

GDOS

80

Grifo[®] Disk Operating System 80 family

USER MANUAL

grifo[®]
ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
Email: grifo@grifo.it



<http://www.grifo.it>

Tel. +39 051 892.052 (a.r.)

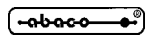
<http://www.grifo.com>

FAX +39 051 893.661

GDOS 80

Edition 5.40

Rel. 19 April 1999

 **GPC[®], grifo[®]**, are trade marks of **grifo[®]**

GDOS

80

Grifo[®] Disk Operating System 80 family

USER MANUAL

GDOS is a complete software tools, developed by **Grifo[®]**, for industrial cards. The suffix 80 denotes the version for Z80, Z180 and compatible cards.

This manual is a complete guide for all **GDOS (Grifo[®] Disk Operating System)** users.

With **GDOS 80** many high level programming languages and many debugger resources can be used on a Z80 based card through the simple addition of a standard personal computer.

grifo[®]
ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
Email: grifo@grifo.it



<http://www.grifo.it>

Tel. +39 051 892.052 (a.r.)

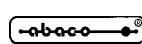
<http://www.grifo.com>

FAX +39 051 893.661

GDOS 80

Edition 5.40

Rel. 19 April 1999

, GPC[®], grifo[®], are trade marks of grifo[®]

DOCUMENTATION COPYRIGHT BY grifo[®], ALL RIGHTS RESERVED.

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, either electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written consent of **Grifo[®]**.

IMPORTANT

Although all the information contained herein have been carefully verified, **Grifo[®]** assumes no responsibility for errors that might appear in this document, or for damage to things or persons resulting from technical errors, omission and improper use of this manual and of the related software and hardware.

Grifo[®] reserves the right to change the contents and form of this document, as well as the features and specification of its products at any time, without prior notice, to obtain always the best product.

For specific informations on the components mounted on the card, please refer to the Data Book of the builder or second sources.

SYMBOLS DESCRIPTION

In the manual could appear the following symbols:

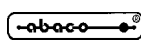


Attention: Generic danger



Attention: High voltage

Trade marks

, **GPC[®]**, **Grifo[®]** : are trade marks of **Grifo[®]**.

Other Product and Company names listed, are trade marks of their respective companies.

GENERAL INDEX

INTRODUCTION 1

RELEASES 1

GENERAL FEATURES..... 2

REQUIREMENTS 3

Z80 BASED CONTROL CARD..... 3

GDOS EPROM OR FLASH EPROM..... 4

PERSONAL COMPUTER 4

SERIAL COMMUNICATION CABLE 4

WORKING SOFTWARE 5

PROGRAMMING SOFTWARE 6

GDOS 80 USER MANUAL 6

DISTRIBUTION DISKS 7

WORKING DISK 7

ZBDEMO = CBZDEMO DISK..... 8

ZBASIC = CBZ 80 DISK 8

NSB8 DISK..... 9

PASCAL DISK 9

MODULA 2 DISK 9

ASSEMBLY DISK 10

HTC DISK 10

HOW TO START 11

GDOS 80 USE 13

GET80: P.C. DEVELOPMENT PROGRAM 13

INSTALLATION 14

EDITOR 14

MENUS AND OPTIONS DESCRIPTION 14

TERMINAL EMULATION..... 17

TERMINAL EMULATION COMMANDS 17

TERMINAL EMULATION CONTROL SEQUENCE..... 18

TERMINAL EMULATION SPECIAL KEY 19

USER STRING 19

GROM 20

DOS2GDOS: P.C. FILE FORMAT TRANSFORMATION PROGRAM..... 20

WATCH DOG 21

GDOS 80 DIRECT COMMANDS 21

GDOS 80 UTILITY PROGRAM..... 22

GDOS 80 UTILITY PROCEDURES..... 24

FORMAT RAM DISK DRIVES 24

BLOCK READ,WRITE SERIAL EEPROM 24



| | |
|---|----|
| LOCAL OPERATOR INTERFACE LEDS ACTIVATION | 25 |
| RAM ROM DISK | 28 |
| AUTORUN PROGRAM | 30 |
| EPROM AND FLASH EPROM PROGRAMMING | 30 |
| GROM: EPROM PROGRAMMING | 31 |
| FGROM.G80: FLASH EPROM PROGRAMMING | 33 |
| DIGITAL I/O INTERFACE | 34 |
| LOCAL PRINTER | 35 |
| MCI 64 RAM DISK | 35 |
| LOCAL OPERATOR INTERFACE | 36 |
| CONFIG.*: GDOS 80 CONFIGURATION | 42 |
| | |
| GDOS 80 TECHNICAL FEATURES | 44 |
| | |
| GDOS 80 VERSIONS | 45 |
| | |
| GDOS 80 DATA STRUCTURES | 46 |
| F.C.B. (FILE CONTROL BLOCK) | 46 |
| T.P.A. (TRANSIENT PROGRAM AREA) | 47 |
| IOBYTE AND IOBYTE EXTENSION | 48 |
| MMU SETTING | 50 |
| DISKS REDIRECTION | 51 |
| | |
| GDOS 80 FUNCTIONS | 52 |
| FUNCTION 0: SYSTEM RESET | 52 |
| FUNCTION 1: PRIMARY CONSOLE INPUT | 52 |
| FUNCTION 2: PRIMARY CONSOLE OUTPUT | 53 |
| FUNCTION 3: AUXILIARY CONSOLE INPUT | 53 |
| FUNCTION 4: AUXILIARY CONSOLE OUTPUT | 53 |
| FUNCTION 5: PRINTER OUTPUT | 53 |
| FUNCTION 6: PRIMARY CONSOLE DIRECT I/O | 54 |
| FUNCTION 7: GET IOBYTE | 54 |
| FUNCTION 8: SET IOBYTE | 54 |
| FUNCTION 9: PRINT STRING ON PRIMARY CONSOLE | 55 |
| FUNCTION 10: READ CONSOLE BUFFER FROM PRIMARY CONSOLE | 55 |
| FUNCTION 11: GET PRIMARY CONSOLE STATUS | 56 |
| FUNCTION 12: RETURN VERSION NUMBER | 56 |
| FUNCTION 13: RESET DISK SYSTEM | 56 |
| FUNCTION 14: SELECT DISK | 56 |
| FUNCTION 15: OPEN FILE | 57 |
| FUNCTION 16: CLOSE FILE | 57 |
| FUNCTION 17: SEARCH FOR FIRST | 58 |
| FUNCTION 18: SEARCH FOR NEXT | 58 |
| FUNCTION 19: DELETE FILE | 58 |
| FUNCTION 20: READ SEQUENTIAL | 59 |
| FUNCTION 21: WRITE SEQUENTIAL | 59 |
| FUNCTION 22: MAKE FILE | 60 |
| FUNCTION 23: RENAME FILE | 60 |
| FUNCTION 24: RETURN LOG IN VECTOR | 60 |

FUNCTION 25: RETURN CURRENT DISK 61
FUNCTION 26: SET DMA ADDRESS 61
FUNCTION 27: GET ALLOCATION ADDRESS 61
FUNCTION 28: WRITE PROTECT DISK 62
FUNCTION 29: GET READ ONLY VECTOR 62
FUNCTION 30: SET FILE ATTRIBUTES 62
FUNCTION 31: GET D.P.B. ADDRESS 63
FUNCTION 32: SET OR GET USER CODE 63
FUNCTION 33: READ RANDOM 63
FUNCTION 34: WRITE RANDOM 64
FUNCTION 35: COMPUTE FILE SIZE 64
FUNCTION 36: GET RANDOM RECORD 65
FUNCTION 37: RESET DRIVE 66
FUNCTION 40: WRITE RANDOM WITH ZERO FILL 66

APPENDIX A: LOCAL OPERATOR INTERFACE DIAGRAM A-1

APPENDIX B: ALPHABETICAL INDEX B-1

FIGURE INDEX

| | |
|---|-----|
| FIGURE 1: SERIAL CABLE BETWEEN P.C. AND TARGET CARD 16 PINS LOW PROFILE CONNECTOR | 4 |
| FIGURE 2: SERIAL CABLE BETWEEN P.C. AND TARGET CARD DB9 MALE CONNECTOR | 4 |
| FIGURE 3: SERIAL CABLE BETWEEN P.C. AND TARGET CARD DB25 FEMALE CONNECTOR | 5 |
| FIGURE 4: SERIAL CABLE BETWEEN P.C. AND TARGET CARD 6 PINS PLUG CONNECTOR | 5 |
| FIGURE 5: SERIAL CONNECTOR AND ACCESSORY | 5 |
| FIGURE 6: GET80 TERMINAL EMULATION CONTROL SEQUENCES | 18 |
| FIGURE 7: GET80 TERMINAL EMULATION REPRESENTATION ATTRIBUTES | 18 |
| FIGURE 8: GET80 TERMINAL EMULATION SPECIAL KEYS CODE | 19 |
| FIGURE 9: UTILITY PROCEDURES ADDRESSES | 24 |
| FIGURE 10: QTP 24P LEDs LOCATION | 27 |
| FIGURE 11: LOCAL DISK DIMENSIONS | 28 |
| FIGURE 12: DIGITAL I/O INTERFACE CONNECTION | 34 |
| FIGURE 13: QTP 16P KEYS NUMERATION | 38 |
| FIGURE 14: QTP 24P KEYS NUMERATION | 38 |
| FIGURE 15: KDX x24 KEYS NUMERATION | 38 |
| FIGURE 16: QTP 24P KEYS LOCATION | 39 |
| FIGURE 17: QTP 16P KEYS LOCATION | 40 |
| FIGURE 18: KEYS ADDRESSES AND DEFAULT CODES | 41 |
| FIGURE 19: DEFAULT GDOS 80 CONFIGURATION | 42 |
| FIGURE 20: FILE CONTROL BLOCK | 46 |
| FIGURE 21: WORK AREA MEMORY MAP | 47 |
| FIGURE 22: DISKS REDIRECTION | 51 |
| FIGURE 23: PRIMARY CONSOLE INPUT BUFFER | 55 |
| FIGURE 24: GDOS FUNCTION SUMMARY (1 ST PART) | 67 |
| FIGURE 25: GDOS FUNCTION SUMMARY (2 ND PART) | 68 |
| FIGURE A-1: KDX x24 ELECTRIC DIAGRAM | A-1 |
| FIGURE A-2: QTP 24P ELECTRIC DIAGRAM (1 ST PART) | A-2 |
| FIGURE A-3: QTP 24P ELECTRIC DIAGRAM (2 ND PART) | A-3 |
| FIGURE A-4: QTP 16P ELECTRIC DIAGRAM | A-4 |
| FIGURE A-5: QTP 16P ELECTRIC DIAGRAM | A-5 |

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 Builder, 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 environment, 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 builder recommendations and the actual safety and health norms.

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 software 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.

Grifo® does not warrant that the operation of the software satisfy the customer requests, that it shall be uninterrupted or error free and that the eventual errors will be corrected. **Grifo®** will not be liable for damages resulting from loss of data, profits, use of products, or incidental or consequential damages, even if advised of the possibility thereof; furthermore any problem caused by changes in personal computer, hardware and operating systems are not solved by **Grifo®**.

RELEASES

This handbook make reference to **GDOS 80** firmware release **4.6** and following ones. The validity of the information contained in this manual is subordinated to the firmware release number, so the user must always verify the correct correspondence between the notations. On the target board, the firmware release number is written on the label stuck on the EPROM or FLASH EPROM or it can be obtained running the operating system. Some others programs are included in **GDOS 80**: each one has an own version that, when necessary, is reported in this documentation. For example the version of the communication program **GET80** described in this manual is the **3.2**. A following chapter describes all the available version of **GDOS 80**, so please read it if further information are necessary.

GENERAL FEATURES

GDOS 80 is a software tools developed for all **Grifo**® industrial cards based on the wide Z80 family of microprocessor.

This software tools allows programming of these cards through high level environments that doesn't require a closely knowledge of the used hardware. With **GDOS 80**, the user employes high level programming languages and many other resources, not available in other different software tools.

It is possible to develop, debug and install any kind of user programs obtaining complete applications in a fast and comfortable way.

GDOS 80 is a simple and efficient operating system for industrial application. It derives from famous Digital Research CP/M operating system and it has similar command and similar programming mode. As well as all the operating systems, **GDOS 80** interfaces hardware to the user, so this one never must develop low level firmware but he can directly use hardware with high level instructions, commands and languages. The power of **GDOS 80** is the result of two features: the possibility to choose between many programming environments (theoretically all CP/M software) and the possibility to use the hardware resource of an external personal computer to simplify the develop and debug phases.

For example **GDOS 80** manages the following hardware resource: serial lines, printers, mass memory devices, user interfaces that can whole be used by programmers through simple instructions of the selected programming language. Normally the operating system uses a little space of memory and it shares the use of one serial line. If this features are not acceptable, there are some hardware solution that solve the problems (for further information, please contact **Grifo**®).

GDOS 80 is composed by some subsystems that create a comfortable develop environment provided of: notes, guides, on line help, examples, etc. Each one of these subsystem has an own version number that must be specified in each contact to **Grifo**® technical staff. In this manual there is a complete description of **GDOS 80** in all its parts, dividing the information according to their use and meaning. As for all software packages also **GDOS 80** evolves and change in the time. Please verify if the file README.ENG is saved on the received disk and if it is you must print it, read it and enclose it to this manual. In README.ENG file you'll find information on addition and improvements not still described in this manual.

REQUIREMENTS

Here follows a brief description of the necessary material (hardware and software) to operate with **GDOS 80**.

Z80 BASED CONTROL CARD

It is the electronic card that belong to **Grifo®** industrial set, based on Z80, Z180, HD64180 and compatible microprocessors, as **GPC® 80F; GPC® 81F; GPC® 011; GPC® 15A; GPC® 15R; GPC® 153; GPC® 183; GPC® 154; GPC® 184;**etc.

N.B. In this manual the indication "**target card**" is always used to refer to one of the over reported cards.

Independently from the application to develop, the target card must be provided of:

EPROM or FLASH EPROM with **GDOS 80** operating system;
at least 64K bytes of RAM;
one asynchronous RS 232 serial line;

The above features list is the minimum work structure, in fact the same system could be expanded, increasing his potentiality. In fact the **GDOS 80** is capable to manage, always with an high level interface, the following target card sections:

- 1) 1 console serial line = necessary any time **GDOS 80** is used also for the application develop and debug phases.
- 2) 1 auxiliary serial line = available for an high level characters reception and transmission with a predefined physical communication protocol.
- 3) 14 TTL I/O lines = managed at high level to drive a CENTRONICS parallel printer (connected for example with **IAC 01** or **DEB 01** modules).
- 4) EPROM or FLASH EPROM that exceed the 16K bytes = managed at high level as a ROM disk drive, where data are organized in read only files.
- 5) RAM that exceed the 64K bytes = managed at high level as a RAM disk drive, where data are organized in read and write files.
- 6) on board serial EEPROM = managed at high level as a buffer where data can be loaded and saved in a byte form.
- 7) 16 TTL I/O lines = managed at high level to drive a RAM CARD PCMCIA (interfaced with **MCI 64**), as a large RAM DISK drive, where data are organized in read and write files.
- 8) 16 TTL I/O lines = managed at high level to drive a local operator interface panel (for example **QTP 24P, KDL x24, KDF 224, QTP 16P**, etc) equipped with alphanumeric display, LEDs and matrix keyboard

The system configuration must be chosen according to requirements of the application to develop.

GDOS EPROM OR FLASH EPROM

The operating system code is always provided on EPROM (**GDOS**) or FLASH EPROM (**FGDOS**) ready to be mounted on the used target card. If **GDOS 80** and remote card are received at the same time the memory device is provided already mounted on the card.

On the EPROM or FLASH EPROM label are reported all the information regarding the target card, the operating system version and the eventual ROM DISK content, saved on the same device.

PERSONAL COMPUTER

An IBM personal computer (or compatible system) provided of: a 3 1/2" floppy disk drive, at least 640K RAM, one RS 232 serial line with V24 standard, Microsoft MS-DOS operating system ver. ≥ 3.3 . An hard disk is not necessary but recommended to speed all the files management operations. The personal computer is not essential but it is normally used in fact it allows to develop and debug the user application program.

In detail the P.C. connected to the target card through a serial line, work as a terminal and the user can employ it to interact with **GDOS 80** (executed on target card) and exploits its hardware resource as memory mass device, printer, etc in a transparent mode.

SERIAL COMMUNICATION CABLE

If the target card console serial line is used to develop and debug the application program, the console device must be connected properly. The target card console serial line always coincides with the serial line A of the used board and the **GDOS 80** require both the reception and transmission signals (TxD and RxD) and the handshake signals (/CTS and /RTS). So the connection must satisfy the V24 standard rules of C.C.I.T.T.

If the console device clashes with the personal computer, the connection is performed by an overturned serial cable (DTE <->DCE), described in the following figures:

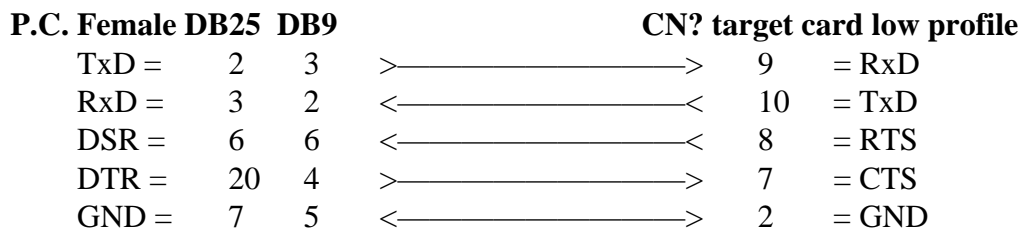


FIGURE 1: SERIAL CABLE BETWEEN P.C. AND TARGET CARD 16 PINS LOW PROFILE CONNECTOR

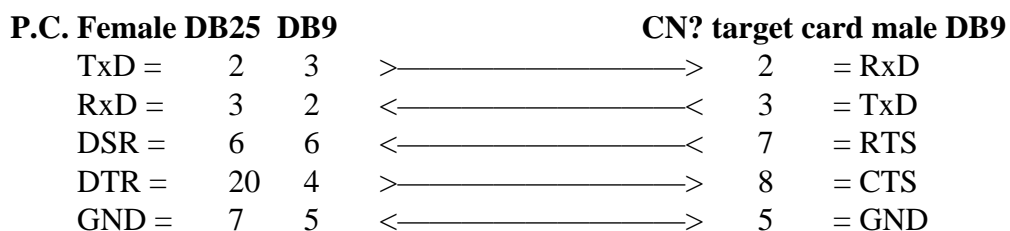


FIGURE 2: SERIAL CABLE BETWEEN P.C. AND TARGET CARD DB9 MALE CONNECTOR

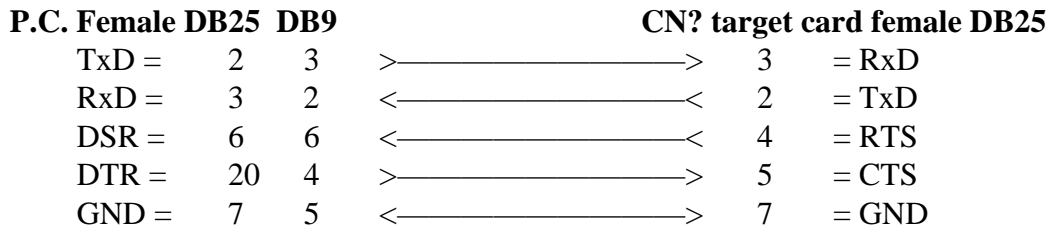


FIGURE 3: SERIAL CABLE BETWEEN P.C. AND TARGET CARD DB25 FEMALE CONNECTOR



FIGURE 4: SERIAL CABLE BETWEEN P.C. AND TARGET CARD 6 PINS PLUG CONNECTOR

The "CN? target card ..." designations refer to Grifo® serial communication standard connectors, described on the target card technical manual. The following table describes these connectors name and the accessory code (cable, cards, etc.) that Grifo® can supply to speed and facilitate the connection steps.

| TARGET CARD | CONNECTOR | SERIAL CONNECTION ACCESSORY CODE |
|-------------|-----------|--|
| GPC® 80F | CN2 | FLT 16+16; NCS 01; CCR 25+25 or CCR 25+9 |
| GPC® 81F | CN1 | FLT 16+16; NCS 01; CCR 25+25 or CCR 25+9 |
| GPC® 15A | CN4 | FLT 16+16; NCS 01; CCR 25+25 or CCR 25+9 |
| GPC® 011 | CN6 | CCR 25+25 or CCR 25+9 |
| GPC® 15R | CN7 | CCR 9+25 or CCR 9+9 |
| GPC® 153 | CN3A | CCR.PLUG25 or CCR.PLUG9 |
| GPC® 154 | CN3A | CCR.PLUG25 or CCR.PLUG9 |
| GPC® 183 | CN7A | CCR.PLUG25 or CCR.PLUG9 |
| GPC® 184 | CN3A | CCR.PLUG25 or CCR.PLUG9 |

FIGURE 5: SERIAL CONNECTOR AND ACCESSORY

WORKING SOFTWARE

By using the Personal Computer to develop and set up the application program are necessary a list of programs and files saved on the GDOS 80 distribution disk. Among these we can remember the intelligent terminal emulation program (GET80.EXE), the GDOS 80 utility programs (DIR.G80, ERA.G80, REN.G80, etc.), the FLASH EPROM writer program and the GDOS 80 operating system file.

For further information on this software list, please read the chapter "GDOS USE".

PROGRAMMING SOFTWARE

This is the software that must be used to write and test the application program to develop. In details the programming software includes all the programming languages (BASIC interpreter and compiler, PASCAL, FORTRAN, C, ASSEMBLY, MODULA 2, etc), the check and test programs (monitor, debugger, etc.) and the editor, compiler, assembler and linker used to obtain the executable version of the program.

The programming software must be ordered separately and it is supplied with the proper technical manual and a disk with some demo programs.

GDOS 80 USER MANUAL

It is this manual and it reports all the technical information on **GDOS 80** operating system. In detail you can find description on: hardware connection, command syntax, function interface, utility program and procedure, memory use, etc.

DISTRIBUTION DISKS

The **GDOS 80** software package includes an EPROM or FLASH EPROM mounted on the target card and a disk set that contain the necessary working software structure.

The following paragraphs briefly describe the received disks content; please remember that all the file with **.G80** extension are **GDOS 80** executable program that can be executed on the target card. Only the most frequently used software tools will be described, but please don't forget that some others programming languages are available (LISP, FORTH, C, etc.) and if further info are necessary, ask them directly to **Grifo®**.

WORKING DISK

It contains all the working software and some utility programs and documentation files. It is divided in some directories to simplify and to better organize the complete structure. As recommend in the chapter "HOW TO START" all the files of all the directory must be copied into a single work directory.

A brief description of the working disk files and directories is below reported; for further info on their use and meaning, please refer to following chapter:

Main root:

It contains the **GDOS 80** most frequently used programs, or on the other hand the main working structure:

| | | |
|--------------|----|--|
| GET80.EXE | -> | Interaction program for GDOS 80 , executed on P.C. |
| GET80.HLP | -> | On line help file of GET80.EXE program. |
| DOS2GDOS.COM | -> | File format transformation program, executed on P.C. |
| DIR.G80 | -> | Utility program executed on target card to obtain a disk files list. |
| ERA.G80 | -> | Utility program executed on target card to delete a file. |
| REN.G80 | -> | Utility program executed on target card to rename a file. |
| COPY.G80 | -> | Utility program executed on target card to copy a file. |
| FORMAT.G80 | -> | Utility program executed on target card to format a target card drive. |
| LEGGIMI.ITA | -> | Last minute documentation changes (Italian version) |
| README.ENG | -> | Last minute documentation changes (English version) |

Z80_EMUL directory :

It contains a powerfull and comfortable program executed on P.C., under MS DOS control, that emulates the CP/M operating system and the Z80 microprocessor.

| | | |
|-----------|----|--|
| Z80MU.EXE | -> | It is a CP/M operating system simulation program for P.C. It can execute all the GDOS 80 command files but the accepted file name extension is .COM instead of .G80 (to execute .G80 programs under Z80MU control, they must be simply renamed). For example it is important to improve experience on available programming languages. Z80MU is provided of on line help and shows on P.C. monitor the command list and their proper syntax. |
|-----------|----|--|

ROM_DISK directory :

It includes the files and programs used to save in EPROM or FLASH EPROM the user application program; this directory contents vary according to acquired **GDOS 80** version (**GDOS** or **FGDOS**) and some of the following files will be available:

- GDO???XX.BIN -> **GDOS 80** operating system binary image for **GPC® ???** card, version XX. It must be used to create the final work EPROM and it is available only in **GDOS 80** EPROM version.
- GHEX2.COM -> Binary file to HEX Intel file transformation program. It must be used when the used EPROM burner is not capable to load binary files and it is available only in **GDOS 80** EPROM version.
- FGROM.G80 -> Target card FLASH EPROM writing program. It saves a ROM DISK into the FLASH EPROM, it is executed on target card and it is available only in **FGDOS 80** FLASH EPROM version.

ZBDEMO = CBZDEMO DISK

It contains a demo version of the efficient ZBASIC = CBZ 80 BASIC compiler that is one of the **GDOS 80** high level programming language.

- CBZDEMO.G80 -> Demo version of BASIC compiler programming language.
- CBZ_80.HLP -> On line help file used by CBZDEMO.
- LEGGIMI.ITA -> Documentation file for fast CBZDEMO use.
- \PC_CROSS*. * -> Directory that includes a complete P.C. cross compiler system for CBZDEMO and **GDOS 80**. The proper documentation file LEGGIMI.ITA describes the operating mode of this cross compiler.
- \ESGPC??? *. * -> Directories that include some demo programs written in CBZDEMO BASIC compiler, in source and executable form. Each directory can have some subdirectories with other general purpose demo programs (i.e. **DEB 01**, TELECONTROLLO, etc.); all these programs can be used to directly manage the on board hardware features.
- \INTERFOP *. * -> Directory with demo programs that use local operator interface driven by **GDOS 80**, written in CBZDEMO.

ZBASIC = CBZ 80 DISK

It contains the efficient ZBASIC = CBZ 80 BASIC compiler that is one of the **GDOS 80** high level programming language.

- ZBASIC.G80 -> BASIC compiler programming language.
- ZBASIC.HLP -> On line help file used by ZBASIC.
- \ESGPC??? *. * -> Directories that include some demo programs written in ZBASIC compiler, in source and executable form. Each directory can have some subdirectories with other general purpose demo programs (i.e. **DEB 01**, TELECONTROLLO, etc.); all these programs can be used to directly manage the on board hardware features.
- \INTERFOP *. * -> Directory with demo programs that use local operator interface driven by **GDOS 80**, written in ZBASIC.

NSB8 DISK

It contains the interesting NSB8 BASIC interpreter that is one of the **GDOS 80** high level programming language.

| | | |
|---------------|----|--|
| NSB8.G80 | -> | BASIC interpreter programming language. |
| BASYM.G80 | -> | Utility program for NSB8 source program: it extracts symbol names. |
| BASTRA.G80 | -> | Utility program for NSB8 source program: it reduces program size. |
| BASYM.DOC | -> | Documentation file for BASYM.G80. |
| BASTRA.DOC | -> | Documentation file for BASTRA.G80. |
| \ESGPC???*.* | -> | Directories that include some demo programs written in NSB8 interpreter, in source and executable form. Each directory can have some subdirectories with other general purpose demo programs (i.e. DEB 01 , TELECONTROLLO, etc.); all these programs can be used to directly manage the on board hardware features. |

PASCAL DISK

It contains the efficient and comfortable PASCAL compiler that is one of the **GDOS 80** high level programming language.

| | | |
|---------------|----|---|
| PASCAL.G80 | -> | Pascal compiler programming language. |
| COMPILER.OVR | -> | Add on program for PASCAL.G80, to manage overlay. |
| COMPILER.MSG | -> | Add on program for PASCAL.G80, with displayed messages. |
| \ESGPC???*.* | -> | Directories that include some demo programs written in PASCAL compiler, in source and executable form. Each directory can have some subdirectories with other general purpose demo programs (i.e. DEB 01 , TELECONTROLLO, etc.); all these programs can be used to directly manage the on board hardware features. |
| \INTERFOP*.* | -> | Directory with demo programs that use local operator interface driven by GDOS 80 , written in PASCAL. |

MODULA 2 DISK

It contains the efficient and comfortable MODULA 2 compiler that is one of the **GDOS 80** high level programming language.

| | | |
|---------------|----|---|
| M2.G80 | -> | MODULA 2 compiler programming language. |
| M2.OVR | -> | Add on program for M2.G80, to manage overlay. |
| ERRMSG.S.OVR | -> | Add on program for M2.G80, with displayed messages. |
| SYSLIB.LIB | -> | MODULA 2 base library. |
| *.CMD | -> | MODULA 2 programming modules. |
| \ESGPC???*.* | -> | Directories that include some demo programs written in MODULA 2 compiler, in source and executable form. Each directory can have some subdirectories with other general purpose demo programs (i.e. DEB 01 , TELECONTROLLO, etc.); all these programs can be used to directly manage the on board hardware features. |

ASSEMBLY DISK

It contains the low level programming language that includes assembler, linker, debugger, disassembler and some demo programs.

| | | |
|---------------|----|--|
| M80.G80 | -> | Z80 macro assembler. |
| L80.G80 | -> | Z80 relocatable linker. |
| ZSID.G80 | -> | Z80 monitor debugger. |
| HIST.UTL | -> | Utility program for ZSID.G80. |
| TRACE.UTL | -> | Utility program for ZSID.G80. |
| REZJ.G80 | -> | Z80 disassembler. |
| REZJ.DOC | -> | Documentation file for REZJ.G80. |
| \ESGPC???*.* | -> | Directories that includes some demo programs written in assembly, in source and executable form. |

HTC DISK

The disk contains a set of directories that include demo programs written in Hi Tech C compiler, in source and executable form. Each directory can have some subdirectories with other general purpose demo programs (i.e. **DEB 01**, TELECONTROLLO, etc.) or subdirectories with library, project, etc; all these files can be used to directly manage the on board hardware features. Some documentation files explain the operating modes and the file functions in a detailed manner.

The real programming structure is not available on this disk in fact it is supplied separately, in a proper software package.

HOW TO START

In this chapter are described the first elementary operations necessary to start working with **GDOS 80**. The description assumes that a P.C. is used as a development system, so first of all read the previous chapter "REQUIREMENTS" and check the availability of the described items.

- 1) Read carefully all the received documentation.
- 2) Prearrange the target card with power supply, right jumpers configuration and insert the **GDOS 80** EPROM or FLASH EPROM, if not already done.
- 3) Perform serial connection between target card and P.C. as described in "SERIAL COMMUNICATION CABLE" chapter.
- 4) Turn on the Personal Computer.
- 5) Create a new directory on P.C. hard disk; if P.C. has not hard disk, make a copy of the received disks and go to point 8.
- 6) Copy the workink software, the programming software and the demo programs in the new directory (see "DISTRIBUTION DISK" chapter).
- 7) Select the created directory as the current directory.
- 8) Install the P.C. development program **GET80.EXE** typing:
...**GET80** /I<ENTER>
and select the right configuration setting for P.C. serial line (COM), baud rate, help language and monitor type. Remember that **GET80** is supplied correctly configured and normally only the COM number must be changed. For further info please read "INSTALLATION" chapter.
- 9) Execute the P.C. development program **GET80.EXE** typing:
...**GET80**<ENTER>
and wait the information window representation.
- 10) Close the information window typing <ENTER> key and then select the "Options|Terminal" menu (<Alt+T>). A clean window appears with the cursor on the high left corner and the following status line on the last monitor row:

F10 Menu | TERMINAL EMULAT. for GDOS80 - GRIFO° Tel. +39-51-892052 |
- 11) Power supply the target card; on P.C. monitor must appear the following presentation and prompt lines:
GDOS - Ver. X.X - Rel. XXX XXX - by GRIFO° 051 892052 Italy

N:ABACO°>
- 12) Work with **GDOS 80** following the information of the following "GDOS 80 USE" chapter. For example the direct command, the external utility programs, the programming language, etc. can be executed and tested.

You can type:

N:ABACO°>**VER**<ENTER>

that shows the **GDOS 80** information line.

N:ABACO°>**DIR C:**<ENTER>

that loads the DIR.G80 utility program from current drive = N = ROM DISK and shows the files list saved on the P.C. current directory.

N:ABACO°>**N:DIR N:**<ENTER>

that loads the DIR.G80 utility program from current drive = N = ROM DISK and shows the files list saved on the target card ROM DISK drive N.

N:ABACO°>**N:ZBASIC**<ENTER>

that loads the ZBASIC.G80 programming language from ROM DISK drive N.

- 13) Exit from **GET80.EXE** program and return to MS DOS operating system typing <F10> key and then select the "File|Exit to DOS" option (<Alt+X>).

On this chapter some P.C. monitor representations include the generic symbol X that corresponds to digits and/or characters used for version and release; hereby these indications vary according to target card and **GDOS 80** type.

GDOS 80 USE

In this chapter it is described the **GDOS 80** operating system use. In detail it is described the P.C. development program, the remote operating system commands, the target card RAM/ROM disk, the EPROM or FLASH EPROM management and the utility programs.

GET80: P.C. DEVELOPMENT PROGRAM

GET80 (Grifo® Editor Terminal 80 family) is used with **GDOS 80** operating system, which is executed on the target card and it can edit a program, transfer program from P.C. to target card, transfer program from target card to P.C. and test program directly on board. It has two main possibilities: to edit application programs developed by user and to manage an intelligent terminal emulation mode. Deeping last feature, **GET 80** manages all console function, as a simple terminal, and moreover it uses P.C. devices (hard disk, floppy disk, printer, etc.) as target card resource. Moreover the program has two utilities, the first one manages user defined strings that can be saved and transmitted to target card, to minimize P.C. keyboard use and to speed develop phase. The second **GET80** utility manages the EPROM image creation with the developed application program or generally it creates an EPROM image with a user defined ROM DISK.

GET80 is an easy to use program provided with high level user interface including on line help, menus, colour identification, dialog boxes, function keys, mouse management, etc. It is completely based on MS DOS functions and it can be executed with version ≥ 3.3 or as a WINDOWS DOS shell program.

Below are described the **GET80** fundamental features regarding P.C. resource use:

| | |
|---------------------------|--|
| Monitor: | Colour or Black and White. |
| Printer: | Parallel CENTRONICS interface on LPT1 |
| Terminal: | ADDS Viewpoint monochrome emulated terminal. |
| Mass memory drive: | Drive A to C in any MS DOS format |
| Serial line: | One RS 232 line (COM 1÷4) following V24 specifications. |
| Mouse: | Microsoft compatible with its software driver installed. |

The user must always remember that the P.C. executing **GET80** program is an indispensable system only during application program develop and debug phase, in fact target card can work alone or with its serial line connected to any other device. Naturally this external device must use the same physical, logical and electric protocol described in this manual.

To execute program, the User must type:

...\b>GET80<ENTER>

directly from MS DOS prompt.

When the program starts, it sets the P.C. hardware and shows an information window. In this window there are: program version number, **Grifo®** information (address, phone, etc.) and the user information defined during **Grifo®** installation phase. Pressing <ENTER> key or clicking with mouse on "OK" button, the presentation window disappear and the main window is shown on P.C. monitor; the main window has six menus with many options described in the following chapter.

INSTALLATION

Before using **GET80.EXE**, the user must correctly install it. For this reason a configuration option has been added at the program. This option is activated typing:

...**GET80 /I**<ENTER>

When it starts a window appears on the screen, asking for six configuration parameter:

- the default serial line (COM) used on P.C. selectable from 1 to 4;
- the default baud rate used for target card communication selectable from 9.6 to 115.2 KBaud;
- the type of P.C. monitor selectable between color and black & white;
- the help language between Italian and English;
- the user name;
- the company name;

All these six parameter are requested only during first installation, in fact if **GET51.EXE** is already installed only the first four parameters can be inserted; so user and company names can be inserted only during first installation normally made by **Grifo**[®]. The other four parameters are used for **GET80.EXE** configuration and can be changed at any time in a permanent (**GET80 /I**) or temporary (**GET80**) way; in any conditions all these configurations must be set according to P.C. and target card features.

During **GET80.EXE** installation, at any time the user can stop the installation with the button "Abort" , or confirm installation with "Install" button.

If **GET80.EXE** is executed without at least one previous installation, it doesn't start and an error message appears, vice versa it starts showing user and company name in the information windows.

EDITOR

GET80.EXE program includes a powerfull and versatile editor capable of ASCII files management; these ASCII files can be directly used by **GDOS 80**.

It is provided of all standard editor functions and many other functions that facilitate its use in any condition. Moreover **GET80** editor is a multi window editor where all the functions can be used contemporaneously on many ASCII files.

The editor has only one restriction: it can't manage windows or files bigger than 64K byte; this dimension is rarely necessary in the standard work condition and in any case it can be overcome by opening and by editing more files and windows.

MENUS AND OPTIONS DESCRIPTION

As previously stated, **GET80** has 6 menus and 35 options concerning its operating modes. Here follows a brief description of all these options and in next chapters there are some deepening:

File menu

| <i>Option</i> | <i>Key</i> | <i>Function</i> |
|---------------|------------|---|
| New | - | Opens a new editor window with the name "Untitled" |
| Fast Open... | F3 | Opens a selected disk file, loads it in memory with fast mode and shows it in the current editor window |

| | | |
|----------------------|--------|---|
| Open... | F4 | Opens a selected disk file, loads it in memory with standard mode and shows it in the current editor window |
| Save | F2 | Saves current editor window content to a disk file. The disk file name is the same of editor window name. |
| Save as... | - | Saves current editor window content to a disk file. The disk file name must be setted by user. |
| Change dir... | Alt+F5 | Changes current MS DOS directory. |
| Dos shell | - | Exits temporarily from GET80 and return to MS DOS operating system. GET80 remains in memory and it is reexecuted with "EXIT" command. |
| Exit | Alt+X | Stops execution of GET80 and return definitely under MS DOS control. |

If the editor window shows some strange characters when a file is opened with "Fast Open" option, the same windows must be closed and then reopen with "Open" option.

Edit menu

| Option | Key | Function |
|-----------------------|-----------|---|
| Undo | - | Restores, if possibile, the last executed function. |
| Cut | Shift+Del | Deletes from current editor window the previously selected text and puts it into clipboard. |
| Copy | Ctrl+Ins | Copies from current editor window the previously selected text and puts it into clipboard. |
| Paste | Shift+Ins | Copies the clipboard content in the current editor window, starting at actual cursor position. |
| Clear | Ctrl+Del | Deletes from current editor window the previously selected text, without putting it into clipboard. |
| Show clipboard | - | Show an editor window with the clipboard content. |

Search menu

| Option | Key | Function |
|---------------------|--------|--|
| Find... | - | Looks for an inserted string inside the current editor window. |
| Replace... | - | Looks for an inserted string inside the current editor window and if present replaces it with a second string. |
| Search Again | Ctrl+L | Repeats the last executed "Find" or "Replace" option. |

Windows menu

| Option | Key | Function |
|------------------|---------|--|
| Tile | - | Shows all open editor windows disposing them on monitor rappresentation area. The window size are selected to allows rappresentation of all windows. |
| Cascade | - | Shows all open editor windows disposing them on monitor rappresentation area. The windows are overlapped and only their frame and name are visible. |
| Size/Move | Ctrl+F5 | Resizes and/or moves the current editor window. |
| Zoom | F5 | Sets current editor window size to maximum dimension. |
| Next | F6 | Moves current editor window to next open window. |

| | | |
|----------|----------|--|
| Previous | Shift+F6 | Moves current editor window to previous open window. |
| Close | Alt+F3 | Closes the current editor window. |

Options menu

| Option | Key | Function |
|----------------|-----------|---|
| Terminal | Alt+T | Activates the intelligent terminal emulation mode. For further information, please refer to "TERMINAL EMULATION" chapter. |
| Reset Terminal | Ctrl+Home | Clears the terminal emulation window and resets serial communication with target card. |
| Serial Port... | - | Selects P.C. serial line and baud rate, in a momentary manner. |
| Video | - | Selects P.C. monitor colour between colour and black & white, in a momentary manner. |
| Help | F1 | Open the main on line help window. |
| Help Language | - | Selects the GET80 on line help language between Italian and English. |
| Information... | - | Shows the information window about GET80 program. |

Utility menu

| Option | Key | Function |
|------------------------|-----|---|
| GROM... | - | Executes the GROM utility that generates the binary image of the EPROM with user application programs and files. |
| String Editor... | - | Edits the 10 user strings used to develop the application program. |
| Save strings | - | Saves the 10 user strings on GET80.FST file; this file will be automatically loaded every time GET80 is executed, to restore the user strings contents. |
| Send strings... | - | Transmits the contents of one of the 10 user strings to target card, through terminal emulation mode. |

In the previous options description, the letters that activate the options in a fast way are written with a bold style; this fast selection is performed simply by opening menu and pressing the letter, without using the arrow keys. The indication "*Key*" is referred to the key, or keys combination, that select the option immediately, even without opening the menu. The indications "..." following the name of some options, means that the option needs other data requested by a specific dialog box (files name, string to find, directory to select, etc.). When a mouse is available, the options selection is really faster and more comfortable, in fact the User has nothing to press on P.C. keyboard.

For further information about options and menus functionality, please read carefully the **GET80** on line help messages.

TERMINAL EMULATION

The intelligent terminal emulation feature of **GET80**, manages all the P.C. hardware resource and make them available to the used target card. In this way it is possible to directly use hard disk, floppy disks, printer, keyboard and monitor with the software executed on target card, through **GDOS 80** operating system.

The communication with remote card is performed through one of P.C. serial line (COM) with standard connection described in chapter "SERIAL COMMUNICATION CABLE" and with the physical protocol described in "TECHNICAL FEATURES" chapter.

When the "Options|Terminal" option is selected, the actual serial line is setted with the actual baud rate and after a clean window is represented with cursor positioned in the upper left corner and with a status line on the last row (number 25). With actual serial line and actual baud rate are denoted the respective parameters selected by user (with "Option|Serial port" option), initially set at the default value defined by **GET80.EXE** installation. To speed the terminal emulation use it is convenient to install or reinstall it properly.

During terminal emulation execution the characters received from target card are displayed on P.C. monitor, while the keys pressed on P.C. keyboard are transmitted to target card. The communication is controlled with a specific logical protocol that manages also file transfer in a transparent mode for the user. This protocol is based on hardware handshakes to arrange fast communication with any type of P.C. The hardware handshake use is enabled or disabled by a flag saved in **GDOS80 IOBYTE EXTENSION** (see proper chapter), with the following correspondence:

| | | |
|------------------------|----|-----------------------------|
| IOBYTE EXTENSION.7 = 1 | -> | Hardware handshake enabled |
| IOBYTE EXTENSION.7 = 0 | -> | Hardware handshake disabled |

During the communication with **GET80**, the hardware handshake must be enabled and for this reason **GDOS 80** sets this flag, by default. If the developed application program can't manage these handshakes (i.e. the target card serial line A is connected to other systems as modem, terminal, network, etc.) the same application program must disable the handshake flag.

All the programs developed with **GDOS 80** can take advantage of the terminal emulation features described in the following paragraphs, obtaining many facilitations in the user interface code development; many programming languages are supplied already configured for **GET80** terminal emulation.

The terminal emulation works in a full asynchronous mode in confront of target card that executes **GDOS80**; for this reason no power on/off procedure must be followed for both the systems.

N.B.

From **MS DOS** operating system it is possible to execute **GET80** and to automatically enable the terminal emulation; this result is obtained with the following syntax:

```
...\GET80 /T<ENTER>
```

Terminal emulation commands

When terminal emulation is enabled only a subset of the 35 **GET80** options are available. These options are selected by proper hot keys or in the standard mode typing <F10> key and then using menus. The list of terminal emulation available option is: "File|Change dir...", "File|Dos shell", "File|Exit to DOS", "Options|Editor", "Options|Reset Terminal", "Options|Serial Port", "Options|Video", "Options|Help", "Options|Help Language...", "Options|Information...", "Utility|GROM...", "Utility|Strings Editor...", "Utility|Save strings" and "Utility|Send string...". Some of these options open a dialog box that is temporarily shown upon the terminal emulation window.

Terminal emulation control sequence

GET80 terminal emulation mode, recognizes some of the standard ADDS Viewpoint control sequences, when received from selected serial line. These sequences are listed in the following table:

| COMMAND | ASCII CODE SEQUENCE | BYTE CODE SEQUENCE |
|----------------------------------|---------------------|--------------------|
| Cursor Home | SOH | 01 |
| Cursor Left | BS | 08 |
| Cursor Right | ACK | 06 |
| Cursor Down | LF | 10 |
| Cursor Up | SUB | 26 |
| Carriage Return | CR | 13 |
| Carriage Return + cursor down | GS | 29 |
| Cursor position (with offset=32) | ESC,Y,row,column | 27,89,row,column |
| Clear screen | FF | 12 |
| Clear Line | EM | 19 |
| Clear to end of line | ESC,K | 27,75 |
| Clear to end of page | ESC,k | 27,107 |
| Cursor off | ESC,P | 27,80 |
| Cursor line style | ESC,M | 27,77 |
| Cursor block style | ESC,Q | 27,81 |
| Attribute type setting | ESC,0,attribute | 27,48,attribute |
| Set attribute | SO | 14 |
| Reset attribute | SI | 15 |
| Audible BELL | BEL | 07 |

FIGURE 6: GET80 TERMINAL EMULATION CONTROL SEQUENCES

The row and column values can vary respectively in the range 0÷23 and 0÷79 and they must be defined with an offset of 32. Therefore if the desired new cursor position is row 10, column 20, the following command sequence must be transmitted by target card:

27,89,42,52.

About representation attributes recognized by GET80 terminal emulation, only a subset of ADDS Viewpoint standard attributes are managed:

| ATTRIBUTE | ASCII CODE | BYTE CODE |
|----------------|------------|-----------|
| Normale | @ | 64 |
| Half intensity | A | 65 |
| Reverse | P | 80 |

FIGURE 7: GET80 TERMINAL EMULATION REPRESENTATION ATTRIBUTES

Terminal emulation special key

The **GET80** terminal emulation translates the special P.C. keyboard keys (arrows, Ins, Del, etc.) and transmit the proper **GDOS 80** standard code to the connected target card.

| KEY | CODE | HEX CODE |
|--------------------|-------|----------|
| UP arrow | 05 | 05 |
| DOWN arrow | 24 | 18 |
| LEFT arrow | 04 | 04 |
| RIGHT arrow | 19 | 13 |
| Page UP | 18 | 12 |
| Page DOWN | 03 | 03 |
| Home | 17,82 | 11,52 |
| End | 17,67 | 11,43 |
| Insert | 22 | 16 |
| Delete | 07 | 07 |

FIGURE 8: GET80 TERMINAL EMULATION SPECIAL KEYS CODE

This feature is really important especially when programming language with an integrated editor (i.e. PASCAL) is used, in fact you can move inside the application program in a fast and intuitive manner.

USER STRING

Starting from 2.4 version of **GET80** the user strings management has been added to speed and to facilitate the user application program development phases. The user strings are 10 strings with a maximum length of 70 characters, that can be edited at any time and then used in the terminal emulation mode, in place of commands typed on the keyboard. It is convenient that the user strings are those most frequently used during develop phase, in fact their main purpose is to reduce keyboard typing and consequently the develop time.

The strings can be permanently saved in **GET80.FST** file, saved on **GET80.EXE** directory; when **GET80** starts it checks if this file exists and if it does, the **GET80.FST** is loaded and the user strings become available for any operations. The user strings modifications are permanently saved on the file, only after the proper save command; if the save command is not executed all modifications are lost when you exit from **GET80**.

The user string management is available in three different **GET80** parts, as below described:

"Utility/Strings Editor..." option:

By selecting this option, an editor window is displayed; this window includes 10 input string lines (for max 70 chr string insertion), 10 enter code sending boolean icons (to add carriage return code at the end of the strings), an "OK" push button (to confirm the currently inserted data and set them into the 10 user strings), a "Save" push button (to confirm the currently inserted data and save them on **GET80.FST** file), a "Cancel" push button (to discard the currently modified data) and an "Help" push button (to open on line help window).

"Utility/Save Strings" option:

This option saves the current 10 user strings on **GET80.FST** file as the previous described "Save" push button. At the end of save operation a window that reports the operation result is displayed and if user strings are saved successfully every time **GET80** is executed the current user strings will be loaded and restored.

"Utility/Send string..." option:

This option transmits the contents of one of the 10 user strings to target card, connected to **GET80** terminal emulation. A submenu is displayed with the 10 user strings list and the user can select which one must be transmitted together with the eventual carriage return code. A more comfortable way to transmit a user string is to press the <Alt+x> keys combination directly from terminal emulation window, where x coincides with user message number. In this way, this two keys pressure physical is the same as all the user string characters pressure.

When you are working with **GDOS 80**, one of the most frequently used operating mode, is: edit the application program source with **GET80** editor, enter the terminal emulation mode, with the selected programming language load the application program, translate it, compile it and execute it, etc. Every described operation performed in the terminal emulation mode, requires many keys pressure and during debug phase these operations are repeated many times, until the application program is completely tested. For this reason the **GET80** user strings are really comfortable, in fact they substitute the develop commands and so they notably reduce the keyboard use, with a big time save results. For example, when ZBASIC is used, the PRGAPP.ZBA development phase can use the following user strings list:

| | |
|-------------------------------|----------------------|
| String 1 = N:ZBASIC | with Carriage Return |
| String 2 = LOAD* C:PRGAPP.ZBA | with Carriage Return |
| String 3 = SAVE C:PRGAPP.ZBT | with Carriage Return |
| String 4 = NEW | with Carriage Return |
| String 5 = RUN C:PRGAPP.ZBT | with Carriage Return |
| etc. | |

GROM

Starting from 2.4 version of **GET80** the binary **GDOS80** EPROM image generation has been added. This EPROM contains the application programs and files developed by the user through the selected programming language and tools and its use is completely described, in "GROM: PROGRAMMING EPROM" chapter.

DOS2GDOS: P.C. FILE FORMAT TRANSFORMATION PROGRAM

The **DOS2GDOS** program saved on the main root of **GDOS 80** working disk transforms a generic MS DOS file in the equivalent **GDOS 80** format. The **GDOS 80** file logical sectors length is 128 bytes, while MS DOS length is 1 byte, so if a MS DOS file must be used by **GDOS 80**, it must be completed with some "end of file" codes till the same file length reaches a multiple of 128 bytes. The **DOS2GDOS** program add this codes with the following syntax:

...\bDOS2GDOS <name>.<ext><ENTER>

The most frequently use of this utility program is the conversion of MS DOS files generated by: editor capable to manage more than 64K bytes, database, cross compiler, etc.
The **GET80** editor already saves file with **GDOS 80** format, so **DOS2GDOS** program must not be used on these files.

WATCH DOG

In all the target cards provided of watch dog circuit, **GDOS 80** retriggers this circuit during the files read operation. This feature allows the use of chain and overlay technique (an executed application program loads and executes another programs) also when watch dog circuit is enabled, without any problems. Thus it is normal that watch dog LEDs status are modified during file system management. The retrigger is performed for each logical sector read operation, so watch dog will not reset target card during RAM and ROM disk files read operations (access time is really low), while during P.C. files read operations the target card watch dog reset depends on computer access time.

GDOS 80 DIRECT COMMANDS

The **GDOS 80** operating system is capable to execute a set of direct commands that increase the full system versatility; these commands can be used only when **GDOS 80** is in command mode highlighted by **<d>:ABACO°** prompt, on a new line, where **<d>** = current drive name.
In the following pages all the direct commands are described, using some examples to suggest the right syntax (this examples assumes that P.C. development program **GET80** is used).

SAVE nn d:name.ext Saves the TPA program memory contents on the name.ext file of drive d. nn indicates the hex codified length of the memory to be saved, in 256 bytes blocks. The command doesn't check the presence of the specified file, so if a same file name exists, it will be rewritten.

To save the first 5K bytes of TPA memory on a file called TEST.G80 located on P.C. drive A, the following command must be typed:

```
..:ABACO°>SAVE 14 A:PROVA.G80<ENTER>
```

d:name Loads the .G80 command file saved on drive d in TPA memory and after executes it. The command loads and executes the program if it exist, viceversa no operations are performed and the prompt is displayed again. If the file exist the **GDOS 80** current drive is setted with drive d.

To execute the programming language NSB8.G80 saved on P.C. drive C, the following command must be typed:

```
..:ABACO°>C:NSB8<ENTER>
```

VER Displays on console device a message with all the **GDOS 80** version and release information. After the information line is shown, the operating system returns in command mode and the prompt is redisplayed. To show the current **GDOS 80** version and release, the following command must be typed:

```
..:ABACO°>VER<ENTER>
```

RIT Rexecutes the program memorized in the target card TPA area, with no modification on the previous content. It executes an absolute jump to address 0100H and it carry on with execution of the code saved in TPA RAM. This command is really comfortable when used together with some programming language, in fact if you exit unintentionally you can then return with no data lost. This command can be used only if the last executed program guarantees a memory image correct for a new execution (i.e. NSB8).

To execute the current TPA code, without load operation, the following command must be typed:

```
..:ABACO°>RIT<ENTER>
```

If the described direct command syntax is not attended, the obtained results should be not correct.

GDOS 80 UTILITY PROGRAM

The **GDOS 80** operating system is capable to execute a set of utility programs that facilitate the system use. The most part these programs manages files and mass memory device and they can be used only when **GDOS 80** is in command mode, highlighted by **<d>:ABACO°** prompt, on a new line, where **<d>** = current drive name.

In the following pages all the direct commands are described, using some examples to suggest the right syntax (this examples assumes that P.C. development program **GET80** is used).

d1:REN d2:name1.ext1=name2.ext2 Changes the file name1.ext1 saved on drive d2 with the new file name2.ext2. The program executes the rename operation only if the file name1.ext1 exists on drive d2 and when operation is completed the prompt is redisplayed; on the contrary if the file doesn't exist no operation is performed and the prompt is immediately redisplayed. The drive name d1 correspond to the drive where **REN.G80** utility program is saved.

To rename the file **TEST.G80** saved on RAM disk of target card in **AUTORUN.G80**, when **REN.G80** program is saved on P.C. drive C, the following command must be typed:

```
..:ABACO°>C:REN M:TEST.G80=AUTORUN.G80<ENTER>
```

d1:DIR d2: Shows the list of files saved on drive d2 on console device. The files name and the extension are displayed left aligned, divided by a space and divided in pages; when the page is full the files list representation is stopped until a key is pressed, as described by a proper message. The drive name d1 correspond to the drive where DIR.G80 utility program is saved.

To display the files list of drive A, when DIR.G80 program is saved on P.C. drive C, the following command must be typed:

```
..:ABACO°>C:DIR A:<ENTER>
```

d1:ERA d2:name.ext Deletes file name.ext from drive d2. The program executes the erase operation only if the file name.ext exists on drive d2 and when operation is completed the prompt is redisplayed; on the contrary if the file doesn't exist no operation is performed and the prompt is immediately redisplayed. The drive name d1 correspond to the drive where ERA.G80 utility program is saved. The erase operation is definitive, so first to use it check carefully its ability.

To delete the file TEST.ZBA saved on P.C. drive A, when ERA.G80 program is saved on ROM disk target card, the following command must be typed:

```
..:ABACO°>N:ERA A:TEST.ZBA<ENTER>
```

d1:FORMAT d2: Formats drive d2. The program checks if the selected drive is a RAM disk drive and if it is not no operation is performed and the prompt is immediately redisplayed; on the contrary if d2 is a RAM disk drive it ask for a confirmation, proceeds or abort format operation and redisplay the prompt. The drive name d1 correspond to the drive where FORMAT.G80 utility program is saved. The format operation is definitive and it remove all the RAMDISK files, so first to use it check carefully its ability.

To format the target card RAM disk drive M , when FORMAT.G80 program is saved on P.C. drive A, the following command must be typed:

```
..:ABACO°>A:FORMAT M:<ENTER>
```

d1:COPY d2:name.ext=d3: Copies the file name.ext from drive d2 to drive d3. The program executes the copy operation only if the file name.ext exists on drive d2 and when operation is completed the prompt is redisplayed; on the contrary if the file doesn't exist no operation is performed and the prompt is immediately redisplayed. The drive name d1 correspond to the drive where COPY.G80 utility program is saved. During the file copy some tests are performed: for example the free space on drive d3 is checked. To copy the file CBZDEMO.G80 saved on P.C. drive A to target card RAM disk, when COPY.G80 program is saved on target card ROM disk, the following command must be typed:

```
..:ABACO°>N:COPY A:NSB8.G80=M:<ENTER>
```

All the utility program above described (ERA, DIR, REN, COPY) can't use the wild cards available in many others operating systems. Furthermore remember that utility programs are real and complete **GDOS 80** command files and often they show an information message with a version number. If the user needs more powerful utility programs, he can ask them directly to **Grifo®** or he can develop them alone, using any programming language.

To speed the utility program load and execution time, they are normally supplied already saved on target card ROM disk.

GDOS 80 UTILITY PROCEDURES

The **GDOS 80** operating system has some general purpose utility procedures that allow high level management of target card on board devices. These procedures can be directly called by programming language and therefore directly used. All the procedures are called through their memory allocation addresses: these addresses are fixed and don't vary when **GDOS 80** version or target card type changes. In this way the application program code is highly portable on all different **Grifo®** cards. The following table shows the allocation addresses value of these utility procedure, while the following paragraphs describe their use.

| PROCEDURA UTILITY | ADDRESS | HEX ADDRESS |
|---|----------------|--------------------|
| Format RAM DISK drive M | 62003 | F233 |
| Format RAM DISK drive O | 62006 | F236 |
| Block read,write serial EEPROM | 62009 | F239 |
| Format RAM DISK drive L | 62012 | F23C |
| Local operator interface LEDs activation | 62015 | F23F |

FIGURE 9: UTILITY PROCEDURES ADDRESSES

FORMAT RAM DISK DRIVES

These utility procedures can format all the target card drives (external RAM card drive too), directly from application program. The procedures have no input and output parameters and they are simply called at the allocation address specified in figure 9, as a standard machine language routine; they are absolute and they unaffact the TPA work memory and the microprocessor registers. About correspondence between drive names (L, M, O) and memory devices, please refer to "RAM ROM DISK" paragraph.

BLOCK READ,WRITE SERIAL EEPROM

This utility procedure can read/write a block of bytes from/to target card serial EEPROM, through all the available programming language of **GDOS 80** operating system. To correctly use the procedure, first you must set the input parameters, then call the allocation address specified in figure

| <i>Address</i> | <i>Parameter</i> |
|----------------|--|
| FF80H = 65408 | -> Local operator interface LEDs 1÷8 status |
| FF81H = 65409 | -> Local operator interface LEDs 9÷16 status |

The input parameters setting must be performed with the proper memory access instructions, available in the selected programming language. The 16 local operator interface LEDs status is joined with the 16 bits of the two input parameters, with the following correspondence:

| | | |
|------------------------------|----|--------------|
| Bit 0 of byte saved at FF80H | -> | LED1 status |
| Bit 1 of byte saved at FF80H | -> | LED2 status |
| : : : : : | : | : : |
| Bit 7 of byte saved at FF80H | -> | LED8 status |
| Bit 0 of byte saved at FF81H | -> | LED9 status |
| : : : : : | : | : : |
| Bit 7 of byte saved at FF81H | -> | LED16 status |

The LEDs status is defined with the following correspondence: Bit reset (0) -> LED turned off
 Bit set (1) -> LED turned on

The user must pay particular attention in this utility procedure use, in fact if target card is not connected to local operator interface, but with some other hardware interfaces, when the procedure is executed, it moves the I/O lines and the effect on the different hardware interface is not predictable. In the following page is available the location of the LEDs managed by this procedure for **QTP 24P** local operator interface.

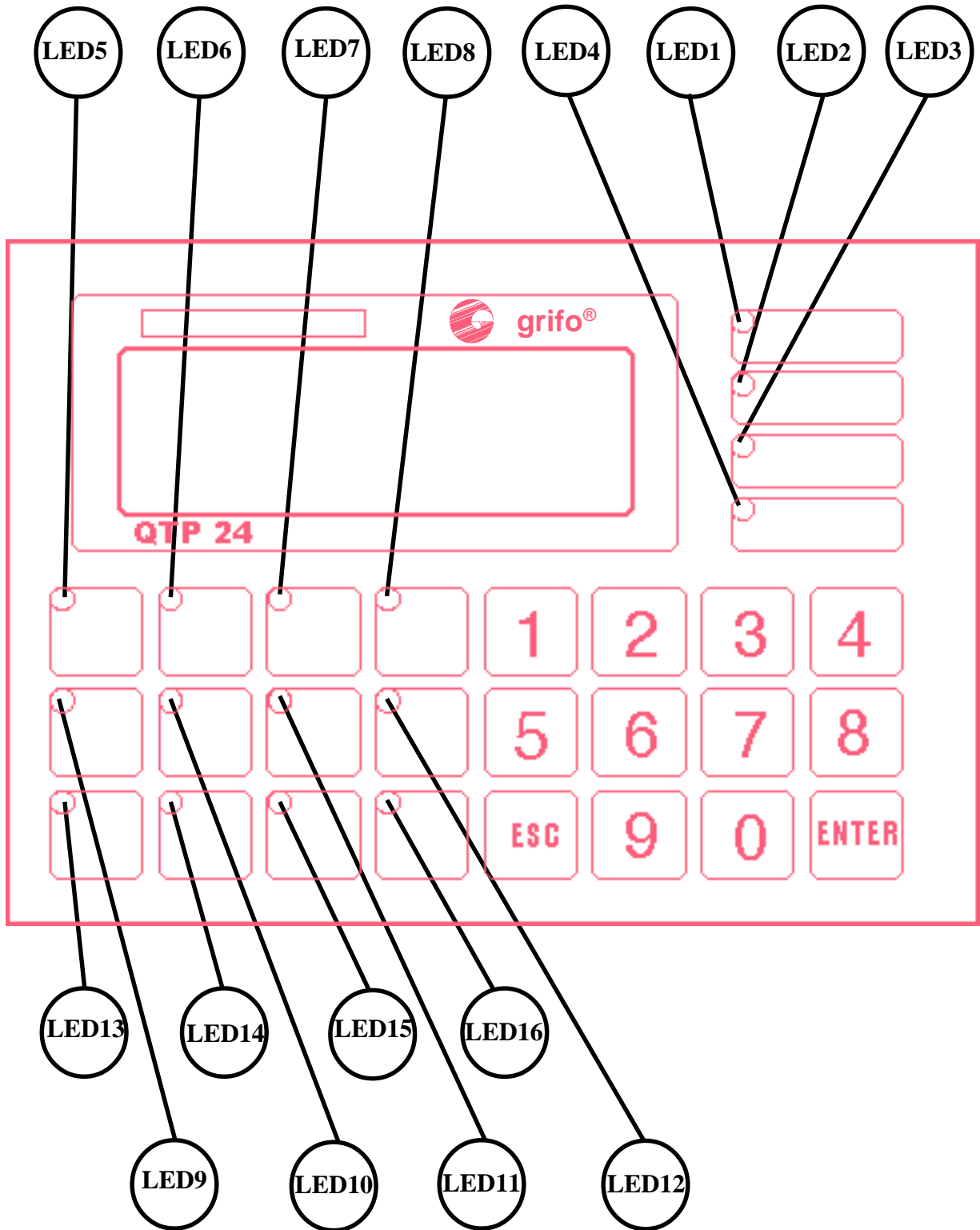


FIGURE 10: QTP 24P LEDs LOCATION

RAM ROM DISK

The **GDOS 80** take care of management of RAM memory that exceeds 64K bytes and of EPROM or FLASH EPROM that exceeds 16K bytes, respectively as RAM disk and ROM disk. In this way all files managed by operating system can be read and write also on the local mass memory resources. This feature notably accelerates all the load and save operation, with a big time economization especially during application development phase.

In details are available:

External RAM card RAM disk, associated at drive 12 = logic drive L
 On board target card RAM disk, associated at drive 13 = logic drive M
 On board target card ROM disk, associated at drive 14 = logic drive N
 On board target card RAM disk, associated at drive 15 = logic drive O (auxiliary)

For example if you want to save the first 10K bytes of TPA memory in a name.ext file saved on RAM disk drive M, the following direct command must be used:

```
..:ABACO°>SAVE 28 M:name.ext<ENTER>
```

Instead if you want to load and execute the command file NSB8.G80 saved on ROM disk drive N, the following direct command must be used:

```
..:ABACO°>N:NSB8<ENTER>
```

Please remember that RAM/ROM disk availability and dimension depend on target card type; in fact any card have a different minimum and maximum memory configuration, as described in figure 11.

| TARGET CARD | DRIVE L | DRIVE M | DRIVE N | DRIVE O |
|-----------------|---------------|---------------|----------------|---------------|
| GPC° 80F | 0,5÷16M bytes | 0÷188K bytes | 109÷237K bytes | not available |
| GPC° 81F | 0,5÷16M bytes | not available | 109÷493K bytes | not available |
| GPC° 011 | 0,5÷16M bytes | 0÷128K bytes | 109÷237K bytes | 0÷58K bytes |
| GPC° 15A | 0,5÷16M bytes | 61K bytes | 109÷493K bytes | not available |
| GPC° 15R | 0,5÷16M bytes | 61÷445K bytes | 109÷493K bytes | not available |
| GPC° 153 | 0,5÷16M bytes | 61÷445K bytes | 109÷493K bytes | not available |
| GPC° 183 | 0,5÷16M bytes | 61÷445K bytes | 109÷493K bytes | not available |
| GPC° 154 | 0,5÷16M bytes | 61÷445K bytes | 109÷493K bytes | not available |
| GPC° 184 | 0,5÷16M bytes | 61÷445K bytes | 109÷493K bytes | not available |

FIGURE 11: LOCAL DISK DIMENSIONS

N.B.

The drive L disk dimension must be reduced of 3K bytes used by disk directory information.

Before using ROM disk and RAM disk it is necessary to check the type of target card and its memory configuration. If on board memory resource are not sufficient, some auxiliary mass memory devices can be used, like **MCI 64**. The **GDOS 80 MCI** version can manage a large dimension RAM disk drive, physically located on a RAM card inserted into **MCI 64** interface, that can be used to solve data loghin, data base, data transformation, etc. problems (for further info please refer to "DIGITAL I/O INTERFACE" and "MCI 64 RAM DISK" paragraphs).

On target card provided of RAM write protection circuit, it is always preferable to enable this features in fact **GDOS 80** drive the write protection circuit and in this condition, data integrity is really guaranteed.

GDOS 80 during reset and power on checks the presence and dimension of RAM and ROM disk drive and after it is able to manage file system on this disk without other preliminary operation of the user. During RAM disk check it verifies also the disk format status and if RAM disk is not formatted (first use, power supply absence or back up battery discharged) it formats all the available space to allows subsequent disk use. If you are in doubt on RAM disk status, you can use proper utility program or procedure as described in "GDOS 80 UTILITY PROGRAM" and "FORMAT RAM DISK DRIVES" paragraphes.

The RAM disk drive can be used both for reading and writing operation, while ROM disk drive can be used only for reading operation. If ROM disk is saved on a FLASH EPROM device it can also be written, but with a dedicated utility program and without **GDOS 80** file system (see "EPROM AND FLASH EPROM PROGRAMMING" chapter).

The user can manage all mass memory disk drive, using proper and powerfull programming language instruction. Data can be read or write in files in any desidered form and organization mode; these modes and instruction syntax must be searched and examined on the programming language technical documentation.

The **GDOS 80** RAM and ROM disks management has some limits in confront of standard operating system. These limits are described in the following points, they are a consequence of the sequential organization of disks and files and they can be easily surpassed.

- 1) A write operation on a file that increase this file dimension is allowed only on one file that is the last created on the drive. Instead, the write operation without dimension increment is allowed with no limits.
You can get over this limit writing the increasing files until their maximum dimension are reached, directly the first time that files are created.
- 2) The free space and the file lenght indication on the local disk are indicative and they can't be used to calculate free and busy space lenght.
You can get over this limit using proper software counters or by testing the eventual operation result.
- 3) The wild cards characters ("*" or "?") can't be used on target card local disk. File names must be always complete and with the right upper or lower case character set.
- 4) On a target card disk can't be saved more than 64 files and there is no possibility to surpass this limit.
- 5) On ROM disk (drive N) can't be opened at the same time more than 3 files and on external RAM card RAM disk (drive L) can't be opened at the same time more than 10 file. There is no possibility to surpass this limit.

AUTORUN PROGRAM

GDOS 80 can execute the application program developed by the user, after a target card reset or power on. This feature allows the system test during start up phase and the final installation of a completely tested application program.

To enable automatic execution it is sufficient to save on target card RAM or ROM disk an executable file with the name: **AUTORUN.G80**. When **GDOS 80** starts execution, it first check the presence of this file on all target card disks and if present it loads and executes the same file, viceversa it enters the command mode and the prompt **N:ABACO°>** is displayed. On target card with more local disks, the **AUTORUN.G80** file is searched on all these disks with the following priority: N, M, O, L.

If **AUTORUN** file is found, the **GDOS 80** loads and executes it but it doesn't transmit the usual presentation message on console device; so only **GDOS 80** hardware inizialization steps are executed and no other operation are added to user application program.

If the user needs to use **GDOS 80** operating system even if an **AUTORUN.G80** is saved on target card RAM/ROM disks, he can change the operating mode of the card. This possibility is really important when interventions on installed systems became necessary: by disabling the **AUTORUN** management the user can change the application program, executes other programs, test hardware system, acquire and check working results, etc. In details:

RUN mode: The **AUTORUN** program if available is executed in automatic mode, otherwise **GDOS 80** command mode is entered.

DEBUG mode: **GDOS 80** command mode is entered also if **AUTORUN** program is available.

The operating mode selection is described in target card technical manual; normally a jumpers is provided and sometimes its status is displayed by proper LEDs (green LED for **RUN** mode and yellow LED for **DEBUG** mode). On the cards not provided of operating mode selection jumper (i.e. **GPC® 80F**, **GPC® 011**) the **DIP 8** of on board dip switch must be used, with the following correspondence:

| | |
|-------------------------------|-------------------|
| DIP 8 in OFF position: | RUN mode |
| DIP 8 in ON position: | DEBUG mode |

The file named **AUTORUN.G80** must be an executable file with **100H** start address (**GDOS 80** command file) and it can be generated directly with the selected programming language. For further info on this command file generation, please refer to programming language documentation.

EPROM AND FLASH EPROM PROGRAMMING

The final step that must be executed to obtain a definitive application with **GDOS 80**, is the programming of the **EPROM** (that will be mounted on target card) or **FLASH EPROM** (already mounted on target card). Both **EPROM** and **FLASH EPROM** are managed by **GDOS 80** as **ROM** disk, so data on this memory devices are always organized in a file system and they will be available through **N** disk read operations.

The **EPROM** must be burned with a proper external programmer and it can be modified only through a replace or erase and reburn operations; anyway it is necessary an hardware intervent to disconnect and reconnect the **EPROM**. For this reason, it is a good practise to save the application program in

EPROM only when it is completely tested.

The FLASH EPROM can be burned directly on target card, so it offers the advantage that no hardware interventions are necessary. The user can modify the application program and then program and reprogram the device for all the requested time, with a simple serial connection to target card.

On ROM disk, besides application programs, other data can be saved: utility programs, messages files, configuration files and/or any other development programs and programming languages, that speed the development phase.

GROM: EPROM PROGRAMMING

As described in the previous paragraphs, to carry out the EPROM has been provided a proper option in **GET80** P.C. development program. This option is named **GROM**, it is available in utility menu and requires the following steps execution:

- a) Check that all the files to be saved on ROM disk, including the **GDOS 80** binary file (saved on the received disks), are available on the P.C. disk drives. In fact **GROM** manages all P.C. drives and directories as loading devices, but it can't use the target card local drive. If the files are saved on target card RAM/ROM disks, please copy them with **GDOS 80** utility programs.
- b) Execute the **GET80** on a personal computer as described in "GET80: P.C. DEVELOPMENT PROGRAM" and selects the "Utility|GROM" options, so as to open the respective dialog box.
- c) Select the used EPROM size. **GROM** checks that the chosen device size is sufficient to contain all the ROM disk files and if it is not, a proper warning message box is displayed. However, according to ROM disk required space, the User can change the EPROM size (compatibly with target card features) obtaining a cost reduction.
- d) Sets the proper flag if the target card is a **GPC® 011**.
- e) Select the binary file containing the **GDOS 80** operating system image for the used target card. This file is saved in ROM_DISK directory of the received disk and it can be chosen through a file selection window associated to "GDOS80 File" button. At the end of selection, the file name and path is displayed on the right side of "GDOS80 File" box.
- f) Insert the binary file name, created by **GROM**, containing the EPROM image to burn. It can be chosen through a file selection window associated to "BIN EPROM File" button and at the end of selection, the file name and path is displayed on the right side of the same button. The inserted file will have the same dimension of the selected device; the optional not used area is filled with no meaning data (OFF Hex).
- g) Insert the file list to be saved on ROM disk; this operation includes the original file name selection and the optional new name insertion. In details when "ROM -Disk" window is selected, the following keys can be used:

| | |
|-------------------------------------|---|
| <arrows>,<PgDn>,<PgUp>,<End>,<Home> | -> to select the current ROM disk file; |
| <Ins> | -> to add a ROM disk file at the end of the list; |
| <ENTER> | -> to modify the current ROM disk file; |
| | -> to delete the current ROM disk file; |
| <Ctrl+Del> | -> to delete all inserted ROM disk files. |

During ROM disk file add and modify operation, firstly is displayed a file selection dialog window and after a second window to insert the file name in ROM disk; this possibility is really usefull especially to save a file with AUTORUN.G80 name on ROM disk starting from a different file name saved on P.C. disks. This file names selection and insertion modalities are completely described in the program together with ROM disk files number and free ROM disk space information. When free space (that depends on selected EPROM size and files list) is not sufficient for the new file addition, the **GROM** shows a proper status message, as described on point **c**. An error message is also displayed when it is inserted a file name that is already saved in ROM disk. The ROM disk file list window is composed by 9 rows and it scrolls up and down to visualize all the eventual maximum 64 files.

- h)** By using the "Cancel" button the window is closed but inserted data are mantained and they will be redisplayed when **GROM** is reselected; this features is really confortable in fact if you start **GROM** use and after you discover that some files are not available or not updated, you can exit **GROM**, generate the requested files and then reexecute **GROM** starting from the previous status.

By using the "Make" button all inserted data are confirmed and a status window is displayed; this window includes a scroll bar that indicates the percentual amount of processed work and an inferior status line that shows the currently executed operation. In this phase the EPROM binary image file is really generated and at the end, on the status line, is reported the final operation result.

- i)** Exit from **GET80** and return to MS DOS environment. Here verify the creation of the file specified at point **f**.
- j)** Prearrange the personal computer for EPROM programmation performing all the necessary operation required to burn an EPROM of dimension specified at point **c** with used EPROM programmer.
- k)** Burn the file obtained at point **h** on a new EPROM of the previously selected size starting from address 00000H. If the EPROM programmers accept only .HEX Intel file format, you can convert the binary file in the equivalent .HEX through GHEX2.COM with the following syntax:

...\b>GHEX2 <name>.<ext><ENTER>

This utility program generates the file <name>.HEX that follows the HEX Intel standard format.

- l)** Check the hardware configuration of the target card, especially for jumpers that select the on board memory configuration, and select the right setting for selected size EPROM management.
- m)** Substitute the obtained EPROM with the one installed on target card. At power on or reset the card executes **GDOS 80** and it makes available the files inserted in ROM disk. If the same ROM disk (drive N) includes a command file named AUTORUN.G80, this be immediately executed as described in "AUTORUN PROGRAM" paragraph.

FGROM.G80: FLASH EPROM PROGRAMMING

To program the target card FLASH EPROM a proper **GDOS 80** program, named **FGROM.G80**, has been developed; it is saved on ROM_DISK directory of the received working disk and it allows a direct management of the FLASH device according to user request. The following steps must be performed:

- a) Check the hardware configuration of the target card, especially for jumpers that select the on board memory configuration, and select the right setting for FLASH EPROM management.
- b) Assure that all the files to be saved in ROM disk are available on the current P.C. disk directories or on target card disk. In fact **FGROM** manages all **GDOS 80** drives as loading devices, but it can't change directories.
- c) Run the **FGROM** program on target card by loading it from one of the **GDOS 80** disks. At the end of the loading phase appears a window that describes the program and shows the version and release information. On this upper status line it is also reported the target card name that is automatically recognized by **FGROM**, thanks to **GDOS 80** information; the user must only check that the card information is correct and proceed with point **d** or abort program execution if target card type is wrong.

- d) At this point an operation type selection menu is displayed on the main window of **FGROM** and the user can choose the required operation. The FLASH EPROM programming can be performed in three different modes:

create a new ROM disk: it is selected typing <C> key and it deletes all the previous ROM disk data and after adds a new file list.

append to a previous ROM disk: it is selected typing <A> key and it adds a new file list to a previous ROM disk.

delete the last ROM disk file and append other files: it is selected typing <D> key and it deletes the last file saved in ROM disk and after adds a new file list. A proper confirmation message precedes the delete file operation, so the user can confirm typing <Y> key, or abort typing <N> key.

When the operating mode is selected, **FGROM** performs some operations on the FLASH EPROM, described by a proper wait message, and then it shows the selected operating mode and the current ROM disk status (files number and free space length). If during FLASH EPROM management some incongruity or errors are encountered, an information message is displayed, the program aborts and control returns to **GDOS 80** operating system.

- e) At the end of the described phases **FGROM** starts a loop composed of some sequential steps that allows file insertion on ROM disk. The first step is a request to continue or to exit from this loop: if you exit, **FGROM** completes the FLASH EPROM programming and control returns to **GDOS 80** operating system, while if you continue, **FGROM** asks for source and destination ROM disk file names. In detail before it asks for the source file name that must be saved on **GDOS 80** disks and after for destination file name to be programmed on ROM disk. This possibility is really useful especially to save a file with AUTORUN.G80 name on ROM disk starting from a different file name saved on P.C. disks. This file names insertion modalities are completely described in the program together with an updated ROM disk files number and free ROM disk space information. When free space (that depends on programmed files number and length) is not sufficient for the new file addition, the **FGROM** shows a proper status

message and returns to point **e**. When free space is sufficient a message with the current read and programmed logic sector is continuously updated, until the entire file is burned. At this point an acoustic signal is generated to attract user attention.

The delete the last ROM disk file and append other files mode described at point **d** is really interesting when the user needs to modify his application program with no modifications on all the other saved files. In this condition it will be sufficient to program the application program in the last position and then delete it and add it.

Please remember that **GDOS 80** checks the target card disk presence only after a reset or power on sequence; so if a FLASH EPROM without ROM disk is programmed with **FGROM** (create new ROM disk mode), the new inserted files will be available only after a target card reset or power on. **FGROM** can burn the only ROM disk area, not the total FLASH EPROM area. For this reason **FGROM** can be used only on FLASH EPROM that already contains **GDOS 80** (i.e. **FGD xxx** described in "GDOS 80 VERSION" chapter). The first sector of the FLASH EPROM is reserved for operating system, to prevent any eventual modifications and or deletion; this ensures the **GDOS 80** functionality in each condition, even if power supply fails during **FGROM** execution.

DIGITAL I/O INTERFACE

The most part of **Grifo**[®] control cards have 16 TTL I/O lines, available on a standard connector that can be directly connected to many interface **Grifo**[®] cards with the same standard connector, through a flat cable. Between these cards you can find didactic boards, power I/O for field signals connection, mass memory devices, operator interfaces, etc. and for some of them **GDOS 80** has proper software drivers. In this way the user can directly manages many powerfull external systems with high level instructions of the selected programming language, without hardware experience or knowledge. In the following table are reported the target card connector and cable list that must be used for digital I/O interface connection:

| TARGET CARD | CONNECTOR | CONNECTION CABLE |
|-------------|---------------|------------------|
| GPC° 80F | CN1 | FLAT 26+20 |
| GPC° 81F | CN3 | FLAT 20+20 |
| GPC° 011 | CN5, CN3 (*1) | FLAT 20+20 |
| GPC° 15A | CN2 | FLAT 20+20 |
| GPC° 15R | CN9 | FLAT 20+20 |
| GPC° 153 | CN3 | FLAT 20+20 |
| GPC° 183 | CN3 | FLAT 20+20 |
| GPC° 154 | CN5 | FLAT 20+20 |
| GPC° 184 | (*2) | FLAT 20+20 |

FIGURE 12: DIGITAL I/O INTERFACE CONNECTION

Normally only one I/O interface can be connected to target card; the cards with two connectors (*1) are an exception: on the first connector you can connect printer and memory card RAM disk, while on the second you can connect local operator interface. The cards without digital I/O lines (*2) anyhow can drive interface cards through **GDOS 80** operating system. This possibility is based on an external PPI 82C55 connected to **ABACO®** I/O BUS allocated at I/O address **BCH**. For example the cards **PIO 01**, **ETI 324**, etc. can be used and for further info you can directly contact **Grifo®** technician.

GDOS 80 doesn't use the digital I/O lines itself to safeguard the system functionality when these lines drive some external system; only the user can enable the operating system lines managements by software, as described in the following paragraphs.

LOCAL PRINTER

Through the interfaces **IAC 01** and **DEB 01** the **GDOS 80** can manage a local printer. It can be connected a generic parallel printer with CENTRONICS standard interface (office printer, panel printer, etc.) that is connected to the listed interfaces through a standard CENTRONICS cable; the interface is then connected to target card as described in figure 12. By software the printer is managed through a comfortable instruction set of the selected programming language ("write(LST,...)" for PASCAL, "LPRINT" for ZBASIC, etc.) and strings, constants, variables, graphics can be directly displayed. The user must only properly sets the IOBYTE data structure as described in the proper chapter of this manual.

The I/O lines used for printer management, are initialized and setted only during the first character print operation, from target card reset or power on, so if printer instructions are not used, **GDOS 80** doesn't change the I/O lines status. The printer software driver is not postponement: if the printer is not ready, after a timeout the same driver exits; moreover it is not possible to acquire the current printer status.

MCI 64 RAM DISK

As described in "RAM ROM DISK" paragraph, the **GDOS 80** can manage a RAM card through the proper PCMCIA memory card interface: **MCI64**. This RAM disk drive is associated to logical drive L and it can have variable dimension from 512K bytes to 16M bytes. The connection must be performed following the figure 12 information and plugging the 20 pins flat cable connector to CN2 of **MCI 64**.

The **MCI 64** LEDs have the following functions:

- LD1 (red) -> signals a write operation performed on memory card (turned on if write in procession and viceversa);
- LD2 (yellow) -> signals the presence of power supply on the memory card (turned on if power supplied and viceversa);
- LD3 (green) -> signals the memory card battery status (turned on if battery charged and viceversa);
- LD4 (red) -> signals the memory card write protect status (turned on if write protect enabled and viceversa);

and they are updated for each RAM card access. The memory card can be extracted at any time, supposing that **GDOS 80** is not driving it, and when the card is out all the LEDs are turned off. Please remember that LD1 is enabled also during read operation on not write protect RAM card, in fact the directory information are updated and then saved on the same card.

On RAM card with write protect enabled, file write operation can't be performed and only 10 files can be opened and used for read operation, at the same time.

The RAM card managed by **GDOS 80** is really similar to standard floppy disk, in fact it can be extracted, moved, used on other systems even if based on different target cards, it is fast and it has an high capacity.

GDOS 80 verifies the AUTORUN.G80 file presence also on drive L (really interesting feature for card not provided of on board RAM disk, as **GPC® 81F**) and to prevent undesirable settings on the 16 digital I/O lines that drive **MCI 64** (when they are used to drive other systems) the operating system must be informed of the **MCI 64** presence. This inform operation consists of a configuration file called CONFIG.SYS that must be saved on a target card local disk (ROM or RAM disk). The CONFIG.SYS file can be generated through the proper **GDOS 80** program: **CONFIG.G80**, and it can be copied with the utility program. So the user, thanks to this configuration file, decides if the **MCI 64** disk drive must be used or not; a simple hardware connection of the interface and of the memory card are not sufficient to manage it correctly. CONFIG.SYS file is verified and used by operating system only during target card reset and power on phase, therefore after a configuration file creation or modification the **GDOS 80** must be restarted.

The "RAM ROM DISK" paragraph indicates that when **GDOS 80** starts, it checks the local disk status and it formats the RAM disk if they are not still formatted; this is not true for RAM disk drive L, in fact RAM card can be removed and used on different systems (not **GDOS 80**) and in this case some data should be incorrectly changed. RAM card disk must be always formatted by the user through the proper utility command and procedure.

By software the RAM card disk is managed through the comfortable instruction set of the selected programming language ("write(..)", "read(..)", "assign(..)", "delete(..)", etc. for PASCAL; "OPEN..", "READ..", "WRITE..", "KILL..", etc. for ZBASIC; "fprintf(..)", "fscanf(..)", "fopen(..)", "remove(..)", etc. for C; etc.) and any data type can be read or write.

For further information on **GDOS 80** operating system configuration and on **MCI 64** management, please refer to following paragraph "CONFIG.*: GDOS 80 CONFIGURATION".

LOCAL OPERATOR INTERFACE

The **GDOS 80** can drive a long list of local operator interfaces that became available through the high level instructions of the programming language. The operator interface problem always is one of the most important problems in many application programs, thus the availability of ready to use systems simplify the user work and reduces the develop time. The primary console device managed by **GDOS 80** can be associated to operator interface hardware systems as **QTP 24P**, **QTP 16P**, **KDx x24**, **IAF 42**, **IAL 42**, **DEB 01**, etc, that include alphanumeric displays, status LEDs and keyboard. These interfaces are provided of a standard I/O connector and they can be directly connected to target card following figure 12 indications. If the **Grifo®** interfaces don't satisfy the user requirements, the user can himself produce a special operator interface following the informations of appendix A, where you can find some electric diagram.

For operator interface LEDs activation the proper utility procedure described in the previous paragraph must be used, while for keyboard and display management the high level instructions for console logic device can be used (i.e. ("write(..)", "read(..)", "kypressed(..)", etc. for PASCAL;

"PRINT..", "INPUT..", "INKEY", etc. for ZBASIC; "printf(..)", "scanf(..)", "kbhit()", etc. for C; etc.). The user must properly advise the **GDOS 80** of the installed local operator interface through the IOBYTE EXTENSION data structure as below described:

IOBYTE EXTENSION: D7 D6 D5 D4 D3 D2 D1 D0

- D0 -> Selects display type
 0 -> Selects LCD display
 1 -> Seleziona display VFD

If display type = LCD (D0=0):

- D1 -> Selects transmission type
 0 -> Selects transmission of characters to LCD display
 1 -> Selects transmission of commands to LCD display

If display type = VFD (D0=1):

- D1 -> Selects interface type
 0 -> Selects **QTP 24P** operator interface
 1 -> Selects **KDx x24, IAL 42, IAF 42, DEB 01** operator interfaces

- D2 -> Associates primary console out logic device
 0 -> Primary console out logic device associated through IOBYTE
 1 -> Primary console out logic device associated to local operator interface display

- D3 -> Associates primary console in logic device
 0 -> Primary console in logic device associated through IOBYTE
 1 -> Primary console in logic device associated to local operator interface keyboard

D4 -> Not used (set to 1)

D5 -> Not used (set to 1)

D6 -> Not used (set to 1)

D7 -> Hardware handshake enable flag for **GET80** communication (see proper paragraph)

The IOBYTE is another **GDOS 80** data structure that defines the association between software logical device and hardware physical device and it is described in "IOBYTE AND IOBYTE EXTENSION" paragraph.

VFD denotes a vacuum fluorescent display (20x1, 20x2, 20x4, 40x1, 40x2), while LCD denotes a liquid crystal display (20x2, 20x4, 40x1, 40x2) that can be mounted or connected to the listed operator interface. If You are using **QTP 16P**, please remember to set D0=0 even if a VFD display is installed, in fact these fluorescent display types are LCD compatible.

GDOS 80 provides only the firmware drivers for characters and/or commands transmission to display, but it doesn't perform display initialization procedure and it doesn't recognize cursor position and display commands. All these not supported features are completely described in display technical documentation and they must be included in the application program.

The operator interface keyboard is a matrix keyboard composed by 6x4=24 keys; only the target card provided of hardware timer sections can associate the primary console in logic device to operator interface keyboard (D3=1). For the cards not provided of timers (**GPC® 81F**) the IOBYTE

EXTENSION bit D3 is a don't care bit and the primary console in logic device is always associated through IOBYTE. For keyboard use, more than timer counter 0, it is enabled also an interrupt management based on a 5 msec. periodic request generated exactly from the timer. The **GDOS 80** sets the I/O lines connected to local operator interface and the keyboard scanning interrupt only after first call to its primary console out or in functions with respectively D2 or D3 = 1. The user only have to:

- not use the timer 0;
- if he uses interrupts in his application program, he must select vectorized interrupt mode (IM 2) and he must save the own interrupt procedure vector in a proper table located in the final 256 bytes of the 64K bytes work area. In fact **GDOS 80** loads the value 0FFH into the I register and this value can't be changed;
- don't use the two bytes saved in FFF8H+FFF9H addresses of the 64K bytes work area where the keyboard scanning interrupt procedure address is saved.

To satisfy all the user requests, the **GDOS 80** provides the possibility to define the key codes of the local operator interface. The following tables show a numeration of the keys reported to keyboard connector and figures 16 and 17 show the physical position of these keys.

| PIN CN3 QTP 16P | 8 | 7 | 6 | 5 |
|--------------------|---|---|----|----|
| 4 | 0 | 4 | 8 | 12 |
| 3 | 1 | 5 | 9 | 13 |
| 2 | 2 | 6 | 10 | 14 |
| 1 | 3 | 7 | 11 | 15 |

FIGURE 13: QTP 16P KEYS NUMERATION

| PIN CN3 QTP 24P | 6 | 5 | 4 | 3 | 2 | 1 |
|--------------------|---|---|----|----|----|----|
| 10 | 0 | 4 | 8 | 12 | 16 | 20 |
| 9 | 1 | 5 | 9 | 13 | 17 | 21 |
| 8 | 2 | 6 | 10 | 14 | 18 | 22 |
| 7 | 3 | 7 | 11 | 15 | 19 | 23 |

FIGURE 14: QTP 24P KEYS NUMERATION

| PIN CN2 KDx x24 | 8 | 7 | 6 | 5 | 9 | 10 |
|--------------------|---|---|----|----|----|----|
| 4 | 0 | 4 | 8 | 12 | 16 | 20 |
| 3 | 1 | 5 | 9 | 13 | 17 | 21 |
| 2 | 2 | 6 | 10 | 14 | 18 | 22 |
| 1 | 3 | 7 | 11 | 15 | 19 | 23 |

FIGURE 15: KDx x24 KEYS NUMERATION

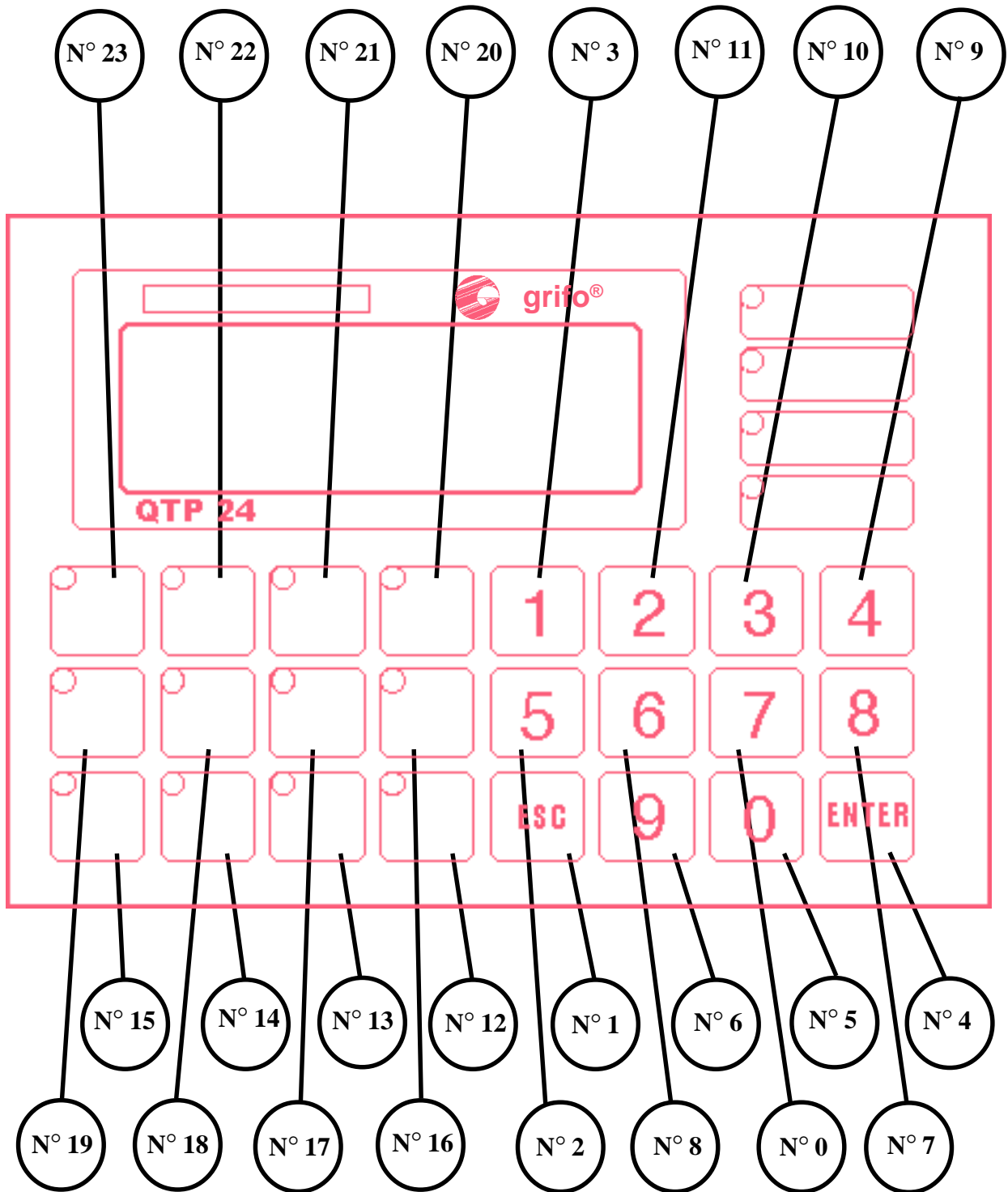


FIGURE 16: QTP 24P KEYS LOCATION

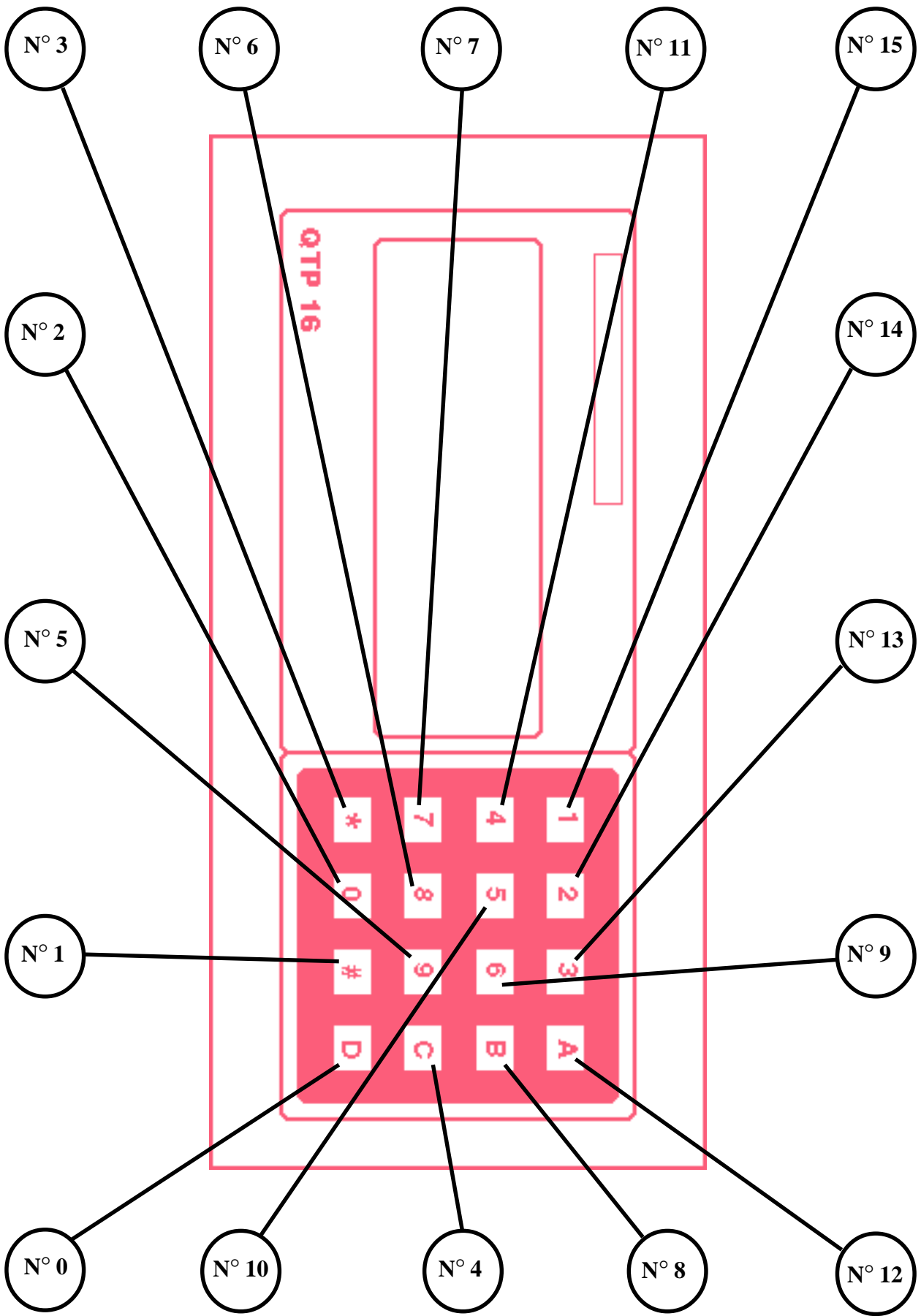


FIGURE 17: QTP 16P KEYS LOCATION

On the top of the work area memory there is a 24 bytes table where the key codes are saved and where the user can change the default values, as described in the following table:

| KEY NUMBER | KEY CODE ADDRESS | DEFAULT KEY CODE |
|------------|------------------|--------------------|
| 0 | FF67H | 37H = 55 = "7" |
| 1 | FF68H | 1BH = 27 = "ESC" |
| 2 | FF69H | 35H = 53 = "5" |
| 3 | FF6AH | 31H = 49 = "1" |
| 4 | FF6BH | 0DH = 13 = "ENTER" |
| 5 | FF6CH | 30H = 48 = "0" |
| 6 | FF6DH | 39H = 57 = "9" |
| 7 | FF6EH | 38H = 56 = "8" |
| 8 | FF6FH | 36H = 54 = "6" |
| 9 | FF70H | 34H = 52 = "4" |
| 10 | FF71H | 33H = 51 = "3" |
| 11 | FF72H | 32H = 50 = "2" |
| 12 | FF73H | 4CH = 76 = "L" |
| 13 | FF74H | 4BH = 75 = "K" |
| 14 | FF75H | 4AH = 74 = "J" |
| 15 | FF76H | 49H = 73 = "I" |
| 16 | FF77H | 48H = 72 = "H" |
| 17 | FF78H | 47H = 71 = "G" |
| 18 | FF79H | 46H = 70 = "F" |
| 19 | FF7AH | 45H = 69 = "E" |
| 20 | FF7BH | 44H = 68 = "D" |
| 21 | FF7CH | 43H = 67 = "C" |
| 22 | FF7DH | 42H = 66 = "B" |
| 23 | FF7EH | 41H = 65 = "A" |

FIGURE 18: KEYS ADDRESSES AND DEFAULT CODES

The key codes are the values returned by **GDOS 80** operating system when it recognizes a key depression in the interrupt scanning procedure and the user can change them to simplify the application program development. The default codes values saved in the table by **GDOS 80** coincide with the standard label of **QTP 24P** panel.

CONFIG.*: GDOS 80 CONFIGURATION

GDOS 80 can manage a configuration file and program to simplify the target card hardware setting. During start phase (after a reset or power on) **GDOS 80** checks if file CONFIG.SYS is available on one of the local disk, with the following order: N, M, O. If this file is not present, it sets the default configuration that are below listed:

| MCI 64: Not used | | | |
|------------------|---------------|---------------|-------|
| SERIAL LINE A | | SERIAL LINE B | |
| BAUD RATE: | max available | BAUD RATE: | 19200 |
| BIT x CHR: | 8 | BIT x CHR: | 8 |
| STOP BIT: | 2 | STOP BIT: | 2 |
| PARITY: | NO | PARITY: | NO |
| HANDSHAKE: | NO | HANDSHAKE: | NO |

FIGURE 19: DEFAULT GDOS 80 CONFIGURATION

while if the file is present, it is loaded and used to set the described hardware and software features. As you can see in the previous configuration list, the settings concern only **MCI 64** interface and the two serial lines: in fact these are the only sections available on all the target card that require a quite complex initialization procedure. The "HANDSHAKE" indication refers to automatic management of the hardware communication handshake signals (/CTS and /RTS), while "max available" denotes the target card higher baud rate. This max value can be found on the target card technical manual and it must be used to set **GET80** communication speed.

The configuration file CONFIG.SYS is generated by a proper **GDOS 80** program named CONFIG.G80, saved on the main root of the working disk, that must be used as described in the following points:

- a) Run the CONFIG.G80 program from **GDOS 80** operating system, by loading it from the drive where it is saved. At the end of the load phase a presentation window appears where it is shown the program version and the used target card.
- b) Select the disk drive where the CONFIG.SYS configuration file must be saved. To make this selection you can simply type the <right arrow> and <left arrow> keys, that automatically change the drive according to used card, and confirm with <ENTER>.
- c) If a previous CONFIG.SYS file is saved on the selected disk then its configuration data are loaded, viceversa the default configuration is loaded. At this point a window similar to figure 19 appears and the user can:
 - move the current configuration entry (display with high intensity attribute) through the <up arrow> and <down arrow> keys;
 - change the current configuration entry with the <right arrow> and <left arrow> keys: the current entry available values are changed and displayed in sequence;
 - exit from configuration program saving the selected configuration on the file CONFIG.SYS on drive selected at point **b**. To perform this operation you can select the "SAVE and EXIT" entry and confirm with <ENTER> key: the monitor is cleaned, the saving configuration file result is displayed and **GDOS 80** is reexecuted;
 - exit from configuration program without saving the selected configuration. To perform this operation you can select the "EXIT without SAVE" entry and confirm with <ENTER> key: the monitor is cleaned and **GDOS 80** is reexecuted.

- d) If you exit program saving the configuration file and the save operation is performed successfully, on the selected disk a CONFIG.SYS file will be available; if this file is copied on a target card disk (or it has been directly generated on a local disk) when the card restart and RUN mode is selected, the configuration data defined at point **c** are loaded and then setted on the hardware. Instead if DEBUG mode is selected only the **MCI 64** configuration is used and the serial lines are always setted with default values.

The HANDSHAKE entries of the configuration program enable or disable the automatic management of /CTS and /RTS hardware handshake signals (for example the autoenable feature of the serial communication device). They don't affect the **GET80** hardware handshake flag, saved in IOBYTE EXTENSION, that remain completely managed by user.

The configuration program CONFIG.G80 is supplied only in the **xGDOS xxx MCI** version, described in "GDOS 80 VERSIONS" chapter but it can be expressly required to **Grifo®**.

GDOS 80 TECHNICAL FEATURES

Here are reported all the technical features of **GDOS 80** software package; these informations are fully detailed in the other manual chapters and this page is only a brief summary:

- 1) Mass memory disks:** P.C. drives from **A** to **D**
RAM disk drive **L** on external memory card
RAM disk drive **M** on target card RAM
ROM disk drive **N** on target card EPROM or FLASH EPROM
RAM disk drive **O** on target card RAM

All these disks have different sizes and formats and for RAM disk it is managed also the write protect circuit.

- 2) Communication:**
- | | | | | |
|------------|---|---|----|---------------|
| | | Primary console device | -> | Serial line A |
| Baud Rate | = | max available, according to target card | | |
| Parity | = | None | | |
| Stop Bit | = | 2 | | |
| Bit x Chr. | = | 8 | | |
| Handshake | = | /CTS (disabled with IOBYTE EXTENSION.7) | | |
| | | Auxiliary console device | -> | Serial line B |
| Baud Rate | = | 19200 Baud | | |
| Parità | = | None | | |
| Stop Bit | = | 2 | | |
| Bit x Chr. | = | 8 | | |
| Handshake | = | None | | |

- 3) Work RAM area:** 64K bytes

- 4) Printer:** P.C. parallel printer
Local parallel printer

- 5) Watch Dog:** Retriggered during local disks files read operations

- 6) EEPROM:** Managed at high level through proper utility procedure

- 7) Operator interface:** LCD or VFD display
Matrix keyboard
Status LEDs

GDOS 80 VERSIONS

As all software and firmware packages, also **GDOS 80** is liable to continuous evolutions and modifications, with the best intent to satisfy the new user applications requirements and to eliminate the contingent discovered problems.

There are three base versions:

GDOS xxx

GDOS 80 software package for target card **GPC® xxx** on EPROM.

FGDOS xxx

GDOS 80 software package for target card **GPC® xxx** on FLASH EPROM.

GDOS xxx MCI and FGDOS xxx MCI

GDOS 80 software package for target card **GPC® xxx** on EPROM or FLASH EPROM, with memory card RAM disk management, through **MCI 64**.

FGD xxx and FGD xxx MCI

FLASH EPROM for target card **GPC® xxx** with the **GDOS 80** operating system saved on its first sector.

For the **GDOS 80** version number, the following brief description can be used:

- Ver. 1.0 -> First release.
- Ver. ≥ 1.9 -> Local mass memory disk drives addition (ROM disk, RAM disks).
- Ver. ≥ 2.5 -> Serial EEPROM and local printer management addition.
- Ver. ≥ 2.6 -> FLASH EPROM management addition.
- Ver. ≥ 3.0 -> External memory card RAM disk drives addition through **MCI 64**.
- Ver. ≥ 4.1 -> Changed communication protocol with **GET80**, to speed the develop phase.
- Ver. ≥ 4.5 -> Local operator interface management addition.
- Ver. ≥ 4.6 -> Data structure MMU setting addition.

Each possible addition or improvement that the user retains interesting, can be suggested contacting directly **Grifo®**.

GDOS 80 DATA STRUCTURES

In this chapter are described the data structures available in **GDOS 80** operating system, that simplify the programming and the use of the software package. Please remember that the structure below described are often used in the previous and next chapter of this manual.

F.C.B. (FILE CONTROL BLOCK)

This data structure is a 36 bytes sequence that identifies a file on all **GDOS 80** mass memory disks. The meaning of each byte is the following one:

| | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| DR | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | T1 | T2 | T3 |

| | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| EX | S1 | S2 | RC | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |

| | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| D8 | D9 | DA | DB | DC | DD | DE | DF | CR | R0 | R1 | R2 |

FIGURE 20: FILE CONTROL BLOCK

where:

- DR** = Drive number where the file is saved: this value changes between 0 (default drive), 1 (drive A) to 16 (drive P), in fact **GDOS 80** can use up to 16 disk drives.
- F1÷F8** = File name: they are 8 upper case ASCII characters that define the file name.
- T1÷T3** = File name extension: they are 3 upper case ASCII characters that define the file name extension.
- EX** = Reserved to **GDOS 80**.
- S1** = Reserved to **GDOS 80**.
- S2** = Reserved to **GDOS 80**.
- RC** = Reserved to **GDOS 80**.
- D0÷DF** = Reserved to **GDOS 80**.
- CR** = Reserved to **GDOS 80**.
- R0-R2** = Current record pointer for random accesses: it is in the range 0÷65535 with overflow to **R2** and low byte in **R0** and high byte in **R1**.

T.P.A. (TRANSIENT PROGRAM AREA)

It is a part of work area memory that is available at the user for his application program. In this area **GDOS 80** saves all the informations (data and code) of any programming language, utility program, application program and so on. The 64K bytes of memory directly addressable by target card that executes **GDOS 80** are so divided:

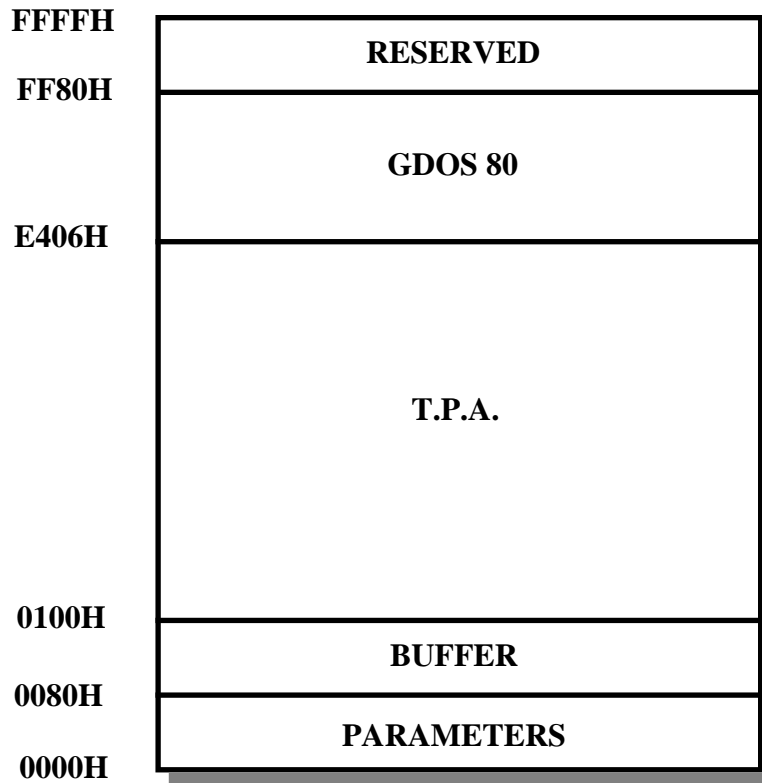


FIGURE 21: WORK AREA MEMORY MAP

The T.P.A. is the greater area of the work area memory in fact it is more than 56K bytes and it is always free for **GDOS 80** application program. When programming languages are used, they are loaded in this area from **GDOS 80** and here executed; also the program developed by the user are saved in T.P.A. area both during develop and final run phase.

To satisfy a frequent users request **GDOS 80** has a little memory area that is commonly not used by the same operating system. This area named **RESERVED** normally is the only area that is not modified by **GDOS 80** nor programming languages; in fact **GDOS 80** uses **RESERVED** area itself only for the interrupt vectors table, when local operator interface keyboard is used (as described in "LOCAL OPERATOR INTERFACE" paragraph). It can be used to save small data quantity, to exchange parameters between programs and procedures, to manage configuration data, etc. Moreover on many target cards the RAM memory is backed, so the contents of this area is mantained also in absence of power supply.

IOBYTE AND IOBYTE EXTENSION

GDOS 80 has a list of functions that manage at high level three logic device: a primary console, an auxiliary console and a printer. The high level programming languages use these functions through dedicated instructions that drive the devices in a transparent mode and simplify the application program development. The functions that manage the logic devices are numerous and they are fully described in next chapter, but they can be so summarized:

| | | |
|-------------------|----|--|
| primary console | -> | suspensive input function, input status function, output function; |
| auxiliary console | -> | suspensive input function, output function; |
| printer | -> | output function. |

With IOBYTE and IOBYTE EXTENSION data structures it is possible to associate the **GDOS 80** logic device with the target card physical devices. The list of these hardware devices and their management functions it is below reported:

| | | |
|--------------------------|----|--|
| serial line A | -> | suspensive input function, input status function, output function; |
| serial line B | -> | suspensive input function, input status function, output function; |
| local operator interface | -> | suspensive input function, input status function, output function; |
| local printer | -> | output function; |
| P.C. printer | -> | output function. |

By a contemporary use of IOBYTE and IOBYTE EXTENSION each logic devices can be associated to any hardware physical devices. In details:

IOBYTE: D7 D6 D5 D4 D3 D2 D1 D0

| | | |
|-------|----|---|
| D0 | -> | Associates primary console input device |
| 0 | -> | Primary console input device associated to serial line A |
| 1 | -> | Primary console input device associated to serial line B |
| D1 | -> | Associates auxiliary console input device |
| 0 | -> | Auxiliary console input device associated to serial line B |
| 1 | -> | Auxiliary console input device associated to serial line A |
| D3 D2 | -> | Associate primary console output device |
| 0 0 | -> | Primary console output device associated to serial line A |
| 0 1 | -> | Primary console output device associated to serial line B |
| 1 0 | -> | Primary console output device associated to P.C. printer |
| 1 1 | -> | Primary console output device associated to local printer |
| D5 D4 | -> | Associate auxiliary console output device |
| 0 0 | -> | Auxiliary console output device associated to serial line B |
| 0 1 | -> | Auxiliary console output device associated to serial line A |
| 1 0 | -> | Auxiliary console output device associated to P.C. printer |
| 1 1 | -> | Auxiliary console output device associated to local printer |

- D7 D6-> Associate printer device
 0 0-> Printer device associated to serial line A
 0 1-> Printer device associated to serial line B
 1 0-> Printer device associated to P.C. printer
 1 1-> Printer device associated to local printer

IOBYTE EXTENSION: D7 D6 D5 D4 D3 D2 D1 D0

- D0 -> Selects display type
 0 -> Selects LCD display
 1 -> Seleziona display VFD

If display type = LCD (D0=0):

- D1 -> Selects transmission type
 0 -> Selects transmission of characters to LCD display
 1 -> Selects transmission of commands to LCD display

If display type = VFD (D0=1):

- D1 -> Selects interface type
 0 -> Selects **QTP 24P** operator interface
 1 -> Selects **KDx x24, IAL 42, IAF 42, DEB 01** operator interfaces

- D2 -> Associates primary console out logic device
 0 -> Primary console out logic device associated through IOBYTE
 1 -> Primary console out logic device associated to local operator interface display

- D3 -> Associates primary console in logic device
 0 -> Primary console in logic device associated through IOBYTE
 1 -> Primary console in logic device associated to local operator interface keyboard

D4 -> Not used (set to 1)

D5 -> Not used (set to 1)

D6 -> Not used (set to 1)

- D7 -> Hardware handshake enable flag for **GET80** communication (see proper paragraph)
 0 -> Hardware handshake disabled
 1 -> Hardware handshake enabled

A physical device can be even associated to more logic devices, without any problems; the association variation (IOBYTE and IOBYTE EXTENSION programming) can be performed at any time with an immediate action. The IOBYTE EXTENSION associations have higher priority than IOBYTE associations, so if primary console logic device is associated to local operator interface physical device, then the IOBYTE D0, D2, D3 bits are not used and therefore their status is don't care.

The IOBYTE EXTENSION D7, D1, D0 bits don't directly concern the association between logic and physical devices, anyhow they concern the primary console device use; for further info please refer to "TERMINAL EMULATION" and "LOCAL OPERATOR INTERFACE".

Both IOBYTE and IOBYTE EXTENSION are located in **GDOS 80** parameters memory area, respectively at address **0003H** and **0008H**. They can be read and written from the user through the high level instructions of the used programming language.

By default **GDOS 80** respectively sets the IOBYTE and IOBYTE EXTENSION at **80H** and **F0H**, that associate primary console device to serial line A (both input and output), the auxiliary console device to serial line B (both input and output) and the printer device to P.C. printer.

Here follows two examples of devices reassociation:

| | | |
|---------------------------------|----|----------------------|
| primary console input device | to | serial line A |
| primary console output device | to | serial line B |
| auxiliary console input device | to | serial line B |
| auxiliary console output device | to | serial line A |
| printer device | to | local printer |
| IOBYTE | = | 11010100 = D4H = 212 |
| IOBYTE EXTENSION | = | 11110000 = F0H = 240 |

| | | |
|---------------------------------|----|--|
| primary console input device | to | local operator interface keyboard on QTP 24P |
| primary console output device | to | local operator interface VFD display on QTP 24P |
| auxiliary console input device | to | serial line B |
| auxiliary console output device | to | serial line B |
| printer device | to | local printer |
| IOBYTE | = | 1100xx0x = C0H = 192 |
| IOBYTE EXTENSION | = | 11111101 = FDH = 253 |

MMU SETTING

Another data structure of **GDOS 80** is the MMU SETTING that contains the current value programmed on the remote card Memory Management Unit circuit and it can be used in user application program to manage hardware interrupts combined with **GDOS 80** file system. On some target card the MMU circuit status can't be completely acquired through the I/O registers and when the **GDOS 80** manages the file system, it programs the MMU and it changes the memory configuration allocated in the 64K bytes addressed by microprocessor. For this reason any possible hardware interrupts can become active when in the 64K of memory, it is not allocated all the TPA area, obtaining unforeseeable results. The MMU SETTING is allocated in **GDOS 80** area at the address **FF66H** and it always contains the current settings of MMU circuit and of the other circuits managed by the same register (these circuits have registers allocated at the same I/O address).

To correctly manage the hardware interrupts combined with file system, the vectorized interrupt mode (IM 2) must be used and the application program developed by the user, in the interrupt service routine, must:

- Set the MMU circuit to allocate all the TPA area in the 64K addressed by microprocessor. This operation changes according to used target card and a detailed description is available on its technical manual.
- Execute the real interrupt service routine.
- Restore the memory configuration present during interrupt activation saved in MMU SETTING data structure. Also this operation changes according to used target card.

DISKS REDIRECTION

To add the opportunity to manage M, N, and O local target card drives also with programming languages that don't drive directly all the **GDOS 80** disks (01=A to 16=P), a drive redirection data structure had been provided. In details this data structure named **DEFDRI** is located in parameters memory area at address **0004H** and it changes the correspondence between logic disks (those managed by high level programming language and passed to **GDOS 80**) and physical disks. Only the first four logic disks can be redirected, as described in the following figure:

| DEFDRI | LOGIC DISK | PHYSICAL DISK | REDIRECTION |
|--------|---------------|---------------|-------------|
| 00H | A = 01 | A = 01 | Disabled |
| | B = 02 | B = 02 | Disabled |
| | C = 03 | C = 03 | Disabled |
| | D = 04 | D = 04 | Disabled |
| | E = 05 | E = 05 | Disabled |
| | : | : | : |
| | : | : | : |
| FFH | P = 16 | P = 16 | Disabled |
| | A = 01 | M = 13 | Enabled |
| | B = 02 | N = 14 | Enabled |
| | C = 03 | O = 15 | Enabled |
| | D = 04 | P = 16 | Enabled |
| | E = 05 | Not available | Enabled |
| | : | : | : |
| : | : | : | |
| P = 16 | Not available | Enabled | |

FIGURE 22: DISKS REDIRECTION

GDOS 80 initializes the redirection byte to **00H**, disabling all the redirection.

For example, if **NSB8** basic interpreter (that manages only the logic disks 01, 02, 03) is used and the target card RAM and ROM disk must be used, it is sufficient to:

- set the DEFDRI byte before any local disk operation: FILL 04,255<ENTER>
- use the RAM disk that is redirected to logic drive 01: SAVE TEST.B,1<ENTER>
- use the ROM disk that is redirected to logic drive 02: LOAD APPL.B,2<ENTER>
- first to enter **GDOS 80** command mode, please ensure that disks redirection is disabled, by resetting DEFDRI byte: FILL 04,00<ENTER>



GDOS 80 FUNCTIONS

The **GDOS 80** does all of the system input output for you. These services are available through a function list, they avoid knowledge of target card hardware and they can be directly used during application program development. All **GDOS 80** functions are requested by issuing a **CALL** instruction to location 0005H; in order to tell the operating system what you need it to do, you must arrange for the internal registers of the CPU to contain the required information before the **CALL** instruction is executed and you must get the returned parameters from the same register. The **GDOS 80** makes no guarantee about the contents of the other registers. If you need to preserve a value that is in a register, either store the value in memory or push it onto the stack. The **GDOS 80** uses its own stack space, so there is no need to worry about it consuming your stack.

All the functions are saved in the **GDOS 80** area and they are always serviceable. They are directly called by high level programming languages, but they are especially useful when the user wants to write assembly application programs; in fact in this case all the target card firmware drivers are already developed and ready to be used.

All the **GDOS 80** functions are described in the following paragraphs; if you need a fast reference please use figures 24 and 25 at the end of this chapter.

FUNCTION 0: SYSTEM RESET

Entry Parameters:

Register C: 00H

The system reset function returns control to the **GDOS 80** operating system at the command mode level. The operating system reinitializes the disk subsystem by selecting and logging in disk drive A.

This function has exactly the same effect as a jump to location **BOOT = 0000H**.

FUNCTION 1: PRIMARY CONSOLE INPUT

Entry parameters :

Register C: 01H

Returned value:

Register A: ASCII character

The primary console input function reads the next console character to register A. The input character is echoed to the primary console, too. The **GDOS 80** does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

FUNCTION 2: PRIMARY CONSOLE OUTPUT

Entry parameters:

Register C: 02H
Register E: ASCII character

The ASCII character from register E is sent to the primary console device.

FUNCTION 3: AUXILIARY CONSOLE INPUT

Entry parameters:

Register C: 03H

Returned value:

Register A: ASCII character

The auxiliary console input function reads the next character from the auxiliary console device into register A, without echo. Control does not return until the character has been read thus suspending execution if a character is not ready.

FUNCTION 4: AUXILIARY CONSOLE OUTPUT

Entry parameters:

Register C: 04H
Register E: ASCII character

This function sends the character from register E to the auxiliary console output device.

FUNCTION 5: PRINTER OUTPUT

Entry parameters:

Register C: 05H
Register E: ASCII character

The printer output function sends the ASCII character in register E to the printer device.

FUNCTION 6: PRIMARY CONSOLE DIRECT I/O

Entry parameters:

Register C: 06H
 FFH (input) or ASCII character (output)

Returned value:

Register A: Character or status

Primary console direct I/O is supported under **GDOS 80** for those specialized applications where basic primary console input and output are required. Upon entry the function 6, register E either contains hexadecimal FF, denoting a primary console input request, or an ASCII character. If the input value is FF, function 6 returns A = 00 if no character is ready, otherwise A contains the next primary console input character, which is not echoed. If the input value is E is not FF, function 6 assumes that E contains a valid ASCII character that is sent to the primary console.

FUNCTION 7: GET IOBYTE

Entry parameters:

Register C: 07H

Returned value:

Register A: IOBYTE value

The get IOBYTE function returns the current value of IOBYTE in register A. See proper paragraph for IOBYTE definition.

FUNCTION 8: SET IOBYTE

Entry parameters:

Register C: 08H
Register E: IOBYTE value

The set IOBYTE function changes the IOBYTE value to that given in register E.

FUNCTION 9: PRINT STRING ON PRIMARY CONSOLE

Entry parameters:

Register C: 09H
Register DE: String adress

The print string function sends the character string stored in memory at the location given by DE to the console device, until a \$ is encountered in the string.

FUNCTION 10: READ CONSOLE BUFFER FROM PRIMARY CONSOLE

Entry parameters:

Register C: 0AH
Register DE: Buffer adress

Returned value:

Primary console characters in buffer

The read buffer function reads a line of edited console input into a buffer addressed by registers DE. Primary console input is terminated when either input buffer overflows or a carriage return is typed. The read buffer take the form:

| | | | | | | | | | | | |
|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|-----------|
| DE: | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | ... | +n |
| | mx | nc | c1 | c2 | c3 | c4 | c5 | c6 | c7 | ... | ?? |

FIGURE 23: PRIMARY CONSOLE INPUT BUFFER

where mx is the maximum number of characters that the buffer will hold (1 to 254) and nc is the number of characters read (set by **GDOS 80** upon return), followed by the characters read from the console. If $nc < mx$, then uninitialized positions follow the last character, denoted by ?? in the above figure. A number of control functions are recognized during line editing:

| | |
|------------------------|---|
| <Ctrl+C> | reboots, entering GDOS 80 command mode |
| <Ctrl+H> = <BACKSPACE> | deletes the last inserted character |
| <Ctrl+M> = <ENTER> | terminates input line |

During buffer input each characters acquired from primary console is sent to the console output device as an echo.

FUNCTION 11: GET PRIMARY CONSOLE STATUS

Entry parameters:

Register C: 0BH

Returned value:

Register A: Console status

The console status function checks to see if a character has been typed on the primary console without suspending execution.

If a character is ready, the value 0FFH is returned in register A. Otherwise a 00H value is returned.

FUNCTION 12: RETURN VERSION NUMBER

Entry parameters:

Register C: 0CH

Returned value:

Registers HL: Version number

Function 12 provides information on the operating system version. To maintain compatibility the returned value is not the **GDOS 80** version but the version of emulated CP/M operating system.

A two bytes value is returned, with H = 00H and L = 22H.

FUNCTION 13: RESET DISK SYSTEM

Entry parameters:

Register C: 0DH

The reset disk function is used to programmatically restore the file system to a reset state where all disk are set to read / write (see functions 28 and 29), and an eventual error condition is cleared.

FUNCTION 14: SELECT DISK

Entry parameters:

Register C: 0EH

Register E: Selected disk

This function designates the disk drive named in register E as the default disk for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so on through 15, corresponding to drive P in a full 16 disk drives system. The drive is placed in an on line status, and all FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

FUNCTION 15: OPEN FILE

Entry parameters:

Register C: 0FH
Register DE: FCB address

Returned value:

Register A: Operation result

The open file operation is used to activate a file that currently exist in the disk specified in FCB data structure. The **GDOS 80** scans the referenced disk directory fo a match in position 1 through 11 of the FCB referenced by DE.

If a directory element is matched, the relevant directory information is copied on the FCB, thus allowing access to the files through susequent read and write operations. The user should note that an existing file must not be accesed until a succesful open operation is completed. Upon return, the open function returns a value with the value 0 if the open was succesful or 0FFH (255 decimal) if the file cannot be found.

Please remember that question marks in the FCB, match any directory character in this position, but it can't be used on local target card disk.

FUNCTION 16: CLOSE FILE

Entry parameters:

Register C: 10H
Register DE: FCB address

Returned value:

Register A: Operation result

The close file function performs the inverse of the open file function. Given that the FCB addressed by DE has been previously activated through an open or make function (see function 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a seccesful close operation is 0 while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to record the new directory information permanently.

FUNCTION 17: SEARCH FOR FIRST

Entry parameters:

Register C: 11H
Register DE: FCB address

Returned value:

Register A: Operation result

Search first scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found; otherwise 0 is returned indicating the file is present. When the file is found, the current DMA address is filled with the record containing the directory entry. Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through t3 matches the corresponding field of any directory entry on the default or auto - selected disk drive, but not for the local target card disks. If the dr fields contains an ASCII question mark, the auto disk select function is disabled and the default disk is searched, with the search function returning any matched entry. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory files.

FUNCTION 18: SEARCH FOR NEXT

Entry parameters:

Register C: 12H

Returned value:

Register A: Operation result

The search next function is similar to the search first function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 (hexadecimal FF) in A when no more directory items match and 0 otherwise.

FUNCTION 19: DELETE FILE

Entry parameters:

Register C: 13H
Register DE: FCB address

Returned value:

Register A: Operation result

The delete file function removes files that match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e. question marks in various positions), but only for P.C. disks. Function 19 returns a decimal 255 if the referenced file or files cannot be found; otherwise, a 0 value is returned.

FUNCTION 20: READ SEQUENTIAL

Entry parameters:

Register C: 14H
Register DE: FCB address

Returned value:

Register A: Operation result

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the read sequential function read the next 128 byte record from the file into memory at the current DMA address. The record is read from position cr of the extent, and the cr field is automatically incremented to the next record position.

The value 00H is returned in the A register if the read operation was succesful, the value 01H is returned if end of file occurs, while a FFH (255 decimal) value is returned if read operation is not succesfully executed.

FUNCTION 21: WRITE SEQUENTIAL

Entry parameters:

Register C: 15H
Register DE: FCB address

Returned value:

Register A: Operation result

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the write sequential function writes the 128 - byte data record at the current DMA address to the file named by the FCB. The record is placed at position cr of the file, and the cr filed is automatically incremented to the next record position. Write operations can take place into an existing file, in which case, newly written records overlay those that already exist in the file.

Register A = 00H upon return from a succesful write operation, while a FFH (255 decimal) value indicates an unsuccessful write, caused by a full disk, file not found, etc.

FUNCTION 22: MAKE FILE

Entry parameters:

Register C: 16H
Register DE: FCB address

Returned value:

Register A: Operation result

The make file operation is similar to the open file operation except that the FCB must name a file that does not exist in the currently referenced disk directory (i.e., the one named explicitly by a nonzero dr code or the default disk if dr zero). The **GDOS 80** creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication.

Upon return, register A = 0 if the operation was successful and FFH (255 decimal) if no more directory space is available or the file exist. The make function has the sideeffect of activating the FCB and thus a subsequent open is not necessary.

FUNCTION 23: RENAME FILE

Entry parameters:

Register C: 17H
Register DE: FCB address

Returned value:

Register A: Operation result

The rename function uses the FCB addressed by DE to changed all occurences of the file named in the firs 16 bytes to the file named in the second 16 bytes. The drive code dr at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is assumed to be zero.

Upon return, register A is set to a value 0 if the rename was successful and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

FUNCTION 24: RETURN LOG IN VECTOR

Entry parameters:

Register C: 18H

Returned value:

Register HL: Log in vector

The log in vector value returned by **GDOS 80** is a 16 bits value in HL, where the least significant bit of L corresponds to the first drive A and the high order bit of H corresponds to the sixteenth drive, labeled P. A 0 bit indicates that the drive is not on line, while a 1 bit marks a drive that is actively on line as a the result of an explicit disk drive selection or an implicit drive select caused by a file operation that specified a nonzero dr field.

FUNCTION 25: RETURN CURRENT DISK

Entry parameters:

Register C: 19H

Returned value:

Register A: Current disk

Function 25 returns the currently selected default disk number in register A. The disk number range from 0 through 15 corresponding to drives A through P.

FUNCTION 26: SET DMA ADDRESS

Entry parameters:

Register C: 1AH

Register DE: DMA address

DMA is an acronym for Direct Memory Address, and in **GDOS 80** comes to mean the address at which the 128 bytes data record resides before a disk write and after a disk read. Upon cold start, warm start, or reset, the DMA address is automatically set to 0080H. The Set DMA function, however, can be used to change this default value to address another area of memory where the data record reside; thus if the DMA address is changed by a subsequent set DMA function, the user doesn't needs to move the data record, between any call to read or write functions.

FUNCTION 27: GET ALLOCATION ADDRESS

Entry parameters:

Register C: 1BH

Returned value:

Register HL: Allocation vector address

An allocation vector is maintained in main memory for each on line disk drive. Various programming languages use the information provided by this vector to determine the amount of remaining storage. Function 27 returns the base address of a dummy fixed allocation vector for the currently selected disk drive, that doesn't contain valid information.

FUNCTION 28: WRITE PROTECT DISK

Entry parameters:

Register C: 1CH

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk before the next cold or warm start operation produces an error condition.

FUNCTION 29: GET READ ONLY VECTOR

Entry parameters:

Register C: 1DH

Returned value:

Register HL: Read only vector value

Function 29 returns a bit vector in register pair HL, which indicates drives that have the temporary read only bit set. As in function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The read only bit is set either by an explicit call to function 28.

FUNCTION 30: SET FILE ATTRIBUTES

Entry parameters:

Register C: 1EH

Register DE: FCB address

Returned value:

Register A: Operation result

The set file attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the read only and system attributes (t1 and t2) can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a match and changes the matched directory entry to contain the selected indicators. This function has effect only on P.C. disks, on local target card disk the attributes can be set but they are not managed.

Register A = 00H upon return from a successful attribute setting, while a FFH (255 decimal) value is returned in the other error conditions.

FUNCTION 31: GET D.P.B. ADDRESS

Entry parameters:

Register C: 1FH

Returned value:

Register HL: DPB address

The address of the **GDOS 80** resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

Function 31 returns the base address of a dummy fixed disk parameter block, that doesn't contain valid information.

FUNCTION 32: SET OR GET USER CODE

Entry parameters:

Register C: 20H

Register E: FFH (get) or user code (set)

Returned value:

Register A: Current user code (get) or no value (set)

An application program can change or interrogate the currently active user number by calling function 32. If register E = FFH, the value of the current user number is returned in register A, where the value is in the range of 0 to 15. If register E is not FFH the current user number is changed to the value of E (modulo 16).

GDOS 80 doesn't manage the user and if it is read a value = 0 is always returned.

FUNCTION 33: READ RANDOM

Entry parameters:

Register C: 21H

Register DE: FCB address

Returned value:

Register A: Operation result

The read random function is similar to the sequential file read operation, except that the read operation takes place at a particular record number, selected by the 24 bits value constructed from the 3 bytes field saved in the FCB (byte position r0 at 33, r1 at 34, and r2 at 35). The user should note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). Byte r2 must be zero, however, since a nonzero value indicates overflows past the end of file. Thus, the r0, r1 bytes pair is treated as a bubble byte, or word value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular records of the 8 megabyte file.

The user should note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record. The user can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Upon return from the call, register A either contains an error code FFH if file doesn't exist, 01 if the specified record is after end of file, or the value 00, indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record.

FUNCTION 34: WRITE RANDOM

Entry parameters:

Register C: 22H
Register DE: FCB address

Returned value:

Register A: Operation result

The write random operation is initiated similarly to the read random call, except that data are written to the disk from the current DMA address. As in the read random operation, the random record number is not changed as a result of the write. The logical extent number and current record position of the file control block are set to correspond to the random record that is being written. Again, sequential read or write operation can be following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. The user can also simply advance the random record position following each write to get the effect of a sequential write operation.

Upon return from the call, register A either contains an error code FFH if file doesn't exist, 06 if the disk is full, or the value 00, indicating the operation was successful.

FUNCTION 35: COMPUTE FILE SIZE

Entry parameters:

Register C: 23 H
Register DE: FCB address

Returned value:

Random record field (r0, r1, r2) set

This function can be used to compute the size of a file, identified from an FCB pointed by DE register pair. The FCB contains an unambiguous file name that it used in the directory scan.

Upon return, the random record bytes contains the file size, which is, in effect, the record address of the record following the end of the file; in details bytes r0 and r1 constitute a 16 bits value (r0 is the least significant byte, as before), that multiplied by a record dimension (128 bytes) gives the real file size. If referenced file doesn't exist, the r0, r1 and r2 bytes are returned zeroed.

Data can be appended to the end of an existing file by simply calling function 35 to get the random record position to the end of file and then performing a random writes starting at the present record address.

FUNCTION 36: GET RANDOM RECORD

Entry parameters:

Register C: 24H
Register DE: FCB address

Returned value:

Random record field (r0, r1, r2) set

The get random record function causes the **GDOS 80** automatically to produce the current record position from the file identified from FCB addressed by DE. The function can be useful in two ways: First, it is often necessary initially to read and scan a sequential file to extract the position of various "key" fields. As each key is encountered, function 36 is called to compute the current random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabulating the keys and their record numbers, the user can move instantly to a particular keyed record by performing a random read, using the corresponding random record number that was saved earlier. The scheme is easily generalized for variable record lengths, since the program need only store the buffer relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called, which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

If referenced file doesn't exist, the r0, r1 and r2 bytes are returned zeroed.

FUNCTION 37: RESET DRIVE

Entry parameters:

Register C: 25H
Register DE: Drive vector

Returned value:

Register A: 00H

The reset drive function allows resetting of specific disk drives. The passed parameter is a 16 bit vector of drives to be reset; the least significant bit is drive A and the resetted drives are those with a corresponding bit setted.

FUNCTION 40: WRITE RANDOM WITH ZERO FILL

Entry parameters:

Register C: 28H
Register DE: FCB address

Returned value:

Register A: Operation result

The write random with zero fill operation is similar to function 34, with the exception that a previously block data is filled with zeros before the data are written.

| FUNCTION | INPUT | OUTPUT |
|---|--|-------------------------|
| System reset | C = 00H | |
| Primary console input | C = 01H | A = ASCII character |
| Primary console output | C = 02H E = ASCII character | |
| Auxiliary console input | C = 03H | A = ASCII character |
| Auxiliary console input | C = 04H E = ASCII character | |
| Printer output | C = 05H E = ASCII character | |
| Primary console direct I/O | C = 06H E = FFH (input) or E = ASCII character | A = Character or status |
| Get IOBYTE | C = 07H | A = IOBYTE value |
| Set IOBYTE | C = 08H E = IOBYTE value | |
| Print string on primary console | C = 09H DE = String address | |
| Read console buffer from primary console | C = 0AH DE = Buffer adress | Update buffer |
| Get primary console status | C = 0BH | A = Console status |
| Return version number | C = 0CH | HL = Version number |
| Reset disk system | C = 0DH | |
| Select disk | C = 0EH E = Selected disk | |
| Open file | C = 0FH DE = FCB address | A = Operation result |
| Close file | C = 10H DE = FCB address | A = Operation result |
| Search for first | C = 11H DE = FCB address | A = Operation result |
| Search for next | C = 12H | A = Operation result |
| Delete file | C = 13H DE = FCB address | A = Operation result |
| Read sequential | C = 14H DE = FCB address | A = Operation result |
| Write sequential | C = 15H DE = FCB address | A = Operation result |
| Make file | C = 16H DE = FCB address | A = Operation result |

FIGURE 24: GDOS FUNCTION SUMMARY (1ST PART)

| FUNCTION | INPUT | OUTPUT |
|------------------------------------|--|--|
| Rename file | C = 17H DE = FCB address | A = Operation result |
| Return log in vector | C = 18H | HL = Log in vector |
| Return current disk | C = 19H | A = Current disk |
| Set DMA address | C = 1AH DE = DMA address | |
| Get allocation address | C = 1BH | HL = Allocation address |
| Write protect disk | C = 1CH | |
| Get read only vector | C = 1DH | HL = Read only vector |
| Set file attributes | C = 1EH DE = FCB address | A = Operation result |
| Get D.P.B. address | C = 1FH | HL = DPB address |
| Set or get user code | C = 20H E = FFH (get) or user code (set) | A = Current user code (get) or no value (set) |
| Read random | C = 21H DE = FCB address | A = Operation result |
| Write random | C = 22H DE = FCB address | A = Operation result |
| Compute file size | C = 23H DE = FCB address | r0,r1,r2 = last record number |
| Get random record | C = 24H DE = FCB address | r0,r1,r2 = current record number |
| Reset drive | C = 25H DE = Drive vector | A = 00H |
| Write random with zero fill | C = 0FH DE = FCB address | A = Operation result |

 FIGURE 25: GDOS FUNCTION SUMMARY (2ND PART)

APPENDIX A: LOCAL OPERATOR INTERFACE DIAGRAM

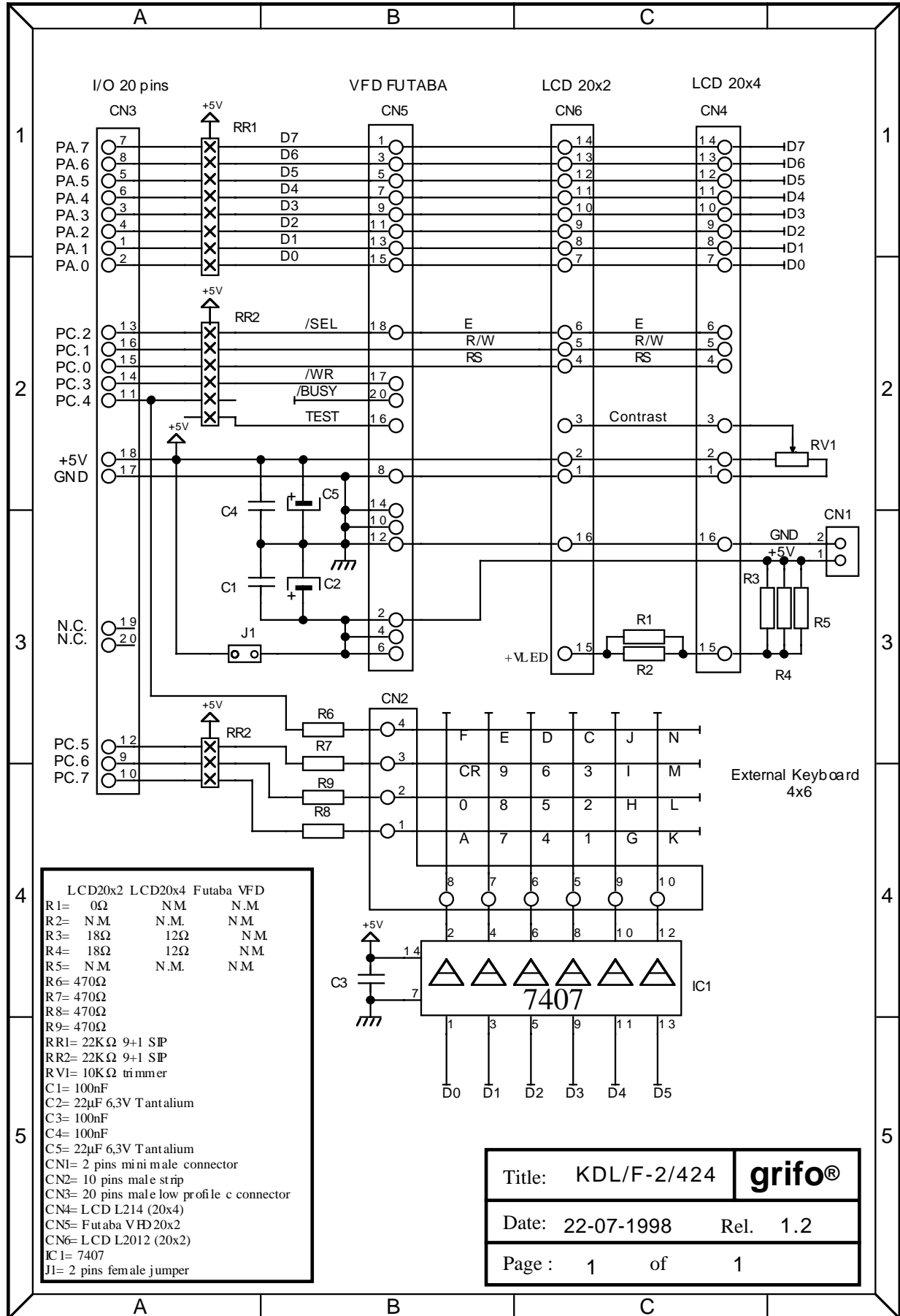


FIGURE A-1: KDX x24 ELECTRIC DIAGRAM



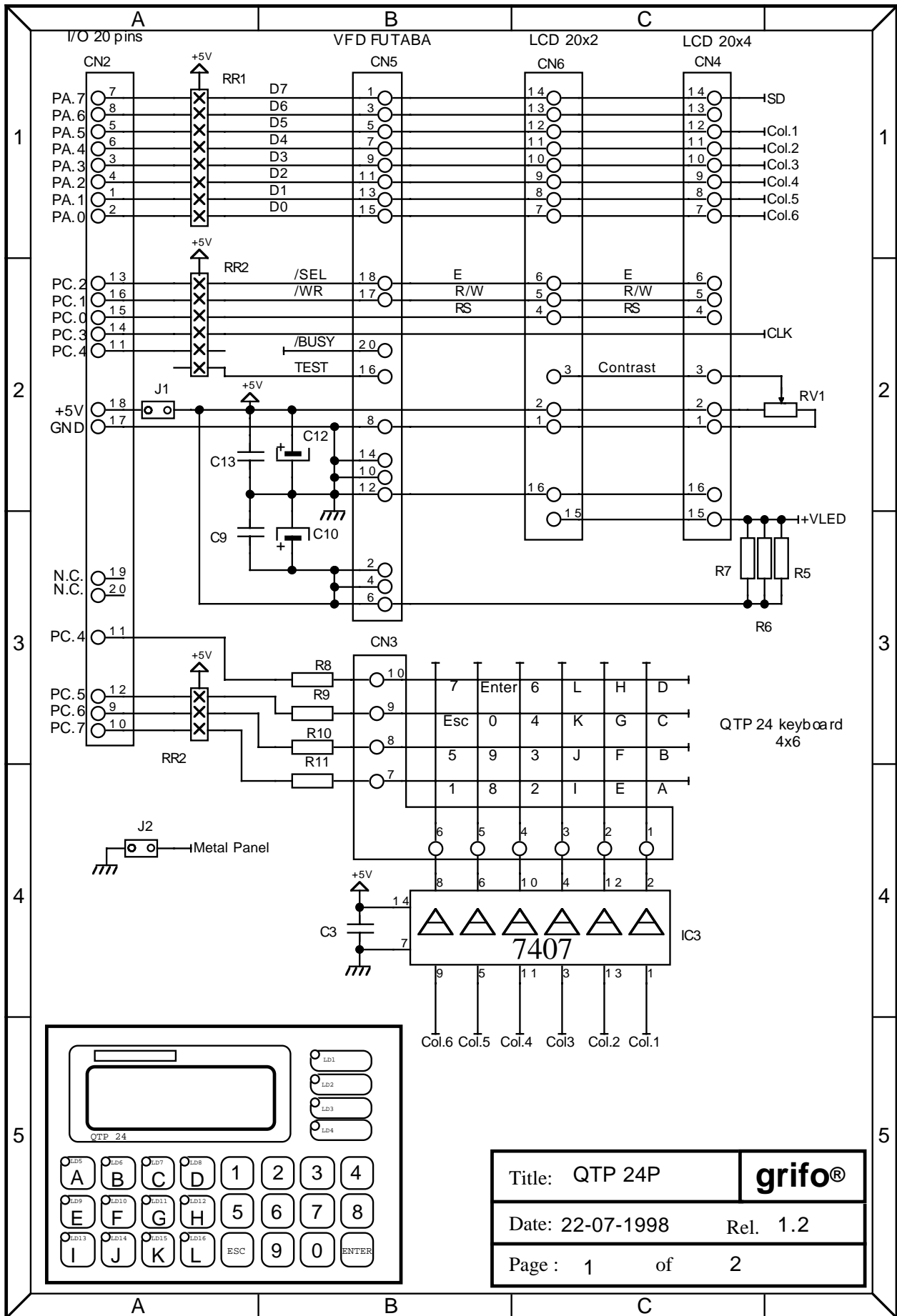


FIGURE A-2: QTP 24P ELECTRIC DIAGRAM (1ST PART)



| | |
|------------------|---------------|
| Title: QTP 24P | grifo® |
| Date: 22-07-1998 | Rel. 1.2 |
| Page : 1 | of 2 |

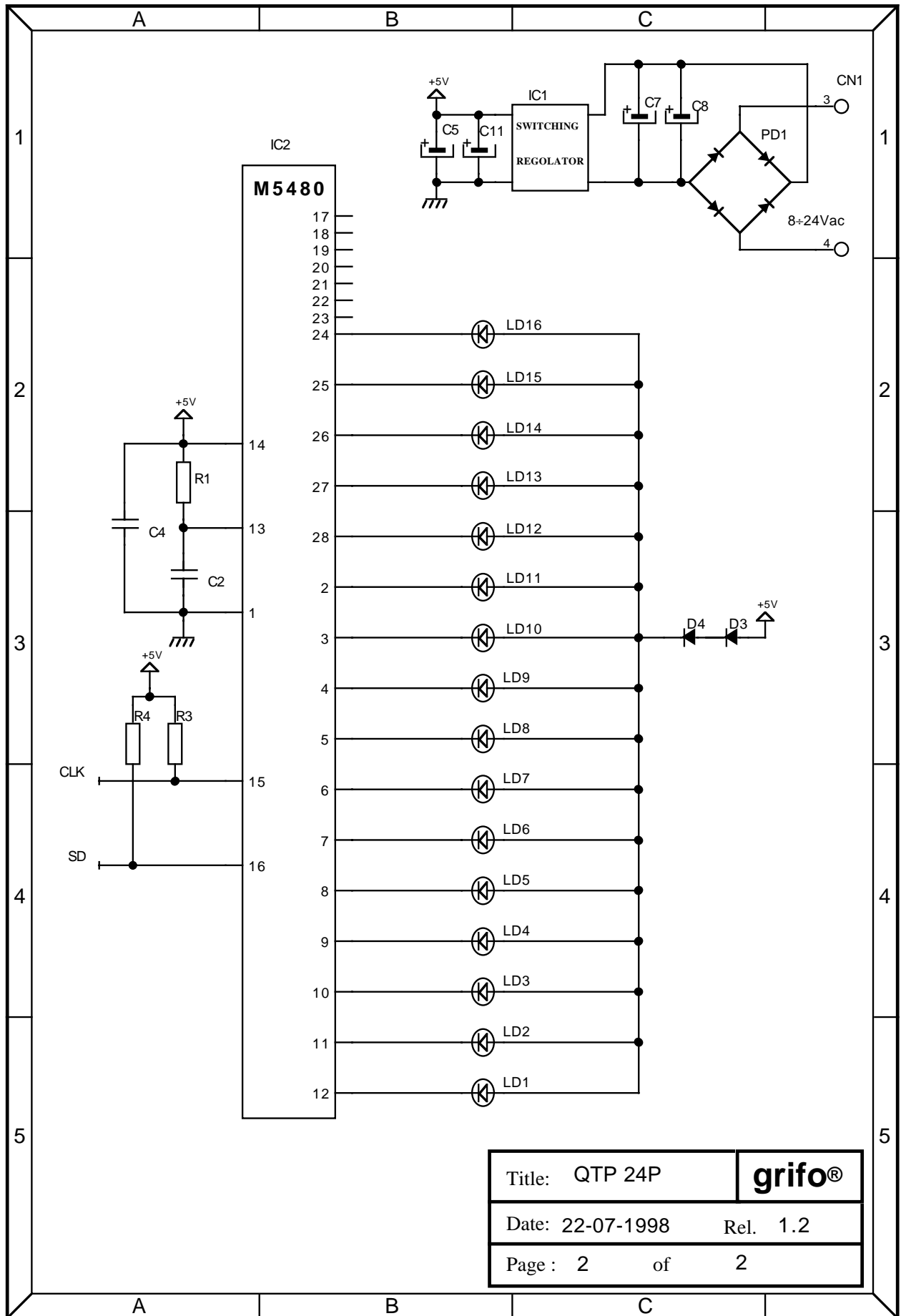
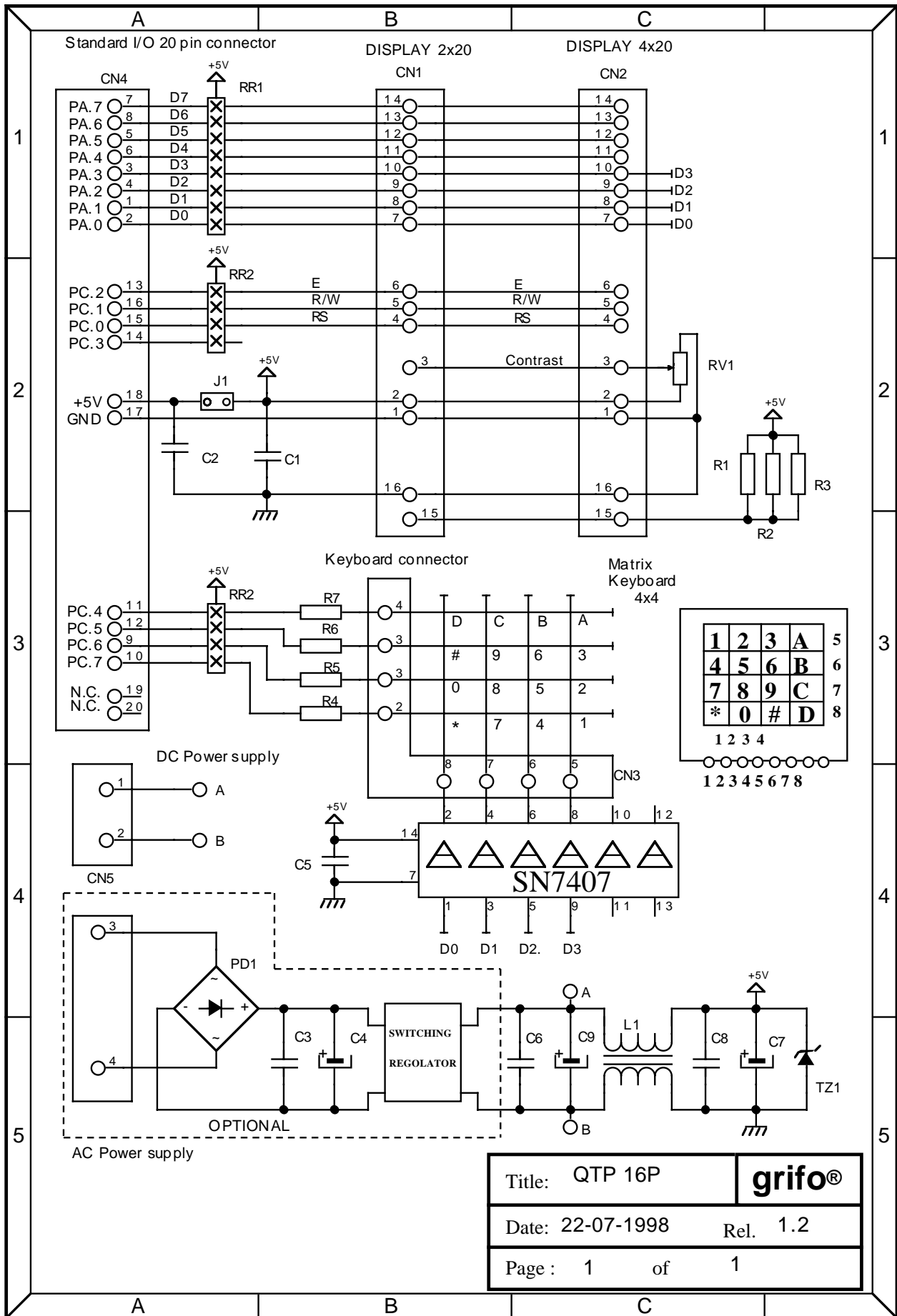


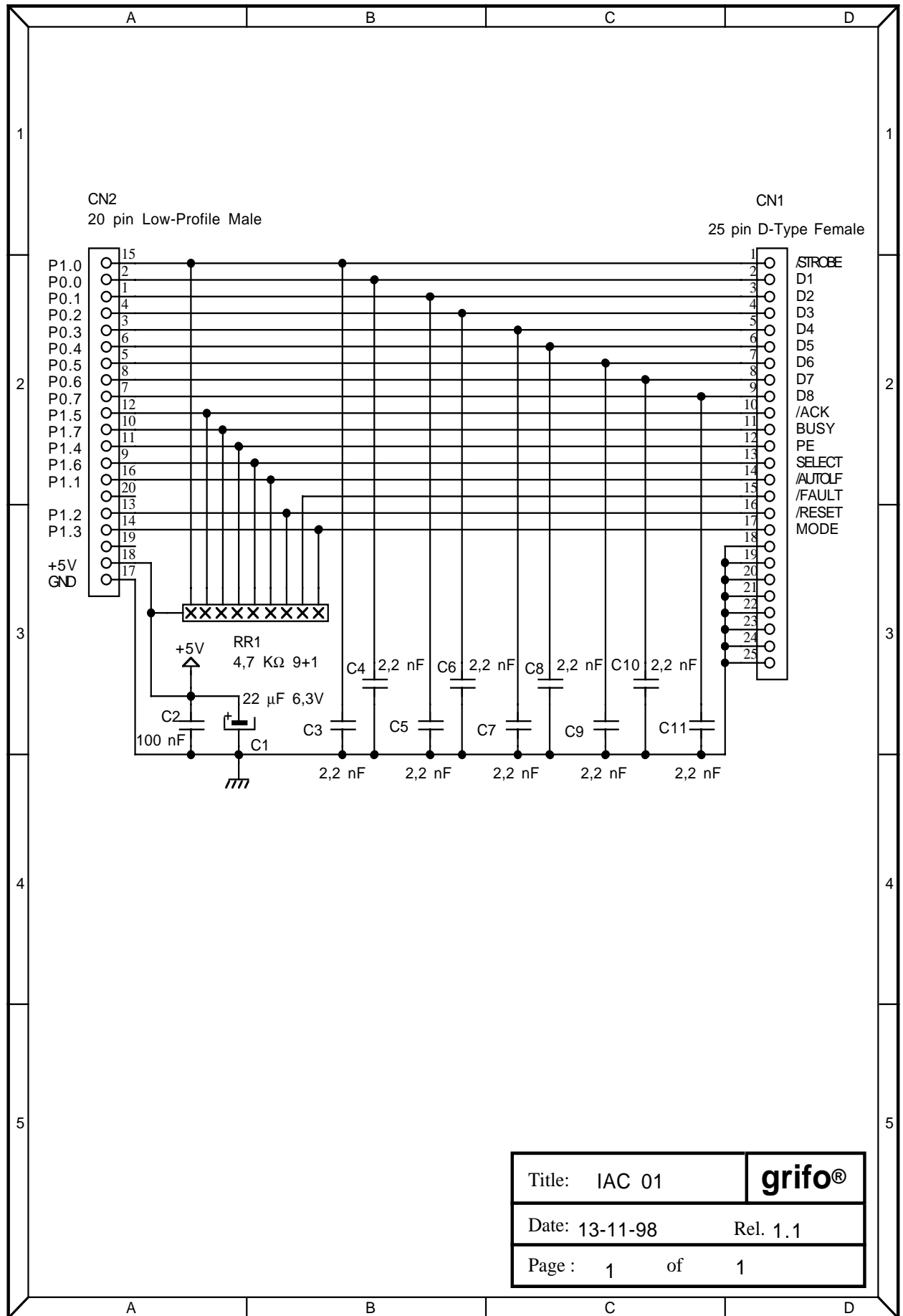
FIGURE A-3: QTP 24P ELECTRIC DIAGRAM (2ND PART)



| | |
|------------------|---------------|
| Title: QTP 16P | grifo® |
| Date: 22-07-1998 | Rel. 1.2 |
| Page : 1 | of 1 |

FIGURE A-4: QTP 16P ELECTRIC DIAGRAM





| | |
|----------------|----------|
| Title: IAC 01 | grifo® |
| Date: 13-11-98 | Rel. 1.1 |
| Page : 1 | of 1 |

FIGURE A-5: IAC 01 ELECTRIC DIAGRAM



APPENDIX B: ALPHABETICAL INDEX

A

ASSEMBLY 10

Assistance 1

AUTORUN 30

C

CBZ 80 8

CBZDEMO 8

CONFIG 36, 42

Console 3, 37, 48, 52, 53, 54, 55, 56

COPY 23

D

Data structures 46

DEBUG mode 30

Digital I/O interface 34

Dip switch 30

DIR 23

Direct commands 21

Disk set 7

Disks 28, 44, 56, 61, 62, 63, 66

Disks redirection 51

Display 37, 49

DOS2GDOS 20

E

Editor 14

EEPROM 3, 24, 44

EPROM 3, 4, 8, 28, 30, 31, 45

ERA 23

F

F.C.B. (File Control Block) 46

Features

General 2

Technical 44

FGROM 8, 33

File 57, 58, 59, 60, 61, 62, 64, 65

FLASH EPROM 3, 4, 8, 28, 30, 33, 45

FORMAT 23, 24

Functions 52, 69

G

GET80 1, 11, 13

GROM 20, 31



H

Handshake 17, 42
How to start 11
HTC 10

I

Installation 14
Introduction 1
IOBYTE 37, 48, 54
IOBYTE EXTENSION 17, 37, 49

K

KDx x24 37, 49, A-1
key 41
Keyboard 37, 49

L

LEDs 25, 27, 35
Limits 29

M

MCI 64 35, 42, 45
Menus 14
MMU setting 50
MODULA 2 9
Monitor 13
Mouse 13

N

NSB8 9

O

Operator interface 3, 25, 36, 44, 49

P

PASCAL 9
Personal computer 4
Printer 3, 13, 35, 44, 49, 53, A-5
Programming software 6

Q

QTP 16P 40, A-4
QTP 24P 26, 37, 39, 49, A-2

R

RAM card 24, 29, 35, 42, 45
RAM disk 3, 24, 28, 35
Release 1, 22, 45
Requirements 3

Reset 52
RIT 22
ROM disk 3, 28, 32, 33
RUN mode 30

S

SAVE 21
Serial cable 4
Serial lines 3, 4, 13, 42, 44

T

T.P.A. (Transient Program Area) 47
Target card 3, 5, 34
Terminal 13
Terminal emulation 17
 Attributes 18
 Commands 17
 Control sequence 18
 Special keys 19
Timer 38

U

Use 13
User manual 6
User string 19
Utility procedures 24
Utility program 22

V

VER 22
Versions 45

W

Warranty 1
Watch Dog 21, 44
Work RAM 44
Working disk 7
Working software 5

Z

Z80MU 7
ZBASIC 8
ZBDEMO 8

