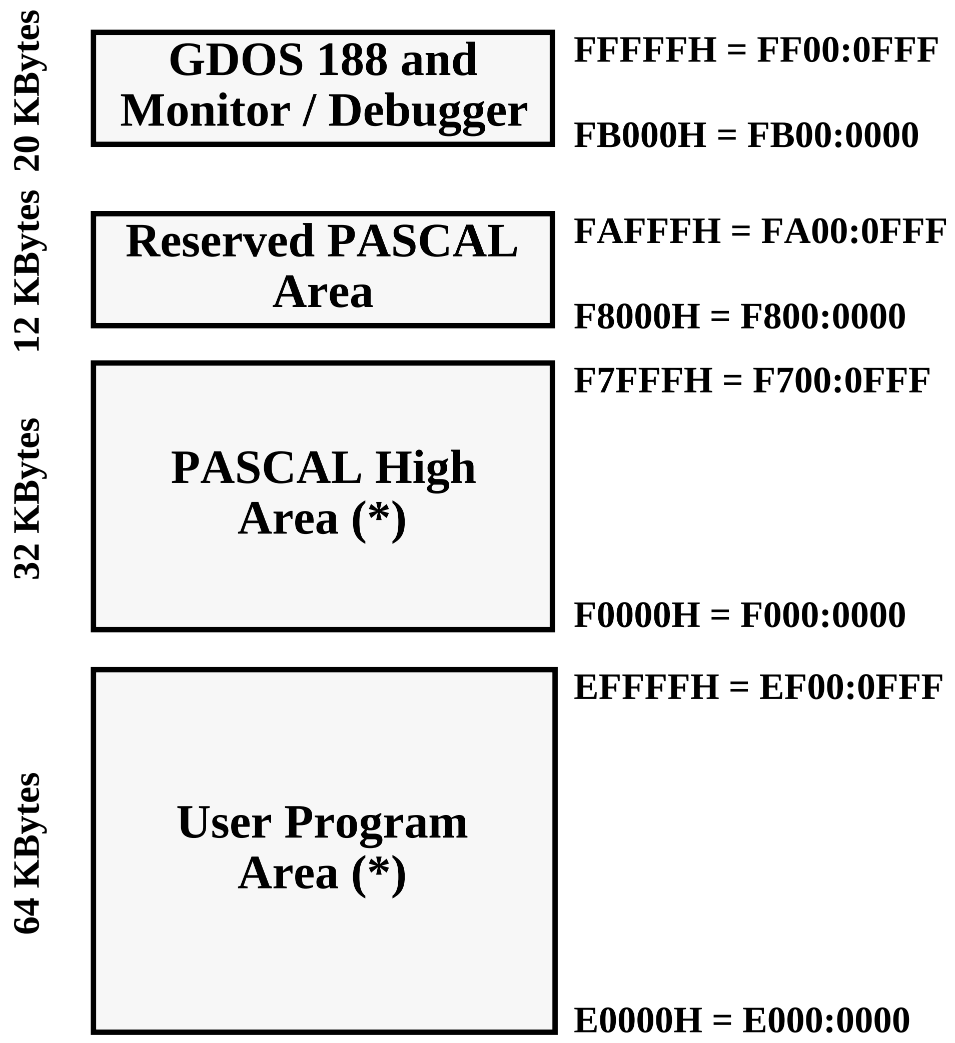

FIGURE 18: DEBUG FLASH EPROM ORGANIZATION

If no data file is employed this is the suggested configuration, because the presence of the PASCAL 188 compiler allows to debug and eventually to modify the application program, without no need to replace the device installed on the target board.

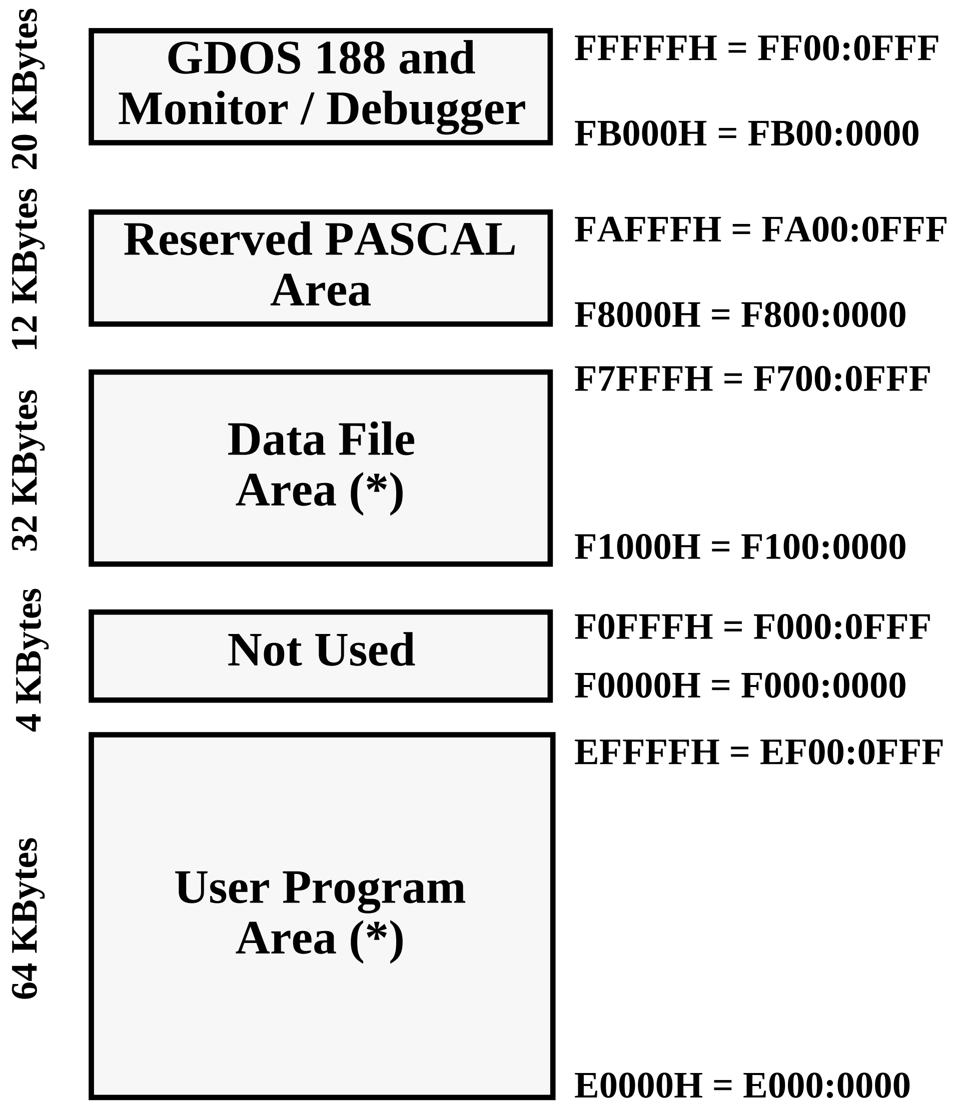

FIGURE 19: USER FLASH EPROM ORGANIZATION

GDOS 188 and Monitor / Debugger
It takes the last 20 KByte of FLASH EPROM and containes the code of the board basic firmware. This code is the one always executed when a Power On or a reset occour and allows to perform all the possible execution and / or development phases for the application program.
The content of this area corresponds to the file **MON?????.BIN** delivered in the distribution disk and, because of its importance, it cannot be modified by the User and it is protected against accidental changes.

TOOLS Area
Low AutoRun priority area, it takes 22 KByte of the work FLASH EPROM and contains the TOOLS 188 utility program, corresponding to the file **TOOLS188.BIN** delivered in the distribution disk.

PASCAL Low area
High AutoRun priority area, it takes the first 64 KByte of work FLASH EPROM and contains the PASCAL 188 compiler, corresponding to the file **PAS188_L.BIN** delivered in the distribution disk.

Reserved PASCAL Area e PASCAL High Area
Low AutoRun prioprity area destined to store PASCAL 188 compiler in its debug confuguration, it takes on the whole 44 KByte of FLASH EPROM. This area has been splitted in two parts, on of which (stored in the **Reserved PASCAL Area**) is always present in the FLASH EPROM, because is stored in the last two sectors that, as previously stated, cannot be changed by the User.
The second part is stored in the PAS188_H.BIN file and must be loaded into the user available 32 KByte area called **PASCAL High Area**.

User Program Area
High AutoRun priority area, it takes the first 64 KByte of user FLASH EPROM, containing the user application program.

Data File Area
It takes the first 28 KByte of user FLASH EPROM (differently from the first 40 KByte of EPROM) and can contain any kind of data (for example configuration data, messages, table, etc.). This area will be managed directly by the application program by memoty read operations starting from the address **F1000H**.
If, under the PASCAL 188 compiler, the User wants to read the first byte of this area, then he/she will have to type the following program line:

```
<byte>:=Mem[$F100:$0000];
```

All the programs stored in the FLASH EPROM areas are executable under the Monitor/Debugger 188 through proper commands (please see the chapter “MONITOR/DEBUGGER 188 DESCRIPTION”), one only amongst them can be executed automatically, according to the priorities described in the paragraphs “AUTORUN PROGRAM”.

FLASH EPROM MANAGEMENT

FLASH EPROM management under **GDOS 188** operating system is an aided operation by which the User, through an utility program, can modify the content of some FLASH EPROM areas, writing into them the content of files stored in any drive managed by **GDOS 188**. All these are high level operations and show help messages that help the User.

BOARD CONFIGURATION

To properly manage the FLASH EPROM, the User must verify and eventually perform the following hardware configurations:

1) Connect the serial line A (consolle) of the target board to a P.C. serial line through the communication calbe described in the paragraph "SERIAL COMMUNICATION CABLE".
2) Install an AMD29F010 (or compliant) FLASH EPROM in the socket of the target board, the FLASH EPROM must already contain **GDOS 188** and Monito/Debugger 188.
3) Configure the target board jumpers to employ a FLASH EPROM, as described in the technical manual of the board.
4) Connect the working modality selection jumper to select the Debug modality, as explained in the paragraph "RUN / DEBUG MODALITY SELECTION".

LOADING AND RUNNING AN APPLICATION PROGRAM

The following steps must be executed to correctly load and run an application program into the FLASH EPROM on the target board:

1) Run **GET188** on the development P.C. connected to the target board.
2) Select the proper configuration of the P.C. serial line (COM and Baud Rate) according to the current hardware configuration of the board through the option "Serial Port" of the "Options" menu of **GET188** program.
3) Activate the intelligent terminal emulation through the option "Terminal" of menu "Options".
4) Reset or supply the target board to execute Monitor/Debugger 188.
5) Select the sub option "HEX Intel File" in option "Transmit file to Monitor" of menu "Utility" then select file WFLASH.HEX in the diloag window that has appeared. This program is delivered in the distribution disk.
6) Wait until the file transmission is completed then press <Enter> key that makes return the Monitor/Debugger 188 prompt.
7) Give the GO command pressing the "G" key followed by these addresses:

GO FROM **0100:0000<Enter>** TILL **0100:FFFF<Enter>**

These are the default addresses, if they are already correct it is enough to confirm them pressing <Enter>.
8) Now the FLASH EPROM management program has started its execution, showing the first help screen.

FLASH EPROM MANAGEMENT PROGRAM STEPS

To correctly manage the FLASH EPROM on the target board, the User must perform a set of operations:

1) Read carefully the help screens, scrolling them by the "**N**" and "**P**" keys then continue the execution pressing <Enter>.

2) Select the requested operation pressing the matchin numeric key ("**0**" ÷ "**4**"), as asked in the menu that has appeared.

3) **Writing into the *User Program Area* or the *PASCAL Low Area***
This operation is selected pressing the "**0**" key, then the name of the file to write into this area is asked. This file must be an executable in binary format (.COM, .BIN) stored in the current directory of the current drive. After selecting the file name, the program checks for its existance and, in case it exists, the execution continues, otherwise the program asks for a new file name. If the file was found, the program checks that it does not exceed **64 KByte** of size, the memory area is cleared then replaced with the content of the selected file.
During the write phase a status message informs the User about the address currently being written, so the User has just to wait until the end of this phase then verify it.
This operatin is normally used to delete the high AutoRun priority program and replace it with the application program.

4) **Writing into the *TOOLS Area* or the *PASCAL High Area***
This operation is selected pressing the "**1**" key, then the name of the file to write into this area is asked. This file must be an executable in binary format (.COM, .BIN) stored in the current directory of the current drive. After selecting the file name, the program checks for its existance and, in case it exists, the execution continues, otherwise the program asks for a new file name. If the file was found, the program checks that it does not exceed **32 KByte** of size, the memory area is cleared then replaced with the content of the selected file.
During the write phase a status message informs the User about the address currently being written, so the User has just to wait until the end of this phase then verify it.
This operation is normally used to write the PASCAL 188 compiler into the Low AutoRun area, to be able to have a FLASH EPROM in Debug configuration.

5) **Writing into the *Data File Area***
This operation is selected pressing the "**2**" key, then the name of the file to write into this area is asked. This file can be any file format , stored in the current directory of the current drive. After selecting the file name, the program checks for its existance and, in case it exists, the execution continues, otherwise the program asks for a new file name. If the file was found, the program checks that it does not exceed **28 KByte** of size, the memory area is cleared then replaced with the content of the selected file.
During the write phase a status message informs the User about the address currently being written, so the User has just to wait until the end of this phase then verify it.
This operation is normally used to build a FLASH EPROM data area, used by the application program.

6) Help screen visualization
This operation is selected pressing the “**3**” key, it allows to read the help screen that appears when the program starts.

7) Program end
This operation is selected pressing the “**4**” key and allows to terminate the execution of WFLASH program returning to Monitor/Debugger 188.

8) In most cases during the execution it is always possible to exit the program pressing the “**ESC**” key, which will return immediatly to Monitor/Debugger 188.

9) During every phase of the program execution possible malfunctions are always checked (file system access error, FLASH EPROM clear error, FLASH EPROM write error, etc.), in case one of these problems occour a proper message will inform the User.

10) All the file names given to the program must be in the format **<drive>:<name>.<extension>**, as drive name any **GDOS 188** manageable drive name can be used, RAM Disk included.

Typical examples of use for this program are:

Work FLASH EPROM creation:
1) Select the *PASCAL Low Area* write operation and load the **PAS188_L.BIN** file, delivered in the distribution disk.
2) Select the *TOOLS Area* write operation and load the **TOOLS188.COM** file, delivered in the distribution disk.

Debug FLASH EPROM creation:
1) Select the *User Program Area* write operation and load the file obtained compiling to disk with PASCAL 188 the User application program.
2) Select the *PASCAL High Area* write operation and load the PAS188_H.BIN file, delivered in the distribution disk.

User FLASH EPROM creation:
1) Select the *User Program Area* write operation and load the file obtained compiling to disk with PASCAL 188 the User application program.
2) If the application program needs a data file, select the *Data File Area* write operation and load the applicatio program data file.

RAM DISK MANAGEMENT

The **GDOS 188** operating system manages the amount of RAM exceeding the work RAM wholly as a **RAM Disk** (**logic drive M**).
This way all the files manageable by the operating system can be read and written also to the mass storage local resources.
This implementation allows to speed up remarkably all the data load and save operations, obtaining a great time saving especially during the phase of the User application development.

Through the PASCAL compilare, it is also possible to use this memory area by the typical high level file management commands as, for example, Assign(<file>, 'M:<file name>'), Reset(<file>), Rewrite(<file>), Write(<file>, ...), Read(<file>, ...), Close(<file>), etc.

GDOS 188, during a Power On or a reset, checks the presence and the formattation status of RAM Disk, to be able to manage this section without any intervent from the User. If the RAM Disk is present but not formatted (first use or data loss), the User can format it through the specific command of TOOLS 188 program, or through several **GDOS 188** operating system function calls.
For further informations please refer to the chapters "TOOLS 188 DESCRIPTION" and "FUNCTIONS OF **GDOS 188** OPERATING SYSTEM".

Here follows several informations about the RAM Disk utilization limits that can be encountered when working with **GDOS 188**, due to the technical implementation of this operating system.

1) File names containing wild card characters (* or ?) are not managed for RAM Disk and so cannot be used.

2) After the RAM Disk formattation, the effective free RAM Disk space is equal to the desired size minus **1 KByte**, used to store disk status informations.

3) The minimun RAM Disk occupation for a file is **1 KByte**, even if the file contains a smaller amount of data.

4) If the target board power supply goes out while one or more files are open in RAM Disk, this last will be damaged and will have to be reformatted, losing any information stored in it. This problem can be avoided opening RAM Disk files only when read or write operations must be performed and closing them immediatly after the completion of these operations.

If you are intrested in some RAM Disk utilization examples, please refer to the specific example programs stored in the distribution disk, delivered with the **GDOS 188** package.

AUTORUN PROGRAM

The **GDOS 188** operating system gives the possibility to execute automatically, during the Power On or the reset, an User developed application program. This way the functionality of the system can be tested also during these phases.
A program is executed in AutoRun mode if contains a particular code sequence, as further shown; the program must be stored starting from specific addresses or it must be sabed in the local RAM Disk with the name: **AUTORUN.COM**.
GDOS 188 operating system during reset or Power On, if configured for the RUN modality as previously described, checks th presence of the AutoRun program, following a determinated priority based sequence, as explained below:

1) *Check for presence of **AUTORUN.COM** in RAM Disk*
The file with this name is executed if contains the AutoRun sequence and is not fragmented; the User, to be sure that this last condition is verified, must assure that no file has been deleted after the last RAM Disk format, or must reformat the RAM Disk before saving the program AUTORUN.COM.

2) *Check for presence of a program in the **high AutoRun priority area***
The program stored in EPROM or FLASH EPROM starting from the address
E0000H = E000:0000 is executed if contains the AutoRun sequence.

3) *Check for presence of a program in the **low AutoRun priority area***
The program stored in EPROM or FLASH EPROM starting from the address
F0000H = F000:0000 is executed if contains the AutoRun sequence.

4) *Check for presence of a program starting from the address **01000H = 0100:0000***
The program stored in stored in the **TPA** is executed starting from this address if contains the AutoRun sequence.

The AutoRun sequence is inserted automatically at the beginning of the program if it is developed using PASCAL 188 compiler; otherwise, if the User wants to develop the application program in Assembly, then he/she will have to insert as first lines of the program the following:

```
xx00 : 0000              Jmp  START
xx00 : 0004              DB   Not ("A")      ; Autorun FLAG
xx00 : 0005              DB   "A"
xx00 : 0006    START:    <Applicativo>
```

The **xx00** refer to the above reported addresses.
If none of the above mentioned conditions is true, the **GDOS 188** operating system executes th Monitor/Debugger 188.

NOTE
Refering to the previously figures, reporting the EPROM or FLASH EPROM organization, it is possible to see that, when a reset or Power On occour, the GDOS 188 operating system, if doesn't find AUTORUN.COM in RAM Disk, executes PASCAL 188 compiler (work structure) or the User application program (user or debug structure).

PERIPHERALS MAPPING

GDOS 188 operating system, when a reset or a Power On occour, sets the 80C188 internal register called **PBA** (**P**eripheral **B**ase **A**ddress) to the value **F000H**. This way the several chip select of the CPU and of the target board peripherals are configured (as explained in the target board technical manual).
The following paragraphs report the complete list of these mapping addresses, according the type of **GPC®** board useable with **GDOS 188**.

GPC® 188D PERIPHERALS MAPPING

The two following tables report the peripherals mapping addresses for the **GPC® 188D** board.

DEVICE	REG.	ADDRESS	R/W	MEANING
W.DOG	RWD	F000H	R	Watch dog retrigger
EEPROM	RE2	F000H	R/W	Serial EEPROM access
SCC 85C30	RSB	F080H	R/W	Serial line B status register
	RDB	F081H	R/W	Serial line B data register
	RSA	F082H	R/W	Serial line A status register
	RDA	F083H	R/W	Serial line A data register
DMA	DMA	F100H	R/W	Disabilitation DMA request register
ABACO® I/O BUS	IOBUS	F180H÷F1FFH	R/W	Addresses for the management of **ABACO®** I/O BUS
PPI 82C55	PA	F200H	R/W	Port A data register
	PB	F201H	R/W	Port B data register
	PC	F202H	R/W	Port C data register
	RC	F203H	R/W	Control and command register

FIGURE 20: GPC® 188D PERIPHERALS MAPPING TABLE 1

NOTE

As shown in the above table, the **ABACO® I/O BUS** section of **GPC®188D** is mapped in **128** addresses, in the range **F180H ÷ F1FFH** (addressed with **A7** signal set to logical level 1).
Because of this, if a board connected to this section uses all the 8 addressing signals **A0 ÷ A7**, it will have to be mapped into an address included in the range **128 ÷ 255** (**80H ÷ FFH**).

DEVICE	REG.	ADDRESS	R/W	MEANING
RTC	S1	F280H	R/W	Units of seconds register
72421	S10	F281H	R/W	Tenth of seconds register
	MI1	F282H	R/W	Units of minutes register
	MI10	F283H	R/W	Tenth of minutes register
	H1	F284H	R/W	Units of hours register
	H10	F285H	R/W	Tenth of hours register; AM/PM
	D1	F286H	R/W	Units of day register
	D10	F287H	R/W	Tenth of day register
	MO1	F288H	R/W	Units of month register
	MO10	F289H	R/W	Tenth of month register
	Y1	F28AH	R/W	Units of year register
	Y10	F28BH	R/W	Tenth of year register
	W	F28CH	R/W	Day of week register
	REGD	F28DH	R/W	D control and status register
	REGE	F28EH	R/W	E control and status register
	REGF	F28FH	R/W	F control and status register
WR PROT	WRP	F300H	W	Write protection register
DIP SWITCH	DSW1	F300H	R	Dip switch status register
LEDS ATTIVITA'	LED	F340H	W	Activity LEDS register

FIGURE 21: GPC® 188D PERIPHERALS MAPPING TABLE 2

For further informations and the description of the above reported registers, please see the technical manual of **GPC® 188D**.

GPC® 188F PERIPHERALS MAPPING

The two following tables report the peripherals mapping addresses for the **GPC® 188F** board.

DEVICE	REG.	ADDRESS	R/W	MEANING
W.DOG	RWD	F000H	R	Watch dog retrigger
EEPROM	RE2	F000H	R/W	Serial EEPROM access
SCC	RSB	F080H	R/W	Serial line B status register
85C30	RDB	F081H	R/W	Serial line B data register
	RSA	F082H	R/W	Serial line A status register
	RDA	F083H	R/W	Serial line A data register
DMA	DMA	F100H	R/W	Disable DMA request register
A/D	IRL0÷7	F180H÷F18EH (EVEN address)	R/W	Sequencer low instructions 0÷7 register
LM12458	IRH0÷7	F181H÷F18FH (ODD address)	R/W	Sequencer high instructions 0÷7 register
	CNTL	F190H	R/W	Low configuration register
	CNTH	F191H	R/W	High configuration register
	INTENL	F192H	R/W	Low interrput abilitation register
	INTENH	F193H	R/W	High interrput abilitation register
	INTSTL	F194H	R	Low interrupt status register
	INTSTH	F195H	R	High interrupt status register
	TMRL	F196H	R/W	Low timer register
	TMRH	F197H	R/W	High timer register
	FIFOL	F198H	R	Low FIFO conversion register
	FIFOH	F199H	R	High FIFO conversion register
	LIMSTL	F19AH	R	Low limit status register
	LIMSTH	F19BH	R	High limit status register

FIGURE 22: GPC® 188F PERIPHERALS MAPPING TABLE 1

DEVICE	REG.	ADDRESS	R/W	MEANING
PPI 82C55	PA	F200H	R/W	Port A data register
	PB	F201H	R/W	Port B data register
	PC	F202H	R/W	Port C data register
	RC	F203H	R/W	Control and command register
RTC	S1	F280H	R/W	Units of seconds register
72421	S10	F281H	R/W	Tenth of seconds register
	MI1	F282H	R/W	Units of minutes register
	MI10	F283H	R/W	Tenth of minutes register
	H1	F284H	R/W	Units of hours register
	H10	F285H	R/W	Tenth of hours register; AM/PM
	D1	F286H	R/W	Units of day register
	D10	F287H	R/W	Tenth of day register
	MO1	F288H	R/W	Units of month register
	MO10	F289H	R/W	Tenth of month register
	Y1	F28AH	R/W	Units of year register
	Y10	F28BH	R/W	Tenth of year register
	W	F28CH	R/W	Day of week register
	REGD	F28DH	R/W	D control and status register
	REGE	F28EH	R/W	E control and status register
	REGF	F28FH	R/W	F control and status register
WR PROT	WRP	F300H	W	Write protection register
DIP SWITCH	DSW1	F300H	R	Dip switch status register
LEDS ATTIVITA'	LED	F340H	W	Activity LEDS register

FIGURE 23: GPC® 188F PERIPHERALS MAPPING TABLE 2

For further informations and the description of the above reported registers, please see the technical manual of **GPC® 188F**.

GPC® 884 PERIPHERALS MAPPING

The two following tables report the complete peripherals mapping addresses for the **GPC® 188F** board.

DEVICE	REG.	ADDRESS	R/W	MEANING
ABACO® I/O BUS	IOBUS	F000H÷F0FFH	R/W	**ABACO®** I/O BUS management registers
RTC 72421	S1	F100H	R/W	Units of seconds register
	S10	F101H	R/W	Tenth of seconds register
	MI1	F102H	R/W	Units of minutes register
	MI10	F103H	R/W	Tenth of minutes register
	H1	F104H	R/W	Units of hours register
	H10	F105H	R/W	Tenth of hours register; AM/PM
	D1	F106H	R/W	Units of day register
	D10	F107H	R/W	Tenth of day register
	MO1	F108H	R/W	Units of month register
	MO10	F109H	R/W	Tenth of month register
	Y1	F10AH	R/W	Units of year register
	Y10	F10BH	R/W	Tenth of year register
	W	F10CH	R/W	Day of week register
	REGD	F10DH	R/W	D control and status register
	REGE	F10EH	R/W	E control and status register
	REGF	F10FH	R/W	F control and status register
CONF. J1	J1	F100H	R	J1 stauts acquisition register
WATCH DOG	WDOG	F600H	R/W	Watch dog retrigger egister

FIGURE 24: GPC® 884 PERIPHERALS MAPPING TABLE

For further informations and the description of the above reported registers, please see the technical manual of **GPC® 188F**.

MONITOR / DEBUGGER 188 DESCRIPTION

The **GDOS 188** package includes a powerful and versatile **Monitor/Debugger**. This software allows to perform operations on the target board hardware resources and to debug at Assembly level the application program. Sole of the possible operations are: input/output, verfy or replace the content of the memory, CPU registers management, single step execution of an application program, break points setting, etc.
When a reset or Power On occour, the **GDOS 188** operating system checks the status of the working modality selection jumper (described in the previous chapter) and cheks for the presence of the AutoRun program; if such file is not present, or the **Debug** mode is selected, the Monito/Debugger 188 is executed, displaying the following informations:

```
MONITOR 188 - GPC° ????     ff MHz      Vx.x

<Some informations about registers>

M188>
```

Here the generic indication **x.x** replaces the version number, the indication **????** replaces the name of the target board used and **ff** its frequence.
The following paragraphs reports several informations about the Monitor/Debugger 188 command format and their parameters.

PARAMETERS FORMAT

Some of the Monitor/Debugger 188 commands require one or more parameters that the User will have to type in an opportune format. The parameters can be of three different formats, in detail they can be:

EXTENDED ADDRESS - ***ssss:oooo***
Both the SEGMENT register (inicated by **ssss**) value and the OFFSET register (indicated by **oooo**) value are required.

OFFSET - ***oooo*** *or* ***SSSS:oooo***
Only the OFFSET register (idicated by **oooo**) value is required; in fact the segment register value cannot be changed even it is displayed.

BYTE or WORD - ***bb*** *or* ***wwww***
The value of a varible, given as byte (indicated by **bb**) or as word (indicated by **wwww**) is required, for example by commands like a data output, registers write, fill of a memory area, etc.

The input of above described parameters is easied by the prompting of a default value or the last value input for a particular command; this indication can be confirmed by pressing <**Enter**>, changed typing directly another value or modified pressing the <**BackSpace**> key. Also, an extended address can be totally deleted simply pressing the <**Space**> key.

INTERRUPT VECTORS

The Monitor/Debugger 188 uses, to insert the Break Points essential for the several debuggin operations, the interrupt vectors **2** and **3**. These vectors must not be used by the application program.

MONITOR / DEBUGGER 188 COMMANDS

The Monitor/Debugger 188 is provided with a set of commands that allow to perform all the typical operations available with this kind of programs; these instructions are invoked by pressing a single key when the prompt string **M188>** is displayed.
The Monitor/Debugger 188 answers with the echo of the command and, eventually, waits for the parameters input; completed this phase the requested command is executed, displaying its output, then the prompt string is displayed again.
Every command can be aborted in any moment pressing the <**Esc**> key.
Here follows a complete list of the commands, the relative key and parameters, near a shot description of the command itself.

LINE ASSEMBLER

Syntax: ***A [Start = EXTENDED ADDRESS] <Enter>***

Assembles the mnemonic codes of **8086** Assembly language directly into the memory, starting from the specified address. To exit from the line assembler and return to the Monitor/Debugger prompt string hit the <**Esc**> key.

Mnemonic codes use The mnemonic code for a FAR return is **retf**. The string manipulation mnemonic codes must specify explicitly the string data size. For example, to move strings of words (16 bits), **movsw** must be used, to move strings of bytes (8 bits), **movsb** must be used.

Jumps and calls assembling The assembler program assembles automatically the SHORT, NEAR or FAR jumps and calls, accoding to the way the destination address is typed. It is possible to override this policy putting the **near** prefix (abbreviatable with **ne**) or the **far** prefix, as shown in the following example:

```
M188> A 0100:0500
A>0100:0500   jmp  502        ; A short jump 2 bytes long
A>0100:0500   jmp  near 505  ; A near jump 3 bytes long
A>0100:0500   jmp  far    50A  ; A far jump 5 bytes long
```

Distinction between byte and word memory locations When a parameter can be both a word or a byte memory location, it is essential to specify the data parameter type using the **word ptr** or the **byte ptr** prefixes. Valid abbreviations are **wo** and **by**. For example:

```
dec  wo          [si]
neg  byte ptr    [128]
```

How to specify operands The line assembler uses the standard convention in conformity with the operands enclosed in sqare brackets refer to a memory location. This is done because the assembler cannot distinguish between an immediate operand and an operand indicating a memory location address. The following exaple shows the difference:

```
mov     ax, 21     ; Move the constant 21h into AX register
mov     ax, [21]   ; Moves the content of the memory location addressed
                   ; 21h into AX register
```

Use of pseudoinstructions The line assembler is provided with two very common and useful pseudoinstructions: db operating code that assembles bye values directly into memory and dw operating code that assembles word values. Here follow examples for both operating codes:

```
db      1, 2, 3, 4, "THIS IS AN EXAMPLE"
db      'THIS IS A DOUBLE QUOTE: " '
dw      1000, 2000, 3000, "BACH"
```

The line assembler can recognize all the indirect register indicization forms. For example:

```
add     bx, 34 [bp+2] . [si-1]
pop     [bp+di]
push    [si]
```

EXECUTION OF THE PROGRAM STORED AT E0000H

Syntax: ***B***

Starts the execution of the program stored in EPROM or FLASH EPROM in the high AutoRun priority area, corresponding to the address **E0000H** (**E000:0000**), without checking for the presence of the AutoRun sequence.
In case the EPROM or FLASH EPROM contains the **GDOS 188** software package, the PASCAL 188 compiler is executed, because, as previously stated, it is stored starting from that location.

MEMORY BLOCK COPY

Syntax:	***C***	***[Block Start = EXTENDED ADDRESS]***	***<Enter>***
		[Block End = OFFSET]	***<Enter>***
		[Destination = EXTENDED ADDRESS]	***<Enter>***

Copies the memory block beginning at “Block Start” address and ending at “Block End” address, starting from the work RAM memory location “Destination”.

Example If the User wants to copy the 64 KByte memory block, from EPROM or FLASH EPROM, delimited in the range E0000H ÷ EFFFFH (from E000:0000 to E000:FFFF), to the location 01000H (0100:0000) in work RAM, then he/she will have to type:

M188> **COPY** FROM **E000:0000<Enter>** TILL E000:**FFFF<Invio>** TO **0100:0000<Enter>**

MEMORY BLOCK DUMP

Syntax: **D** ***[Block Start = EXTENDED ADDRESS]*** ***<Enter>***
[Block End = OFFSET] ***<Enter>***

Displays (dumps) the content of the memory block delimited by the “Block Start” and “Block End” addresses.

Exapmle If the User wants to display the 32 KByte memory block in RAM Disk delimited by the addresses 20000H ÷ 27FFFH (from 2000:0000 to 2000:7FFF) the he/she will have to type:

```
M188> DISPLAY FROM 2000:0000 <Enter> TILL 2000:7FFF <Enter>
```

ONE WORD INPUT

Syntax: **E** ***[Peripheral Address = WORD]*** ***<Enter>***

One word is input from the I/O mapped peripheral, located at the specified address, then the 16 bits result is displayed.

Example If the User wants to acquire the status of PPI 8255 port A and B installed on the **GPC® 188F** or **GPC® 188D** target boards, then he/she will have to type:

```
M188> E  --->  INPUT (WORD)  F200 <Enter> =  "PB PA"
```

MEMORY BLOCK FILL

Syntax: **F** ***[Block Start = EXTENDED ADDRESS]*** ***<Enter>***
[Block End = OFFSET] ***<Enter>***
[Data = BYTE] ***<Enter>***

Fills the work RAM memory block, delimited by the “Block Start” and “Block End” addresses, with the byte specified as “Data”.

Example If the User wants to fill the 64 KByte memory block in the work RAM delimited by the addressed 01000H ÷ 10FFFH (from 0100:0000 to 0100:FFFF), then he/she will have to type:

```
M188> FILL FROM 0100:0000 <Enter> TILL 0100:FFFF <Enter> WITH 55 <Enter>
```

PROGRAM BLOCK EXECUTION

Syntax: ***G*** ***[Program Block Start = EXTENDED ADDRESS] <Enter>***
[Program Block End = EXTENDED ADDRESS] <Enter>

This command executes the memory resident program block delimited by the addresses “Program Block Start” and “Program Block End”. In detail a **Break Point** (interrupt code 3) is placed at the “Program Block End” address, replacing the code there present; if this block end parameter is omitted, then no Break Point is set and the program is executed wholly.
If the application to be executed is stored out of the work RAM it can be only executed wholly, because in this case it is not possible to place the Break Point.

NOTE When the program execution reaches the Break Point the Block End location is restored with its previous value; If, for some cause, the Break Point location is not reached, it is always possible to restore its previous value restarting the Monitor/Debugger 188, resetting the target board, and restore the original code answering **Y** to the request “**RESTORE OLD BREAK AT ssss:oooo**” that is prompted.

Example If the User wants to execute a 4Kbyte program block, stored in the work RAM and delimited by the addresses 01000H ÷ 01FFFH (from 0100:0000 to 0100:0FFF), then he/she will have to type:

M188> **GO** FROM **0100:0000 <Enter>** TILL **0100:FFFF <Enter>**

COMMANDS LIST PRINT

Syntax: ***H***

The complete list of Monitor/Debugger 188 commands is displayed, followed by a short description of the commands themselves.

ONE BYTE INPUT

Syntax: ***I*** ***[Peripheral Address = WORD]*** ***<Enter>***

One byte is input from the I/O mapped peripheral, located at the specified address, then the 8 bits result is displayed.

Example If the User wants to acquire the status of Dip Switch bank installed on the **GPC® 188F** or **GPC® 188D** target boards, then he/she will have to type:

M188> **E** ---> INPUT (BYTE) **F300 <Enter>** = "DIP"

EXECUTION OF THE PROGRAM STORED AT F0000H

Syntax: ***J***

Starts the execution of the program stored in EPROM or FLASH EPROM in the high AutoRun priority area, corresponding to the address **F0000H** (**F000:0000**), without checking for the presence of the AutoRun sequence.
In case the EPROM or FLASH EPROM contains the **GDOS 188** software package, theTOOLS 188 program is executed, because, as previously stated, it is stored starting from that location.

MEMORY BLOCK DISASSEMBLE

Syntax:	***L***	***[Block Start = EXTENDED ADDRESS]***	***<Enter>***
		[Block End = OFFSET]	***<Enter>***

Disassembles then displays, in Assembly **8086** language, the content of the memory block delimited by the addresses "Block Start" and "Block End".

Example If the User wants to display, in Assembly **8086** language, the content of the memory block delimited by the addresses 01000H ÷ 010FFH (from 0100:0000 to 0100:00FF), then he/she will have to type:

M188> **L**IST FROM **0100:0000 <Enter>** TILL 0100:**00FF <Enter>**

ONE BYTE OUTPUT

Syntax:	***O***	***[Peripheral Address = WORD]***	***<Enter>***
		[Data = BYTE]	***<Enter>***

The byte "Data" is output to the I/O mapped peripheral corresponding to the specified address.

Example If the User wants to activate the LD3 activity LED installed on **GPC® 188F** or **GPC® 188D** target boards, then he/she will have to type:

M188> **O**UTPUT (BYTE) **F340 <Enter>** WITH **01 <Enter>**

ONE STEP PROGRAM EXECUTION

Syntax: ***P*** *<**Enter**>*

One step of the program is executed, starting from the extended address determined by **CS:IP** register; if the step contains a CALL instruction, then the called procedure is executed completely; otherwise the single instruction is executed. At the completion this command displays the values of all the registers and the next instruction disassembled.

NOTE This command requires the insertion of a Break Point in the application program, so the program must be stored in thework RAM.

GDOS 188 RESTART

Syntax: ***Q***

Restarts completely the **GDOS 188** operating system, checking again the jumper status and for the presence of the AutoRun program, as if a reset or a Power On had occoured.

REGISTER DUMP OR MODIFY

Syntax: ***R*** ***[Register Name (optional)]*** *<**Enter**>*

This command displays or modifies the content of the **80c188** CPU (or derivates) registers.
If the “Register Name” parameter is omitted, then the content of all the registers will be displayed, otherwise the content of the specified register may be modified.

Example If the User wants to store the word 55AAH into the AX registern then he/she will have to type:

M188> **R**EGISTER name (<Enter> show all registers): **AX** **<Enter>** = **55AA** **<Enter>**

MEMORY DATA REPLACEMENT

Syntax: ***S*** ***[Start = EXTENDED ADDRESS]*** *<**Enter**>*

This comman allows to replace a serie of bytes in the work RAM, starting from the address specified by “Start”. After this command has been confirmed, the following informations will be displayed:

S> **ssss:oooo** = **bb** WITH

Where **ssss:oooo** is the starting location, previously typed, and **bb** is the data currently stored in it.

At this point, the following operations are possible:

<- key>	To display the previous memory location
<Enter>	To display the next memory location
[New Data = BYTE] <- key>	Replaces the data in the current location with "New Data" and displays the previous memory location.
[New Data = BYTE] <Enter>	Replaces the data in the current location with "New Data" and displays the previous memory location.
<Esc>	Return to the Monitor/Debugger prompt

NOTE After a byte replacement, a verify of the write operation is performed; in case of error Monitor/Debugger 188 presumes that the address of the failed write operation is inside an area provided with a write protectin circuitry (like RAM Disk on **GPC® 188F** e **GPC® 188D** boards). Then the program asks a confirmation before repeating the operation, which is executed managing opportunely the write protection circuitry (essential to write a byte into a proteced memory area); after the completion of this operation no more checks are performed.

Example If the memory locations 01000H and 01001H (0100:0000 and 0100:0001) of work RAM contain the byte FFH and the User wants to change these values with, respectively, 55H and AAH, then he/she will have to type:

```
M188> SUBSTITUE  AT  0100:0000  <Enter>
S>  0100:0000 = FF     WITH   55 <Enter>
S>  0100:0001 = FF     WITH  AA <Enter>
S>  0100:0002 = xx     WITH   <Esc>
```

EXECUTION OF ONE OR MORE INSTRUCTIONS

Syntax: ***T [Number of steps = WORD]*** *<**Enter**>*

Executes the number or program instructions, indicated by "Number of steps", starting from the extended address stored in the **CS:IP** registers.
At the completion of each step this command displays the values of all the registers and the next instruction disassembled.

Example If the User wants to execute three instructin of the application program, then he/she will have to type:

```
M188> TRACE  0003    <Enter>
```

ONE WORD OUTPUT

Syntax: ***U*** ***[Peripheral Address = WORD]*** ***<Enter>***
[Data = WORD] ***<Enter>***

The word "Data" is output to the I/O mapped peripheral corresponding to the specified address.

Example If the User wants to write the bytes 55H and AAH to the PPI 8255 ports A and B , then he/she will have to type:

```
M188> U ---> OUTPUT (WORD) F200 <Enter> WITH AA55 <Enter>
```

INTEL HEX FORMAT VISUALIZATION

Syntax: ***W*** ***[Block Start = EXTENDED ADDRESS]*** ***<Enter>***
[Block End = OFFSET] ***<Enter>***

This command displays,in Intel HEX format, the content of the memory block delimited by the addresses "Block Start" and "Block End".

Example If the User wants to display, in Intel HEX format, the 32 KBytes memory block in work RAM delimited by the addresses 01000H ÷ 08FFFH (frol 0100:0000 to 0100:7FFF), then he/she will have to type:

```
M188> WRITE FROM 0100:0000 <Invio> TO 0100:7FFF <Invio>
```

FILE RECEPTION

Syntax: ***X*** ***<Space Key (file type selection)>*** ***<Enter>***

This command receives a file, in **Binary** or **Intel HEX** format, from the target board consolle line and stores it in work RAM or in RAM Disk. The file type selection is performed through the <**Space**> key, then it must be confirmed by pressing <**Enter**>.

Binary format file The program is stored starting from the location indicated by **CS:IP** registers, which contain the default value **0100:0000** when a reset or a Power On occour. When the Monitor/Debugger 188 begins to reveive a binary file, it waits for the transmission of the first byte, stores it into the specified address, then waits for a Time Out of about **600 msec**, if no other character is received by this time amount, the file transfer is considered completed.

Intel HEX format file The program is stored starting from the location indicated inside the file itself; if it doesn't contain informations about the segment, then the default segment value **0100** is assumed. The Monitor/Debugger 188 end the file transfer when it receives the closing record described by the Intel Hex standard; it is also possible to abort the transfer operation in any moment pressing the <**Esc**> key.

COMMANDS SUMMARIZING TABLE

Here follows the Monitor/Debugger 188 commands summarizing table.

COMMAND	KEY	PARAMETERS
Line Assembler	A	[Start = EXTENDED ADDRESS] <Enter>
Execution Of The Program Stored At E0000H	B	----
Memory Block Copy	C	[Block Start = EXTENDED ADDRESS]<Enter> [Block End = OFFSET] <Enter> [Destination = EXTENDED ADDRESS]<Enter>
Memory Block Dump	D	[Block Start = EXTENDED ADDRESS]<Enter> [Block End = OFFSET] <Enter>
One WORD Input	E	[Peripheral Address = WORD] <Enter>
Memory Block Fill	F	[Block Start = EXTENDED ADDRESS]<Enter> [Block End = OFFSET] <Enter> [Data = Byte] <Enter>
Program Block Execution	G	[Program Block Start =EXT.ADDRESS]<Enter> [Program Block End = OFFSET] <Enter>
Comands List Print	H	----
One BYTE Input	I	[Peripheral Address = WORD] <Enter>
Execution Of The Program Stored At F0000H	J	----
Memory Block Disassemble	L	[Block Start = EXTENDED ADDRESS]<Enter> [Block End = OFFSET] <Enter>
One BYTE Output	O	[Peripheral Address = WORD] <Enter> [Dato = BYTE] <Enter>
One Step Program Execution	P	<Enter>
GDOS 188 Restart	Q	----
Register Dump Or Modify	R	[Register Name (optional)] <Enter>
Memory Data Replacement	S	[Start = EXTENDED ADDRESS] <Enter>
Execution Of One Or More instructions	T	[Number of steps = WORD] <Enter>
One WORD Output	U	[Peripheral Address = WORD] <Enter> [Dato = BYTE] <Enter>
Intel HEX Format Visualization	W	[Block Start = EXTENDED ADDRESS]<Enter> [Block End = OFFSET] <Enter>
File Reception	X	<Space Key (file type selection)> <Enter>

FIGURE 25: MONITOR/DEBUGGER 188 COMMANDS SUMMARIZING TABLE

TOOLS 188 DESCRIPTION

GDOS 188 package includes a powerfull utility program called **TOOLS 188**; the purpose of this software is to easy the management of some target board resources; in fact it is possible to format RAM Disk, copy P.C. files to RAM Disk and viceversa, rename or delete files, set the Real Time Clock, etc. Please remark that the several operations that involve a P.C. file manipulation will be possible only if the GET188.EXE program is running on th P.C. itself.
Here follows a description about how to execute TOOLS 188, the several commands available and a short explanation of their use.

EXECUTING TOOLS 188

TOOLS 188, as explained in the paragraph “EPROM OR FLASH EPROM ORGANIZATION”, is stored in the low AutoRun priority area (addressed at **F0000H = F000:0000**); to execute it is essential to follow these steps:

1) Connect serial line A(consolle) of the target board to a P.C. serial line thorough the communication cable described in the paragraph “SERIAL COMMUNICATION CABLE”.
2) Connect the working modality selection jumper to activate the Debug modality, as described in the paragraph “RUN / DEBUG MODALITY SELECTION”.
3) Execute **GET188** program on the development P.C. connected to the target board.
4) Select the opportune configuration for the P.C. serial line (Com and Baud Rate), according to the current hardware configuration of the target board, through the “Serial Port” option of the “Options” menu of **GET188** program.
5) Activate the intelligent terminal emulation through the “Terminalt” option of the “Options” menu of **GET188** program.
6) Reset or supply the target board to execute Monitor/Debugger 188.
7) Give the execution command typing “**J**”.
8) TOOLS 188 starts its execution, displaying the following informations:

```
TOOLS 188  -  Ver x.x  -  GPC° ????  ff MHz

CHANGE DRIVE
COPY FILE
DELETE FILE from M:
RENAME FILE on M:
FORMAT M:
SET RTC
EXIT

<Informations about RAM Disk M>
```

Where the generic indication **x.x** replaces the version number, the indication **????** replaces the name of the target board used and **ff** its frequence; “Informations about RAM Disk M” reports indications about the RAM Disk status (if it is present, if it is formatted, number of KByte available, etc.) and eventually the file directory.

TOOLS 188 COMMANDS

TOOLS 188 makes available several commands, that can be selected in an easy and intuitive way. In fact, through the <**Up Arrow**> and <**Down Arrow**> keys, it is possible to move the selection inside the main menu, highlighting the desired command, then confirm it pressing <**Enter**>.
Any selected operation can be aborted during the parameter input phase pressing the <**Esc**> key.
Here follows a short description of all the several commands executable and their parameters.

DRIVE SELECTION AND DIRECTORY VISUALIZATION

This option is executed selecting the **CHANGE DRIVE** command from the main menu and allows to select the drive, both a P.C drive or RAM Disk, then to display its content.
To perform this operation it is essential to specify the letter associated to the desired drive (**Enter DRIVE**); the list of the files stored in the disk current directory will be displayed.
This list is structured in pages of **45 file names** each,the pages can be scrolled through the <**Page Up**> and <**Page Down**> keys.

NOTE
This command can be executed only if the **GET188.EXE** program is running its intelligent terminal emulation on the development P.C.

FILE COPY

This option is executed selecting the **COPY FILE** command from the main menu and allows to copy one or more files between two drives, both a P.C. drive or RAM Disk.
To perform the file copy it is essential, as first, to type the letters specifing the source drive (**SOURCE DRIVE**) and the destination drive (**DEST DRIVE**); next, the current directory of the source disk will be displayed and through the <**Up Arrow**>, <**Down Arrow**>, <**Page Up**>, <**Page Down**> and <**Enter**> keys it will be possible to select and highlight all the files to be copied, then perform the copy operation pressing the <**G**> key. If a selected file is already stored into the destination disk, a confirm request will be prompted before overwriting it.

NOTE
This command can be executed only if the **GET188.EXE** program is running its intelligent terminal emulation on the development P.C., in addition to this, please remark that file copy to RAM Disk is possible only if it is present and formatted.

FILE ERASE FROM RAM DISK

This option is executed selecting the **DELETE FILE from M:** command from the main menu and allows to erase one or more files from the target board RAM Disk.
Executing this command, the current directory of the M drive (RAM Disk) will be displayed and thorugh the <**Up Arrow**>, <**Down Arrow**>, <**Page Up**>, <**Page Down**> and <**Enter**> keys it will be possible to select and highlight all the files to be erased, then perform the erase operation pressing the <**G**> key, that will prompt a confirm request.

FILE RENAME ON RAM DISK

This option is executed selecting the **RENAME FILE on M:** command from the main menu and allows to rename one file on the target board RAM Disk.
Executing this command, the current directory of the M drive (RAM Disk) will be displayed and thorugh the <**Up Arrow**>, <**Down Arrow**>, <**Page Up**>, <**Page Down**> and <**Enter**> keys it will be possible to select and highlight the file to be renamed, then perform the rename operation pressing the <**G**> key, that will prompt a confirm request.

NOTE
This command will be executed only if just one file will be selected.

RAM DISK FORMATTATION

This option is executed selecting the **FORMAT M:** command from the main menu and allows to format the target board RAM Disk, to let the User be able to employ it as a normal disk drive for all the disk operations.
Executing this command a request message will be displayed if the RAM Disk is already formatted, then the User will have to input the RAM Disk new size in KByte (**Enter KBYTE RAM Dik size (1..256):**).
This data will not have to exceed the amount of RAM that GDOS 188 has reserved to RAM Disk, otherwise an error message will be displayed.
At the end of the formattation a message reporting the result of the operation and eventual errors occoured is displayed.

REAL TIME CLOCK SETTING

This option is executed selecting the **SET RTC** command from the main menu and allows to initialize the target board's Real Time Clock time and date.
Executing this command, as first, the current date and timewill be displayed in the format:

ww dd/mm/yy

Where **ww**, **dd**, **mm**, **yy** indicate respectively day of week, day number, month and year. Next, this date may be confirmed, switching to the new time input, or a set of new values for the date may be input.

In this case the new values for day (dd), month(mm) and year (yy) will be requested in the format:

dd/mm/yy

followed by the request to input the day of week in the following format:

0	**->**	**Domenica (Su)**	**1**	**->**	**Lunedì (Mo)**	**2**	**->**	**Martedì (Tu)**
3	**->**	**Mercoledì (We)**	**4**	**->**	**Giovedì (Th)**	**5**	**->**	**Venerdì (Fr)**
6	**->**	**Sabato (Sa)**						

Next, current time will be displayed according to the format:

hh:mm:ss

Where **hh**, **mm** and **ss** indicate respectively hour, minutes and seconds. At this point it is possible to confirm the displayed informations and return to the main menu or insert new values, repeating the above described operations.

EXIT FROM TOOLS 188

This option is executed selecting the **EXIT** command from the main menu or pressing the <**Esc**> key and allows to terminate the execution of TOOLS 188, returning to Monitor/Debugger 188.

FUNCTIONS OF GDOS 188 OPERATING SYSTEM

GDOS 188 operating system is provided with several functions by which the User can interface directly to the system hardware, without having to know it in detail. Every procedure is accessible directly by the application program and has specific input and output parameters, but they can be accessed the same wy, as here explained:

1) Parameter initialization
2) Function selection loading the proper value into 80c188 **AH** register
3) Call to **3BH** or **21H** interrupt vector
4) The procedure restore automatically the application program execution

The User can use any function to develop the application program, but the following rules must be respected:

1) Procedures containing **GDOS 188** function calls must not be recursive
2) Interrupt vectors **3BH** and **21H** must not be modified
3) The 80c188 **PBA** register must not be changed

NOTE
Several **GDOS 188** functions are already used by PASCAL 188 compiler instructions like, for example, Write(...), WriteLn(...), Write(AUX,...), Write(LST,...), Assign ...., Write(<file>,...), ClrScr, ClrEol, GotoXY(...), etc.
This means that developing an application program with high level commands it is not needed to perform direct calls to **GDOS 188** functions; these function can be used with profit to develope Assembly applications, because the User doesn't have to implement the drivers to interface the target board's hardware.
The RAM Disk, RTC and EEPROM management functions are useful in any programming environment, because they allow to reduce the amount of source code and to use these devices in a program written by an high level language.

GDOS 188 operating system's functions are divided into two groups, the first group of functions is available through calls to the interrupt **3BH** , the second through calls to the interrupt **21H**.
The following pages report their complete description.

INTERRUPT 3BH FUNCTIONS

The **GDOS 188** operating system functions here described are available through calls to **interrupt 3BH**; the number indicating the requested procedure will have to be stored into **AH** register, while other tegister, specifically for each function, are use to pass the parameters.
If the User is developing the application program using Assembly language, then, to call a function, he/she will have to insert the following instructions:

```
MOV     AH, <Function Number>
MOV     .......                 ; Loading eventual other registers
INT     3BH                     ; Call to interrupt vector 3BH
```

If the User isdeveloping the application using PASCAL 188 compiler, then he/she will have to insert, for example, the following instructions:

```
type RegPack = record
                          AX,BX,CX,DX,BP,DI,SI,DS,ES,Flags : integer;
               end;

var Reg : RegPack;

.......

Reg.AX := (<Function number> shl 8) + <AL>;  { Loading AH ed AL }
Reg.?? := .......                            { Loading eventual registers }
intr( $3B , Reg);                            { Call to 3BH interrupt vector}
```

Examples of interrupt 3BH function calls are reported into the several example programs, stored in the distribution disk.
Here follows the complete list of interrupt 3BH function calls.

FUNCTION 0: KEY ACQUISITION

Input:	***AH = 0***		
Output:	***AL = <KeyCode>***	*or*	***AL = 0***
			AH = <Special KeyCode>

Waits for a key to be pressed on the keyboard of the P.C. connected to the consolle line (serial line A) of the target board.
The code of the key is returned into **AL** register, or into **AL**, **AH** registers in case of a special key. For this last case the coding is shown in the table reported in paragraph “TERMINAL EMULATION”; in detail the second code of these sequences, reported in AH, has the <u>most signigicant bit reset</u>.

FUNCTION 1: CONSOLLE LINE STATUS

Input:	***AH = 1***		
Output:	***ZERO flag***	***= Status of receiver***	***(Z = 0 -> character received)***
	CARRY flag	***= Status of transmitter***	***(C = 1 -> ready to transmit)***

This function checks the consolle line status register (serial line A) and reports the informations about receiver and transmitter, respectively in **ZERO** and **CARRY** flag.

FUNCTION 2: CONSOLLE LINE INPUT

Input:	***AH = 2***
Output:	***AL = <Data received>***

This function waits for a character from the consolle line (serial line A) of the target board and returns it into **AL** register.

FUNCTION 3: CONSOLLE LINE OUTPUT

Input:	***AH = 3***
Output:	***AL = <Data to send>***

This function sends the character stored in **AL** register to the consolle line (serial line A) of the target board .

FUNCTION 4: AUXILIARY LINE STATUS

Input:	***AH = 4***		
Output:	***ZERO flag***	***= Status of receiver***	***(Z = 0 -> character received)***
	CARRY flag	***= Status of transmitter***	***(C = 1 -> ready to transmit)***

This function checks the auxiliary line status register (serial line B) and reports the informations about receiver and transmitter, respectively in **ZERO** and **CARRY** flag.

FUNCTION 15: BELL CONSOLLE LINE OUTPUT

Input:	***AH = 15 (0FH)***
Output:	*None*

This function sends the ASCII code of "BEL" (**07**) character to the consolle line (serial line A) of the target board.

FUNCTION 16: CR+LF CONSOLLE LINE OUTPUT

Input:	***AH = 16 (10H)***
Output:	*None*

This function sends the ASCII code of "CR+LF" (**13** and **10**) character s to the consolle line (serial line A) of the target board.

FUNCTION 20: CURSOR POSITIONING

Input:	***AH = 20 (14H)***
	DH = <Row>
	DL = <Column>
Output:	*None*

This function sends, to the consolle line of the target board, the colmun and row cursor positioning codes, indicated in **DH** and **DL** registers (**0, 0** corresponds to the **Home** position).
The code sequence transmitted is the **GET188** own sequence and is reported in the paragraph “TERMINAL EMULATION”.

NOTE
This function can work properly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulation.

FUNCTION 21: CLEAR SCREEN

Input:	***AH = 21 (15H)***
Output:	*None*

This function sends, to the consolle line of the target board, the **clear screen** codes (the whole screen is deleted). The code sequence transmitted is the **GET188** own sequence and is reported in the paragraph “TERMINAL EMULATION”.

NOTE
This function can work properly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulation.

FUNCTION 22: CLEAR END OF LINE

Input:	***AH = 22 (16H)***
Output:	*None*

This function sends, to the consolle line of the target board, the **clear end of line** codes (the line is deleted from cursor position to the end). The code sequence transmitted is the **GET188** own sequence and is reported in the paragraph “TERMINAL EMULATION”.

NOTE
This function can work properly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulation.

FUNCTION 23: DELETE LINE

Input:	***AH = 23 (17H)***
Output:	*None*

This function sends, to the consolle line of the target board, the **delete line** codes (the line where the cursor finds isa deleted). The code sequence transmitted is the **GET188** own sequence and is reported in the paragraph “TERMINAL EMULATION”.

NOTE
This function can work properly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulation.

FUNCTION 24: INSERT LINE

Input:	***AH = 24 (18H)***
Output:	*None*

This function sends, to the consolle line of the target board, the **insert line** codes (a new line is inserted at cursor position). The code sequence transmitted is the **GET188** own sequence and is reported in the paragraph “TERMINAL EMULATION”.

NOTE
This function can work properly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulation.

FUNCTION 25: HIGH BRIGHTNESS

Input:	***AH = 25 (19H)***
Output:	*None*

This function sends, to the consolle line of the target board, the **highvideo** attribute codes (high brightness). The code sequence transmitted is the **GET188** own sequence and is reported in the paragraph “TERMINAL EMULATION”.

NOTE
This function can work properly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulation.

FUNCTION 26: NORMAL BRIGHTNESS

Input:	***AH = 26 (1AH)***
Output:	*None*

This function sends, to the consolle line of the target board, the **lowvideo** attribute codes (normal brightness). The code sequence transmitted is the **GET188** own sequence and is reported in the paragraph "TERMINAL EMULATION".

NOTE
This function can work properly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulation.

FUNCTION 31: ENABLE RAM DISK FORMATTATION

Input:	***AH = 31 (1FH)***
Output:	*None*

This function enables the target board RAM Disk formattation (drive **M:**).
Only after calling this function it is possible to paerform the effective formattation, calling function **32**.

FUNCTION 32: RAM DISK FORMATTATION

Input:	***AH = 32 (20H)***
	AL = <Size in KByte>
Output:	***AL = <Result of th operation>***

This function performs the target board RAM Disk formattation (drive **M:**).
The size in KByte, indicated by AL register, has the following meaning:

AL = 1÷255	->	1÷255 KBytes
AL = 0	->	256 KBytes

The **AL** register also return the result of the operation:

AL = 0	->	formattation completed correctly.
AL = 1	->	Formattation disabled (call function **31**).
AL = 2	->	Error during formattation.
AL = 3	->	Error during formattation verify.
AL = 4	->	Amount of RAM insuffîcent.

NOTE
Before attempting to format the RAM Disk, the previously described function **31** must always be called.

FUNCTION 33: RAM DISK VERIFY

Input:	***AH = 33 (21H)***
Output:	***AL = <Result of verify>***

This function performs several verifies on the target board RAM Disk (drive **M:**) and returns the result into **AL** register, in detail:

AL = 0	->	RAM Disk valid and error free.
Bit 0 AL = 1	->	RAM Disk not present or not formatted.
Bit 1 AL = 1	->	RAM Disk iDirectory inconsistent.
Bit 2 AL = 1	->	RAM Disk write operation not completed.

FUNCTION 34: EEPROM READ

Input:	***AH = 34 (22H)***
	BX = <Address where to read>
Output:	***AH = <Result of read>***
	AL = <Byte read>

This function performs a read operation of one byte from the target board EEPROM, at the address indicated by the **BX** register.
Registers **AL** and **AH** return respectively the byte read and the result of the operation, according to the following codes:

AH = 0	->	Read completed correctly.
AH = 255	->	Errore during read.

Please remark that, in case of read error, the content of **AL** is meaningless.

NOTE
This function performs no check to detect if the address to read is out of the range acceptable by the installed EEPROM, so the User must be careful not to give values over the last existing location.

FUNCTION 35: EEPROM WRITE

Input:	***AH** = 35 (23H)*
	***BX** = <Address where to write>*
	***AL** = <Byte to write>*
Output:	***AH** = <Result of write>*

This function performs a write operation of the content of **AL** register to the target board EEPROM, at the address indicated by the **BX** register.
Register **AH** returns the result of the operation, according to the following codes:

AH = 0	->	Write completed correctly.
AH = 255	->	Errore during write.

Please remark that, in case of correct write, this function also generate a **10 msec** delay, essential to store correctly the byte in the EEPROM.

NOTE
This function performs no check to detect if the address to read is out of the range acceptable by the installed EEPROM, so the User must be careful not to give values over the last existing location.

FUNCTION 36: PROTECTED RAM WRITE

Input:	***AH** = 36 (24H)*
	***ES:BX** = <Address where to write>*
	***AL** = <Byte to write>*
Output:	*none*

This function writes the byte contained in **AL** register into the write protected RAM at the address specified by the registers **ES:BX**.
It is useful especially useful when the RAM area normally destined to RAM Disk must be managed directly; in fact this area is write protected through a opportune circuitry (**GPC® 188F** and **GPC® 188D**). The function manages properly the write protection circuitry, without no need for the User to perform specific operations.

NOTE
For compatibility, this function is available in all the **GDOS 188** versions, also those which are designed for boards without a write protection circuitry.
In this case its effect is a simple write operation, so its use is pointless.

INTERRUPT 3BH FUNCTIONS SUMMARIZING TABLE

Here follows the interrupt **3BH** functions summarizong table.

FUNCTION	INPUT	OUTPUT

Key Acquisition	AH = 0	AL = <KeyCode>
		or:
		AL = 0
		AH = <Special KeyCode>

Consolle Status	AH = 1	Z = 0 (Character received)
		C = 1 (Ready to transmit)

Consolle Line Input	AH = 2	AL = <Data Received>

Consolle Line Output	AH = 3
	AL = <Data to send>
		---- 	

Auxiliary Line status	AH = 4	Z = 0 (Character received)
		C = 1 (Ready to transmit)

BELL Consolle Line Output	AH = 15 (0FH)	----
CR+LF Consolle Line Output	AH = 16 (10H)	----

Cursor Positioning	AH = 20 (14H)
	DH = <Row>
	DL = <Column>
		---- 	

Clear Screen	AH = 21 (15H)	----
Clear End of Line	AH = 22 (16H)	----
Delete line	AH = 23 (17H)	----
Insert Line	AH = 24 (18H)	----
High Brightness	AH = 25 (19H)	----
Normal Brightness	AH = 26 (1AH)	----
Enable RAM Disk Formattation	AH = 31 (1FH)	----

RAM Disk Formattation	AH = 32 (20H)
	AL = <Size (KByte)>
		AL = <Result of operation>

RAM Disk Verify	AH = 33 (21H)	AL = <Result of verify>

EEPROM Read	AH = 34 (22H)
	BX = <Address>
		AL = <Byte read>
		AH = <Result of read>

EEPROM Write	AH = 35 (23H)
	AL = <Byte to write>
	BX = <Address>
		AH = <Result of write>

Protected RAM write	AH = 36 (24H)
	AL = <Byte to write>
	ES:BX = <Address>
		---- 	

FIGURE 26: INTERRUPT 3BH FUNCTIONS SUMMARIZING TABLE

INTERRUPT 21H FUNCTIONS

The **GDOS 188** operating system functions here described are available through calls to **interrupt 21H**; the number indicating the requested procedure will have to be stored into **AH** register, while other tegister, specifically for each function, are use to pass the parameters.
If the User is developing the application program using Assembly language, then, to call a function, he/she will have to insert the following instructions:

```
MOV     AH, <Function Number>
MOV     .......                     ; Loading eventual other registers
INT     21H                         ; Call to interrupt vector 21H
```

If the User isdeveloping the application using PASCAL 188 compiler, then he/she will have to insert, for example, the following instructions:

```
type RegPack = record
                    AX,BX,CX,DX,BP,DI,SI,DS,ES,Flags : integer;
               end;

var Reg : RegPack;

.......

Reg.AX := (<Function number> shl 8) + <AL>;  { Loading AH ed AL }
Reg.?? := .......                            { Loading eventual registers }
intr( $21 , Reg);                            { Call to 21H interrupt vector}
```

Examples of interrupt 21H function calls are reported into the several example programs, stored in the distribution disk.
Here follows the complete list of interrupt 3BH function calls.

FUNCTION 0: PROGRAM EXECUTION END

Input:	***AH = 0***
Output:	*None*

Terminates the current program execution, returning to Monitor/Debugger 188.

FUNCTION 3: AUXILIARY LINE INPUT

Input:	***AH = 3***
Output:	***AL = <Data Received>***

This function waits for a character from the auxiliary line (serial line B) of the target board and returns it into **AL** register.

FUNCTION 4: AUXILIARY LINE OUTPUT

Input:	***AH = 4***
	DL = <Data to send>
Output:	*None*

This function sends the character stored in **DL** register to the auxiliary line (serial line B) of the target board .

FUNCTION 5: LOCAL PRINTER OUTPUT

Input:	***AH = 5***
	DL = <Data to print>
Output:	*None*

This function sends the character stored in **DL** register to the CENTRONICS parallel printer connected to the target board I/O connector.

FUNCTION 13: DISK BUFFER FLUSH

Input:	***AH = 13 (0DH)***
Output:	*None*

This function flushes all the current disk file buffers (RAM Disk pr P.C. disk). Please remark that this function doesn't update the directory and doesn't close the open files.

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 14: CURRENT DRIVE SELECTION

Input:	***AH = 14 (0EH)***
	DL = <Drive number>
Output:	***AL = <Number of drives installed>***

This function selects the new current drive; it also returns the number of installed drives. The disk unit to select is indicated in the **DL** register, in detail:

DL = 0	->	P.C. drive **A**
DL = 1	->	P.C. drive **B**
DL = 2	->	P.C. drive **C** (Hard Disk)
DL = 3	->	P.C. drive **D** del P.C. (Hard-Disk, CD-ROM or removable unit)
.....		
DL = 12 (0CH)	->	Drive **M** (RAM Disk)

The **AL** register returns the number of disk units installed; in case the **M** disk is selected as new current drive, **AL** returns **1**.

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 25: CURRENT DRIVE ACQUISITION

Input:	***AH = 25 (19H)***
Output:	***AL = <Current drive>***

Returns in AL the current drive code, using the standard numeric code previously shown:
drive A=**0**, drive B=**1**, drive C=**2**, drive D=**3**, ..., drive M=**12**.

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 26: SET DTA ADDRESS

Input:	***AH = 26 (1AH)***
	DS:DX = <DTA Address>
Output:	*None*

This function allows to set the **D**isk **T**ransfer **A**rea address (**DTA**), that will be used to perform the I/O file operations. The address of this memory area is specified by the **DS:DX** registers. It is essential to specify a **DTA** address before using any interrupyt **21H** function.

FUNCTION 42: DATE ACQUISITION

Input:	***AH = 42 (2AH)***
Output:	***AL = <Day of week>***
	CH = 0
	CL = <Year>
	DH = <Month>
	DL = <Day>

The current date is acquired from the target board Real Time Clock and returned into the **AL**, **CX** and **DX** registers. **DH** contains the month number (from 1 to 12); **DL** contains the day of month (from 1 to 28, 29, 30 or 31 according to the month); **CL** contains the year (from 00 to 99); **CH** always returns **0**; **AL** contains the day of the week (0 = Sunday, 1 = Monday, ... , 6 = Saturday).

FUNCTION 43: DATE SETTING

Input:	***AH = 43 (2BH)***
	AL = <Day of week>
	CL = <Year>
	DH = <Month>
	DL = <Day>
Output:	***AL = 0***

The current date is set into the target board Real Time Clock according to the content of **AL**, **CX** and **DX** registers. **DH** contains the month number (from 1 to 12); **DL** contains the day of month (from 1 to 28, 29, 30 or 31 according to the month); **CL** contains the year (from 00 to 99); **CH** always returns **0**; **AL** contains the day of the week (0 = Sunday, 1 = Monday, ... , 6 = Saturday).
This function returns **0** in **AL**.

FUNCTION 44: TIME ACQUISITION

Input:	***AH = 44 (2CH)***
Output:	***CH = <Hours>***
	CL = <Minutes>
	DH = <Seconds>

The current time is acquired from the target board Real Time Clock and returned into the **CX** and **DH** registers. **CH** contains the hour (from 0 to 23, according to the military standard time); **CL** contains the minutes (from 0 to 59); **DH** contains the seconds (from 0 to 59).

FUNCTION 45: TIME SETTING

Input:	***AH = 45 (2DH)***
	CH = <Hours>
	CL = <Minutes>
	DH = <Seconds>
Output:	***AL = 0***

The current date is set to the target board Real Time Clock according to the **CX** and **DH** registers. **CH** contains the hour (from 0 to 23, according to the military standard time); **CL** contains the minutes (from 0 to 59); **DH** contains the seconds (from 0 to 59).
This functions always returns **0** in **AL**.

FUNCTION 48: VERSION NUMBER ACQUISITION

Input:	***AH = 48 (30H)***
Output:	***AX = <Version number>***
	BX:CX = <Serial Idendification Number>

The **GDOS 188** operating system version number is returned into the **AX** register. It will always be **0001** (corresponding to the release **1.0**), for compatibility to the existing software. **BX** and **CX** registers contain the Serial Identification Number that, for the same reason, will always be **0**.

FUNCTION 54: FREE DISK SPACE ACQUISITION

Input:	***AH = 54 (36H)***
	DL = <Drive number>
Output:	***AX = <Sectors per Cluster> or AX = FFFFH (Wrong drive number)***
	BX = <Unused Clusters>
	CX = <Bytes per Sector>
	DX = <Total Clusters on disk>

Several informations about the disk unit indicated in **DL** register are reported:

DL = 0	->	Drive corrente
DL = 1	->	Drive **A** del P.C.
DL = 2	->	Drive **B** del P.C.
DL = 3	->	Drive **C** del P.C. (Hard Disk)
DL = 4	->	Drive **D** del P.C. (Hard-Disk, CD-ROM o unità removibile)
.....		
DL = 13 (0DH)	->	Drive **M** (RAM-Disk)

If an invalid disk number is given, then this function returns **FFFFH** in **AX** register. Otherwise **AX** contains the number of Sectors per Cluster (minimun assigned disk space), **CX** contains the number of bytes per Sector, **BX** contains the number of Clusters available and **DX** contains the total number of Clusters.

These values allow to calculate easily many useful data:

*CX * AX*	*= Byte per cluster*
*CX * AX * BX*	*= Total bytes free*
*CX * AX * DX*	*= Total available disk space in bytes*
*(BX * 100) / DX*	*= Percentage of free space*

If **S** is the size of a file in bytes, the number of cluster it occupies is:

*(S + CX * AX - 1) / (CX * AX)*

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 60: NEW FILE CREATION

Input:	***AH = 60 (3CH)***	
	CX = <Attribute>	
	DS:DX = <ASCIIZ file name address>	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	***AX = <File Handle>***

This function creates a new file using the specified file name. If a file having the specified name already exist then it is truncated to length zero, otherwise a new file is created using the specified name; the name must be provided as an **ASCIIZ** string (an ASCII string terminated by a **0** code). The registers pair **DS:DX** contains the address of the name string, while **CX** contains the file attributes (<u>not managed by RAM Disk</u>), according to the following list:

Bit 0 CX = 1	->	Sets the Read Only attribute
Bit 1 CX = 1	->	Sets the Hidden File attribute
Bit 2 CX = 1	->	Sets the System File attribute
Bit 5 CX = 1	->	Sets the Archive File attribute

The remaining bits are not managed and must be set to **0**.
When the function returns succesfully, the **CARRY flag** is reset (set to **0**) and **AX** register contains the **Handle** to the file created. Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX =03	->	Path not found
AX = 04	->	No Handle available
AX = 05	->	Access denied (no more space available or read inly file)

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 61: FILE OPEN

Input:	***AH = 61 (3DH)***	
	AL = <Access code>	
	DS:DX = <ASCIIZ file name address>	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	***AX = <File Handle>***

This function opens an existing file having name and path indicated; they must be provided as an **ASCIIZ** string (an ASCII string terminated by a **0** code). The registers pair **DS:DX** contains the address of the name string, while **AL** register contains a code that specifies the use of the file , in detail:

AL = 0	->	Read Only file access
AL = 1	->	Write Only file access
AL = 2	->	Read and Write file access

The remaining possible values of **AL** register are not managed and generate an error during the file open.
When the function returns succesfully, the **CARRY flag** is reset (set to **0**) and **AX** register contains the **Handle** to the file created. Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX = 02	->	File not found
AX = 03	->	Path not trovato
AX = 04	->	No Handle available
AX = 05	->	Access denied
AX = 12 (0CH)	->	Invalid access code

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 62: FILE CLOSE

Input:	***AH = 62 (3EH)***	
	BX = <File Handle>	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	***AX = <File Handle>***

This function closes the file whose Handle is stored into **BX** register. The buffer files are flushed and the directory is updated, if necessary; then when the function returns succesfully, the **CARRY flag** is reset (set to **0**). Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains the error code, that can be only AX=6 (Invalid file Handle).

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 63: READ FROM FILE

Input:	***AH** = 63 (3FH)*	
	***BX** = <File Handle>*	
	***CX** = <Number of bytes to read>*	
	***DS:DX** = <File buffer address>*	
Output:	*CARRY flag = 1 (Error)*	*CARRY flag = 0 (No Error)*
	AX = <Error Code>	*AX = <Number of bytes read>*
		DS:DX = <File buffer of read bytes address>

This function read bytes from the file matched to the Handle code stored in **BX**; **CX** register specifies the number of bytes to be read, while **DS:DX** point to the buffer where the data to be read will be stored. If the read operation successes, the **CARRY flag** is reset (set to **0**) and **AX** register contains the number of bytes read (if **AX=0** the function attempted to read over the file end). Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX = 05	->	Access denied (Write Only file)
AX = 06	->	Invalid Handle number

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 64: WRITE TO FILE

Input:	***AH** = 64 (40H)*	
	***BX** = <File Handle>*	
	***CX** = <Number of bytes to write>*	
	***DS:DX** = <File buffer address>*	
Output:	*CARRY flag = 1 (Error)*	*CARRY flag = 0 (No Error)*
	AX = <Error Code>	*AX = <Number of files written>*

This function read bytes from the file matched to the Handle code stored in **BX**; **CX** register specifies the number of bytes to be written, while **DS:DX** point to the buffer where the data to be written are stored. After the write operation the file pointer is updated to point the byte after the last written byte.If the write operation successes, the **CARRY flag** is reset (set to **0**) and **AX** register contains the number of bytes written (if this is lower than the previous value of CX then there is not enough room on disk to complete the write operation). Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX = 05	->	Access denied (Read Only file)
AX = 06	->	Invalid Handle number

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 65: FILE ERASE

Input:	***AH** = 65 (41H)*	
	***DS:DX** = <ASCIIZ file name address>*	
Output:	***CARRY flag** = 1 (Error)*	***CARRY flag** = 0 (No Error)*
	***AX** = <Error Code>*	

This function erases the file whose name is specified; the name must be provided as an **ASCIIZ** string (an ASCII string terminated by a **0** code). The registers pair **DS:DX** contains the address of the name string.
When the function returns succesfully, the **CARRY flag** is reset (set to **0**). Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX = 02	->	File not found
AX = 03	->	Path not found
AX = 05	->	Access denied (Read Only file)

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 66: MOVE FILE POINTER

Input:	***AH** = 66 (42H)*	
	***AL** = <Movement type>*	
	***BX** = <File Handel>*	
	***CX:DX** = <File pointer offset>*	
Output:	***CARRY flag** = 1 (Error)*	***CARRY flag** = 0 (No Error)*
	***AX** = <Error Code>*	***DX:AX** = <New file pointer>*

This function changes the logic file read/write position (file pointer). To call this function, **BX** must contain the file Handle, **CX** (high word) and **DX** (low word) must contain the 32 bit integer number that indicates the new file pointer position. This offset can be applied to the current file pointer in three different ways, according to the content of **AL** register:

AL = 0	->	The offset is intended from the first byte of the file and the file pointer is moved **CX:DX** byte from there.
AL = 1	->	The offset is intended from the current file pointer position.
AL = 2	->	The offset is intended from the last byte of the file.

When the function returns succesfully, the **CARRY flag** is reset (set to **0**) and returns the new file pointer position indented as distance in bytes from the first byte of the file into **DX:AX**. Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX = 01	->	Codice di spostamento, in **AL**, non valido.
AX = 06	->	Numero di handle non valido.

This function can be used for many purposes:

1) To let the file pointer point a random position; set **AL = 0** and specify into **CX:DX** this new position intended from the first byte of the file.

2) To point the end of a file set **AL = 2** and **CX:DX = 0**.

3) To calculate the current file pointer position as distance in bytes from the first byte of the file; set **AL = 1** and **CX:DX = 0**.

NOTE

Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 68: DEVICES I/O CONTROL

Input:	***AH = 68 (44H)***	
	AL = <Subfunction code>	
	[Other registers according to the selected subfunction]	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	***[Registers according to the subfunction]***

This function acts as the MS DOS 68 (44H) function; so the User can refer to its manual for further informations about use modalities and registers to employ.

NOTE

Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 71: GET CURRENT DIRECTORY

Input:	***AH = 71 (47H)***	
	DL = <Drive number>	
	DS:SI = <64 bytes buffer address>	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	***DS:SI = <Directory in ASCIIZ>***

The current directory of the specified drive is returned, the drive number is stored in **DL** register and has the meaning explained previously in function **54**: (**0** = current drive, **1** = P.C. drive **A**, **2** = P.C. drive **B**, **3** = P.C. drive **C**, etc., 1**3 (0DH)** = RAM Disk **M**). The DS:SI registers must store the address of a 64 bytes wide buffer. When the function returns succesfully, the **CARRY flag** is reset (set to **0**) and fills the buffer with an ASCII string that inidcates the current directory of the specified drive. Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains error code **15 (0FH)**.

NOTE

Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 78: TERMINATE PROGRAM EXECUTION

Input: *AH = 76 (4CH)*
Output: *None*

Like function **0**, terminates the execution of current program returning to **Monitor/Debugger 188**.

FUNCTION 78: FIND FIRST FILE IN A DIRECTORY

Input:	***AH = 78 (4EH)***	
	CX = <File attribute>	
	DS:DX = <ASCIIZ file name address>	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	***DTA = <Informations abut directory>***

This function explores a directory to find the first file having the specified name and attribute. The **DS:DX** registers must contain the address of an **ASCIIZ** string that contains the pathname of the file to find (in case of a P.C. drive both the ***** and **?** jolly characters are valid, while in RAM Disk only the *.* writing is valid). Also, the **CX** register must contain the attributes of the file to find (please see function **60** for the attributes list).

NOTE
Before calling this function it is essential to set a **DTA** area at least **43 bytes** wide calling function **26**, previously described.

If this function finds a file corresponding to the specified characteristics, it resets the **CARRY flag** (sets it to **0**) and fills the first **43 bytes** of the **DTA** with the following informations:

Byte 0÷20:	Area used by **GDOS 188** operating system for function **79** (find next file in a directory).
Byte 21:	Attributes of found file.
Byte 22÷23:	Date of found file.
Byte 24÷25:	Time of found file.
Byte 26÷29:	Size of found file..
Byte 30÷42:	File name and extension (**ASCIIZ** string).

If the function returns without success, the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX = 02	->	File not found.
AX = 03	->	Path not found.
AX = 18 (12H)	->	There are no more files; no file matching found.

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 79: FIND NEXT FILE IN A DIRECTORY

Input:	***AH = 79 (4FH)***	
	DTA = <Result of function 78>	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	***DTA = <Informations abut directory>***

This function continues the exploration of a directory, it exists because a name containing jolly characters could fine more than one matching file name. **DTA** must contain the result of a previuos call to function **78** or **79**.
If this function finds a file corresponding to the specified characteristics, it resets the **CARRY flag** (sets it to **0**) and updates **DTA** content with opportune informations. Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains error code 18 (12H): There are no more files; no file matching found.

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

FUNCTION 86: FILE RENAME

Input:	***AH = 86 (56H)***	
	DS:DX = <ASCIIZ file name address>	
	ES:DI = <ASCIIZ new file name address>	
Output:	***CARRY flag = 1 (Error)***	***CARRY flag = 0 (No Error)***
	AX = <Error Code>	

This function renames a file; it may also move a file from a directory to another. The file however is not physically copied, only the pathname is changed. So, old and new pahtnames must refer to the same drive.
The pathnames are provided as **ASCIIZ** strings. The registers pair **DS:DX** points to the old file name string, **ES:DI** point to the new file name string.
When the function returns succesfully, the **CARRY flag** is reset (set to **0**) and returns the new file pointer position indented as distance in bytes from the first byte of the file. Otherwise the **CARRY flag** is set (to **1**) and **AX** register contains one of the following error codes:

AX = 02	->	File not found
AX = 03	->	Path not found
AX = 05	->	Access denied (Read Only file)
AX = 17 (11H)	->	Files are not on the same drive

NOTE
Executing this function may require P.C. files manipulation; so it will be performed correctly only if the target board is connected, through the consolle line, to the **GET188** intelligent terminal emulator.

INTERRUPT 21H FUNCTIONS SUMMARIZING TABLE

Here follows the interrupt **21H** functions summarizong table.

FUNCTION	INPUT	OUTPUT
Program Execution End	AH = 0	----
Auxiliary Line Input	AH = 3	AL = <Data received>

Auxiliary Line Output	AH = 4
	AL = <Data to send>
		---- 	

Local Printer Output	AH = 5
	AL = <Data to send>
		---- 	

Disk Buffer Flush	AH = 13 (0DH)	----

Current Drive Selection	AH = 14 (0EH)
	DL = <Drive number>
		AL = <N. Drives installed>

Current Drive Acquisition	AH = 25 (19H)	AL = <Current drive>
Set DTA Address	AH = 26 (1AH) DS:DX = <DTA address>	----

Date Acquisition	AH = 42 (2AH)	AL = <Day of week>
		CH = 0
		CL = <Year>
		DH = <Month>
		DL = <Day>

Date Setting	AH = 43 (2BH)
	CL= <Year>
	DH = <Month>
	DL = <Day>
		AL = 0

Time Acquisition	AH = 44 (2CH)	CH = <Hours>
		CL = <Minutes>
		DH = <Seconds>

Time Setting	AH = 45 (2DH)
	CH = <Hours>
	CL = <Minutes>
	DH = <Seconds>
		AL = 0

Version Number Acquisition	AH = 48 (30H)	AX = <Version number> BX:CX = <Serial number>

Free Disk Space Acquisition	AH = 54 (36H)
	DL = <Drive number>
		AX = <Sectors per Cluster>
		BX = <Unused Clusters>
		CX = <Bytes per Sector>
		DX = <Tot. Cluster on disk>
		or:
		AX = FFFFH (Wrong drive)

FIGURE 27: INTERRUPT 21H FUNCTIONS TABLE 1

FUNCTION	INPUT	OUTPUT

New File Creation	AH = 60 (3CH)
	CX = <Attribute>
	DS:DX = <ASCIIZ file name address>
		CARRY flag = 0
		AX = <Handle>
		or:
		CARRY flag = 1
		AX = <Error code>

File Open	AH = 61 (3DH)
	CX = <Access code>
	DS:DX = <ASCIIZ file name address>
		CARRY flag = 0
		AX = <Handle>
		or:
		CARRY flag = 1
		AX = <Error code>

File Close	AH = 62 (3EH)
	BX = <Handle>
		CARRY flag = 0
		or:
		CARRY flag = 1
		AX = <Error code>

Read From File	AH = 63 (3FH)
	BX = <Handle>
	CX = <Number of bytes to be read>
	DS:DX = <File buffer address>
		CARRY flag = 0
		AX = <Handle>
		DS:DX = <Read bytes address>
		or:
		CARRY flag = 1
		AX = <Error code>

Write To File	AH = 64 (40H)
	BX = <Handle>
	CX = <Number of bytes to be read>
	DS:DX = <File buffer address>
		CARRY flag = 0
		AX = <Handle>
		or:
		CARRY flag = 1
		AX = <Error code>

File Erase	AH = 65 (41H) DS:DX = <ASCIIZ file name address>	CARRY flag = 0 or: CARRY flag = 1 AX = <Error code>

Move File Pointer	AH = 66 (42H)
	AL = <Movement type>
	BX = <Handle>
	CX:DX = <File pointer offset>
		CARRY flag = 0
		DX:AX= <New file puntatore>
		or:
		CARRY flag = 1
		AX = <Error code>

Devices I/O Control	AH = 68 (44H)
	AL = <Subfunction code>
	[Other registers according to subf.]
		CARRY flag = 0
		[Other registers for subf.]
		or:
		CARRY flag = 1
		AX = <Error code>

Get Current Directory	AH = 71 (47H)
	AL = <Drive number>
	DS:SI = <64 byte buffer address>
		CARRY flag = 0
		DS:SI= <Directory in ASCIIZ>
		or:
		CARRY flag = 1
		AX = <Error code>

FIGURE 28: INTERRUPT 21H FUNCTIONS TABLE 2

FUNCTION	INPUT	OUTPUT
Terminate Program Execution	AH = 76 (4CH)	----

Find First File In A Directory	AH = 78 (3EH)
	CX = <Attribute>
	DS:DX = <ASCIIZ file name address>
		CARRY flag = 0
		DTA= <Directory information>
		or:
		CARRY flag = 1
		AX = <Error code>

Find next File in A Directory	AH = 79 (3FH)	CARRY flag = 0 DTA= <Directory information> or: CARRY flag = 1 AX = <Error code>
File Rename	AH = 78 (3EH) DS:DX = <ASCIIZ old file name address> ES:DI = <ASCIIZ new file name address>	CARRY flag = 0 or: CARRY flag = 1 AX = <Error code>

FIGURE 29: INTERRUPT 21H FUNCTIONS TABLE 3

APPENDIX A: DIGITAL I/O INTERFACES DIAGRAM

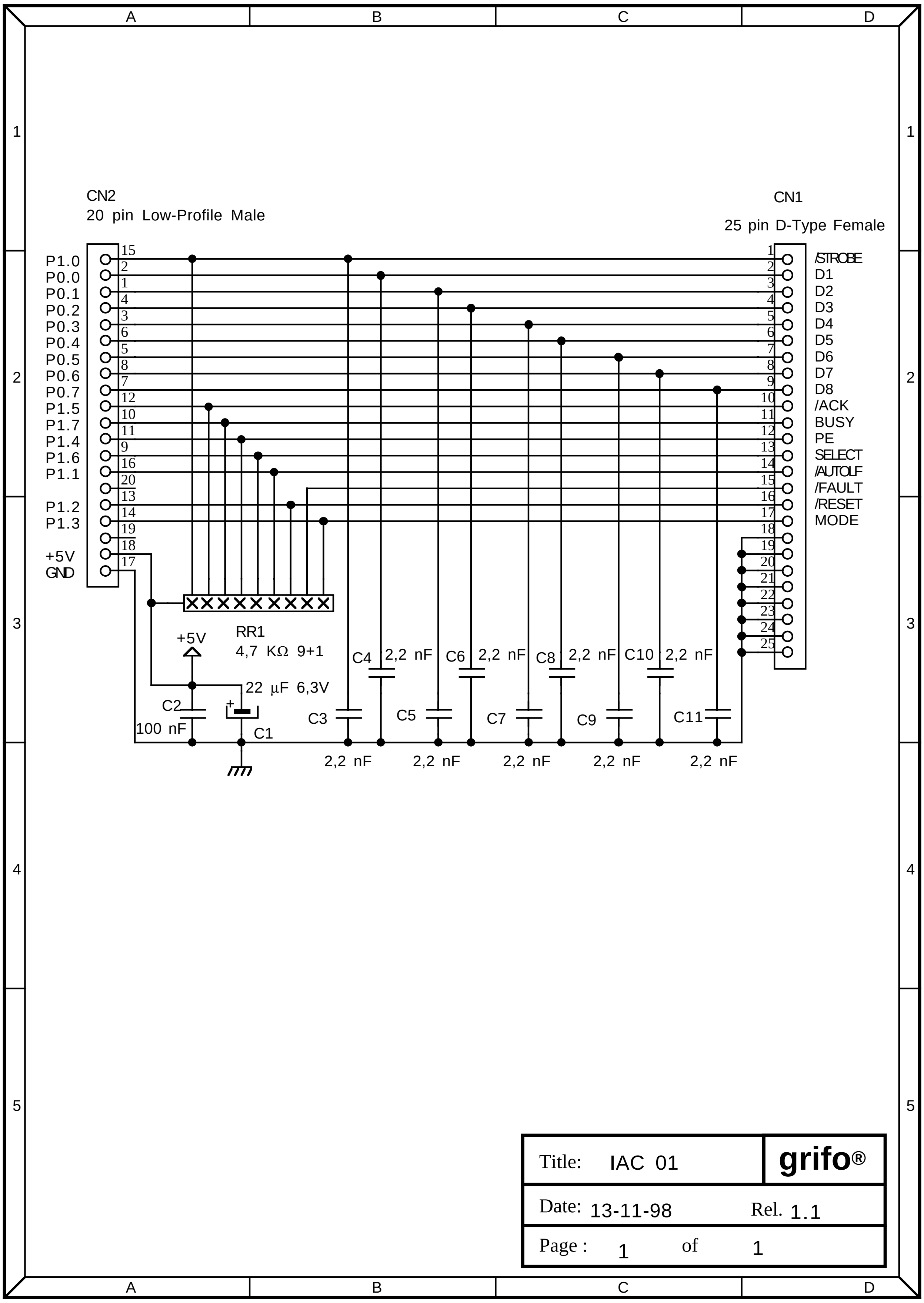

FIGURE A1: IAC 01 CARD ELECTRIC DIAGRAM

APPENDIX B: ALPHABETICAL INDEX