M20 PERSONAL COMPUTER

PCOS

System Programmer's Guide

olivetti

M20 PERSONAL COMPUTER

PCOS

System Programmer's Guide

olivetti

PREFACE

This book describes the architecture and design concepts of the Professional Computer Operating System (PCOS). Moreover, it provides information on how to customize PCOS for particular installations or applications.

The book is directed at system programmers, system analysts, and, in parts of the book, interested non-programmers. Some parts of the book assume knowledge of assembly language concepts.

The book comprises three parts.

Part I provides a general overview of the software components that make up PCOS, and of the hardware options that PCOS manages.

Part II contains detailed descriptions of the software modules that make up PCOS, and of how those modules relate to the hardware.

Part III provides information that enables the user to customize PCOS.

REFERENCES:

PCOS (Professional Computer Operating System) User Guide
Code 3985280 D

Assembler User Guide
Code 3987670 L

DISTRIBUTION: General (G)

EDITION: December 1983

RELEASE: 3.0

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)

CONTENTS

PART I

1. INTRODUCTION

ABOUT THIS CHAPTER

This chapter provides an introduction to the manual and to the major features of PCOS.

CONTENTS

OVERVIEW

This section provides an introduction to this manual and to the major features of PCOS.

THIS MANUAL

This manual is intended to serve as a complete reference to PCOS for the system designer or system programmer who needs detailed knowledge of the internal functioning of the system. The information contained herein should enable a qualified person to modify the system so as to adapt or enhance its performance for a particular user environment or particular application. The manual provides a functional description of the overall system, its component parts, and the relationships among the parts. This information is different from that in the User Guide, but because some topics overlap, this manual can also be useful to systems analysts, and some parts to non-programmers.

Whereas the User Guide provides details on each command, the System Programmer's Guide differs in that it gives information and strategy on the use of commands in general approaches. For example, it explains how settings within the commands Set System (SSYS) and Set BASIC (SBASIC) interract and how the two commands interract together. With the wrong settings in SBASIC for memory and files, it is possible to run out of memory, although PCOS has plenty of memory available. These interractions are outside the scope of the User Guide.

The System Programmer's Guide is in three major parts: Part 1 gives a conceptual overview of PCOS, Part 2 gives an extended functional description, and Part 3 provides additional information on PCOS and gives reference information in a convenient form.

The contents of Part 1 provide the following information:

- The Introduction gives an overview of the manual and the major features of PCOS
- The System Overview explains the relationship of PCOS to other major elements of M20 software, and describes the major modules within PCOS
- The Command Overview lists the PCOS commands in functional groups
- The Hardware Configuration Options section gives a concise description of M20 standard configurations, options, and enhancements. Information is given from a programmer's viewpoint, in terms of system functions and capacities.

FEATURES OF PCOS

Because PCOS was developed for the existing, well-defined M20 computer system, its design resulted in an integrated and very flexible operating system.

Flexibility for programmers and users is assured by built-in commands that permit custom tailoring for specific installation or for specific applications. Configuration is easily accomplished by non-programmers. Commands required for a particular task can be loaded and saved, then written to diskette, so that a customized PCOS system can be used for a particular application.

Commands can be loaded from diskette into memory and retained so that they are readily available without the need for reloading each time they are required.

New commands and utilities are easily developed and configured into the system. Commands can be developed in assembly language or in BASIC.

PCOS provides internal consistency and design economy. The system, as supplied, supports 16 national keyboards, and has a built-in mechanism which permits design of new keyboards and displays. Interpretation of key codes, display of characters, and printing of characters are all controlled by common tables; therefore, a programmer can design a particular set of characters for use throughout the system. The national keyboards use the Roman or Greek alphabet plus any exclusive characters such as German umlauts and French accents. The system can also provide support for keyboards which have Cyrillic, Semitic, and Katakana (Japanese) symbols.

Similarly, the use of graphics is well integrated into the system. Graphics are supported by the monochrome and color monitors and by the thermal printer and some dot-matrix printers. The fundamental graphics capabilities, which are extensive, can be interfaced by the novice or non-programmer through BASIC and by the programmer through the system calls. A graphics package for PASCAL is also available. The package can be used separately with Assembly Language.

PCOS system design also provides drivers for extending system capabilities. The RS232-C driver supports communication with remote devices, and allows interfacing of peripherals or equipment using a serial bus. An IEEE driver supports interfaces with instrumentation for use in laboratory and engineering environments, and requires a parallel bus.

Certain internal system parameters can be adjusted by use of the set system global commands. Non-programmers can use these commands to make useful system modifications. These modifications can be temporary or can be made permanent.

The PCOS system provides extensive data protection, in addition to physical write-protection of diskettes. PCOS provides write protection of files and volumes of files on diskettes or the hard disk. In addition, files and volumes can be password-protected against being read or copied.

2. SYSTEM OVERVIEW

ABOUT THIS CHAPTER

This chapter provides an overview of the major elements of M20 software and the major functional modules of the Professional Computer Operating System (PCOS).

CONTENTS

OVERVIEW

This section provides an overview of the major elements of M20 software and the major functional modules of the Professional Computer Operating System (PCOS).

INTRODUCTION

Relationships of M20 software elements are illustrated by Figure 2-1.

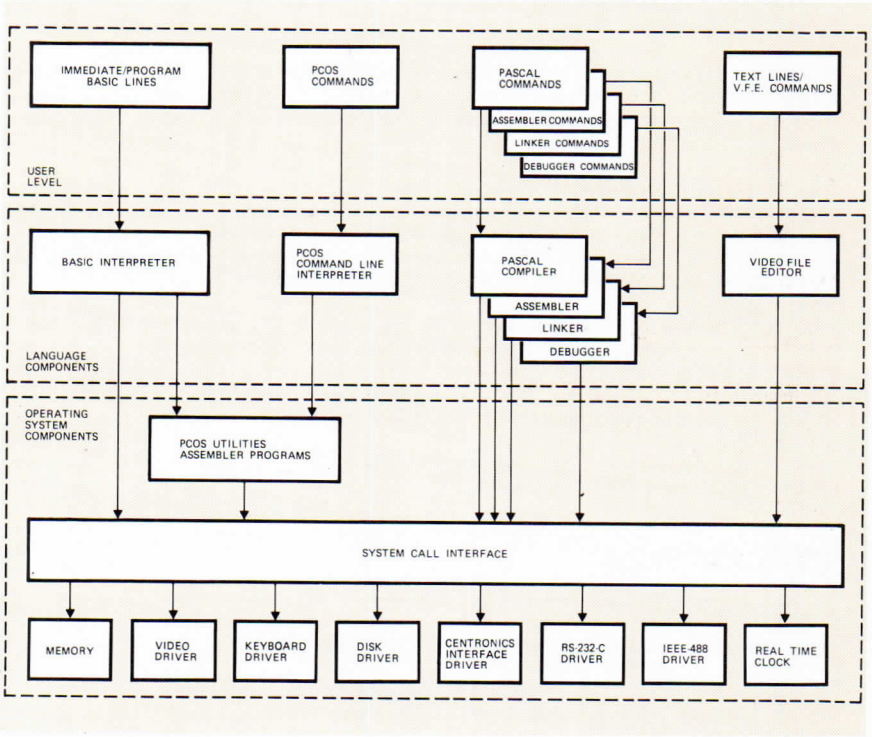


Fig. 2-1 Relationship of Software Elements

Users enter lines of text, which may be PCOS commands, BASIC program lines, assembly language statements, or text input to an editor program. PCOS commands are fundamental. The BASIC interpreter, the assembler, and the text editor are each brought into action by being executed as a PCOS command that is processed by the command line interpreter before execution. This manual describes the ramifications of PCOS commands, how they are entered and processed, and how they are supported by the operating system.

COMMAND LINE INTERPRETER

The command line interpreter examines a line of text input and interprets it as a command that may be followed by parameters that provide additional information for use in execution. The command line interpreter breaks up the line into the name of a command and its optional parameters, which are edited into a standard form for use by the command routine. The interpreter then checks for the presence of the named command routine in system memory or in the system directory for its diskettes or optional hard disk. This command routine is a program, usually written in assembly language but possibly in BASIC. If the command routine is available, the command line interpreter turns system control over to it. If not, it gives an error message.

A secondary function of the command line interpreter is to edit input lines. The control functions of backspace, line delete, etc., are taken care of. The command routine receives edited parameters rather than raw input.

From a user's viewpoint, a command is an instruction to the system to perform some action. The command may be the only entry on the line or it may have additional information. From the system viewpoint, a command is a module that may accept user-specified parameters and that calls on lower-level system elements to perform the desired action.

A unique feature of PCOS is the integration of BASIC with the PCOS system. If the command line interpreter sees a command file that is implemented in BASIC (shown by the .bas extension) then the command line interpreter will check for the presence of BASIC and load it if necessary before turning control over to the command routine.

Also, PCOS commands can be executed from a BASIC program through the use of the "EXEC" and "CALL" commands.

COMMANDS AND UTILITIES

In general data-processing usage, the term "command" typically refers to a built-in general-purpose system operation and the name "utility" to some useful special-purpose routine or program.

In PCOS, no distinction is made between "commands" and "utilities." The system design allows the user to configure the system, incorporating those commands needed for their specific "utility."

Commands are divided into two categories, "resident" and "transient." Resident commands are within the operating system and are always available. A transient command or utility is on a disk file and is available when the disk is present in the system.

In some systems, a good deal of manipulation of diskettes is necessary to configure a convenient set of commands and utilities on a diskette in order to have an appropriate set of commands and utilities available for a particular task. Doing this copying may require some time and thought.

PCOS has unique features to solve this problem. Two special commands, PLOAD and PSAVE allow any transient command or utility to be turned into a resident command. Using PLOAD, command routines can be taken from a diskette, loaded into memory, and retained by the system. The PLOADED commands are then 'resident' until the system is rebooted.

If the user so desires, this configuration can be PSAVED, which copies PCOS onto a new diskette, providing a customized operating system whenever the new diskette is booted. The PSAVE command thus allows permanent configuration of command sets. An extra value of this feature is that it can be executed by someone who has no knowledge of system programming.

To ensure maximum user memory capacity, standard PCOS has only three resident commands that are always loaded into memory during system initialization. They cannot be removed:

PLOAD loads transient commands into memory.
PUNLOAD removes PLOADed commands from memory.
LTERM, in BASIC programs, differentiates among the line-terminator keys /CR/, /S1/, and /S2/.

In Part 3 of this manual a section titled "Creating A Utility" explains how to write an assembly-language program that can be called on by the command line interpreter and can be used as a utility or a command.

SYSTEM CALL INTERFACE

PCOS commands and user-written assembly-language routines perform all functions such as data input and output, file management, memory allocation, etc., by the use of system calls passed through a general module, the system call interface. The interface is given the number of a system call and, if applicable, parameters that provide additional information.

Assembly-language commands specifying physical port addresses could be used for input/output functions such as reading from the keyboard and writing to the printer. Using the system call interface, however, allows the system to provide a common method for execution of such commands and for error recovery. The system call interface obviates the need for repetitious development of these functions and provides comprehensive services with these functions. The system call interface also provides internal scheduling in the handling of system functions.

The system call interface and system calls are described in part 2 of this manual.

DRIVERS

System drivers control input/output functions such as reading keyboard codes and writing files to a diskette or hard disk. Driver functions may be divided into low-level, or physical, and high-level, or logical functions. At the physical level, drivers control details of device operation; at the logic level, drivers control general system functions.

For example, at the physical level, the printer driver can cause the printer to advance paper to the top of a fresh page in either of two ways. If the printer is equipped with top-of-form capability, the physical driver can send the proper code to the printer to cause this action. If the printer does not have top-of-form capability, the physical function uses carriage returns to move the paper to the top of form a line at a time. The driver must keep track of its location on the page and be able to issue the correct count of carriage returns when called on by the logical driver for the top-of-form function.

The logical function for the printer (or printers) is insulated from such details of printer functioning. It deals with an idealized or abstract printer. At the system level, the system call and associated parameters are processed without regard for printer configuration.

The system call approach works at the logical functional level; the programmer is only concerned with identifying the device, the type of character, and certain control registers. The physical functional level is handled by utilities, which identify the printer model (SFORM), or modify the font (RFONT or SLANG), etc.

Similarly, the physical disk driver functions incorporating such information as size of sectors, number of sectors per track, number of tracks per surface, timing requirements to move from one part of the surface to another, etc, are handled by the utility commands. At the logical-driver level, with the appropriate system calls, the programmer merely treats the hard disk or diskette as a group of sectors or blocks that can be separately addressed.

Interaction among drivers is facilitated by logical design. For example, the keyboard driver supports any of a number of national keyboards. Each national keyboard has an associated table that gives the ASCII character for all keys and for combintations of keys. Combinations result from pressing a shift or control key in combination with another. The screen display driver, using the font table, produces the appropriate symbols. The symbols are defined by font tables. These symbols are stored on the disk file as standard ASCII codes. The ASCII code is associated with the raw keyboard input by the conventional relationship between the keyboard driver tables and the display driver. The keyboard tables and the font tables are accessible by system utilities. By calling the appropriate utility, existing keyboards can be modified, new keyboards can be developed, new character fonts and graphic symbols can be developed.

The printer driver can also interact with the display driver. Certain printers can output graphic images as well as text. The design of the drivers allows the printer driver to use screen-driver information. Driver interaction of the type described in these examples ensures efficient coordination of system resources.

Part 2 of this manual discusses system drivers and lists their associated system calls.

PCOS KERNEL AND MEMORY MANAGEMENT

The PCOS kernel, so named because it is the nucleus or heart of the system, is always resident. The kernel provides essential routines that permit interpretation of commands, memory management, and essential input/output. Other PCOS elements can be loaded as needed, but the kernel is required to load those elements.

Memory management is part of the kernel. Memory management keeps track of what system memory is in use and what is available. Other system routines call on memory management for memory space as needed and release space to memory management when done. User programs call on memory management by means of the appropriate system calls. Most system modules also use these system calls. To optimize capacity, transient commands are removed after execution, temporary PCOS tables are purged, and when a command or user program stops executing its memory space is automatically freed.

The assembly-language programmer accesses memory through memory management rather than directly. A special utility, DCONFIG (%M option), allows the programmer to find out the actual memory locations.

Part 2 of this manual provides more information on the system kernel and on memory management.

3. COMMAND OVERVIEW

ABOUT THIS CHAPTER

This chapter lists PCOS commands in functional groups. Each command is listed with its two-character short form, its full name, and a short description of its function.

CONTENTS

OVERVIEW

This section lists PCOS commands in functional groups. Each command is listed with its two-character short form, its full name, and a short description of its function. A PCOS command can be invoked by entering its first two letters or by entering more of its name. Only the first two letters are needed for most commands in searching for the command routine; the exceptions are FDISK, MSCOPY, and PLOT; for each of these at least three letters are required.

The section "Commands and Utility Programs" in Part 2 of this manual briefly explains how PCOS processes a command. In Part 3, a section called "Creating a Utility" explains how to develop new commands for PCOS. For detailed information on individual commands, see the PCOS Operating System User Guide, Section 13.

The three resident commands have no extensions. They are PLOAD, PUNLOAD, and LTERM. PLOAD loads commands with the .cmd extension into memory without executing them. PUNLOAD removes PLOADED commands. Commands with the .sav extension are SAVED the first time called, and stay in memory. They cannot be PUNLOADED. LTERM is called only from BASIC, and can be used to distinguish terminator values. This function can be useful in data-entry applications.

A few commands have .bas extensions. These commands run under the BASIC interpreter. When the user enters a .bas command, if BASIC is not already in the system PCOS will load BASIC to support execution of the .bas.

New PCOS commands and special PCOS utilities are continually being added. The following information cannot remain in current, but does provide a general overview of commands. For the most current commands, refer to current release diskettes.

Commands that can be used with a TV adaptor are flagged with (*). See the explanation at the end of the section.

PROGRAMMING TOOLS

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
as	ASM.CMD	Runs the Assembler
ba	BASIC.CMD	Loads the BASIC interpreter
bk	BKEYBOARD.BAS	Enables BASIC verbs to be entered by using single alpha key plus /COMMAND/
bv	BVOLUME.SAV	Allows BASIC to call specific subroutines
ed	EDIT.CMD	Loads the Video File Editor
hd	HDUMP.CMD	Prints a file in hexadecimal notation
li	LINK.CMD	Runs the Linker
ml	MLIB.CMD	Creates a library of object files
pd	PDEBUG.SAV	Loads the Program Debugger
te	TEXTDUMP.CMD	Prints a text file
ci	CI.SAV	Provides BASIC interface to RS232-C driver functions

PCOS CONFIGURING COMMANDS

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
pl	PLOAD (resident)	Loads commands from diskette or hard disk into semipermanent memory
pr	PRUN.CMD	Reloads an operating system; used to load an alternate version of PCOS
ps	PSAVE.CMD	Saves the current configuration of PCOS in memory to diskette or disk
pu	PUNLOAD (resident)	Unloads commands that were PLOADed

SET SYSTEM GLOBAL COMMANDS

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
sb	SBASIC.CMD	Sets the BASIC environment
sc	SCOMM.CMD*	Sets the RS232-C communications port environment
sd	SDEVICE.CMD	Changes device names
sf	SFORM.CMD*	Sets the printer environment
sl	SLANG.CMD*	Sets the national keyboard language
ss	SSYS.CMD*	Sets the system environment

KEYBOARD-RELATED COMMANDS

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
ck	CKEY.CMD	Changes the ASCII value of a key
lt	LTERM (resident)	Returns an integer (0, 1, or 2) depending on which of the three carriage return keys (/CR/, S1, S2) was last used
pk	PKEY.CMD*	Assigns a string to a key

The following utilities also relate to keyboards:

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
fo	FONT.ALL	Data file used by set language utility
ka	KANA.SAV	For Katakana keyboards
kb	KB.ALL	Data file used by set language utility

FILE MANAGEMENT COMMANDS - VOLUME HANDLING

KEYWORD		COMMAND FUNCTION
SHORT FORM	FULL NAME	
bv	BVOLUME.CMD	Searches the volume directory for file name string, or returns free disk space, or returns the name of the current volume (from BASIC only)
va	VALPHA.CMD	Alphabetizes a directory
vc	VCOPY.CMD*	Copies a volume (drive to drive)
vd	VDEPASS.CMD	Removes a password from a volume
vf	VFORMAT.CMD*	Formats a volume
vl	VLIST.CMD*	Lists a volume directory (full form)
vm	VMOVE.SAV*	Copies a volume (using one drive)
vn	VNEW.CMD*	Initializes a volume
vp	VPASS.CMD	Assigns a password to a volume
vq	VQUICK.CMD	Lists a volume directory (filename only)
vr	VRENAME.CMD	Renames a volume
vv	VVERIFY.CMD*	Checks the hard disk for faulty blocks

FILE MANAGEMENT COMMANDS - FILE HANDLING

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
fc	FCOPY.CMD*	Copies a file
fd	FDEPASS.CMD	Removes a password from a file
ff	FFREE.CMD	Frees unused file blocks
fk	FKILL.CMD	Deletes a file
fl	FLIST.CMD	Lists ASCII files, optionally lists hexadecimal files
fm	FMOVE.CMD*	Copies a file (disk to disk on single drive system)
fn	FNEW.CMD	Creates a new file (reserves blocks)
fp	FPASS.CMD	Assigns a password to a file
fr	FRENAME.CMD	Renames a file
fu	FUNPROT.CMD	Removes write-protection from a file
fw	FWPROT.CMD	Assigns write-protection to a file
rk	RKILL.CMD	Recovers a killed file
fdi	FDISK.CMD	Partitions the hard disk unit so it can be shared by PCOS, MS-DOS, CP/M-86, and UCSD p-system
msc	MSCOPY.CMD	Display a MS-DOS directory (from PCOS)
msd	MDIR.CMD	Copy a file from PCOS to MS-DOS, and vice versa

STANDARD INTERFACE HANDLING COMMANDS

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
ci	CI.SAV	Provides the BASIC interface to the RS232-C driver
ie	IEEE488.SAV	Loads the IEEE-488 package
rs	RS232.SAV	Loads the RS232-C package

PCOS GRAPHIC FACILITY COMMANDS

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
dp	DPALETTETV.CMD	Defines colors (TVA only)
la	LABEL.CMD*	Displays, magnifies, orients, and positions a label string
ls	LSCREEN.CMD	Transfers contents of display screen to printer
rf	RFONT.CMD*	Creates a new ASCII font matrix file from the currently active font
sp	SPRINT.CMD*	Prints the text and graphic contents of a specified window
wf	WFONT.CMD*	Makes the new font matrix file active

USER AIDS

KEYWORD		
SHORT FORM	FULL NAME	COMMAND FUNCTION
dc	DCONFIG.CMD*	Displays the hardware and/or memory configuration
ep	EPRINT.SAV*	Displays error messages

TV ADAPTOR COMMANDS (*)

Another version of these commands, (commands with *), is available for the TV adaptor. This adaptor enables a regular TV set to be used in place of the video screen. Because of distortions caused by the different interface used, these commands have been modified for the TVA. When both versions of the command are on the same diskette, the TV version will have TV in the command name; for example, SCOMMTV.CMD.

The command DPALETTETV.CMD may be used only with the TVA.

The command LABEL.CMD is available in two additional versions, depending on the TV screen refresh rate. LABELT5.CMD is for 50 megahertz and LABELT6.CMD is for 60 megahertz.

4. HARDWARE CONFIGURATION OPTIONS

ABOUT THIS CHAPTER

This chapter gives a concise description of M20 hardware configurations, including options and enhancements. Information is given from a programmer's viewpoint, in terms of system functions and capacities.

CONTENTS

OVERVIEW

This section gives a concise description of M20 hardware configurations. It comprises standard configurations, options, and enhancements. The section includes a list of available printers.

Information is given from a programmer's viewpoint, that is in terms of system functions and capacities. Hardware specifications are not given.

MINIMUM CONFIGURATION

The M20 system consists of a display screen and keyboard and a chassis containing the Z8000 CPU mounted on the motherboard.

The motherboard also contains 128 Kbytes of RAM, serial (RS232-C) and parallel (industry standard) interfaces, and floppy disk control logic. This board also contains five expansion slots. Two slots are for optional interface boards or an alternate processor board, and three slots are for memory expansion.

The standard video display screen is monochrome.

The minimum configuration is one diskette drive, nominal capacity 160 Kbytes, 320 Kbytes, or 640 Kbytes. However, the unit is usually sold equipped with two drives.

KEYBOARD

There are 16 national keyboards available, based on the Roman alphabet. Non-Roman versions have been developed for Semitic alphabets, Katakana characters, etc.

The national keyboards support 95 characters, with additional characters software-definable. Software utilities allow reconfiguring the keyboard, reassigning characters and keys, and creating new character fonts and keyed graphic display elements.

DISPLAY SCREEN

The screen can be monochrome or color. The color display supports eight possible colors: red, green, yellow, blue, magenta, cyan, black, and white. Monochrome and color versions have the same alphanumeric and graphics characteristics and are software compatible. The system supports two color configurations, four-color and eight-color. Four-color supports any four colors at a time, eight-color supports all colors.

Color displays require one or two additional memory boards, either 32 Kb or 128 Kb. One expansion board can support four colors, two boards are required for eight. Within each 32 Kb or 128 Kb board, 16 Kb are reserved to support four colors and the remainder is available for system memory use.

The screen is 12 inch diagonal with 256 horizontal rows (scanlines) containing 512 dots each. These dots are called "pixels," short for "picture elements." Text and graphics characters are both built from pixels and are treated alike by the system. This allows development of new character fonts for screen display by developing font tables.

PCOS supports either 16 lines of text with up to 64 characters per line, or 25 lines of text with up to 80 characters per line.

DISK DRIVES

The unit typically has two floppy disk drives, and can support an optional 5-1/4" Winchester hard disk. The hard disk unit may replace one of the floppy disk units or may be in a separate enclosure. It requires additional hardware control logic, which occupies one expansion slot. The software disk driver program supports the following floppy disk and hard disk configurations:

1 160-kbyte floppy drive
2 160-kbyte floppy drives

1 320-kbyte floppy drive
2 320-kbyte floppy drives
1 320-kbyte floppy drive, 1 hard disk drive
2 320-kbyte floppy drives, 1 hard disk drive

1 640-kbyte floppy drive
2 640-kbyte floppy drives
1 640-kbyte floppy drive, 1 hard disk drive
2 640-kbyte floppy drives, 1 hard disk drive

As provided by the factory, the hard disk unit is placed in one of the diskette drive compartments.

ROM 2.0 is required for support of the hard disk, and for 160 Kb and 640 Kb floppy disk drives.

The floppy disk capacities are nominal unformatted capacities. Formatted capacities are:

Unformatted	Formatted
160 Kb	143 Kb
320 Kb	286 Kb
640 Kb	592 Kb

The hard disk, nominally 11.26 megabytes, has a formatted capacity of 8.85 megabytes.

Configurations with floppy disk drives of different sizes intermixed are not supported by the disk driver. The driver supports any floppy disk drive in combination with the hard disk drive. However, Olivetti does not necessarily market all possible combinations.

The disk driver will work with all memory and display configurations.

The drive identifiers are 0 and 1 for floppy disks and 10 for the hard disk.

MEMORY

The central processing unit board includes 128 Kbytes of RAM. The motherboard has three memory expansion slots. Memory expansion boards have a capacity of 32 Kbytes or 128 Kbytes. Expansion boards cannot be mixed; all must be 32 Kbytes or 128 Kbytes, giving a system maximum of 224 Kbytes or 512 Kbytes. The color monitor uses expansion memory, one board for 4-color and two boards for 8-color.

Jumpers are used to inform the system of the memory configuration. The seven fundamental cases are shown in the table below.

Case	Configuration	Expansion Boards	Jumper Code
1	Standard 128 Kbyte memory only	0	5
2	32 Kbyte expansion board(s), black and white display	1-3	7
3	32 Kbyte expansion board(s), 4-color display	1-3	3
4	32 Kbyte expansion board(s), 8-color display	2-3	2
5	128 Kbyte expansion board(s), black and white display	1-3	6
6	128 Kbyte expansion board(s), 4-color display	1-3	1
7	128 Kbyte expansion board(s), 8-color display	2-3	0

Table 4-1 Memory Configuration

As the table shows, there are seven fundamental configurations, each with either no expansion or a range of possible expansion. Actual memory configuration within the possible range is determined by the system diagnostics when the system is started up.

Details of memory configuration are given in the "Memory Configuration" section of Part 2.

PRINTERS

There are four kinds of printers: spark ink jet, dot-matrix impact, thermal, and daisy wheel. Printers are connected to the parallel port, using the industry-standard Centronics interface method, except for certain cases where the serial port with the RS232-C interface is used.

The thermal, inkjet, and certain dot-matrix printers support printing of graphics, and one supports color printing. Printers available are listed later in this section under the heading "System Printers Available."

It is possible to connect two printers to the system, one on the parallel interface and another on the serial RS232 interface. The printer driver routines can be set to support either printer, in alternation, by using the SFORM utility. Another approach is to use the printer driver to support the parallel printer and use the RS232 driver, with appropriate separate software and support routines for a serial printer.

AUXILIARY INPUT/OUTPUT

These optional interface boards are in addition to the serial and parallel interfaces provided in the basic system. One or both boards can be placed in the two interface expansion slots on the motherboard.

RS232

Dual serial interface (RS232-C) and/or Current Loop. The board supports any of these combinations:

-- Two RS232-C interfaces
-- Two current loop interfaces
-- One RS232-C and one current loop interface

IEEE

IEEE 488 parallel interface.

ALTERNATE PROCESSOR BOARD

Uses the 8-MHz 8086 CPU and supports alternate operating systems, CP/M-86, MS-DOS, and UCSD p-system. This board is placed in one of the interface expansion slots, and requires ROM 2.0 or later.

TRADEOFFS

System configuration tradeoffs are summarized below:

- Color displays require one or two memory expansion boards. Four-color systems require 16 Kbytes of dedicated memory; eight-color systems require two boards and use 16 Kbytes from each.
- The two interface expansion slots support any two of the following four boards. Only one board of each kind is supported:
  - Hard disk controller board
  - Dual serial board
  - IEEE board
  - Alternate processor board
- As provided by the factory, the hard disk replaces one diskette drive.

SYSTEM PRINTERS AVAILABLE

DOT MATRIX IMPACT

Mnemonic	Description
PR 1450	Dot-matrix printer with graphics capability -normal characters (7 x 7) printed within a 9 x 7 dot matrix -10, 12.5, or 16.6 characters per inch -80 or 132 characters per line -6 printed lines per inch Speed: 80 characters/second at 10/inch; 100 characters/second at 12.5/inch; 132 characters/second at 16.6/inch

PR 1471	Dot-matrix printer with graphics capability -normal characters (7 x 7) printed within a 9 x 7 dot matrix -double width characters (under program control) -10, 12, or 16.6 characters per inch -132, 158, or 220 characters per line -6 or 8 lines printed per inch -140 characters per second print speed
PR 1481	Dot-matrix printer with graphics capability -normal characters (7 x 7) printed within a 9 x 7 dot matrix -double width characters (under program control) -10, 12, or 16.6 characters per inch -132, 158, or 220 characters per line -6 or 8 printed lines per inch -140 characters per second print speed -two and four color printing

SPARK INK JET

Mnemonic	Description
PR 2300	Dot-matrix printer with graphics capability -characters printed within either 7 x 7 or 7 x 5 dot dot matrix -80, 97, or 147 characters per line -10, 12.2, or 18 characters per inch -double width and double height character options

	-6, 8, or program controlled printed lines per inch -50 lines (of 80 characters) per minute, average

THERMAL

Mnemonic	Description
PR 2400	Dot-matrix printer with graphics capability -characters printed within a 7 x 5 dot matrix -80 characters per line -10 characters per inch -6 printed lines per inch -240 lines per minute print speed

DAISY WHEEL

Mnemonic	Description
PR 320	Bidirectional printer -10, 12, or 15 characters per inch or proportional spacing -132 characters per line at 10 characters per inch; 158 characters/line at 12 characters/inch; 197 characters/line at 15 characters/inch -6 printed lines per inch 25 characters per second print speed, average; 30 characters/second peak

PART II

5. OVERVIEW

ABOUT THIS CHAPTER

This chapter describes the contents of Part 2, which is the heart of the System Programmer's Guide, and contains an extended functional description of PCOS.

CONTENTS

GENERAL OVERVIEW

Part 2 is the heart of the System Programmer's Manual. It contains an extended functional description of PCOS, starting from the outside and moving in.

Part 2 begins with general discussion of major system elements: the command line interpreter, commands and utilities, memory management, the system call interface, and device rerouting. Then the device drivers, which underly the system calls, are discussed with a section each for the major drivers.

At the end of Part 2 is an explanation of the graphics subsystem, which is implemented for PASCAL, and its library of subroutines which can be used by assembly-language programmers. The graphics subsystem is based on the graphics system calls, which are described briefly in the Video Driver section. The graphics subsystem, useful in its own right, is also an example of how the system resources present in PCOS can be used to develop further resources.

DETAILED CONTENTS

COMMAND LINE INTERPRETER

The command line interpreter is the part of PCOS that works directly with the user. It gives the user access to PCOS commands and utilities and to other programs that make use of PCOS capabilities. This section describes the functions of the command line interpreter and provides some information about its implementation.

COMMANDS AND UTILITY PROGRAMS

This section gives background information on the distinctions between resident and transient commands and on the use of the PCOS utilities PLOAD, PUNLOAD, and PSAVE. The section includes information about command name conventions and requirements.

MEMORY CONFIGURATION

This section explains the physical implementation of system memory in functional terms. The information is provided for background, and is not necessary for making use of memory when using the PCOS system calls for storage allocation. This section bridges the gap between information about Z8000 memory handling concepts in the vendor literature and the PCOS implementation of memory management.

MEMORY MANAGEMENT

This section is a companion to the prior one, and gives practical information about the use of system storage allocation. It provides some information about how these functions are implemented.

SYSTEM CALLS

System calls are PCOS procedures used to handle input/output and to manage system resources such as memory and the real-time clock. System calls can be accessed by assembly language calls. They provide indirect access to privileged instructions that cannot be directly used.

This section gives an overview of the system calls grouped by general functions, and provides background information on their use.

DEVICE REROUTING

Device rerouting allows the rerouting of standard input (the keyboard), or standard output (the display screen), or both. Certain other input or output devices can be substituted for these two devices or used to supplement them.

This section gives information on how and why to use device rerouting, and provides some information on how this capability is implemented.

THE KEYBOARD DRIVER

This section is the first of five driver sections. It describes the keyboard driver routines and their related utility programs. The driver takes raw codes generated by the keyboard and interprets them as ASCII characters. PCOS supports numerous national keyboards. A utility called SLANG (Set Language) provides the keyboard driver with the information needed to support any national keyboard. Two other utilities, CKEY (Change Key) and PKEY (Program Key) allow easy user modification and enhancement of keyboard functions.

VIDEO DISPLAY DRIVER

The video display driver supports both text and graphic display. This section describes the capabilities of the driver and its related utilities, RFONT and WFONT. RFONT allows creating customized characters and small graphics characters, and can be used to create entire alphabets. WFONT is used to select an alternate display font. The section includes information about the system calls used to display text and to provide graphics capabilities.

DISK DRIVER AND FILE MANAGEMENT

The disk driver controls the floppy disk drives and the optional hard disk drive. Based on the disk drive functions, PCOS provides file management capabilities. This section explains the capabilities of the disk driver, and gives information about the system calls for using it and the file management capabilities.

OTHER DRIVERS

This section contains background information on two device drivers, the RS232-C driver and the IEEE-488 driver. These drivers are not seen directly by the user. The RS232 driver supports the SCOMM package and the CI calling BASIC. The IEEE driver supports the IEEE commands in BASIC. The programmer could, if necessary, access these drivers through system calls. System call information is given in this section.

THE PRINTER DRIVER AND PRINTER MANAGEMENT

This section describes the capabilities of the printer driver and the use of its associated utility programs. The driver supports the printing of ASCII text and (on certain printers) of graphics, using either a parallel or serial interface. The SFORM utility gives the user control over printer configuration parameters. LSCREEN allows the printing of display screen text and SPRINT allows the printing of the entire screen contents, including graphics.

GRAPHICS SUBSYSTEM

This section describes an extensive set of library routines that are used in PASCAL to provide graphics. These routines can also be accessed via assembly language calls. In addition to describing the capabilities of these routines, the contents of this section provide an example of the use of the PCOS graphics system calls which underlie these routines.

6. COMMAND LINE INTERPRETER

ABOUT THIS CHAPTER

This chapter provides a functional description of how the command line interpreter works.

CONTENTS

OVERVIEW

The command line interpreter (CLI) receives keyboard input from the user, interprets the user request, and calls on PCOS command routines to fulfill the request. This section provides a functional description of how the command line interpreter works. The section "Creating A Utility," in Part 3 of this manual, contains more detailed information on the internal appearance of the parameter information that the command line interpreter passes to the command routine.

THE COMMAND LINE

The command line interpreter operates on a command line, which is a string of characters ended with a carriage return. The command line is assumed to begin with a command and may have parameters to be passed to the command routine for processing or use. It may also have directives for use in device rerouting. (A later section, "Device Rerouting," gives information about this system facility.)

COMMAND LINE SOURCES

Commands are most commonly received from the keyboard by direct user input. Commands can also be be received from the RS232-C interface, from a disk file, or from the IEEE-488 interface, by means of the device rerouting capability. Commands can be received from BASIC, via the EXEC statement. All characters between the double-quote marks in the EXEC statement are sent to the CLI as a command line. Commands can be received from assembly language programs using the Call User (77) system call.

THE BASIC CALL STATEMENT

The CALL statement in BASIC can also be used to call a command routine for execution. However, it bypasses the CLI and handles the command and its parameters directly. BASIC implements the CALL statement by processing the parameters itself and issuing a Call User (77) system call. There are several differences between EXEC and CALL in the handling of commands and their parameters:

- The maximum number of parameters allowed is different
- The identification of hexadecimal numbers is different. PCOS uses an ampersand (&) and the CALL statement an ampersand H (&H) to identify hexadecimal numbers
- Parameter syntax is different

For information on these syntactical differences, see the BASIC manual. There is another difference in the capabilities of CALL and EXEC. Only CALL can use BASIC program variables as parameters.

COMMAND LINE PARSING

From the viewpoint of the CLI, command lines from the keyboard, from an RS232-C or IEEE-488 interface, from a disk file, from a Call User (77) system call, and from a BASIC EXEC statement, are all indistinguishable. All send the CLI a command line with the same format.

The CLI scans the command line and separates strings, numbers (decimal and hex), and rerouting directives (parameters beginning with '+' or '-'). Commas or spaces are the separators which break the command line into separate elements. The first (non-directive) word in a line will always be interpreted as the name of the command to be executed, the other characters on the line as parameters for the command itself or as rerouting directives. The command name will be first searched for in memory and, if not found, will be searched for as a disk file.

The parsing logic assumes that items starting with + or - are rerouting directives, that the first string encountered identifies the command routine, and that other non-directive items on the line are command parameters. These parameters can be identified by the first character. A decimal digit implies a decimal number, an & implies a hexadecimal number, and other items are strings. If appropriate, the string might be a file name. If it contains special characters, the string must be within quotation marks.

DEVICE REROUTING

The effect of, and implementation of, rerouting directives is described in "Device Rerouting" in Part 2. The rerouting directives go to the PCOS routines which support device rerouting. Rerouting is independent of the fundamental CLI functions, which are to process command parameters for the command routine.

PARAMETERS

The CLI checks for syntax consistency of every parameter, processes the parameters into a standard form, and then pushes all the parameters onto the stack. A brief overview of the types of parameters allowed is given below. For more details, see "Creating A Utility," in Part 3.

REROUTING DIRECTIVES

Device rerouting directives begin with '+' or '-' and may appear anywhere in the command line. Currently, the directives are:

+|-prt

+|-Sdevicename:

+|-Ddevicename:

+|-Sfilename

+|-Dfilename

These directives may be in uppercase or lowercase. The default device names are shown below. Other names can be assigned by using SDEVICE.

prt:	com1:
cons:	com2:
com:	ieee:

The file name can be any valid disk file name.

STRINGS

Strings may be enclosed in single or double quotes, or may be any alphanumeric sequence containing some character which is not a space, comma, ampersand, plus sign, minus sign, or quote character. These characters can be inserted only in a quoted string. A quoted string can also contain the quote sign of the opposite type from that used to delimit the string. That is, a string delimited by double-quotes can contain a single quote, and vice versa.

NUMBERS

Numbers must be integers, either decimal or hexadecimal (shown with the '&' identifier).

NULL PARAMETERS

Commas in sequence, or separated only by spaces, are evaluated as null parameters. If there are N + 1 commas in a sequence of spaces and commas, N 'null' parameters will be pushed onto the stack.

MAXIMUM COUNT

The maximum number of parameters allowed in a command line is 20 (PCOS 2.0f and following releases). The maximum number of parameters allowed in a command line for previous versions of PCOS was 11.

FILENAMES AND EXTENSIONS

Only a portion of the command routine filename is needed by the CLI (at least the first two characters, and sometimes more). The CLI checks first for the resident commands, including those commands made resident by PSAVE or by being PLOADED during the current working session. Then it searches the disk files. Both diskettes are searched, the current diskette first.

The search logic treats upper and lower case letters as equal. It looks for the portion of the name given and (except for three built-in resident commands) assumes a filename extension unless the full command name is supplied. Command extensions are .cmd and .sav, and .bas, when the routine is implemented in BASIC. The search logic will make up to three passes, looking first to match the filename portion and .cmd, then to match with the .sav extension, and finally with the .bas extension. In the case of ambiguous names, the CLI uses the first one it finds. For example, "fc" will match either "fcopy.cmd" or "fcopynew.cmd."

PASSING PARAMETERS

Parameters are processed into a standard form and pushed on the stack. They conform in format to the specifications described in "Data Passing Mechanism" in Part 3. A parameter appears on the stack as a three-word entry; the high byte of the first word contains a type identifier (0: null, 2: integer, 3: string -- and others not used by the CLI), and the second and third words contain a pointer to the data item. The item must be in the same segment.

If the item is an integer, the pointer points directly to the high byte of a 16-bit integer which is not necessarily word-aligned. If the item is a string, the pointer points to a 3-byte string descriptor: the first byte contains the string length in bytes, and the second and third bytes contain the offset of the first byte of the string in the same segment as above. If the item is null, the value of the offset of the pointer will be -1 (%FFFF).

When the CLI calls the command routine, the stack contains the CLI return address, a one-word parameter count, and the parameters (processed into Microsoft standard parameters as described above). See the section "Creating A Utility" or "Data Passing Mechanism" in Part 3 of this manual for a more detailed description of parameter format.

COMMAND EXECUTION

If the command routine is present in memory, the CLI turns control over to it. If the command routine is on a disk file, the CLI reads it into memory and turns control over to it.

If the command routine is in BASIC (has a .bas extension) the CLI checks to see if BASIC is in memory, and loads it first if necessary.

The command routine makes use of any parameter information on the stack. At the conclusion of its processing, it returns control to the CLI by using the return address given on the stack.

RELATIONSHIP OF THE CLI AND CALL USER (77)

Call User (77) is recursive; it can call itself. When Call User (77) is invoked with a single string parameter, it passes control to the CLI. The CLI parses the string, puts the individual items on the stack, and returns control to Call User (77). Call User (77) then calls the command routine, which executes using the parameter information on the stack.

The CLI handles rerouting directives by invoking the appropriate PCOS routines. If the command line contains only rerouting directives, only the rerouting routines are called and the effect is to set global rerouting.

CLI and Call User (77) are designed to work together, and can be thought of as co-routines. When Call User (77) is invoked with one string parameter, it calls on CLI to do its parsing; when it is invoked with parameters in a stack, it invokes the appropriate command routine which then executes using the parameter. So, the CLI is a subroutine for Call User (77). The two work together as the complete PCOS command line interpreter. When PCOS receives a command line (from any source) it passes the command line to Call User (77) as one string. Call User (77) calls CLI to parse the string into a parameter stack, and when control returns to it, Call User (77) calls itself with that parameter stack.

SPECIAL CHARACTERS

The characters listed below have special meaning. They are not allowed in a string, unless the string is delimited by quote marks.

SYMBOL:	USED FOR:
plus sign	device rerouting identifier
minus sign	device rerouting identifier
ampersand	hex number identifier
space	parameter separator
comma	parameter separator
double quote	string delimiter
single quote	string delimiter

Either single or double quotes can delimit a string. Quotes inside a string must be of the opposite type.

7. COMMANDS AND UTILITY PROGRAMS

ABOUT THIS CHAPTER

This chapter gives background information on the distinction between resident and transient commands and on the PCOS utilities PLOAD, PUNLOAD, and PSAVE. It includes information about command name conventions and requirements.

CONTENTS

OVERVIEW

This section gives background information on the distinction between resident and transient commands and on the PCOS utilities PLOAD, PUNLOAD, and PSAVE. It includes information about command name conventions and requirements.

BACKGROUND INFORMATION

PCOS has resident and transient commands. Resident commands are built into PCOS or are placed into PCOS by use of the PSAVE command. Transient commands are resident on disk. They are brought into system memory and executed by the command line interpreter when the user enters the command name.

The user can PLOAD the command, which brings the transient command routine into system memory as a temporary resident command. The routine stays in system memory, available for immediate use, until it is removed using the PUNLOAD command or until the system is reset or turned off. PLOADED commands can be made resident by PSAVING them.

A command routine is loaded into space allocated from the memory heap. During the time it is in memory, the amount of heap space remaining for the user is diminished. The command routine must be relocatable (must not depend on being in a fixed memory location), because the memory location assigned for it can vary.

A command routine is a utility program written in such a way that it can be called by the command line interpreter, can make use of the parameters passed to it by the command line interpreter, and can properly return control to PCOS when it has finished its task. For information on developing such a utility program, see "Creating A Utility" in Part 3.

RELATED SECTIONS

"Command Overview" in Part 1 provides an overview of PCOS commands grouped by general functions. "Commmand Line Interpreter" in Part 2 explains the mechanism by which commands are invoked. "Creating A Utility" in Part 3 gives information on developing a utility program that can be called as a PCOS command.

COMMAND NAMES

The minimum identification necessary to invoke a command is the first two characters. The PCOS standard requires that these characters be unique. It is not always possible to meet this requirement. Some special purpose commands may have duplicate characters in the first two positions. However, in most applications duplicates will not be found because the commands in general use meet this standard. Letters may be lower case or upper case. For example, the command for file copying is "fcopy." It can be invoked by "fc," "fcopy," "FC," or "FCOPY." If the first two letters of a command are the same, then more letters must be used to

properly identify that command. In any event, the command line interpreter makes use of as much command identifications as it is given.

The parameters to be used by the command routine are included in the command line. For example, if a user wanted to copy from 0:file1 to 1:file2 the command line could be

```
fc    0:file1 1:file2
```

or

```
fcopy 0:file1 1:file2
```

COMMAND LINE INTERPRETER

The PCOS command line interpreter (CLI) interprets the first item on a line as the command, and everything following the command as parameters. Then it relays the parameter information to the utility by pushing onto the stack the number of parameters given and what they are, before transferring control. The utility itself is responsible for parameter error checking.

The command line interpreter does have the responsiblity for finding the proper command. If it is already PLOADED, then nothing more is necessary but to transfer control there. However, the search of the disk files needs to be explained. Disk files which are PCOS utilities have one of these extensions: .sav, .cmd, or .bas. The command line interpreter searches for the first file with one of those extensions that begins with the same two (or more) characters the user gave. It takes the first matching entry it finds. Letters are matched exactly, treating upper and lower case as equivalent. Any number of characters greater than two can be specified to distinguish a command, and the last character used can be the first character in the file name which is not a duplicate. For example, to distinguish between fcopy.cmd and fconnect.cmd, the commands can be entered as fcop and fcon.

COMMAND SUFFIXES

.cmd and .sav

The difference between the .cmd and .sav extensions deserves some explanation. Most utilities are in the category which use .cmd as a suffix. Utilities with .cmd extensions are removed from system memory after they return control to the system. Utilities with .sav extensions remain in memory (the CLI PLOADS them). Usually, the .sav extension indicates utilities which, when loaded, alter the PCOS system in some permanent way which remains in effect until the system is reset or powered off. For example, RS232 is a driver that provides RS-232-C communications. Once loaded, it must remain to service communications it has begun.

.bas

The .bas suffix indicates that the command routine is written in BASIC. The CLI will check for the presence of the BASIC interpreter, and if necessary will load BASIC and then the command routine. A BASIC routine cannot have a .sav extension.

PUNLOAD EXCEPTIONS

The PUNLOAD command, available in PCOS 2.0 and later verions, reverses the effect of PLOAD (or in theory, the .sav extension) and frees, if possible, the memory space formerly occupied by the command routine.

EXCEPTION LIST

However, some commands change PCOS tables and cannot be removed after having been PLOADED. Their TLOC attribute is set to 9. PUNLOAD examines the attribute and does not remove them. The commands are:

ci.sav	RS232 driver user routines
rs232.sav	RS232 driver basic routines
ieee.sav	IEEE driver routines
eprint.sav	Print error messages
kana.sav	Katakana keyboard table
vmove.sav	Volume copy on one drive system command
pdebug.sav	Debugger

Resident commands cannot be PUNLOADED:

lterm pload punload

THE .sav EXTENSION

Usually, commands with the .sav extension cannot be removed from memory. However, the .sav extension does not imply the command is unloadable. This extension means only that the system does not automatically unload the command after its execution. The user can use the .sav extension as desired. Unloadability is controlled by the TLOC attribute.

LEGAL FILENAMES

Legal filenames must start with a letter. The remaining characters can be letters or numbers. Both upper and lower case characters are allowed as filenames, and are treated as equivalent. The following special characters are NOT allowed within the filename:

```
= (equals)        - (minus)                 + (plus sign)
, (comma)         : (colon)                 # (pound or hash)
\ (backslash)     / (slash)                 ' (single quote)
* (asterisk)      ? (question mark)         " (double quote)
/space/           /any control character/
```

COMMAND NAME AVAILABILITY

Despite a few exceptions, the PCOS standard requires characters of a command name to be unique for the command to be properly accessed when called in its short form. The following table shows the commands generally furnished with PCOS. All other two letter combinations are available. In fact, all combinations of a letter followed by a number are also available.

as	ASM.CMD	fu	FUNPROT.CMD	sc	SCOMM.CMD
ba	BASIC.CMD	fw	FWPROT.CMD	sd	SDEVICE.CMD
bk	BKEYBOARD.BAS	hd	HDUMP.CMD	sf	SFORM.CMD
bv	BVOLUME.SAV	ie	IEEE.SAV	sl	SLANG.CMD
ci	CI.SAV	la	LABEL.CMD	sp	SPRINT.CMD
ck	CKEY.CMD	li	LINK.CMD	ss	SSYS.CMD
dc	DCONFIG.CMD	ls	LSCREEN.CMD	te	TEXTDUMP.CMD
ed	EDIT.CMD	lt	LTERM	va	VALPHA.CMD
ep	EPRINT.SAV	ml	MLIB.CMD	vc	VCOPY.CMD
fc	FCOPY.CMD	pd	PDEBUG.SAV	vd	VDEPASS.CMD
fd	FDEPASS.CMD	pk	PKEY.CMD	vf	VFORMAT.CMD
ff	FFREE.CMD	pl	PLOAD	vl	VLIST.CMD
fk	FKILL.CMD	pr	PRUN.CMD	vm	VMOVE.CMD
fl	FLIST.CMD	ps	PSAVE.CMD	vn	VNEW.CMD
fm	FMOVE.CMD	pu	PUNLOAD	vp	VPASS.CMD
fn	FNEW.CMD	rf	RFONT.CMD	vq	VQUICK.CMD
fp	FPASS.CMD	rk	RKILL.CMD	vr	VRENAME.CMD
fr	FRENAME.CMD	rs	RS232.SAV	vv	VVERIFY.CMD
		sb	SBASIC.CMD	wf	WFONT.CMD

8. MEMORY CONFIGURATION

ABOUT THIS CHAPTER

This chapter explains the physical implementation of system memory in functional terms, providing background information.

CONTENTS

OVERVIEW

This section explains the physical implementation of system memory in functional terms. The information is provided for background, and is not necessary for making use of memory when using the PCOS system calls for storage allocation. These calls are described in the companion section, "Memory Management," which follows. "Memory Configuration" bridges the gap between information provided about Z8000 memory handling concepts in the vendor literature and the PCOS implementation of memory management. The section also has information about the actual storage location of certain PCOS system elements, which is not usually needed by the programmer but which may be needed for certain programming tasks. This section does not contain hardware information except for a small amount of background information necessary to explain certain memory functions.

PHYSICAL MEMORY BLOCKS

The M20 system memory is allocated in 16 Kb blocks. Standard memory on the motherboard is 128 Kb (8 blocks). Up to three memory expansion boards can be added. All expansion boards added must be of the same capacity, which is either 32 Kb (2 blocks) or 128 Kb (8 blocks).

Jumper options on the motherboard inform the system software that the system contains one of the seven fundamental memory configurations shown below.

Case	Configuration	Expansion Boards	Jumper Code
1	Standard 128 Kbyte memory only	0	5
2	32 Kbyte expansion board(s), black and white display	1-3	7
3	32 Kbyte expansion board(s), 4-color display	1-3	3
4	32 Kbyte expansion board(s), 8-color display	2-3	2
5	128 Kbyte expansion board(s), black and white display	1-3	6
6	128 Kbyte expansion board(s), 4-color display	1-3	1
7	128 Kbyte expansion board(s), 8-color display	2-3	0

As the above list shows, the jumper code informs the system either that no memory expansion boards are present or that expansion capacity exists with a particular maximum potential. The system determines the actual size within the potential size when it is started up. It does this by writing a word of zeros at the low address for each block and reading the value at that address. If the value of all ones is read, no memory block exists. (The unconditioned value of the data lines will be read as ones.)

Z8001 MEMORY CONCEPTS

The Z8001 CPU allows extensive manipulation of actual memory addresses in order to provide system designers with flexibility, compartmentalization, and security. The fundamental memory addressing scheme is segment and offset: A memory segment can be up to 64 Kb (addressed by 16 bits), and there can be up to 128 segments, (addressed by 7 bits). The M20 uses 32 segments, addressed by 5 bits. In the interests of execution speed, PCOS does not use the Z8001 security code provisions for user memory. Because the M20 is a single-user system, such provisions should not be necessary.

Furthermore, memory can be differentiated between "code" and "data" segments. The Z8001 has a code/data signal which changes state according to whether the CPU is executing an instruction code or accessing data. This distinction is meant to provide the designer with a tool for system security. The designer can reserve certain logical memory locations for data only, and the machine will refuse to allow code execution in those locations.

In practice, there are some difficulties with this distinction. When the loader is placing programs in memory, those programs are data (they are not executing). When those programs are executing, they must be in "code" segments. Therefore, certain "code" and "data" segments must be equivalent for the system loader to work.

The distinction between code and data segments is useful in high-level languages, where the code may be processing buffers of data or large arrays of numbers. In assembly language programs, code and data information is often intermingled and must all be placed in a code segment for actual execution.

PCOS design makes use of the code/data distinction in certain cases. One example, discussed in "Language Support" in Part 3, is the implementation of the BASIC interpreter. As developed, the interpreter and its user area were both fitted within 64 Kb. The PCOS enhancement places the user area in a second 64 Kb segment which is a data segment. The user-entered BASIC statements can all be treated as data by the interpreter which actually executes code.

LOGICAL-TO-PHYSICAL MEMORY DECODING

The M20 provides for the physical memory (up to 512 Kb, 32 blocks) to be addressed within a logical framework of two megabytes, one megabyte for "code" memory and one for "data." Many physical blocks have more than one logical address.

Figure 1, below, shows how logical addresses are decoded into physical block addresses.

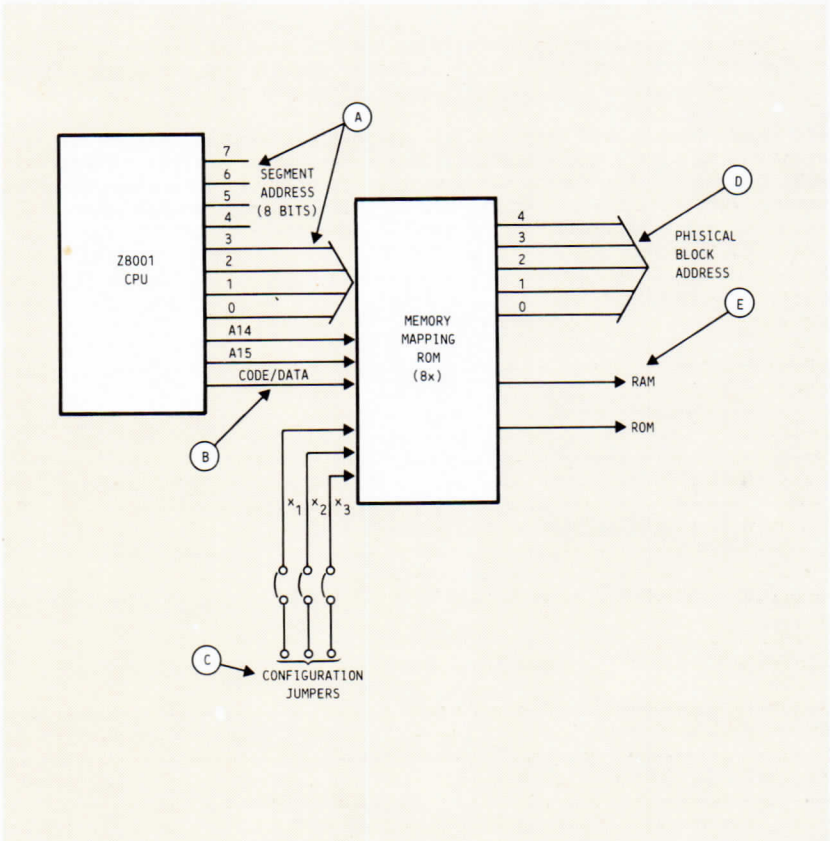


Fig. 8-1 Logical to Physical Address Decoding

A The Z8001 provides 8 bits for segment address. The low four bits are sent to the memory mapping ROM and the high bits are ignored. Two address bits, A14 and A15, are also sent.

B The Z8001 code/date signal is sent to the memory mapping ROM.

C The configuration jumper setting is read by the memory mapping ROM.

D These input bits (A, B, and C) are used as an index into a table and the ROM table entry provides 8 bits of output. There are 16 tables. The table is selected by the three configuration jumper settings and the code/data signal. Within the selected table, the four segment address bits index a table entry which is output. Output consists of a 5-bit physical block address (0-1F; 0-31 in decimal) and three signal settings described below.

E These two signal lines are used by the M20 hardware. They indicate whether the physical block is ROM or dynamic RAM. A third signal, currently unused, is available for future development.

The figure below shows the logical-to-physical configuration for the eight blocks on the motherboard. The portions shown are the same for all PCOS systems. Other logical-to-physical assignments vary according to memory configurations. At the end of this section, logical-to-physical memory maps are provided for the seven fundamental cases.

Code Segments

4															
3	4	7				4									
2	3	6				3		3	7	3					
1	2	5	S0	-R0-		2	-R0-	2	6	2					
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

	4	4													
	7	7			4	4									
	6	6			3	3		7	7	3					
1	5	5	S0	-R0-	2	2	-R0-	6	6	2					
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Fig. 8-2 Motherboard System Memory

The figure above shows the logical access paths for 8 Kb of ROM (R0) and eight blocks of system memory (S0, S1, ... S7). As can be seen, some physical blocks can be accessed in more than one path and may be accessed in either a code segment or a data segment. The diagrams also show that logical segments may contain no memory or from one to four blocks (16 Kb) of physical system memory.

LOGICAL ADDRESSES

A logical address consists of a segment and an offset. Segment values range from 0 through %F and offsets from 0 through %FFFF. The distinction between code and data segments depends on the state of the code/data signal from the Z8001.

Physical blocks within a segment start at one of these offsets: 0, %4000, %8000, or %C000. For example, physical block 6, which is in the same location in both CS2 and DS2, starts at <<2>>%4000 and ends at <<2>>%7FFF. The block boundary does not matter to the programmer so long as it is not an end of the segment. A data string lying from <<2>>%7FE0 through <<2>>%813A could be manipulated without regard for block boundaries.

When using the storage handling system calls, the programmer does not need to be concerned with logical boundaries. PCOS treats all memory as a "heap" without boundaries, and takes care of the boundary effects internally.

SEGMENT USAGE

The following discussion gives some information about usage of ROM and some of the system memory segments. This information is provided as background. The application programmer would never need to know these details. The information may be useful to the system programmer in certain cases.

ROM (READ ONLY MEMORY)

The startup routines, including startup diagnostics and the bootstrap loader, are in ROM. They occupy 8 Kb of the physical block shown as R0, which is present in logical segments CS4, CS7, DS4, and DS7. The ROM routines must be in a code segment in order to execute, and in a data segment in order for the system routines to extract data provided there.

SCREEN BIT MAP

S0 is a 16 Kb block which is dedicated for screen display, and is present in both CS3 and DS3. The video display is memory-mapped, which means that data placed in S0 displays on the screen.

Further screen bit-map blocks to support color graphics would go into the same segment.

For display purposes, S0 is in a data segment. It is also in a code segment because during startup the ROM routines use half of S0 for work space. (Therefore, only half the display screen is active during startup diagnostics.)

PCOS BLOCKS

PCOS code is executed from physical block 4, which is the third block in CS1 and CS6. The bootstrap routines and other loader routines can also access this block via a data segment address in order to load the PCOS nucleus or other PCOS routines.

PCOS uses physical block 1 for data. This can be located as the first block of both CS0 and DS0.

LANGUAGE BLOCKS

The BASIC interpreter goes into CS1 and its user area into DS2. The loader routines place other languages in the highest segment available and work down. The loader skips the BASIC segments, and uses them only as a last resort.

UTILITY AND ASSEMBLY PROGRAMS

These routines use physical blocks 5, 6, and 7, which can be accessed in both code and data segments. See, for example, CS2 and DS2.

Z8001 BACKGROUND INFORMATION

The memory concepts discussed briefly here are provided for background information. The information is available in reference documentation on the Z8000 family of CPU and supporting integrated circuit chips. The PCOS memory management software handles these hardware capabilities and buries them from view of the programmer.

SYSTEM MODE AND NORMAL MODE

The Z8001 has two modes of operation, system mode and "normal" mode. In system mode, all Z8001 instructions can be executed. In normal mode, certain instructions, such as direct input/output, are not allowed. PCOS runs under system mode and user programs under normal mode. The PCOS system calls allow user programs to perform, indirectly, system mode operations.

The memory management subsystem of PCOS performs operations in system mode to control memory allocation. These operations include setting of logical-to-physical translation and setting of attributes for portions of memory. Memory attribute settings allow reserving portions of memory for system use only and the prevention of unauthorized memory access.

The stack control registers, R14 and R15, are actually different registers in system mode and normal mode. A programmer using system mode and a programmer using normal mode can both set values into R14 and R15 and work with stack operations. The stack handling instructions appear to refer to the same registers, but actually do not. The two stacks will be handled independently.

SEGMENTED MODE AND NON-SEGMENTED MODE

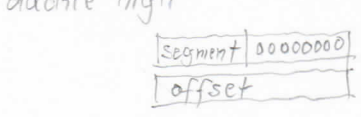
Memory addressing in the Z8001 is done using "segments" and "offsets." A segment can contain a maximum of 64 kilobytes of memory. The offset pointer is 16 bits and can address any byte within the maximum segment size. A complete segmented memory address consists of a 7-bit segment identifier and the 16-bit offset:

```
-----------------------
|   Seg    |  Offset  |
-----------------------
```

Segment	0-127
Offset	0-65534

In segmented mode, the programmer provides both segment and offset values. In non-segment mode, the programmer provides only the offset value. The "current segment" is implied. The current segment value is obtained from the program status word (PSW). The segment value is maintained in the PSW in either mode. Specifying a segment in segmented mode does not change the PSW segment identifier.

The M20 design uses the low four bits of the segment address, and therefore has segment values from 0 through 15.



THE SEVEN FUNDAMENTAL CONFIGURATIONS

OVERVIEW

The tables that follow give the seven fundamental configurations possible for system memory. The expansion configurations show the maximum possible expansion. Actual capacity is determined by the startup diagnostic routines and saved for use by the system memory management routines.

The tables show three underlying conditions: no expansion. 32 Kb expansion, and 128 Kb expansion. When expansion boards are available, the layout depends on the type of display because 4-color and 8-color displays require one or two blocks to be dedicated for additional screen memory.

CONFIGURATION OVERVIEW

Case	Configuration	Expansion Boards	Jumper Code
1	Standard 128 Kbyte memory only	0	5
2	32 Kbyte expansion board(s), black and white display	1-3	7
3	32 Kbyte expansion board(s), 4-color display	1-3	3
4	32 Kbyte expansion board(s), 8-color display	2-3	2
5	128 Kbyte expansion board(s), black and white display	1-3	6
6	128 Kbyte expansion board(s), 4-color display	1-3	1
7	128 Kbyte expansion board(s), 8-color	2-3	0

The jumper code shown refers to motherboard jumpers, and is used when computing the effective address. Strapping or jumper arrangements on the expansion boards are not discussed.

CONFIGURATION 1:

Standard 128 Kb, no expansion, black and white display.

Jumper Selection: Switches Read: 5

X1 to X8 = OFF
X2 to X7 = ON
X3 to X6 = OFF

Code Segments

4															
3	4	7				4									
2	3	6				3		3	7	3					
1	2	5	S0	-R0-		2	-R0-	2	6	2					
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

	4	4													
	7	7			4	4									
	6	6			3	3		7	7	3					
1	5	5	S0	-R0-	2	2	-R0-	6	6	2					
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Physical arrangement:

Main M20 board = R0, S0, 1, 2, 3, 4, 5, 6, 7
No Expansion boards

Note: R0 is the ROM Memory on the main board
S0 indicates screen bitmap area

CONFIGURATION 2:

32 Kb expansion board(s) (1-3), black and white display.

Jumper Selection: Switches Read: 7

X1 to X8 = OFF
X2 to X7 = OFF
X3 to X6 = OFF

Code Segments

4		8						10	12	10					
3	4	7				4		9	11	9					
2	3	6				3		3	7	3					
1	2	5	S0	-R0-		2	-R0-	2	6	2	13				
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

11	8	8						12	12	10					
10	7	7		13	4	4		11	11	9					
9	6	6		12	3	3		7	7	3					
1	5	5	S0	-R0-	2	2	-R0-	6	6	2	13				
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Physical arrangement:

Main M20 board	=	R0, S0, 1, 2, 3, 4, 5, 6, 7
First expansion board	=	8, 9
Second expansion board	=	10, 11
Third expansion board	=	12, 13

Note: R0 is the ROM Memory on the main board
S0 indicates screen bitmap area

CONFIGURATION 3:

32 Kb expansion board(s) (1-3), 4-color display

Jumper Selection:

X1 to X8 = OFF
X2 to X7 = OFF
X3 to X6 = ON

Switches Read: 3

Code Segments

4		9						11	13	11					
3	4	7				4		10	12	10					
2	3	6	S8			3		3	7	3					
1	2	5	S0	-R0-		2	-R0-	2	6	2					
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

12	9	9						13	13	11					
11	7	7			4	4		12	12	10					
10	6	6	S8		3	3		7	7	3					
1	5	5	S0	-R0-	2	2	-R0-	6	6	2					
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Physical arrangement:

Main M20 board	=	R0, S0, 1, 2, 3, 4, 5, 6, 7
First expansion board	=	S8, 9
Second expansion board	=	10, 11
Third expansion board	=	12, 13

Note: R0 is the ROM Memory on the main board
S0, S8 indicate screen bitmap areas

CONFIGURATION 4:

32 Kb expansion board(s) (2-3), 8-color display

Jumper Selection: Switches Read: 2

X1 to X8 = ON
X2 to X7 = OFF
X3 to X6 = ON

Code Segments

4		9	12				13	12		12					
3	4	7	S10			4		11	13	11					
2	3	6	S8			3		3	7	3					
1	2	5	S0	-R0-		2	-R0-	2	6	2					
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

12	9	9	12				13			12					
11	7	7	S10		4	4		13	13	11					
10	6	6	S8		3	3		7	7	3					
1	5	5	S0	-R0-	2	2	-R0-	6	6	2					
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Physical arrangement:

Main M20 board = R0, S0, 1, 2, 3, 4, 5, 6, 7
First expansion board = S8, 9
Second expansion board = S10, 11
Third expansion board = 12, 13

Note: R0 is the ROM Memory on the main board
S0, S8, S10 indicate screen bitmap areas

CONFIGURATION 5:

128 Kb expansion boards (1-3), black and white display

Jumper Selection: Switches Read: 6

X1 to X8 = ON
X2 to X7 = OFF
X3 to X6 = OFF

Code Segments

4		8						10	12	10	16	20	24	28	24
3	4	7				4		9	11	9	15	19	23	27	31
2	3	6				3		3	7	3	14	18	22	26	30
1	2	5	S0	-R0-		2	-R0-	2	6	2	13	17	21	25	29
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

17	8	8						12	12	10	16	20	24	28	24
16	7	7		25	4	4		11	11	9	15	19	23	27	31
9	6	6		24	3	3		7	7	3	14	18	22	26	30
1	5	5	S0	-R0-	2	2	-R0-	6	6	2	13	17	21	25	29
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Physical arrangement:

Main M20 board = R0, S0, 1, 2, 3, 4, 5, 6, 7
First expansion board = 8, 9, 10, 11, 12, 13, 14, 15
Second expansion board = 16, 17, 18, 19, 20, 21, 22, 23
Third expansion board = 24, 25, 26, 27, 28, 29, 30, 31

Note: R0 is the ROM Memory on the main board
S0 indicates screen bitmap area

CONFIGURATION 6:

128 Kb expansion board(s) (1-3), 4-color display

Jumper Selection: Switches Read: 1

X1 to X8 = OFF
X2 to X7 = ON
X3 to X6 = ON

Code Segments

4		9						11	13	11	17	21	25	29	25
3	4	7				4		10	12	10	16	20	24	28	24
2	3	6	S8			3		3	7	3	15	19	23	27	31
1	2	5	S0	-R0-		2	-R0-	2	6	2	14	18	22	26	30
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

24	9	9					25	13	13	11	17	21	25	29	25
17	7	7			4	4		12	12	10	16	20	24	28	24
16	6	6	S8		3	3		7	7	3	15	19	23	27	31
1	5	5	S0	-R0-	2	2	-R0-	6	6	2	14	18	22	26	30
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Physical arrangement:

Main M20 board = R0, S0, 1, 2, 3, 4, 5, 6, 7
First expansion board = S8, 9, 10, 11, 12, 13, 14, 15
Second expansion board = 16, 17, 18, 19, 20, 21, 22, 23
Third expansion board = 24, 25, 26, 27, 28, 29, 30, 31

Note: R0 is the ROM Memory on the main board
S0, S8 indicate screen bitmap areas

CONFIGURATION 7:

128 Kb expansion board(s) (2-3), 8-color display

Jumper Selection: Switches Read: 0

X1 to X8 = ON
X2 to X7 = ON
X3 to X6 = ON

Code Segments

4		9	24				25	11	13	11	18	22	26	30	26
3	4	7	S16			4		10	12	10	17	21	25	29	25
2	3	6	S8			3		3	7	3	15	20	24	28	24
1	2	5	S0	-R0-		2	-R0-	2	6	2	14	19	23	27	31
CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

Data/Stack Segments

24	9	9	24				25	13	13	11	18	22	26	30	26
17	7	7	S16		4	4		12	12	10	17	21	25	29	25
16	6	6	S8		3	3		7	7	3	15	20	24	28	24
1	5	5	S0	-R0-	2	2	-R0-	6	6	2	14	19	23	27	31
DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15

Physical arrangement:

Main M20 board = R0, S0, 1, 2, 3, 4, 5, 6, 7
First expansion board = S8, 9, 10, 11, 12, 13, 14, 15
Second expansion board = S16, 17, 18, 19, 20, 21, 22, 23
Third expansion board = 24, 25, 26, 27, 28, 29, 30, 31

Note: R0 is the ROM Memory on the main board
S0, S8, S16 indicate screen bitmap areas

9. MEMORY MANAGEMENT

ABOUT THIS CHAPTER

This section presents the memory management functions available to the programmer that allow use of system memory resources as a "heap", without regard for segment boundaries. It includes implementation concepts.

CONTENTS

OVERVIEW

This section is a companion to the prior section, "Memory Configuration." "Memory Management" presents the system functions available to the programmer that allow use of system memory resources as a "heap," without regard for segment boundaries. It gives some information on implementation and briefly describes the system calls for storage allocation.

PCOS MEMORY CONCEPTS

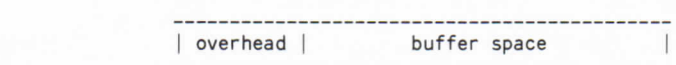
The "heap" concept allows the programmer to request a block of memory of a particular size and to release it back to the system when done. Available system memory is treated as a heap from which portions can be taken and to which portions can be added. Most of the details given in the prior section are hidden. The system calls given at the end of the section operate without regard for segment boundaries.

IMPLEMENTATION OF MEMORY MANAGEMENT

The PCOS nucleus includes the memory management routines. When PCOS is initialized the memory management routines are initialized also and used by the nucleus to load the rest of PCOS and the associated routines and tables (resident commands, RFONT tables, etc.).

Memory management takes all of memory available after initialization and links it together as large buffers forming one large heap. Then it allocates memory from the heap and, when allocated memory is returned to it (by the Dispose call), returns the memory block to the heap. Memory management starts by allocating chunks of memory constrained only by segment boundaries, which it hides from users. As user programs and system programs receive memory buffers, the allocation of memory begins to resemble a patchwork quilt.

Memory buffers look like this:



The overhead information is concise. It contains a special marker, the length of the buffer space, and information linking it to other buffers. When a buffer is requested, memory management takes space from the heap, sets up a buffer of the requested size, and allocates it to the caller. Unused space is allocated to the heap in the same fashion as to any other user. When a using program disposes of a buffer, memory management allocates it to the heap. When a program finishes execution, PCOS informs memory management which then releases all buffers used by the program and allocates them to the heap. (StickyNew allocations are an exception.)

Certain points can be summarized from this.

* All system memory is allocated, either to a user program, system program, or the heap.

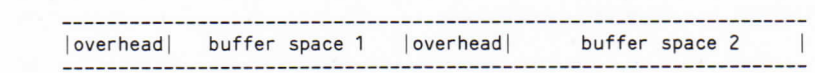
* Memory management hides details. Users request memory of a particular size, receive it, use it, and return it. The actual location of memory buffers is not a matter of user concern.

The DCONFIG utility program can be used to find actual memory locations, if desired.

WARNING ON BUFFER USE

As described above, there is no unallocated space in the system and there is no guard space around buffers. The design of the memory management system emphasizes small overhead and speed of execution. Memory management does not guard against a program accessing memory past its buffer boundaries.

In the figure below, we see that at the end of a buffer is the overhead informtion for the next.



If the user of buffer space 1 writes information past the end of the buffer, the overhead information on buffer space 2 is destroyed.

This can happen in assembly language programs through programmer error, and to some degree in languages such as BASIC when using arrays or similar data structures and attempting to access with a subscript incremented or decremented just past the proper range. Such errors are less likely in PASCAL, which has rigorous internal checks.

To some degree, memory management can recover from difficulties caused by desroyed overhead information. However, the buffer space information that depended on the lost overhead information is lost. The effects of this may not show up until later, usually when the program terminates, and the cause may not be understood. That is, it is possible to destroy buffer information and not realize it, and to later suffer the consequences and not realize why. This kind of problem can be one of the most difficult to detect and remedy. A special command, TEXTHEAP.SAV, can be very valuable in this situation. It can give the address of destroyed or invalidated information. Contact Olivetti for this program.

In summary, be careful not to exceed buffer boundaries.

PCOS NUCLEUS

The PCOS nucleus, or kernel, is a fundamental part of PCOS required to handle input/output for the system peripherals (keyboard, display, printer, and disks), to decode command lines and execute commands, and to manage memory. Other system software modules are loaded by the kernel when needed.

The kernel resides in permanent memory. The permanent memory area also contains the resident commands and any system elements made permanent by the PSAVE utility, such as command routines, programmed key (PKEY) definitions, and user designed fonts (RFONT utility).

PCOS STARTUP

The diagnostic ROM determines the size of memory for each segment and the total size of memory and saves these values for memory management. If there is not enough memory to load the current PCOS, PCOS will not execute. This can happen when a user has saved material on a larger system and attempts to load it on a smaller one.

When a PSAVED PCOS is booted onto a system with more memory, the memory management descriptor table is modified to include the additional free blocks in the heap.

OBSOLETE STORAGE ALLOCATION CALLS

Certain obsolete system calls operate within segment boundaries. They are listed in "System Calls" in Part 2, under "Obsolete Calls." These calls are presently supported, but should not be used for development work. When modifying older programs, these calls should be replaced with the system calls described at the end of this section. For convenience in understanding the obsolete calls in order to replace them, some information on segment handling is given below.

The major constraint is the segment boundary and therefore the remaining memory capacity in the segment. Most of the obsolete system calls operate within the "current segment." The current segment can be changed to a new segment by issuing a new segment call requesting memory. The request can be for zero length, which merely changes the segment and allows determining such matters as how much memory is available in the new segment.

Remember that although a segment has a maximum size of 64 Kb, it may actually be made up of fewer physical blocks or even be empty. Moreover, even in a full-sized segment the system may have taken memory space for its own use or for memory management overhead.

STORAGE ALLOCATION CALLS

The following system calls are the programmer's interface to the memory management system. These calls treat all of system memory as a heap, without regard for segment boundaries. For further details, see the Assembly Language Manuals.

Dispose (34)

Releases heap space.

```
    Input:

            RR8   <-  address of block pointer

    Output:

            @RR8  ->  hex FFFFFFF
            R5    ->  error status
```

New (120)

Allocates a block of bytes from heap.

```
    Input:
            RR8   <-  address of block pointer
            R10   <-  length

    Output:
            R5    ->  error status
            @RR8  ->  block pointer
```

BrandNewAbsolute (121)

Allocates a block at a specified address.

```
    Input:
            RR8   <-  address of block pointer
            R10   <-  length

    Output:
            R5    ->  error status
            @RR8  ->  block pointer
```

NewLargeBlock (122)

Allocates the largest free block of bytes from heap.

```
Input:
        RR8    <-  address of block pointer

Output:
        @RR8   ->  block pointer
        R10    ->  length
        R5     ->  error status
```

StickyNew (123)

Allocates a block of bytes from heap that remains allocated after the program doing this call terminates.

```
Input:
        RR8    <-  address of block pointer
        R10    <-  length

Output:
        @RR8   ->  block pointer
        R5     ->  error status
```

Examples

The following examples show how the stack can be used to hold the block pointer.

Example 1. All calls except Dispose

```
dec     r15, #4              //make room for pointer
ldl     rr8, rr14            //addr of ptr to rr8
ld      r10, #SIZE NEEDED
sc      #New
pop     rr2, @rr14           //rr2 has ptr to block
```

Example 2. Dispose

```
push    @rr14, rr2
ldl     rr8, 4414           //rr8 has addr of ptr
sc      Dispose
pop     rr2                 //rr2 will contain nil (-1)
```

10. SYSTEM CALLS

ABOUT THIS CHAPTER

This chapter gives an overview of the PCOS system calls grouped by general functions and provides background information on their use by the programmer. System calls are used to handle input/output and to manage system resources.

CONTENTS

OVERVIEW

System calls are PCOS procedures used to handle Input/Output and to manage system resources such as memory or the real-time clock. System calls can be accessed by assembly language programs via the Z8000 System Call instruction. The System Call instruction includes a one-byte request code which indicates the function to be performed. For example:

```
sc #3     System call, request code = 3
```

System operations done by BASIC, PASCAL, and other high-level languages make use of system calls, as do PCOS command routines and utility programs.

Parameters to be used by the system call are generally passed in registers numbered from R5 to R13. If strings or other large data structures are to be passed, pointers to the structures are passed as parameters in the registers.

In general, parameters are passed as 16-bit unsigned values. ASCII characters are passed occupying the lower bytes of a register.

All system calls that return an error condition use register R5. Zero indicates no error; a non-zero integer gives the error code. Error codes are listed in Part 3, under "PCOS Error Codes."

TYPES OF CALLS

In the discussions which follow, the system calls have been grouped by type as follows:

a. Bytestream Calls
b. Block Transfer Calls
c. Storage Allocation Calls
d. Data Manipulation Calls
e. Time and Date Calls
f. User Call to PCOS
g. System Management
h. File Management
i. IEEE-488 Calls
j. Obsolete Calls
k. Graphics Calls

The section also has supplementary discussions on Device Rerouting and on the RS232 driver support. These topics are related to the bytestream calls.

All system calls from PCOS 1.X through PCOS 3.X are included in the discussion in this section.

NUMBERING AND LABELS

For purposes of identification, each system call has been assigned a label as well as a number. A list of these labels is given at the end of the section.

It is recommended that this list be made into an include file and that programmers use the symbolic label rather than the number when coding system calls. For example, SC 9 can also be referenced by its label: 'LookByte.'

FURTHER INFORMATION

The remainder of this section gives an overview of system calls and general information about them. For detailed information on each system call, see the Assembly Language Manual.

BYTESTREAM I/O CALLS

GENERAL

Bytestream I/O calls are used to interface with the disk, printer, RS232 communications port, and console (keyboard and video). These calls are used to:

a) Transfer bytes of data to or from an I/O device

b) Send control information to a device or to a device driver

c) Receive status information from a device

The bytestream calls are:

LookByte (9)	Close (19)
GetByte (10)	SetControlByte (20)
PutByte (11)	GetStatusByte (21)
ReadBytes (12)	OpenFile (22)
WriteBytes (13)	DSeek (23)
ReadLine (14)	DGetLen (24)
Eof (16)	DGetPosition (25)
ResetByte (18)	

FILE IDENTIFIER (FID) NUMBERS

A FID is a small integer used to identify the keyboard, the console, the printer, an open disk file, or other I/O device. The operating system maintains a table associating FIDs with a File Pointer. That pointer refers to a control structure comprised of pointers to data structures and to routines. The FID is required with bytestream calls.

FILE AND DEVICE POINTERS

Opening a disk file creates a stream data structure, and places a pointer to it in the File Pointer Table (FPT). Closing the disk file sets this pointer to nil, and releases any table space associated with the file. Some "files" or devices are always open. For example, the keyboard and the screen (the default window) are always open. They can, however, be closed and re-opened by use of the PCOS Device Rerouting feature.

The following table describes the allocation of FIDs. Some of these FIDs represent devices which are always open: others are assigned to files or screen windows by system calls.

0-15	BASIC files
16	Reserved system file
17	Console
18	Printer
19, 25, 26	RS232 Communications (Com, Com1, Com2)
20-24	PCOS files

BASIC file numbers translate simply into PCOS FIDs, but BASIC window numbers for the screen are distinct from FIDs. The PCOS file ID's cannot be accessed in BASIC.

BYTESTREAM I/O CALL OVERVIEW

LookByte (9)

Returns the next byte from designated device buffer without removing the byte from the buffer.

Input:

R8	<-	FID

Output:

RL7	->	returned byte
RH7	->	buffer status
R5	->	error status

GetByte (10)

Returns the first byte from designated device, removing the byte from the device buffer.

```
Input:
        R8      <-      FID

Output:
        RL7     ->      returned byte
        (H7 always zero)
        R5      ->      error status
```

PutByte (11)

Transmits a byte to specified device.

```
Input:
        R8      <-      FID
        RL7     <-      input byte

Output:
        R5      ->      error status
```

ReadBytes (12)

Reads and counts bytes, from a device, into a buffer in memory.

```
Input:
        R8      <-      FID
        R9      <-      count to be read
        RR10    <-      ptr to memory buffer

Output:
        R7      ->      count returned
        R5      ->      error status
```

WriteBytes (13)

Writes a specified number of bytes from memory to a file or device.

```
Input:
        R8      <-      FID
        R9      <-      count
        RR10    <-      start

Output:
        R7      ->      count returned
        R5      ->      error status
```

ReadLine (14)

Reads and counts bytes input from keyboard, until the first <CR>, into a memory buffer (at a specified address).

```
Input:
        R8      <-      FID
        R9      <-      count
        RR10    <-      destination

Output:
        R6      ->      count returned
        R5      ->      error status
```

Eof (16)

Checks if input character is available from file.

```
Input:
        R8      <-      FID

Output:
        R9      ->      returned status
        R5      ->      error status
```

ResetByte (18)

Resets input file or device.

```
Input:
        R8      <-      FID

Output:
        R5      ->      error status
```

Close (19)

Closes specified disk file or device.

```
Input:          R8      <-      FID number

Output:         R5      ->      error status
```

SetControlByte (20)

Writes a word into device parameter table.

```
Input:
        R8      <-      FID
        R9      <-      word number
        R10     <-      word

Output:
        R5      ->      error status
```

GetStatusByte (21)

Reads a single word from the Device Parameter Table.

```
Input:
        R8      <-      FID
        R9      <-      word number

Output:
        R10     ->      word read
        R5      ->      error status
```

OpenFile (22)

Opens designated file or device for read, write, etc.

```
Input:
          (Files)                          (Device)

  R6    <-  extent length
  R7    <-  mode
  R8    <-  FID                    R8    <-  FID
  R9    <-  fileidentifier
            length
  RR10  <-  address

Output:
  R5    ->  error status           R5    ->  error status
```

Dseek (23)

Positions file pointer as specified.

```
Input:
        R8      <-      FID
        RR10    <-      position

Output:
        R5      ->      error status
```

DGetLen (24)

Returns length of file or number of bytes in the input buffer.

Input:

(Files)			(Device)		
R8	<-	FID	R8	<-	FID

Output:

RR10	->	length	R10	->	zero status
R5	->	error status	R11	->	number
			R5	->	error status

DGetPosition (25)

Gets postion of next byte to be read or written.

Input:

R8	<-	FID

Output:

RR10	->	position
R5	->	error status

BYTESTREAM CALLS AND APPLICABLE DEVICES

The table below shows which bytestream calls can be used with which devices. The devices are:

Console	FID 17. (File Identification 17.) The console includes the keyboard (key) and the screen display (disp).
Disk	BASIC disk files (B), FIDs 0-15; PCOS disk files (P), FIDs 20-24.
RS232	The RS232-C ports; COM, COM1, and COM2; FIDs 19, 25, and 26.
Printer	The system printer, FID 18.

Bytestream Calls and Applicable Devices

Bytestream Call	Console key	Console disp	Disk B	Disk P	RS232	Printer
LookByte (9)	X				X	
GetByte (10)	X		X	X	X	
PutByte (11)		X	X	X	X	X
ReadBytes (12)			X	X	X	
WriteBytes (13)		X	X	X	X	
ReadLine (14)	X					
Eof (16)	X		X	X	X	
ResetByte (18)	X				X	
Close (19)			X	X	X	
SetControlByte (20)					X	
GetStatusByte (21)					X	
OpenFile (22)			X	X	X	
DSeek (23)			X	X		
DGetLen (24)			X	X	X	
DGetPosition (25)			X	X		

DEVICE REROUTING

Standard M20 system devices are the keyboard, the display screen, the disks and their files, the printer, and an RS232 interface. Optional system devices include extra RS232 interfaces and an IEEE-488 interface. The PCOS Device Rerouting commands permit the user to declare any of these devices or files as a replacement or additional source or destination, depending upon its type and the parameters of the command. The source or destination can replace or supplement the fundamental input source and output destination which is the keyboard and the display screen (the console).

In general, the bytestream commands that can use FID 17 (the console) are capable of being modified by the device rerouting capability to accept another bytestream FID in addition to or in place of FID 17.

For further information, see the "Device Rerouting" section in Part 2.

RS232 DEVICE DRIVER

The RS232 device driver is a general purpose asynchronous communication package that allows the user to specify baud rate, parity, stop bits, and data bits for the communication line. The bytestream calls interface with this driver. The user accesses this driver by the SCOM command, described in the PCOS User Manual. The programmer can make use of the SCOM package (via Call User) or the CI command in BASIC, or can access the driver via the appropriate FIDs and bytestream calls.

This driver is discussed in the "Other Drivers" section of Part 2.

BLOCK TRANSFER CALLS

GENERAL

The block transfer system calls allow the programmer to set portions of memory to a fixed value, to transfer data from one portion to another, and to clear memory. For example, block transfer calls are used by the PCOS system to transfer the BASIC interpreter's fixed tables from ROM to RAM, and by BASIC to transfer other initialization tables from ROM to RAM.

BLOCK TRANSFER CALL OVERVIEW

BSet (29)

Sets a block of bytes to a specified value.

```
Input:
        RL7     <-      n (byte value)
        RR8     <-      start
        R10     <-      length

Output:
        R5      ->      error status
```

BWSet (30)

Sets a block of words to a value.

```
Input:
        R7      <-      n (word value)
        RR8     <-      start
        R10     <-      length

Output:
        R5      ->      error status
```

BClear (31)

Sets specified block of memory to zeros.

```
Input:
        RR8     <-      start
        R10     <-      length

Output:
        R5      ->      error status
```

BMove (32)

Moves a block of bytes from one location to another.

```
    Input:
            R7      <-      length
            RR8     <-      start
            RR10    <-      destination

    Output:
            R5      ->      error status
```

STORAGE ALLOCATION CALLS

GENERAL

The storage allocation calls are supported by the memory management routines. User programs and system programs make use of these calls to request and release buffer space in system memory. Memory management makes system memory available as a "heap" without regard for segment boundaries.

LIST OF CALLS

The Storage Allocation calls are:

```
        Dispose (34)                NewLargestBlock (122)
        New (120)                   StickyNew (123)
        BrandNewAbsolute (121)
```

For background information and for details on these calls, see the "Memory Management" section in Part 2.

DATA MANIPULATION CALLS

GENERAL

Numerical Display calls convert the internal form of numeric values into a displayable form. They operate on bytes, words, long words, and integers. String handling calls display a string, perform a carriage return and line feed, and provide the length of an input string.

NUMERIC DISPLAY CALL OVERVIEW

The system calls are:

DHexByte (91)

Displays a byte in hexadecimal.

Input:

```
R12   <-  byte
```

Output:

```
R5    ->  error status
```

DHex (92)

Displays a word in hex.

Input:

```
R12   <-  word
```

Output:

```
R5    ->  error status
```

DHexLong (93)

Displays long word in hexadecimal.

Input:

```
RR12  <-  long word
```

Output:

```
R5    ->  error status
```

DNumW (94)

Displays integer.

Input:

```
R12   <-  integer
R13   <-  field width
```

Output:

```
R5    ->  error status
```

DLong (95)

Displays number as unsigned integer.

Input:

```
RR12  <-  long integer
```

Output:

```
R5    ->  error status
```

STRING HANDLING CALL OVERVIEW

DString (89)

Displays a string message.

Input:

```
RR12  <-  address
```

Output:

```
R5    ->  error status
```

CrLf (90)

Does a carriage return <CR> and linefeed <LF>.

Input:

```
(there are no parameters)
```

Output:

```
R5    ->  error status
```

StringLen (105)

Returns the length of the input string.

Input:

```
RR12  <-  pointer
```

Output:

```
R7    ->  length
R5    ->  error status
```

TIME AND DATE CALLS

The M20 has a real-time clock which maintains both date and time. This clock must be reset each time the system is turned on.

Setting the time or date is done by passing the address of an ASCII string to the operating system. Likewise, the time or date may be read by receiving an ASCII string from the operating system. The formats of these strings are defined by the calls listed below. These correspond to the string values passed in BASIC by TIME$ and DATE$.

These system calls read and set data and time:

SetTime (73)

Sets the system clock.

```
Input:
        RR8    <-  address
        R10    <-  length

Output:
        R5     ->  error status
```

SetDate (74)

Sets the system date-clock.

```
Input:
        RR8    <-  address
        R10    <-  length

Output:
        R5     ->  error status
```

GetTime (75)

Returns the system time.

```
Input:
        RR8    <-  address
        R10    <-  length

Output:
        R5     ->  error status
```

GetDate (76)

Returns the system date.

```
    Input:
            RR8   <-  address
            R10   <-  length

    Output:
            R5    ->  error status
```

USER CALL TO PCOS

One system call has been provided to allow the user to execute any utility or command routine available that could be executed from the PCOS command line. The utility or routine may be transient or resident. The call is:

CallUser (77)

Calls user or PCOS utility or command.

```
    Input:
            RR14  <-  pointer

    Output:
            R5    ->  error status
```

The call can be used in assembler utilities to process PCOS user commands. The "Command Line Interpreter" section in Part 2 gives further information about this call. A detailed explanation of the call is given in the discussion of CallUser (77) in the Assembly Language Manual.

SYSTEM MANAGEMENT

These calls are used for internal PCOS management and are available for general use as needed.

SYSTEM MANAGEMENT CALL OVERVIEW

BExit (0)

Exit from Basic.

(this procedure has no parameters)

Error (88)

Displays standard error message.

```
    Input:
            RH5   <-  parameter number #
            RL5   <-  error code

    Output:
            (no outputs)
```

BootSystem (107)

Reboots (initializes) system.

```
    Input:
            (this call has no parameters)

    Output:
             R5    ->  error status
```

SetSysSeg (108)

Returns caller to segmented system mode.

```
    Input:
            (this call has no parameters)

    Output:
            R5    ->  error status
```

SearchDevTab (109)

Searches system device table.

```
    Input:
            RR10  <-  ptr to device name
            R9    <-  device name length

    Output:
            RL5   ->  entry number
            RH5   ->  device type
            RR8   ->  ptr table entry
            R5    ->  error status
```

KbSetLock (114)

Sets the state of both the shift lock and the cursor lock flags.

```
Input:
        R6    <-  integer flag

Output:
         R7    ->  previous flag
         R5    ->  error status
```

EXPLANATION

BExit exits from BASIC to PCOS. Error displays system call error messages. BootSystem initializes or reinitializes the system. SetSysSeg returns the system to segment mode. SearchDevTab is used to search the device table. RbSetLock sets the state of the shift-lock and cursor-lock flags.

FILE MANAGEMENT

GENERAL

These calls are used intenally to manage volumes and files, and are available for general use.

The file management calls are:

DRemove (26)	CheckVolume (97)
DRename (27)	Search (98)
DDirectory (28)	SetVol (102)
DisectName (96)	DiskFree (106)

EXPLANATION

DRemove removes a file name from a directory. DRename renames a file in a directory. DDirectory displays a directory. DisectName parses file names. CheckVolume forces a check of disk volumes. Search searchs a disk directory for a file name. SetVol sets the active volume for next access. DiskFree returns the number of free sectors on a disk. For further information, see the "File Management" section in Part 2.

IEEE-488 CALLS

GENERAL

The IEEE-488 package supports use of an optional IEEE-488 interface board. The package consists of a group of programs which execute the following BASIC IEEE statements:

ISET, IRESET, ON SRQ GOSUB, POLL, PRINT@,
WBYTE, RBYTE, INPUT@, and LINE INPUT@.

These statements allow the user to perform the following operations on an IEEE-488 bus:

a) Control the IFC (interface clear) and REN (remote enable) lines

b) Receive a service request from another device on the bus, identify the requesting device through serial pooling, and process the service request

c) Write control bytes (e.g., "Device Clear", "Device Trigger", etc.) to other devices

d) Address, write data to, and read data from, other devices

e) Allow the devices within an IEEE-488 network to transfer data on the bus (that is, assign "Talker" status to one de-vice, and "Listener" status to one or more devices)

SUMMARY OF IEEE SYSTEM CALLS

The following system calls are assigned for the IEEE package:

IBSrQ0 (78)	IBPrnt (83)
IBSrQ1 (79)	IBWByt (84)
IBPoll (80)	IBInpt (85)
IBISet (81)	IBLinpt (86)
IBRSet (82)	IBRByt (87)

For further information, see the "Other Drivers" section in Part 2.

If the system does not have an IEEE option board, register R5 will contain error 34 on exiting from any IEEE system call.

OBSOLETE CALLS

The following calls are obsolete. Their functions have been replaced with newer system calls. These calls are still active to provide support for older software. However, they should not be used for current development work and should be replaced when encountered in programs that are being redone. Later versions of PCOS may no longer support these calls.

NewSameSegment (33)

Allocates a block of bytes from heap in the current segment.

```
    Input:
            RR8     <-      pointer to address variable
            R10     <-      size

    Output:
            R5      ->      error status
    address variable->      buffer start address
```

MaxSize (99)

Returns maximum free heap size.

```
    Input:
            (there are no parameters)

    Output:

            R8    ->  size
            R5    ->  error status
```

TopFree (100)

Gets top of heap.

```
    Input:
            (there are no parameters)

    Output:
            RR8   ->  top
            R5    ->  error status
```

ProtRead (101)

Verify protection pattern on track 35.

```
    Input:
            R7    <-  volume number
            R9    <-  string length
            RR10  <-  ptr to string

    Output:
            R5    ->  error status
```

InitHeap (103)

Sets new address for top of heap.

```
    Input:
            R9    <-  address

    Output:
            R5    ->  error status
```

NewAbsolute (104)

Allocates a block at a specified address.

```
    Input:
            RR8   <-  address of block pointer
            R10   <-  length
            @RR8  <-  block pointer

    Output:
            R5    ->  error status
```

GRAPHICS CALLS

The graphics system calls are discussed in detail, with background, in the "Video Display" section of Part 2. A summary of the calls is given below.

SUMMARY OF GRAPHICS CALLS

System Call No.	Mnemonic	Description
35	CLS	Clears current window
36	Chg Cur 0	Positions text cursor
37	Chg Cur 1	Positions graphics cursor
38	Chg Cur 2	Sets text cursor blink rate
39	Chg Cur 3	Sets graphics cursor blink rate
40	Chg Cur 4	Sets text cursor shape
41	Chg Cur 5	Sets graphics cursor shape
42	Read Cur 0	Returns text cursor positon (column, row) and blink rate in current window
43	Read Cur 1	Returns graphics cursor position (column, row) and blink rate in current window
44	Select Cur	Selects graphics or text cursor, or turns off current cursor
45	Grf Init	Initializes screen and sets defaults
46	Palette Set	Selects a global four-color set (only for four-color systems)
47	Define Window	Creates a new window
48	Select Window	Selects another window
49	Read Window	Reads attributes of current window
50	Chg Window	Changes window colors
51	Close Window	Closes the selected window
52	Scale XY	Checks coordinates against window bound-boundaries
53	Map XY	Converts x-y coordinates to absolute values and stores results in graphics accumulator
54	Map CXY	Converts C-value in graphics accumulator to X-Y coordinates
55	Fetch C	Returns contents of graphics accumulator
56	Store C	Sets graphics accumulator to a specified C-value saved by 'fetch'
57	Up C	Moves position (as stored in graphics accumulator) up by one pixel
58	Down C	Moves position (as stored in graphics accumulator) down by one pixel
59	Left C	Moves position (as stored in graphics accumulator) left by one pixel
60	Right C	Moves position (as stored in graphics accumulator) right by one pixel
61	Set Atr	Sets the current color value
62	Set C	Plots a single point
63	Read C	Returns the color attribute of the current point
64	NSet Cx	Draws a horizontal line
65	NSet Cy	Draws a vertical line

66	NRead	Reads a screen rectangle into an array
67	NWrite	Transfers a graphics rectangle from an array to the screen
68	Pnt Init	Specifies global color attributes for PAINT routines
69	TDown C	Moves graphics accumulator down by one pixel after checking the window boundary
70	TUp C	Moves graphics accumulator up by one pixel after checking the window boundary
71	Scan L	Paints left on a scanline up to a border
72	Scan R	Paints right on a scanline up to a border
113	Close All Windows	Closes all existing windows (from 2 to 16)
115	Clear Text	Clear a specified rectangle of text in the current window
116	Scroll Text	Copies a rectangle of text characters in a window to another position of the same window

SYSTEM CALL LABELS

On the following pages is a sample include file that gives a suggested symbolic label for each system call number. This sample can be made into an include file so that programmers may use labels rather than numbers when coding system calls. Using a standard set of labels makes source code easier to read and maintain. The sample shows certain labels "commented out." Some managers may wish to do this as a matter of policy, in order to discourage the use of obsolete calls and the inadvertant use of calls not required within a programming group, such as the graphics calls or ieee calls.

THE MASTER TABLE

After the list of system calls a sample PCOS master table is provided for background information. PCOS internal routines make use of the master table and system calls. They read and update master table values. The sample table gives an idea of the information used by and available to system calls. Master table locations change with every release of PCOS. Therefore, the master table is not available for programmer use, except for certain programmers working directly on the internal PCOS routines. Location address are not given in the sample table in order to avoid confusion.

The PCOS system calls allow programmers access, indirectly, to all master table values necessary for developing programs under PCOS. System calls will continue to work, regardless of PCOS changes, because they are not location dependent.

```
//
//  Sample System Call Include File
//

CONSTANT

//BExit          := 0    // exit to PCOS
LookByte         := 9    // look at what is in buffer
GetByte          := 10   // get byte from file
PutByte          := 11   // put byte in files
ReadBytes        := 12   // read n bytes from file
WriteBytes       := 13   // write n bytes to file
ReadLine         := 14   // read a line from screen
//GetFP          := 15   // get file pointer
Eof              := 16   // test for end of file
//Eom            := 17   // test for end of medium
ResetByte        := 18   // clear ring buffer for keyboard
Close            := 19   // close specified file
SetControlByte   := 20   // set comm port device param table entry
GetStatusByte    := 21   // get comm port device param table entry
OpenFile         := 22   // open specified file
DSeek            := 23   // seek to nth byte
DGetLen          := 24   // get length of file
DGetPosition     := 25   // get current file position
DRemove          := 26   // remove specified file
DRename          := 27   // rename specified file
DDirectory       := 28   // list volumne directory
//BSet           := 29   // block memory seg
BWSet            := 30   // block word set
//BClear                 := 31   // clear memory
//BMove          := 32   // block memory move
NewSameSegment   := 33   // allocate heap storage in same segment
Dispose          := 34   // dispose of heap storage
Cls              := 35   // clear screen
ChgCur0          := 36   // update text cursor position
ChgCur1          := 37   // update graphics position
ChgCur2          := 38   // set text-cursor blinkrate
ChgCur3          := 39   // set graphics-cursor blinkrate
ChgCur4          := 40   // set text-cursor shape
ChgCur5          := 41   // set graphics-cursor shape
ReadCur0         := 42   // fetch attributes of text cursor
//ReadCur1       := 43   // fetch attributes of graphics cursor
SelectCur        := 44   // select graphics or text cursor
GrfInit          := 45   // initialize graphic system
//PaleteSet      := 46   // select color palete
//DefineWindow   := 47   // define a new window
//SelectWindow   := 48   // change window
ReadWindow       := 49   // read window characteristics
ChgWindow        := 50   // change window forground and background color
//CloseWindow    := 51   // close window
//ScaleXY                := 52   // check on window boundaries
//MapXYC                 := 53   // map x&y into graphic accumulator
//MapCXY                 := 54   // map graphic accumulator into x&y
//FetchC                 := 55   // fetch graphic accumulator
//StoreC                 := 56   // store graphic accumulator
```

```
//UpC           := 57   // move graphic accumulator up one pixel
//DownC         := 58   // move graphic accumulator down one pixel
//LeftC         := 59   // move graphic accumulator left one pixel
//RightC                := 60   // move graphic accumulator right one pixel
//SetAtr                := 61   // set current attribute
//SetC          := 62   // plot graphic accumulator position on screen
//ReadC         := 63   // read current attribute
//NSetCX                := 64   // set n pixels on x axis
//NSetCY                := 65   // set n pixels on y axis
//NRead         := 66   // read n pixels into memory
//NWrite                := 67   // write n pixels from memory to screen
//PntInit               := 68   // paint initialization
//TDownC                := 69   // move graphic acc down one pixel & check
//TUpC          := 70   // move graphic accumulator up one pixel & check
//ScanL         := 71   // scan to the left
//ScanR         := 72   // scan to the right
SetTime         := 73   // set time
SetDate         := 74   // set date
GetTime         := 75   // get time
GetDate         := 76   // get date
CallUser        := 77   // call on PCOS transient file
//IBSrQ0                := 78   // enable ieee-488 interrupt
//IBSrQ1                := 79   // disable ieee-488 interrupt
//IBPoll                := 80   // poll listeners
//IBISet                := 81   // ISET call
//IBRSet                := 82   // IRESET call
//IBPrnt                := 83   // output parameter to IEEE-488
//IBWByt                := 84   // write n bytes out to IEEE-488
//IBInpt                := 85   // input off IEEE-488 bus
//IBLinpt               := 86   // input a line off IEEE-488
//IBRByt                := 87   // read bytes of IEEE-488
Error           := 88   // print error message
Dstring         := 89   // print string
CrLf            := 90   // print carriage return and line-feed
DHexByte        := 91   // display hex byte
DHex            := 92   // display hex word
DHexLong        := 93   // display hex long
DNumW           := 94   // display decimal word
DLong           := 95   // display decimal long
DisectName      := 96   // file system
CheckVolume     := 97   // force check of disk volume
Search          := 98   // file system
MaxSize         := 99   // display max free heap block size
TopFree         := 100  // display top of heap
ProtRead        := 101  // verify protection pattern on track 35
Setvol          := 102  // set active volume
InitHeap        := 103  // set top of heap
NewAbsolute     := 104  // allocate block at specified address
StringLen       := 105  // return length of input string
DiskFree        := 106  // free sectors on disk
BootSystem      := 107  // boot default system with no disk parameters
SetSysSeg       := 108  // caller is returned in segmented system mode
SearchDevTab    := 109  // Routine to search system device table
SetHFlag        := 110  // set heap flag value (for PSAVE, etc)
DsplCtlChar     := 111  // switch screen driver mode to display control chars
```

%84000000 ≙ <4>0000 Rom bis %2000 incl. ≙ 8192
%83000000 ≙ <3>0000 Schirm bis %4000 incl. ≙ 16384

```
InitWTree        := 112  // calls pwindo_initwtree (for PSAVE)
DispWRoot        := 113  // calls pwindo_disproot (for PSAVE)
kb_set_lock      := 114  // set shift and cursor lock.
ClearText        := 115  // clear a rectangle of characters to background
ScrollText       := 116  // copy a rectangle of characters within a window
//WarmBootSys    := 117  // Warm boot of system with disk parameters
NetCall          := 118  // Access the Local Area Net
GetVol           := 119  // Returns current volume number as a string
New              := 120  // allocate block in any segment
NewAbsAnySegment:= 121  // newabs not restricted to seg 2
NewLargestBlock := 122  // new for largest block anywhere in mem
StickyNew        := 123  // block to stick around forever
//
//
//


//
//      Sample of Master Table -- Without Addresses
//

CONSTANT

//      This is the PCOS PSA Table location

        psabase          := %8200xxxx    // Master Table Pointer Location

//      These are linkages to the rom routines  %84000000 - %84008192

                                  0000
//      disk_init        := %8400xxxx    // Disk Driver Initialization Call
//      dsk_io           := %8400xxxx    // Disk Driver I/O Call
//      rtc_init         := %8400xxxx    // Real Time Clock Initialization Call
//      rtc_intr         := %8400xxxx    // Real Time Clock Interrupt routine
//      scn_init         := %8400xxxx    // Rom Screen Driver Initialization Call
//      scn_putbyte      := %8400xxxx    // Rom Screen Driver Character display
//      scn_putstr       := %8400xxxx    // Rom Screen Driver String display
        cold_boot        := %8400xxxx    // Reboot PCOS from first file on disk
        warm_boot        := %8400xxxx    // Reboot PCOS from specified file

//      System data size constants

        mail_box_size    := 11           // 11 words
        sc_table_size    := 124          // system calls 0 .. 123
        fp_table_size    := 27           // file pointers of double longs
        nil              := -1           // system nil definition (ffffffff)

TYPE

mastab record   [%82000100 enthält Adresse auf 1
     1  mtBoot      long    // system start address -- one of following:
     2  mtInit      long 4  // either: normal start address
     3  mtChkpt     long 8  // or: checkpoint restart address
     4  mtRomtab    long 12 // address of Rom linkage table [1]
     5  mtMaxsc     word 16 // maximum system call
     6  mtSCtab     long 18 // address of system call table [2]
```

long 32 bits byte 8 bits
word 16 bits

```
    mtExtsc          long  // address of 'extra' system call handler
    mtSaveregs       long  // address of register save area (sc_r0) [3]
    mtCursc          word  // contains current system call number
    mtMaxfp          word  // largest file ID allowable
    mtFPtab          long  // address of FP table [4]
    mtSegtab         long  // address of segment table for PSAVE [7]
    mtHeaptab        long  // address of heap table [8]
    mtInitlist       long  // head of initialization list [9]
    mtChktab         long  // pointer to (future) checkpoint table [10]
    mtPkey           long  // pointer to Pkey Link Table [11]
    mtMonlink        long  // pointer to Monitor Link Table [12]
    mtMailbox        long  // pointer to mailbox [33]
    mtCurwind        long  // pointer to current window table
    mtFonttab        long  // pointer to font table
    mtScntran        long  // pointer to screen translation table
    mtKbdtran        long  // pointer to keyboard translation table
    mtConfig         long  // ptr to system configuration table
    mtWFheap         long  // ptr to Write Font heap allocation record
    mtPrinter        long  // ptr to printer driver variables
    mtClock          long  // ptr to real time clock variables
    mtCurwindadr     long  // adr(current window num. in pwindo.p)
    mtWindtab        long  // adr(table of window pointers in pwindo.p)
    mtSysGetatt      long  // entry point of 'getatt'
    mtSysPutatt      long  // entry point of 'getatt'
    mtFsVol          long  // pointer to current-volume structure
    mtSysinitfils    long  // file system initialization entry
    mtSysxfrblk      long  // file system remap block transfer entry
    mtSysbitinmap    long  // file system bit map routine (?)
    mtKbdEntries     long  // pointer to table of keybd routine entries
    mtKbdVar         long  // pointer to keybd variables
    mtSysfstime      long  // file system "random" number generator.
    mtExtErrProc     long  // external error handler
    mtSysdsk_io I/O  long  // system dsk_io routine
    mtDiskTrace      long  // external disk trace routine
    mtSysVar         long  // system extent and display type pointer
    mtDevTable       long  // system device table
    mtGetSymbol      long  // routine to get two char symbol of last cmd
    mtExtUsage       long  // external usage printing utility
    mtExtDsk         long  // External disk driver hook
    mtFsVoltab       long  // Pointer to voltableadr (cur voltab ptr)
    mtRamfont        long  // Pointer to system font data area
    mtPSaveBlockFlag word  // tells PSave whether it can proceed or not
    mtVolTable       long  // Pointer to volume table for psave deallocate
    mtRealErr        word  // The last error code reported by "error"

  ] // end of 'mastab' type declaration
```

46 Long ×4 = 180 = 190 Bytes ≙ % BE
5 word ×2 = 10 % 100 ≙ 256

11. DEVICE REROUTING

ABOUT THIS CHAPTER

This chapter gives information on how and why to use device rerouting, and provides some information on how this capability is implemented.

CONTENTS

OVERVIEW

Device rerouting allows the rerouting of standard input (the keyboard) and standard output (the display screen), or both. Certain other devices can be substituted for these two devices or used to supplement them. Rerouting can be local (for the duration of a single command) or global (for all commands entered during a work session or until other rerouting is specified).

This section gives information on how and why to use device rerouting, and provides some information on how this capability is implemented. Further information on the use of device rerouting is available in the PCOS Operating System User Guide.

LOCAL AND GLOBAL DEVICE REROUTING

Local device rerouting changes input and/or output devices for one PCOS command only. Following execution of the command, the rerouting command is cancelled. Global device rerouting remains in effect for all commands entered during the rest of the working session or until other rerouting is specified. The difference in specifying local and global rerouting is simple. Rerouting parameters entered with a command take effect only for that command. Rerouting parameters entered without a command take effect globally.

REROUTING PARAMETERS

Device rerouting is implemented by specifying parameters for the devices involved. A plus sign (+) indicates that a device is to be enabled; a minus sign (-) indicates that a device is to be disabled (cancelled). A second indicator--S for source or D for destination--specifies whether the device will be used for input or output.

SYNTAX ELEMENT	MEANING
command	a PCOS command to be executed using the rerouting capability. (Implies local rerouting.)
command parameter	a parameter for the PCOS command
+	the device or file specified is to be enabled
-	enabling of the device or file specified is to be cancelled
S	specifies the source (input) device (upper or lower case)
D	specifies the destination (output) device (upper or lower case).
device name	a string of 13 or fewer printable ASCII characters, the first character alphabetic, specifying the device to be used. The device name must be followed by a colon (:), with the exception of prt.
file identifier	any valid file identifier. A destination file is created, if none exists.

No spaces separate + or -, S or D, and device or filename; these elements constitute one parameter. Rerouting parameters are separated by commas. Upper- and lower-case letters are equivalent.

If additional devices are enabled without disabling the currently active device(s), devices are active simultaneously. Caution must be exercised in input rerouting to prevent intermixing of data from several devices.

The keyboard can be disabled by specifying "-SCONS:". Control cannot be regained, however, unless a "+SCONS:" command is issued by an external

active device or the system is reset.

DEVICE NAMES

The standard (default) device names are:

prt:	Printer
cons:	Keyboard input, video output
com:	Standard RS232-C communications port
com1:	Second optional RS232-C port
com2:	Third optional RS232-C port
ieee:	IEEE-488 optional communications

The com1, com2, and ieee devices require optional hardware.

Because the printer can be a destination device only, the D prefix is optional for prt.

The standard device names can be changed by using the SDEVICE utility. For information, see the PCOS Operating System User Guide. If a standard device name has been changed, the new name must be used in rerouting.

FILE NAMES

A file name must meet PCOS standards. A source file contains text with the desired commands and parameters. A destination file will receive the output. If a file of the specified name does not exist, one will be created on the disk specified or the disk in the drive last selected. Only one source and one destination file may be open at one time.

REROUTING EXAMPLES

The position of the rerouting parameter in the command line is arbitrary.

```
      +DPRT: FL 1:prun.cmd
or
      FL 1:prun.cmd +DPRT:
```

+DPRT: may also be written as +PRT
(in upper- or lower-case letters).

The examples below show local rerouting.

ENTRY	MEANING
VL 0:,-DCONS:,+DPRT:	the directory of the disk in drive 0 is printed (+DPRT:) but is not displayed (-DCONS:)
SS +PRT /CR/	the Set System global parameters are displayed and printed

The examples below show global rerouting.

ENTRY	MEANING
+SCOM:,+DCOM: /CR/	input is received from both the keyboard and and the built-in RS-232-C communications port (provided it has been initialized. Output is displayed and rerouted to the RS-232-C communications port.
-DCOM:,+DPRT: /CR/	the RS232-C port (previously enabled) is cancelled as a destination device and the printer is enabled. Other devices designated as source or destination devices remain enabled.
+D1:fileA /CR/	fileA on the disk in drive 1 is enabled for output. If no fileA exists, it is created. All output is displayed and rerouted to fileA.
+SANYFILE/CR/	the system loads the file, ANYFILE, and takes input from it and the keyboard. If ANYFILE contains the following lines: BA LOAD MYFILE LIST 100-200

	the system will execute the command in sequence, first loading the BASIC interpreter, then loading the BASIC program stored as MYFILE, and then listing lines 100-200. The system remains in the BASIC environment.
+DPRT: /CR/	the printer is enabled for output (+DPRT:)
+D1:output /CR/	the file named "output" on the disk in drive 1 is enabled to receive output (+D1:output)
-DPRT: /CR/ -D /CR/	the printer (-DPRT) and the file (-D) are cancelled. When a file is cancelled, the identifier can be omitted because it it not checked.

DEVICE REROUTING FROM A BASIC PROGRAM

All device rerouting in BASIC is global and remains effective until a command is issued to alter it or until the system is no longer operating in BASIC. The EXEC or call statements are used to execute rerouting commands in BASIC.

ENTRY	MEANING
ba /cr/ EXEC "vl 1:,+D1:OUT" /cr/ . . . . EXEC "-D1:OUT" /cr/	the BASIC mode is entered (ba). The first EXEC statement executes a VLIST command on the disk in drive 1 and routes the volume list to the OUT file on the same disk and to the display. All subsequent output is routed to the file until the command is cancelled by the second EXEC statement.
ba /cr/ EXEC "vl 1:,+dprt:" /cr/ . . . SYSTEM /cr/	the BASIC mode is entered (ba). The directory of the disk in drive 1 is printed (EXEC statement). All subsequent output is also routed to the printer until a SYSTEM command cancels the printer.

IMPLEMENTATION

Any bytestream call that can be used for FID 17 (the console) can be used with device rerouting. PCOS maintains a table of input devices and output devices to be used with these calls when FID 17 is specified. The tables contain entries for the standard devices: the console (keyboard and screen) and the printer. The tables also have space reserved for disk filenames, one for input and one for output. When the RS232-C driver is loaded, or the IEEE-488 driver, additional table entries are added. The table entries use the standard device names or the names specified by the SDEVICE utility.

Associated with the device tables are flags used for rerouting. The flags show whether the device or file is enabled or disabled, and if enabled the flag shows whether the device is a source or destination. Ordinarily, only the console is enabled, for keyboard input and screen output. Device rerouting adds and/or removes flags.

For local rerouting, the flags are reset when the command finishes executing to use only the console entries. For global rerouting, the flags are not reset unless changed. However, their settings are not saved by PCOS after the working session ends. Local rerouting can supercede global rerouting for one command, and then return the system to the specified global rerouting.

USE OF DEVICE REROUTING

Device rerouting can be used for minor conveniences, such as printing information displayed on the screen or saving console output for debugging. Device rerouting is far more than a convenience. It is a very powerful and general tool for programmer use. The opportunities available can be grasped when the programmer realizes how general it is. For example, one system can control another system connected by an RS232-C or IEEE-488 communications facility.

Entire files of commands can be set up and run. Canned procedures can be developed for use by entry of a simple command. This can be done to simplify repetitive operations and to assure accuracy.

Remember that device rerouting can be done from within programs. Examples of rerouting in BASIC have been given. Rerouting can be done within assembly language by using Call User (77).

12. THE KEYBOARD DRIVER

ABOUT THIS CHAPTER

This chapter describes the keyboard driver functions and capabilities and its related utilities. This chapter is the first of five chapters devoted to system drivers.

CONTENTS

OVERVIEW

The keyboard driver is a set of routines devoted to handling the input of characters from the keyboard. The keyboard driver has several tasks. It interprets keystroke input according to the requirements of many national keyboards with different keys and keyboard layouts. PCOS comes with 17 national keyboards, selectable with the set language (SLANG) utility. The driver allows redefinition of keys (done by the CKEY utility): it allows one-key entry of "alias strings" (strings defined with PKEY commands); and it handles some control characters (such as control C, control S, and reset) by taking direct action. Finally, it supports the LTERM utility, which returns a code corresponding to the last line terminator key pressed. These utilities, SLANG, CKEY, PKEY, and LTERM, are described in this section.

RELATIONSHIP OF KEYBOARD DRIVER AND VIDEO DISPLAY DRIVER

Text characters from the keyboard driver's buffer are displayed by the video driver using font tables that describe the appearance of characters on the display screen. A font table contains a matrix for each displayable character. Within each matrix is a pattern of zeros and ones used to generate a pattern of pixels (picture elements = individual dots) on the display screen. Text characters are passed to the video display driver by kernel routines. The two drivers are independent and do not interact.

The programmer can create new font tables for custom character sets or for graphic use. The RFONT and WFONT utilities, which allow font-table creation and change, are described in the next section, "Video Display." The keyboard utilities CKEY and PKEY and the display utilities RFONT and WFONT can be used together to extend system capabilities, to support customized keyboards, and to create entirely new keyboard configurations.

KEYBOARD DRIVER INTERNAL LOGIC

Every time a key is pressed, an interrupt is generated. When the CPU acknowledges the interrupt, it jumps to a memory location defined by the "interrupt table" and executes a routine called "keyboard service interrupt routine." This routine receives the character from the serial interface and acts as follows:

1. Translates the raw (physical) keyboard code into the code associated with the key. The raw code is used as an index into a translation table. The associated code may be one specified by the current national keyboard or it may have been changed by use of CKEY. (The current national keyboard may be specified by SLANG.)

2. If the character is a control character (in the range %A0 - %AF), the driver executes the routine for this specific character.

3. If the character is a printable character, it adds this character to the keyboard buffer. Displayable characters will be sent to the screen by the video display driver.

4. However, there may be an "alias" created by use of PKEY. If so, the driver replaces the single ASCII character in its buffer with the alias string. This string might contain a complete command, which will be executed by PCOS as if it had all been entered character by character.

5. Finally, if the keystroke entered was one of the three terminator characters, the driver sets the associated integer value in a mailbox location available to LTERM. The value %0D (the ASCII /CR/) is placed in the keyboard buffer.

After these operations the routine completes and returns control to PCOS, which returns to the interrupted program.

ASCII characters are listed in Part 3; ASCII character equivalences are listed in Appendix B to the PCOS User Guide.

WHAT THE KEYBOARD DRIVER DOES

The M20 keyboard sends the keyboard driver a code called "raw key code." This code depends merely on the physical position of the key on the keyboard and on whether or not a shift or control key was pressed together with the key itself. Every key in the same position on all keyboards will generate the same raw code in all countries. The raw codes are shown later in this section. Refer to the PCOS Operating System User Guide, Appendix B, for the corresponding codes for each national keyboard. The raw codes and the associated translation codes are independent. Because of a design artifact, the raw key codes and the translation codes for the USA national keyboard happen to be in the same order. The USA national keyboard is given at the end of the section.

The translation table contains, for each raw key code, the corresponding ASCII character to be inserted into the buffer. This table contains two elements:

- A small table (3 words) called cap-lock table

- A large table (256 bytes) containing the translation codes.

In the cap-lock table, every key is represented by a bit; if the bit is 1 this means that this key must be reversed when in cap-lock state. If this bit is 0, this means that the key will not be affected by the cap-lock operation.

The keyboard tables for all countries are contained in a file called "kb.all" in the PCOS diskette. The PCOS diskette officially distributed contains the USA ASCII table. The SLANG utility allows the user to change the table and customize the PCOS with other country keyboards. SLANG replaces both the cap-lock table and the translation table. The only differences among national keyboards are that these tables are different and different markings are on the keytops.

The keyboard buffer is a 64-character ring buffer used by the driver to store the characters just typed and not yet required from PCOS. When the buffer is full, all further characters typed in are lost and a beep signals the buffer-full condition. The buffer contains the translated codes from the national language table and the CKEY changes to that table. PKEY strings are picked up indirectly. If the translated code in the buffer refers to a PKEY string, the driver then looks into a second table that holds PKEY strings, picks up the associated alias string, and makes the alias string available when responding to a request for keyboard input.

If the translated code is a line terminator code, the keyboard driver inserts a %0D into the buffer in its place. It also updates the mailbox area with the code for this particular line terminator. A layout of the mailbox is given in the "Other Drivers" section of Part 2, in the discussion of the IEEE-488 driver.

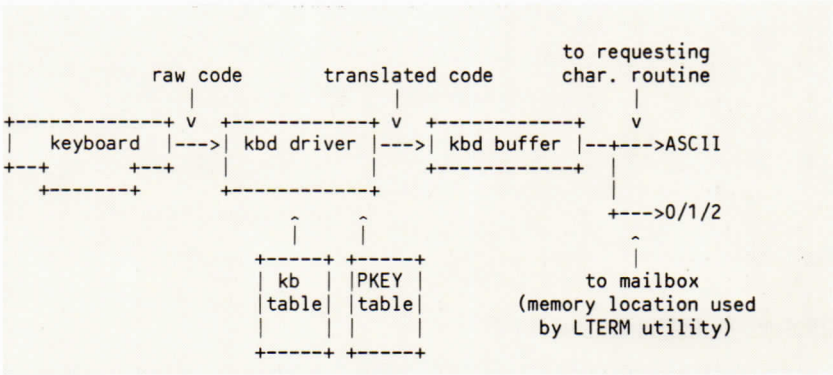


Fig. 12-1 Key Code Generation

The keyboard driver services requests for keyboard input by providing pointers into its internal key buffer. The pointer, when an outside routine requests characters from the keyboard, looks into the buffer and sees the code of the next character.

RAW CODES

The raw codes are the same for all keyboards. The illustration below shows the raw codes, in hexadecimal.

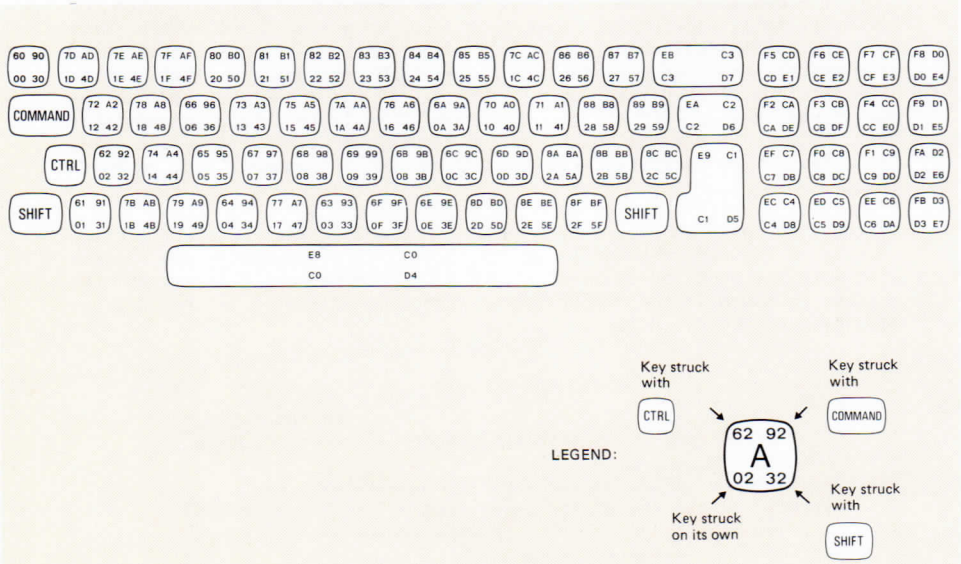


Fig. 12-2 Raw Key Codes

THE CONTROL CHARACTERS

The control characters are keys that cause the keyboard driver to take a specific action. These characters are executed directly by the driver and are never seen by a requesting routine. The control characters correspond to ASCII codes in the range %A0 to %AF and can be disabled by using CKEY to change (delete) those codes in the keyboard translation table. The table is given below.

KEY	ASCII CODE	FUNCTION
/CTRL/ /RESET	%A0	Logical reset. Boot PCOS again
/CTRL/ /B/	%A1	Reserved. Jump into debug routine, if present.
/CTRL/ /C/	%A2	Break facility. Clear keyboard buffer, place 03 in first location.

/CTRL/ /C/	%A0	Halt display. Any key causes scrolling to resume.
/CTRL/ /?/	%A4	Cursor lock. Equivalent to shift lock for numeric keyboard.
/COMMAND/ /?/	%A5	Shift lock. All keys go to shifted mode. Pressing again returns to unshift.
/00/	%A6	Two zeros. Two zeros are placed in keyboard buffer.
/S1/	%A7	End of line. CR in keyboard buffer, 0 in LTERM buffer.
/S2/	%A8	End of line. CR in keyboard buffer, 1 in LTERM buffer.
/CR/	%A9	End of line. CR in keyboard buffer, 2 in LTERM buffer.
/S1/	%AA	End of line. CR in keyboard buffer, 0 in LTERM buffer. (DATEV keyboard)
/S2/	%AB	End of line. CR in keyboard buffer, 1 in LTERM buffer. (DATEV keyboard)
/CR/	%AC	End of line. CR in keyboard buffer, 2 in LTERM buffer. (DATEV keyboard)
	%AD	Reserved
	%AE	Reserved
	%AF	No operation

Table 12-3 Control Characters

THE KEYSTROKE UTILITIES

Keystroke utilities are described briefly below. More extensive descriptions of SLANG, CKEY and PKEY are given later.

THE SLANG UTILITY

The set language utility selects one of the national keyboards to be the current keyboard. It replaces the cap-key table and the translation table.

THE CKEY UTILITY

The change key utility is used to change a single code in the keyboard translation table. CKEY takes effect after SLANG. It is possible to rearrange all the keyboard, changing the meaning of each key including the control keys.

THE PKEY UTILITY

The PKEY utility is used to replace a character generated by the keyboard with a string. PKEY takes effect after CKEY.

THE LTERM UTILITY

The LTERM utility returns the contents of a memory location called mailbox. The keyboard driver maintains the mailbox location. It provides an integer value of 0, 1, or 2 for the three line terminators.

SLANG UTILITY

OVERVIEW

The M20 is marketed with 17 different physical keyboard layouts identified with national requirements. The software generation of the font patterns for all these national keyboards is included with PCOS and can be invoked by the SLANG utility regardless of the actual keyboard used.

The SLANG utility may be invoked

- in direct command mode
- in BASIC
- by an Assembly Language subroutine

USE OF THE SLANG UTILITY

The SLANG utility allows displaying the country codes available or selecting a country code for use as the current keyboard. Entering sl/CR/ will display the menu, similar to the example below.

Country Code Numbers

Italy	0	Yugoslavia	10
West Germany	1	Norway	11
France	2	Greece	12
Great Britain	3	Switzerland/France	13
United States	4	Switzerland/Germany	14
Spain	5	Germany (Original)	15
Portugal	6	Datev	16
Sweden/Finland	7	Delta	17
Denmark	8		

To select a keyboard, provide SLANG with a country code.

```
sl [country code #]
```

To maintain the new keyboard translation after powering off the system, the PSAVE utility can be used to make the condition permanent.

CHANGE KEY UTILITY

OVERVIEW

The change key (CKEY) utility is used to change a single code in the keyboard translation table. It is possible to rearrange the complete keyboard, including the control keys, by using this utility.

The raw key code is used as an index into a table identified with the particular national keyboard in use. CKEY changes the character code associated with any or all the raw key codes in the table.

The M20 keyboard generates 252 raw codes, 0 through 251. The associated translation table entry can be any 8-bit value, that is, any integer ranging from 0 to 255.

USER INTERFACE DESCRIPTION

The user specifies a raw key code in the range 0 to 251 and an associated translation code in the range 0 to 255. Refer to the chart of raw key codes shown above.

If the user enters only the raw key code, the current character code is printed out.

When changing key codes, be cautious with these special cases:

a) The ASCII codes for 0 through 31 (hex 0 through 1F), which ASCII uses for control purposes.

b) The special PCOS system codes described above. They are in the range 160 through 175 (hex A0 through Af).

The form of the change key command is as follows:

```
ck {[%f SHIFTFLAG%[,OLDVALUE%]] | [RAWKEY%  [,NEWCODE%  | NEWCODE$]]}*
```

where:

SHIFTFLAG% = AN INTEGER FROM 0 TO 3.

0 = both flags cleared

1 = shift lock (alphabetic characters) set, cursor lock (numeric characters) cleared

2 = cursor lock set, shift lock cleared

3 = both flags set

OLDVALUE% = previous setting of flags.

RAWKEY% = an integer that defines the desired key.

NEWCODE% = an integer that defines the new desired character code.

NEWCODE$ = a one-character string that defines the new character.

The following examples illustrate the use of CKEY from BASIC:

```
CALL "ck"  ("%F", shiftflag% , @OLDVALUE%)

CALL "ck"  (RAWKEY% , NEWCODE%)
```

or

```
CALL "CK"  (rawkey% , NEWCODE$)
```

Here are a few of the more useful examples of CKEY:

PCOS	BASIC	COMMENT
ck &C3,8	CALL "CK" (&HC3 , 8)	S2 to backspace
ck &64,&A8	CALL "ck" (&H64 , &HA8)	disable CTL C
ck &64,&A3	CALL "CK" (&H64 , &HA3)	enable CTL C
ck &60,&A8	CALL "ck" (&H60 , &HA8)	disable CTL RESET
ck &60,&A1	CALL "ck" (&H60 , &HA1)	enable CTL RESET

CKEY works on all M20 configurations except KATAKANA.

THE PKEY UTILITY

OVERVIEW

The PKEY utility allows replacing a single keyboard character with a string of characters. When the user presses the redefined key, the PKEY function replaces the entered character with the string of characters. The PKEY function can be used to create one-key command entries, for example. For practical reasons, PKEY is often used to redefine a shift or control version of a key, rather than the fundamental keystroke entry.

The replacement string could be a single character. However, for a single character change, using CKEY would be recommended.

The string that substitutes for a character is called the "alias" string and is returned automatically when the character is typed. Note that CKEY changes the translation between the physical keyboard code and the generated translation code, while PKEY converts a translated code into a different code or string. In other words, PKEY works after CKEY.

When PKEY is invoked, a new table is created in memory; in this table will be stored the code that we want to change and the memory address for the alias string. This 26-entry table will contain the first 26 characters changed by PKEY and their associated strings. When there are more than 26 alias strings, the table can chain to another table, and so on, using a link mechanism. When outside routines request characters from the keyboard driver, the driver returns the first character ready on buffer after checking the alias table. If this character is substituted with an alias string, the driver returns all the characters in the alias string. In this manner, an alias string can be larger than the keyboard buffer: 255 bytes compared to 64.

PKEY has four functions that are described below:

DEFINE KEY
DELETE KEY
DELETE ALL
DISPLAY ALL

DEFINE KEY

The syntax is:

```
pk char|int,string...
```

The char|int portion specifies the keystroke entry, and the string portion specifies the alias string.

To define the keystroke entry, specify a character or an integer in the range of 0 - 255. Any key may be defined. In the case of control shift or command shift, an integer must be specified because they do not generate displayable characters.

The string portion of the syntax may be a series of integers or strings or a combination of each. The maximum number of bytes used must not exceed 255.

Examples

The following examples are written in BASIC and assume a USA ASCII keyboard is being used. These examples all do exactly the same thing. They define the capital "A" key to print "clear" followed by CR which causes immediate execution.

```
CALL "pk" ("A",clear,13)

CALL "pk" (65,clear,13)

CALL "pk" (65,99,108,101,97,114,13)

CR% = 13
A$ = "clear"
CALL "pr"  ("A",A$,CR%)
```

To re-define a key it is not necessary to clear it first. Simply invoke PKEY using the integer format to specify the key since its original character is no longer in effect.

DELETE KEY

The syntax is:

```
pk int
```

The parameter is an integer representing the key to be cleared. If the key has not previously been defined, the key retains the original code and the next prompt appears.

DELETE ALL

The syntax is:

```
pk %c
```

All previously defined keys will be returned to their normal state. If no keys have been defined then all keys retain their original code and the next prompt appears.

DISPLAY KEYS

The syntax is:

```
pk
```

Invoking PKEY with no parameters will cause all the currently defined keys to be displayed with their definitions. If no keys have been defined the next prompt appears.

If there are defined keys, the screen will be cleared and set up for 80 column display mode. (All windows will be closed.)

At this point a small window at the top of the screen is created to prevent the heading from scrolling off the screen in the event that there is more than a screenfull of information to display.

The keys and their definitions are then displayed in the main window. There are three fields in the display, the Code field, the Char field, and the String field.

The Code field represents the numeric value of the key that has been defined.

The Char filed will be blank unless the key defined has a displayable character associated with it.

The String field will display the definition of the key as PCOS sees it. This will not always appear exactly as entered. For example, if one enters the following command:

```
pk 65,66                          | define "A" to be "B"
```

the display will show:

```
Code Char                                    String
-----------------------------------------------------
 65   A : B
```

THE USA ASCII KEYBOARD

The keyboard translation table for USA ASCII is given below. The layout and tables are organized in ascending numerical order based on the raw key code. This corresponding order is a design artifact. The order of the translation tables is independent of the raw code order, as is shown in other national keyboards.

Main keyboard keys.

```
 RE      A   B   C   D   E   F   G   H   I   J   K   L   M   N
 0   P   Q   R   S   T   U   V   W   X   Y   Z   0   1   2   3
 4   5   6   7   8   9   -   Ω   @   [   ;   :   ]   ,   .   /
```

This is the cap-lock table; each bit corresponds to a key in the table.

```
    0011111111111111 1111111111110000 0000000000000000
```

Main keyboard UNSHIFTED. Raw key range %00 - %2F.

```
%DD,' ','a','b','c','d','e','f','g','h','i','j','k','l','m','n',
'o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3',
'4','5','6','7','8','9','-','Ω','@','[',';',':',']',%2C,'.','/',
```

Main keyboard SHIFTED. Raw key range %30 - %5F.

```
%DE,'|','A','B','C','D','E','F','G','H','I','J','K','L','M','N',
'O','P','Q','R','S','T','U','V','W','X','Y','Z','_',%21,'"','#',
'$',%25,'&',%27,'(',')','=','±','~','{','+','*','}','<','>','?',
```

Main keyboard CONTROL shift. Raw key range %60 - %8F.

```
%A0,%7F,%01,%A1,%A2,%04,%05,%06,%07,%08,%09,%0A,%0B,%0C,%0D,%0E,
%0F,%10,%11,%12,%A3,%14,%15,%16,%17,%18,%19,%1A,%E0,%E1,%E2,%E3,
%E4,%E5,%E6,%E7,%E8,%E9,%EA,%EB,%00,%FB,%1E,%1F,%1D,%FE,%FF,%A4,
```

Main keyboard COMMAND shift. Raw key range %90 - %BF.

```
%DF,%F8,%80,%81,%82,%83,%84,%85,%86,%87,%88,%89,%8A,%8B,%8C,%8D,
%8E,%8F,%90,%91,%92,%93,%94,%95,%96,%97,%98,%99,%EC,%ED,%EE,%EF,
%F0,%F1,%F2,%F3,%F4,%F5,%F6,%F7,%13,%1C,%FC,%FD,%9F,%F9,%FA,%A5,
```

The following keypad keys generate only three unique codes each. Pressing COMMAND and one of these keys generates the same code as unshifted.

```
                SP  CR  S1  S2

  Keypad:       .   0   00  1
                2   3   4   5
                6   7   8   9
                +   -   *   /
```

UNSHIFTED - raw key range %C0 - %D3.

```
%20,%A7,%A8,%A9,          (Note 1)

%2E,%30,%A6,%31,          (Note 2)
%32,%33,%34,%35,
%36,%37,%38,%39,
%2B,%2D,%2A,%2F,
```

SHIFT - raw key range %D4 - %E7.

```
%20,%A7,%A8,%A9,          (Note 1)

%2E,%30,%A6,%1C,          (Note 2)
%9A,%1D,%9B,%9C,
%9D,%1E,%9E,%1F,
%2B,%2D,%2A,%2F,
```

CONTROL - raw key range %E8 - %FB.

```
%20,%A7,%A8,%A9,          (Note 1)

%B0,%B1,%B2,%B3,
%B4,%B5,%B6,%1B,
%B8,%B9,%BA,%BB,
%BC,%BD,%BE,%BF,
```

Note 1. %A7, %A8, and %A9 will all be interpreted as %0D, an ASCII /CR/. The distinction among these keys is maintained in the mailbox and is accessible by LTERM.

Note 2. %A6 will be interpreted as %3030, two ASCII zeros.

NATIONAL KEYBOARD DIFFERENCES

All national keyboards are shown in Appendix B of the PCOS Operating System User Guide. The translation table and font display is the same for national keyboards, except for eleven keys. The table below shows them.

SYSTEM CALLS

Bytestream Calls are used for reading text from the keyboard. The keyboard FID is 17. The system calls are:

```
LookByte (9)          Eof (16)
GetByte (10)          ResetByte (18)
ReadLine (14)
```

For further information, see the discussion of Bytestream Calls in the chapter entitled "System Calls" in Part 2. Details on using these calls are in the Assembler User Guide.

13. VIDEO DISPLAY

ABOUT THIS CHAPTER

This chapter describes the capabilities of the driver and its related utilities, RFONT and WFONT. The chapter includes information about the display screen characteristics and about the system calls used to display text and to provide graphics capabilities.

CONTENTS

OVERVIEW

The video display driver supports both text and graphic display. This section describes the capabilities of the driver and its related utilities, RFONT and WFONT. RFONT allows creating customized characters and small graphics characters, and can be used to create entire alphabets. WFONT is used to select an alternate display font. The section includes information about the display screen characteristics and about the system calls used to display text and to provide graphics capabilities.

DRIVER FUNCTIONS

The video display is memory-mapped. The screen display hardware reads a pattern from an area in system memory called the screen bit-map and displays that pattern. The video display driver maintains the screen bit-map. It receives both text and graphics material to display.

Text is received as a code, usually in the ASCII range 32-127, and is displayed by means of a font table which has a bit-map pattern for each displayable code. (RFONT can extend the range of displayable codes past 127.) Text codes are passed in R7 to the driver by various kernel routines, system utilities, and user programs. Most of these sources send text via byte-stream system calls. One kernel routine monitors the keyboard buffer and sends newly-entered displayable characters to the video display driver.

Graphics patterns are drawn on the screen bit-map. All graphics input comes to the screen driver by means of system calls. Screen driver routines interpret the graphics call requests in terms of the current state of the driver pointers to the screen bit-map and other internal information, and modify the bit-map to fulfill the request.

DISPLAY SCREEN

The video display screen contains 256 horizontal scanlines, each consisting of 512 pixels (picture elements). There are two display modes:

Mode 0: 16 lines of 64 characters (64 columns)

Mode 1: 25 lines of 80 characters (80 columns)

Mode 0 is mapped across all pixels in a scanline (256 x 512). Mode 1 is mapped across 480 pixels (256 x 480). These modes are set as a global parameter choice by the Set System (SSYS) Utility and their implementation is handled by M20 hardware and PCOS software. For example, when mode 0 is changed to mode 1, the trailing 32 pixels for each scanline are automatically cleared.

SCREEN BIT-MAPS AND COLOR

The video display is memory-mapped, which means that an image is constructed within system memory, read by the display hardware, and shown on the display screen. Black and white displays use 16 K of screen bit-map memory. Each bit represents one pixel; 0 is black, 1 is white. Mapping starts at the upper left of the screen, which is the low memory address, and proceeds across and down. Scanline lengths are according to the mode setting, 512 or 480.

Color systems use two or three 16 K bit-map memories, for four-color or eight-color displays. Mapping is the same, the additional screen or screens provide additional color information. As the figure below shows, for every pixel position on the screen there are either two or three bits in the same relative location in the screen bit-maps. These provide either four or eight values which determine the pixel color.

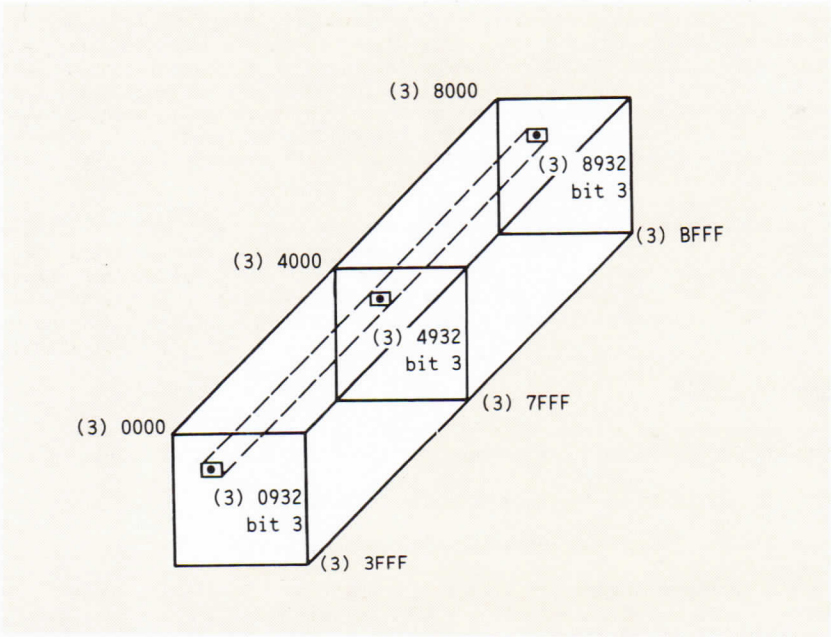


Fig. 13-1 Color Bit Plane Coding

Screen bit-maps are in memory segment 3. The figure shows three bits in three bit-maps that together specify the color of one pixel. Color values are:

Four-Color	Eight-Color
0 color A	0 black
1 color B	1 green
2 color C	2 blue
3 color D	3 cyan
	4 red
	5 yellow
	6 magenta
	7 white

Four-color values are selected from the set of eight. More information on color is given later in this section, in the discussion of graphics.

SCANLINE SKIPPING

Pixels are organized into display fonts that are 8 pixels wide by 10 pixels high in mode 0, 6 x 10 in mode 1. In both modes, text lines are 10 scanlines high. In mode 0, the 16 textlines are automatically spaced within bands of 16 scanlines. The extra scanlines are automatically skipped. In mode 1, the 25 textlines leave only 6 extra scanlines. These extra lines are automatically skipped at appropriate positions on the display screen.

DISPLAY FONT AND CHARACTER FONT

The programmer who is defining display fonts must be aware of the different use of the display font matrix in these two modes. The display font matrix is 8 columns by 10 rows. The character font is 5 x 7 and is placed in the 5 right-hand columns as shown below. When the 8 x 10 font is displayed in 25 x 80 display mode, the two left-most columns are not used. The result is as follows:

16 by 64 Characters (8 x 10)	25 by 80 Characters (6 x 10)
. . . . .	. . . . .
. . . X X X X X	. X X X X X
. . . X X X X X	. X X X X X
. . . X X X X X	. X X X X X
. . . X X X X X	. X X X X X
. . . X X X X X	. X X X X X
. . . X X X X X	. X X X X X
. . . X X X X X	. X X X X X
. . . . .	. . . . .
. . . . .	. . . . .

All characters used in the 16 national keyboards are delineated within the 5 x 7 character font in the position shown. This convention assures vertical and horizontal spacing for text lines. However, it is possible to use the full font when spacing is not required. Font displays can be used to define primitive graphic elements that can be used in constructing graphic displays. These graphic elements provide another method of

implementing graphics displays, independent of the BASIC or PASCAL graphics facilities and the underlying graphics system calls used to implement them.

Such fonts would be suitable for defining symbols to be used singly or in groups as an adjunct to text or for simple displays. Because of automatic scanline skipping in mode 0 display, or the shortened width of the screen in mode 1, such characters cannot be used for full-screen displays.

PCOS makes available two utilities, RFONT and WFONT, that can be used to define extensions or replacements for the display characters furnished in the national keyboards. These facilities have been used to define entire non-Roman alphabets. As mentioned above, they can also be used to define graphic elements.

FONT TABLES

The standard national font tables correspond to the national keyboards. Most of them have 95 displayable characters, corresponding to the ASCII values from 32 to 127. (The Greek, Katakana, and Datev tables are non-standard and have more. Greek and Katakana support display of both Roman and national characters, and Datev has additional special characters.)

The tables are kept internally in binary form. A 95-character font set consists of 95 display fonts in sequence, starting with the font for ASCII 32. In other words, a 95 character font table can be considered as 10 rows of binary values, each row 8 times 95 bits in length.

To display a text value, the driver indexes into the font table and picks up the character font corresponding to the value. (Taking either an 8x10 or 6x10 font, according to mode setting.) The driver places the character font at the current text character location in the bit-map and updates its location pointer, which displays as the text cursor.

READ AND WRITE FONT UTILITIES

The RFONT and WFONT utilities, combined with CKEY and PKEY, enable a programmer to define, call, and use any character fonts desired. These fonts can be customized character sets, non-Roman character sets, or graphics characters. A personalized font thus created can be called from the PCOS environment, called from BASIC or an assembler subroutine, or, if PSAVED, initialized on booting the system.

The standard character sets of the M20 national keyboards use the codes for ASCII 32 through 127 (hexadecimal 20 - 7F). In general, additional character or display fonts are assigned within the range of codes 80 hex to 9F hex, and B0 hex to FF hex. These new characters or display elements would be supplementary to the existing national keyboard font. If it is desired to create an entirely new font, the range of codes from 20 hex to 9F hex and B0 hex to FF hex would be available. Codes 0 hex through 19 hex are used by ASCII for control functions, and PCOS makes special use of the codes from A0 hex to AF hex.

A brief description of RFONT and WFONT usage follows. For details, see the PCOS Operating System User Guide.

RFONT

The RFONT utility is used to create a text file that forms the base for the customized font set. When invoked, a file is created using the character set of the current keyboard or personalized font pattern. To create a customized font, use RFONT (rf) to read the existing font into a file, then edit the file.

RFONT reads the selected display font table and converts the binary display fonts into an ASCII file. The RFONT file is a sequential file that uses a pattern of "-" and "X" to display the fonts. This file can be edited to change or add display fonts.

The ASCII file can be converted to binary for system use by the WFONT utility, which is described later.

For information on the use of RFONT, see the PCOS Operating System User Guide.

RFONT FILE STRUCTURE

The RFONT file has the structure shown in the example below. In this example, the current character set is the USA ASCII. The first character is ASCII 32, the SPACE character (20 hex). Each character is shown in dot-matrix form, with "-" and "X" used to delineate the character.

For reference, the "ASCII" section in part 3 shows the full USA keyboard values. The RFONT utility can be used to display the character fonts for any national keyboard of interest.

```
USA

country 4

matrix height = 10

95 characters

32

--------
--------
--------
--------
--------
--------
--------
--------
--------
-------

33

--------
-----X--
-----X--
-----X--
-----X--
--------
-----X--
-----X--
--------
--------

34

--------
-----...

etc.
```

The meanings of the first four lines of the file are as follows:

Line 1 Reference text name, not used by WFONT and may be used as convenient.

Line 2 Country code from which the original RFONT utility was invoked. Not used by WFONT and is useful as reference.

Line 3 The line height of a valid character font matrix. Reminder only, not used by WFONT.

Line 4 The character count. A number followed by a word (i.e., "character"). This character count must be equal to the total number of matrices in the file to be read by WFONT. Any characters beyond this figure will be ignored.

The remaining lines of the file, from line 5 to the end, are sets of 11-line matrix blocks, each set comprising the character code in decimal followed by an 8 x 10 matrix describing the actual character.

Characters must be defined in sequence and cannot be skipped. Blank displays can be provided for unused values, such as the characters for 160-175 (hexadecimal A0-AF).

WFONT

The WFONT utility does two tasks. First is converts an RFONT file from ASCII to a binary display font set which it places in system memory. Then it changes a system pointer causing PCOS to use the new font set in place of the prior. The other font set is not harmed, and can be restored to use.

The new font set can be stored on disk and saved for later use. The PSAVE command can be used to configure the system so that the new font is the initial font available.

For information on using WFONT, see the PCOS Operating System User Guide.

RFONT AND WFONT - INTERNAL INFORMATION

PCOS reserves a 6-byte area for use by WFONT. This global pointer holds the size and location of the last font set generated by WFONT. When WFONT is invoked it clears that pointer and releases the space to memory management. If WFONT is invoked without a filename, this has the effect of restoring the initialization font to use. When invoked with a filename it converts the ASCII file to a font table and places the size and location information in that pointer space. PCOS then uses the WFONT pointer in place of the pointer to its initialization font set. PSAVE preserves the pointer and the font set.

Any number of RFONT files can be saved as text files and made active by WFONT whenever desired.

SYSTEM CALLS

TEXT

For writing text to the display screen, use these Bytestream Calls with FID 17:

PutByte (11)
WriteBytes (13)

For information, see the discussion of Bytestream Calls in the section on "System Calls" in Part 2. For details on the system calls, see the Assember User Guide.

The text cursor can be modified by system calls (as can the graphics cursor). The cursor shape may be changed and the rate at which it blinks. For information on both cursors, see the discussion on cursors under "Graphics Calls," below. System calls for modifying the text cursor are grouped with the calls for modifying the graphics cursor and explained below.

GRAPHICS CALLS

Graphics system calls are discussed in this section. Later in Part 2 of this manual is an extended discussion of a graphics package that runs under the PASCAL language, and can be used by assembly language programs. That graphics package uses these system calls, and so do the BASIC graphics routines.

GENERAL

The screen area for the M20 display has 256 scanlines by 512 pixels, for either black-and-white or (optional) color. "Pixel," or "picture element" is the fundamental unit of screen display. It is a dot capable of being set to black or white, or to a color on color screens. Both monochrome and color displays have two different display modes: 256 scanlines by 512 pixels, used for 64 character by 16 row text mode; and 256 scanlines by 480 pixels, used for 80 character by 25 row text mode.

A brief overview of the graphics system calls, including background information, is given below. A summary table of calls follows the overview.

CLEAR WINDOW (SCREEN)

System call CLS (35) clears the screen (or current window) and positions the cursor(s).

CURSORS

The PCOS system provides two cursors for the screen, one for text and one for graphics. These may be placed anywhere and XORed with the normal contents of the screen. The cursor may be blinking or nonblinking. There is only one cursor displayed for the whole screen at a given time. However, each window will maintain two cursor positions and two cursor bitmaps.

The standard cursor bitmap is 8 bits wide and 12 scanlines high. The shape may be changed to suit the user's preference, and each cursor may have its own blinkrate. The blinkrate value defines the number of "state changes" per second, and is twice the number of blinks. (Blink and no blink are two states.) A blink rate of 6 gives 3 blinks per second.

During BASIC INPUT and Line INPUT, the text cursor is used. If it was off, it will be turned on, and upon exit will be turned off again. When using BASIC, the cursor is always a standard block, and non-blinking.

System calls 36 through 44 provide the capability to select the text or graphics cursor, select blinkrate, and update its position:

ChgCur0 (36)	ChgCur1 (37)	ChgCur2 (38)
ChgCur3 (39)	ChgCur4 (40)	ChgCur5 (41)
ReadCur0 (42)	ReadCur1 (43)	SelectCur1 (44)

WINDOWS

The screen is initialized to obtain one window by GrfInit (45) which sets default global attributes for both screen and windows, and returns a color flag and a pointer to the "mailbox." The mailbox also contains other flags, indicating the M20 model, etc. A layout of the mailbox is given with the IEEE-488 driver described in the "Other Drivers" section of Part 2 of this manual.

The screen may be divided into windows by splitting along horizontal or vertical lines. There may be a maximum of sixteen windows on the screen, which are assigned window numbers 1 to 16 in order of creation.

A new window is created by splitting the current window into two parts. The current window remains the one selected. A quadrant system is used to identify the new window and the part of the old from which it was created.

In addition to BrfInit(45), system calls 47 through 51 and 113 are provided to define, select, return attributes and/or close windows:

GrfInit (45)
DefineWindow (47)
SelectWindow (48)
ReadWindow (49)
ChgWindow (50)
CloseWindow (51)
CloseAllWindows (113)

GRAPHICS ACCUMULATOR

The graphics routines make use of a global variable referred to as the "graphics accumulator" to define the current absolute screen location.

This graphics accumulator is said to be of type "C". A C-variable is a 32-bit variable containing a memory address and a bit mask for the specified group of pixels at that address. The memory address and the bit mask are each 16 bits in size.

The memory address selects a word in the Bit-Map area, and is in the range %0 to %3FFE (8192 words). The bit mask relates a pixel on the screen to a bit in that area of the Bit-Map specified by the memory address. A bit value of one means ON, zero means OFF.

For example, if the graphics accumulator is assigned the value %20208000, then the first word identifies the sixteen pixels at the center of the screen and the second word sets the first of these sixteen pixels ON.

Conversion routines are provided for converting local x-y coordinates for windows to or from the C-type variable in the graphics accumulator. Most plotting routines manipulate the graphics accumulator in an abstract and machine-independent way. In general, the plotting of a point is at the position defined by the contents of the graphics accumulator.

Likewise, the "current attribute" is a global variable representing the current foreground color. Any plotting or painting routine will set this to the color specified in the high-level BASIC (or other) routine by using SetAtr (61) (set attribute), or is assumed to be the current window's current foreground color by default.

Several system calls (52 through 67, 115, and 116) are provided for scaling or converting coordinates, for manipulating the accumulator, and for drawing lines:

ScaleXY (52)
MapXYC (53)
MapCXY (54)
FetchC (55)
StoreC (56)
UpC (57)
DownC (58)
LeftC (59)
RightC (60)
SetAtr (61)
SetC (62)
ReadC (63)
NSetCX (64)
NSetCy (65)
NRead (66)
NWrite (67)
ClearText (115)
ScrollText (116)

PAINT GRAPHICS CALLS

The PAINT operation fills an area of a window bounded by a specified boundary color (and the window boundaries) with another specified brush color. These system calls implement the PAINT operation:

PntInit (sc 68)	ScanL (sc 71)
TDownC (sc 69)	ScanR (sc 72)
TUpC (sc 70)	

PntInit (68) is called first, and sets the paint and border colors. The colors selected must be legal screen colors (see Color, below).

The remaining calls move the position of the graphics accumulator up or down (checking first if the move is within the boundaries of the current window; if not, an error is returned), and scan left or right to paint the window.

COLOR

There are two color systems, one allowing the display of four colors simultaneously and the other allowing eight colors. In the four-color system, the colors are selected from the full set of eight.

A color code is a value from 0 to 7, expressed in three bits (2,1,0). The color codes for the different systems are:

Black/White	Four-Color	Eight-Color
0 black	0 color A	0 black
1 white	1 color B	1 green
	2 color C	2 blue
	3 color D	3 cyan
		4 red
		5 yellow
		6 magenta
		7 white

The four-color selections are chosen from the eight color possibilities by using PaletteSet (46).

Where a color code exceeds the specified range, the assignment is computed as follows. For a black and white system, codes 2 through 7 are computed by ORing the three bits together. For all cases the result is one. For a four-color system, codes 4 through 7 are computed by ORing bit 2 with bit 0. The result is:

100 = 01	4 = 1
101 = 01	5 = 1
110 = 11	6 = 3
111 = 11	7 = 3

Cls (35)

Clears the current window.

(this call has no parameters)

ChgCur0 (36)

Positions the text cursor.

Input:

```
R8    <-  column
R9    <-  row
```

Output:

```
R5    ->  error status
```

ChgCur1 (37)

Positions the graphics cursor.

Input:

```
R8    <-  x
R9    <-  y
```

Output:

(no output)

ChgCur2 (38)

Sets blink rate of the text cursor.

Input:

```
R8    <-  rate
```

Output:

(no output)

ChgCur3 (39)

Sets blink rate of the graphics cursor.

Input:

```
R8    <-  rate
```

Output:

(no output)

ChgCur4 (40)

Sets shape of the text cursor.

Input:
```
RR8    <-  address
```

Output:
```
(no output)
```

ChgCur5 (41)

Sets shape of the graphics cursor.

Input:
```
RR8    <-  address
```

Output:
```
(no output)
```

ReadCur0 (42)

Returns the position (column and row), and the blinkrate of the current window's text cursor.

Input:
```
RR10   <-  address
```

Output:
```
R7     ->  blinkrate
R8     ->  column
R9     ->  row
```

ReadCur1 (43)

Returns the position (column and row), and the blinkrate of the current window's graphics cursor.

Input:
```
RR10   <-  address
```

Output:
```
R7     ->  blinkrate
R8     ->  x position
R9     ->  y position
```

SelectCur (44)

Selects graphics or text cursor, or turns off current cursor.

```
Input:
        R8    <-  select

Output:
        (no output)
```

GrfInit (45)

Initializes screen and sets defaults.

```
Input:
        (no inputs)

Output:
        R8    ->  color flag
        RR10  ->  pointer
```

PaletteSet (46)

Selects a global four color set (only for four color systems).

```
Input:
        R8    <-  color A
        R9    <-  color B
        R10   <-  color C
        R11   <-  color D

Output:
        R5    ->  error status
```

DefineWindow (47)

Creates a new window.

```
Input:
        R8    <-  quadrant
        R9    <-  position
        R10   <-  vertical spacing
        R12   <-  horizontal spacing

Output:
        R11   ->  window number
        R5    ->  error status
```

SelectWindow (48)

Selects another window.

```
    Input:
            R8    <-  window number

    Output:
            R5    ->  error status
```

ReadWindow (49)

Returns attributes of current window.

```
    Input:
            (no inputs)

    Output:
            R7    ->  window
            R8    ->  x
            R9    ->  y
            R10   ->  foreground
            R11   ->  background
            R5    ->  error status
```

ChgWindow (50)

Changes window colors.

```
    Input:          R8    <-  foreground
                    R9    <-  background

    Output:          R5    ->  error status
```

CloseWindow (51)

Closes the selected window.

```
    Input:
            R8    <-  window

    Output:
            (no outputs)
```

ScaleXY (52)

Checks coordinates against window boundaries.

```
    Input:
            R8    <-  x
            R9    <-  y

    Output:
            R10   ->  return value
```

MapXYC (53)

Converts x-y coordinates to absolute values and stores result in the graphics accumulator.

```
    Input:
            R8    <-  x
            R9    <-  y

    Output:
            (no outputs)
```

MapCXY (54)

Convert C-value in graphics accumulator to x-y coordinates.

```
    Input:
            (no inputs)

    Output:
            R8    ->  X
            R9    ->  Y
```

FetchC (55)

Returns contents of graphics accumulator.

```
    Input:
            (no input)

    Output:
            RR8   ->  C-value
```

StoreC (56)

Sets graphics accumulator to a specified C-value saved by 'fetchc'.

Input:
RR8 <- C-value

Output:
(no outputs)

UpC (57)

Moves position (as stored in graphics accumulator) up by one pixel.

(this procedure has no parameters)

DownC (58)

Moves the position (as stored in graphics accumulator) down one pixel.

(this procedure has no parameters)

LeftC (59)

Moves the position (as stored in graphics accumulator) left one pixel.

(this procedure has no parameters)

RightC (60)

Moves the position (as stored in graphics accumulator) right one pixel.

(this procedure has no parameters)

SetAtr (61)

Sets the current color value.

Input:
R8 <- color

Output:
R5 -> error status

SetC (62)

Plots a single point.

```
    Input:
            R8    <-  operation

    Output:
            (no outputs)
```

ReadC (63)

Returns the color attribute of the current point.

```
    Input:
            (no inputs)

    Output:
            R8    ->  color
```

NSetCx (64)

Draws a horizontal line.

```
    Input:
            R8    <-  count
            R9    <-  operation

    Output:
            (no outputs)
```

NsetCY (65)

Draws a vertical line.

```
    Input:
            R8    <-  count
            R9    <-  operation

    Output:
            (no outputs)
```

NRead (66)

Reads a screen rectangle into an array.

```
Input:
        R8    <-  width (in pixels)
        R9    <-  height (in pixels)
        RR10  <-  pointer to byte array

Output:
        @RR10 ->  address of byte array
        R5    ->  always cleared (no error
                    conditions)
```

NWrite (67)

Transfers a graphics rectangle from an array to the screen.

```
Input:
        R7    <-  logical operation
        R8    <-  maximum width of rectangle
                    in pixels
        R9    <-  maximum height of rectangle
                    in scanlines
        RR10  <-  pointer to a byte array

Output:
        R5    ->  always cleared (no error
                    conditions)
```

PntInit (68)

Specifies global color attributes for PAINT routines.

```
Input:
        R8    <-  paint color
        R9    <-  border color

Output:
        R5    ->  error status
```

TDownC (69)

Moves graphics accumulator down by one pixel after checking the window boundary.

```
Input:
        (no inputs)

Output:
        R8    ->  check value
```

TUPC (70)

Moves graphics accumulator up by one pixel after checking the window boundary.

Input:

(no inputs)

Output:

```
R8    ->  check value
```

ScanL (71)

Paints left on a scanline up to a border.

Input:

(no inputs)

Output:

```
R9    ->  count-1
R10   ->  margin flag
R11   ->  painted flag
```

ScanR (72)

Paints right on a scanline up to a border.

Input:

```
R8    <-  maxcount
```

Output:

```
RR6   ->  C-type
R8    ->  maxcount
R9    ->  count-r
R10   ->  margin flag
R11   ->  painted flag
```

CloseAllWindows (113)

Closes all existing windows (from 2 to 16).

Input/Output:

This call has no parameters

ClearText (115)

Clears a specified rectangle of text in the current widow.

```
Input:
        R10     <- Column (left edge of
                     cleared rectangle)
        R11     <- Row (top row of cleared
                     rectangle)
        R12     <- Column count (width of
                     rectangle)
        R13     <- Row count (height of rectangle)

Output:
        R5      -> error status
```

ScrollText (116)

Copies a rectangle of text characters in a window to another position of the same window.

```
Input:
        R6   <- Color Plane Mask
        R7   <- Logical function (0 for
                  normal copy)
        R8   <- Source column (Left edge of
                  source)
        R9   <- Source row (top row of source)
        R10  <- Destination column (left edge
                  of destination)
        R11  <- Destination row (top row of
                  destination)
        R12  <- Column count (width of rectangle)
        R13  <- Row count (height of rectangle)

Output:
        R5   -> Error status
```

14. DISK DRIVER AND FILE MANAGEMENT

ABOUT THIS CHAPTER

This chapter explains the capabilities of the disk driver and PCOS file management. It provides information about related utilities, system calls used for doing disk file input/output, and system calls used for file management.

CONTENTS

OVERVIEW

The disk driver supports floppy disk and the optional hard disk drives. Based on the disk drive functions, PCOS provides file management capabilities. This section explains the capabilities of the disk driver and gives information about the system calls for doing disk file input/output and file management.

DISK DRIVER AND FILE MANAGEMENT FUNCTIONS

The disk driver manages the physical resources of diskettes and the hard disk. File management routines are structured on top of the driver capabilities in a hierarchical fashion. At the lower levels, the driver provides file management with the functions that convert logical blocks to physical address. At a higher level, file management handles files and directories of files. Some file management utilities operate on a diskette or disk full of files, which is called a volume.

This section starts with a discussion of the driver and then gives information about file management, including an overview of file management utilities. System calls are given at the end.

DISK DRIVER CAPABILITIES

The driver features include write precompensation for the 640KB drive, and support for disk operation on the 512 byte sector diskettes (CPM-86 and MSDOS types) as well as 256 byte sectors.

This driver supports the following disk drive configurations, not all of which are actually provided by Olivetti.

1 160-kbyte floppy drive
2 160-kbyte floppy drives
1 160-kbyte floppy drive, 1 hard disk drive
2 160-kbyte floppy drives, 1 hard disk drive

1 320-kbyte floppy drive
2 320-kbyte floppy drives
1 320-kbyte floppy drive, 1 hard disk drive
2 320-kbyte floppy drives, 1 hard disk drive

1 640-kbyte floppy drive
2 640-kbyte floppy drives
1 640-kbyte floppy drive, 1 hard disk drive
2 640-kbyte floppy drives, 1 hard disk drive

ROM 2.0 is required for support of the hard disk, 160KB, and 640KB floppy disk drives.

Configurations with floppy disk drives of different sizes intermixed are not supported by the disk driver. The driver supports any floppy disk drive in combination with the hard disk drive. However, not all possible drive combinations are actually marketed. For example, the 160-kbyte floppy drives are not used with the hard disk.

This driver will work with all memory and display configurations.

DISKETTE AND HARD DISK CHARACTERISTICS

DISKETTES

All diskettes are 5-1/4 inches in diameter, have a transfer rate of 250 Kbits per second, and an average access time of 303 ms. The individual characteristics are:

	160 Kbytes	320 Kbytes	640 Kbytes
Sides	single-sided	double-sided	double-sided
Density	double-density	double-density	quadruple-density
Read/write heads	1	2	2
Tracks per surface	40 (35 used)	40 (35 used)	80

Ø-34 ai vformat

All diskettes have 16 blocks per track, with 256 bytes per block, except for track zero of side zero. This track is not used. Its 16 blocks have 128 bytes per block, filled with Olivetti control data.

HARD DISK

The hard disk is also 5-1/4 inches in diameter, and can be substituted for a 5-1/4 diskette drive. It is a Winchester-type drive, with the rotating memory and the access heads sealed inside a protective casing. It has three platters with six recording surfaces. Its characteristics are:

Read/write heads	6
Track density	10 tracks per mm (254 tracks per inch)
Tracks per surface (cylinders)	180
Access times	track to track -- 1.1 ms average -- 66 ms maximum -- 158 ms
Rotation speed	3,600 revolutions per minute
Latency time	average -- 8.33 ms
Data transfer rate	5 Mbits per second
Capacity, nominal	11.26 Mbytes
Capacity, formatted	8.85 Mbytes
	256 bytes per block 32 blocks per track 8.192 Kbytes per track 1.475 Mbytes per surface

Drive identifiers:

0, 1 floppy
10 hard disk

INTERFACE DESCRIPTIONS

The following information is provided for background. User calls to the disk driver are actually done by bytestream system calls. These calls are listed later in this section.

DRIVER INITIALIZATION

The global name for this procedure is "disk_init". This routine is called at system initialization time. Any active disk drives are turned off and RAM variables are initialized so that upon the first disk operation the restore operation will be done.

ASSEMBLY LANGUAGE INTERFACE

The global call name of this procedure is "disk_io". This call is the general interface that allows all of the commands described in the Commands section below to be executed. All parameters to and from the driver are passed in registers.

The following registers are used for parameter passing to the driver and returning information from the driver. The driver does not save register values.

```
    Parameters:
                RL7     Driver command
                RH7     Physical drive number (0, 1, or 10)
                R8      Number of blocks to transfer
                R9      Logical block number
                RR10    Buffer address
        Return:
                RH6     If retry count is not zero, this is the
                        type of error
                RL7     Error code: final result after retry
                        attempts
                RH7     Number of retries attempted
```

COMMANDS

The following commands can be issued using the general assembly language call:

```
                0       Read block(s) command
                1       Write block(s) command
                2       Format track command
                3       Verify block(s) command
                4       Initialize driver
```

VERIFY AFTER WRITE FLAG OPTION

A global flag byte with the name "dsk_vfy_flag" will determine if a verify operation should be done after a write operation. If this flag is zero no verification will take place. If the flag is non-zero the verification will take place.

This flag is initially zero but can be changed with the SSYSTEM PCOS command and PSAVED.

Buffer 256 Bytes Ø-255 (Ø-FF)

FLOPPY DISK ERROR CODES

The following is a list of bits that can be set by the driver in the return status byte to reflect error conditions. A zero status byte reflects no errors. NOTE: Bit 0 is the least significant bit.

Bit 0 Illegal parameter(s) error
Bit 1 Not track 0 after restore error
Bit 2 Seek error
Bit 3 Data transfer error
Bit 4 Record not found error
Bit 5 Write fault error
Bit 6 Write protect error
Bit 7 Drive not ready error

HARD DISK ERROR CODES

The following is a list of bits that can be set by the driver in the return status byte to reflect error conditions. A zero status byte reflects no errors. NOTE: Bit 0 is the least significant bit.

Bit 0 Illegal parameter(s) error
Bit 1 Not track 0 after restore error
Bit 2 Abort error
Bit 3 Data transfer error
Bit 4 Record (sector) not found error
Bit 5 CRC on sector id error
Bit 6 CRC on data error
Bit 7 Bad block error OR Drive not ready error

CONCEPTS AND BACKGROUND INFORMATION

The following discussion gives general concepts and specific information useful in understanding the disk drives.

LOGICAL BLOCK NUMBERS

Many operations performed by the driver require the calling program to furnish a "logical block number". This number is a device independent way of representing block of data. Each valid logical block number refers to a physical disk address (a set of sector head, track, and numbers). Valid logical block numbers comprise an unbroken sequence, beginning with 0. The file system may therefore increment or decrement any logical block number within this sequence and produce another valid logical block number designating the next or previous sector, respectively.

The PCOS 1.0 driver mapped logical block numbers into physical disk addresses as follows:

```
          XXXXX    TTTTTT H SSSS
bit:   15   11  10    5 4 3  0
```

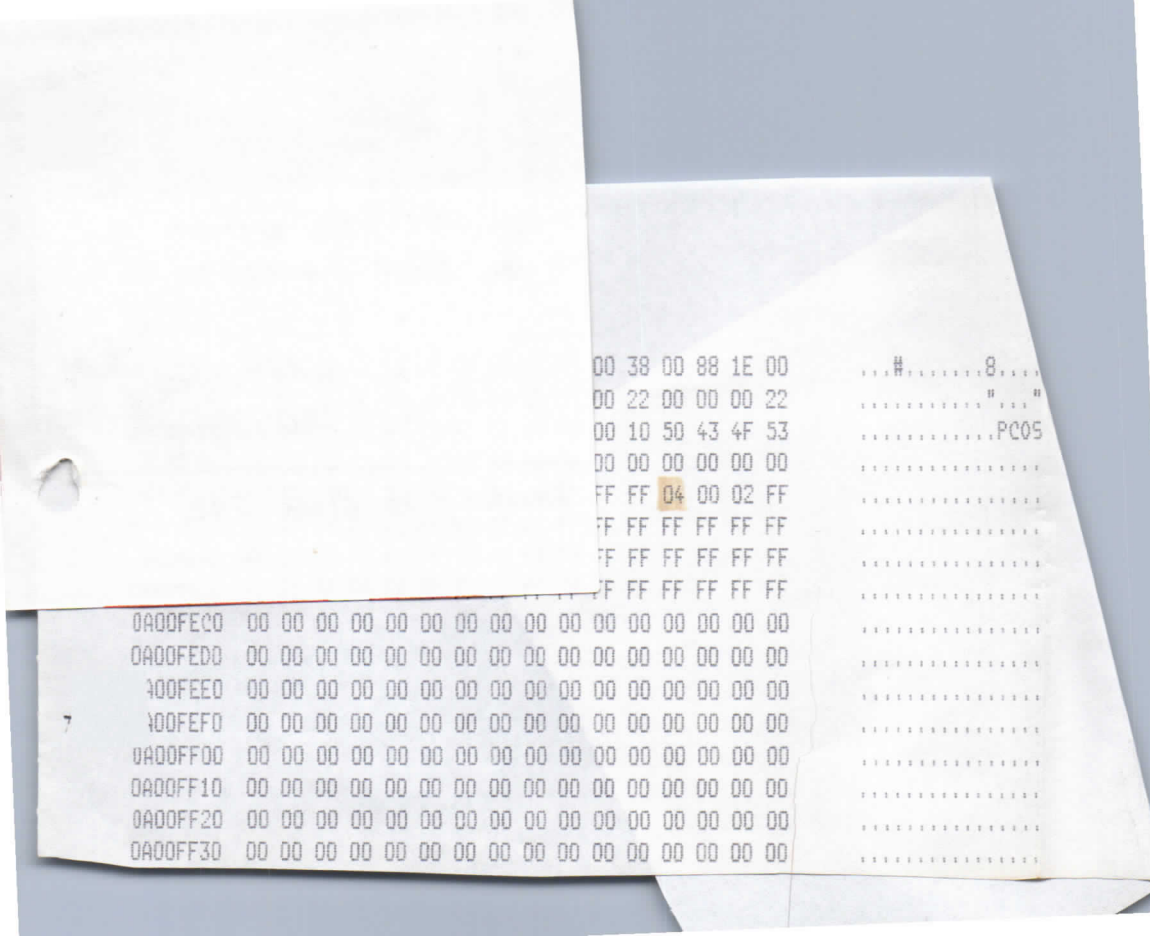
where S = sector number (range 0 - 15), H = head number (range 0 - 1), T = track number (range 0 - 34), and X = unused. This scheme accommodated all valid disk addresses on the 320-kbyte disk, which had 16 sectors/track, 2 sides, and 35 tracks/side.

A new method accommodates all valid disk addresses on the optional drive types. Their parameters are:

Media Type	Floppy Disk						Hard Disk
Drive Type	160 Kbyte		320 Kbyte		640 Kbyte		8 Mbyte
Disk Format	ECMA	IBM	ECMA	IBM	ECMA	IBM	ECMA
Bytes/sector	256	512	256	512	256	512	256
Sectors/track	16	8	16	8	16	8	32
Bytes/track	4,096		4,096		4,096		8,192
Tracks/head(side)	40		40		80		180
Heads(sides)/drive	1		2		2		6
Tracks/drive	40		80		160		1,080
Capacity (KB)	160		320		640		8,640
Capacity (Bytes)	163,840		327,680		655,360		8,847,360

As shown above, an independent "track number" field requires 7 bits for the 640-kbyte floppy drive, and 8 bits for the hard disk. An independent "heads" field requires 3 bits for the hard disk.

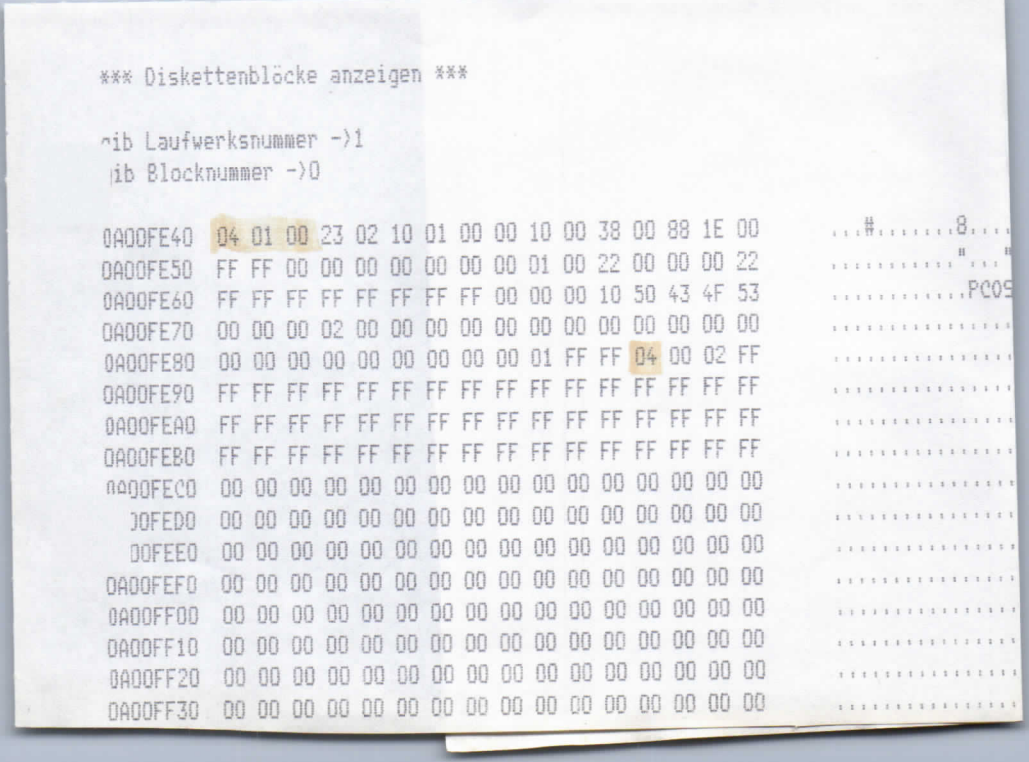
The method currently used maps disk addresses as follows:


00 38 00 88 1E 00 ...#.....8....
00 22 00 00 00 22 .....".....
00 10 50 43 4F 53 .....PCOS
00 00 00 00 00 00 ....
FF FF 04 00 02 FF ....
FF FF FF FF FF FF ....
FF FF FF FF FF FF ....
FF FF FF FF FF FF ....
0A00FEC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0A00FED0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0A00FEE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0A00FEF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0A00FF00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0A00FF10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0A00FF20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0A00FF30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....

vf gibt Track 1-34 an

vqtvl gibt 1088 Blöcke an

Original Hilcin
System Sept. 88



Hard disk:

```
                AAAAAAAAAAA       SSSSS
        bit:   15          5      4   0
```

Floppy disks: (ECMA type)

160-kbyte:

```
                XXXXXX   TTTTTT   SSSS
        bit:   15    10  9    4   3  0
```

320-kbyte:

```
                XXXXX    TTTTTT H SSSS
        bit:   15   11  10    5 4 3  0
```

640-kbyte:

```
                XXXX    TTTTTTT H SSSS
        bit:   15  12  11     5 4 3  0
```

Floppy disks: (IBM PC type)

160-kbyte:

```
                XXXXXXX   TTTTTT   SSS
        bit:   15     9   8    3   2 0
```

320-kbyte:

```
                XXXXXX    TTTTTT H SSS
        bit:   15    10   9    4 3 2 0
```

640-kbyte:

```
                XXXXX    TTTTTTT H SSS
        bit:   15   11  10     4 3 2 0
```

For the hard disk, A = (track X 6) + head; (range = 0 - 2047). Track and head values are determined by dividing the A-field by 6; track value is the quotient, head value the remainder.

WRITE PRECOMPENSATION

Write precompensation is used both on the 640KB floppy disk drive and with the hard disk drive.

For the 640KB drive, the write precompensation bit (bit 7 in general purpose disk output port) is set to one whenever a write operation is occurring on tracks 43 through 80.

For the hard disk, the write precompensation byte is written to the Western Digital controller at initialization time. The track at which the write precompensation is begun is track 128.

DISK FORMATS

Both floppy disks and the hard disk are formatted one track at a time, by repeated "format track" commands issued from the utility VFORMAT.

Any sector interleaving is done when the diskette is formatted. This allows for many different interleave formats that can be optimized for a particular application. The driver software is independent of the interleave and will work with any interleave scheme. Therefore the driver supports the ECMA standard of having the sectors in numerical order, as well as supporting different interleave schemes.

The format program contains tables that have interleave schemes for the various types of disks. For the PCOS 3.0 floppy diskettes, VFormat uses the following interleave table:

```
Physical 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
Logical  09 01 10 02 11 03 12 04 13 05 14 06 15 07 16 08
```

For the PCOS 3.0 hard disk, VFormat uses the following interleave table:

```
Physical 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
Logical  00 04 08 12 16 20 24 28 01 05 09 13 17 21 25 29

Physical 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Logical  02 06 10 14 18 22 26 30 03 07 11 15 19 23 27 31
```

The floppy disk interleave is an interleave by 2. This allows about 12 milliseconds of processing time between sectors. The hard disk interleave is an interleave by 8. This allows about 4 milliseconds of processing time between sectors. These interleaves have been optimized for the PCOS file system.

ECMA COMPATIBILITY

The physical format of the diskette can be written according to the ECMA-70 Standard if the format program that uses the write track command sets up the data to be written properly. NOTE: The floppy disk driver writes and reads only the FB data mark and does not support the F8 deleted data mark on normal sector reads and writes. However, when the write track command is executed. The driver will write whatever data is supplied by the user, so deleted (F8) type sectors can be written in this way.

The format program currently used by the PCOS operating system creates an interleave by 2, which IS NOT the ECMA-70 standard. If the ECMA standard is required, a user supplied format program can be used. Also, ECMA-70 diskettes generated on other machines can be used as long as they do not use the F8 deleted data mark. This may cause a difference in operating efficiency because there may not be any interleave on diskettes made on other machines.

MSDOS, CPM-86, AND IBM PC DISK FORMATS

The APB1086 board using MSDOS and CPM-86 and the IBM PC all use disk formats with 8 sectors of 512 bytes per track. The PCOS 3.0 disk driver is also capable of reading MS-DOS and CP/M-86 diskettes. This is done by specifying diskette type 4 for the 160KB diskette, type 5 for the 320KB diskette, and type 6 for the 640KB diskette.

SYSTEM INTERFACE DESCRIPTION

INITIALIZATION

At initialization time, the attached drive types are identified. Two flags, "drive_0_type" and "drive_1_type", are set, using these values:

0: drive not present or type unknown
1: 160-kb floppy
2: 320-kb floppy 5: MSDOS
3: 640-kb floppy.

A third flag, "hard_dsk_drives", is set to 0 if no hard disk drives are attached, 1 if one is attached. In future releases, this flag may indi- attached, if a multiple-hard-disk

*** Diskettenblöcke anzeigen ***

gib Laufwerksnummer -)1
gib Blocknummer -)0

```
0A00FE40  02 00 00 23 02 10 01 00 00 10 00 38 00 88 1E 00   ...#.......8....
0A00FE50  FF FF 00 00 00 00 00 00 00 01 00 22 00 00 00 22   ..........."...
0A00FE60  FF FF FF FF FF FF FF FF 00 00 00 10 50 43 4F 53   ............PCOS
0A00FE70  00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FE80  00 00 00 00 00 00 00 00 00 00 01 FF FF 02 00 02   ................
0A00FE90  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
0A00FEA0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
0A00FEB0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
0A00FEC0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FED0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FEE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FEF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FF00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FF10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0A00FF30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
sys_conf_tab:
country         4 byte   // keyboard country
mem_size        5 byte   // total number of 16k memory
                         // blocks
key             6 byte   // saves any key pressed at startup
num_drives      7 byte   // number of ready disk drives
stack_top       8 word   // offset of system stack top
colorsyst       A byte   // color/bw flag
dsk_vfy_flag    B byte   // verify after write flag
drive_0_type    C byte   // drive 0 floppy drive type
                         // 0 = type unknown
                         // 1 = 160-kb floppy
                         // 2 = 320-kb floppy
                         // 3 = 640-kb floppy
drive_1_type    D byte   // drive 1 floppy drive type
disk_0_type     E byte   // drive 0 floppy disk type
                         // 1 = 160-kb diskette (ECMA type)
                         // 2 = 320-kb diskette (ECMA type)
                         // 3 = 640-kb diskette (ECMA type)
                         // 4 = 160-kb diskette (IBM PC type)
                         // 5 = 320-kb diskette (IBM PC type)
                         // 6 = 640-kb diskette (IBM PC type)
disk_1_type     F byte   // drive 1 floppy disk type
hard_dsk_drives 10 byte  // number of hard disks in system
```

The system configuration table can be accessed through the master table pointer "mtConfig".

FLOPPY DISK ERROR RECOVERY

The following is a list of various types of floppy disk drive errors and error recovery.

Data Lost Error

This error will result when an interrupt occurs just prior to the beginning of a sector read or write. This error does not increment the retry count and does not in any way reflect on the performance of the floppy disk interface. Whenever this error occurs an immediate retry will be done.

Seek Error

This error results when the track desired is not reached correctly. The recovery for this error is to first restore the drive and then attempt the seek again. This will be repeated up to 6 times.

Büroprogramm beleg
pp
pl alis
pl mc

Main entry ØØ-8A3E

Block Ø D-8A
2
1

sys_config-tab

mtTab

: mtConfig master 82 + ;

5¼ %8000 000 F ⌴ stecke 1
übernimmt MSDOS-Floppy in 1

Einmal ist jetzt dort lesbar
(Test HG-Hintergr. - D(6))
Block Ø-638 ok im Lesen.

Urlader
ab Block 416 - 639 korrekt
in Station 1 lesbar

ab Block Ø-831 korrekt
in station Ø (PCOS-Typ) lesbar

:NT

done to recover from this error.

tempted without head movement.

rformed and the desired track is reached
ration.

en attempted.

ntelligent controller has built in error
t programmable. The disk driver adds another

rite command, if the WD-1000 does not find a
s a CRC error in either the ID field or the
sequence is done. First the operation is
ement 16 more times. A restore operation is
sued. The operation is then attempted
er will cause the above sequence to happen
returned to the caller.

ot track 0 errors, the driver returns the
no retries.

t of the hard disk, 160KB, and 640KB floppy
works properly with the ROM 1.0 if only the
.

f the Bootstrap ROM 1.0 is present, the
o 320KB disk drive type, without looking at

Y SYSTEM CONTAINING BOOTSTRAP ROM 1.0 WILL
IVES. Note, though, that the software will
ootstrap ROM 1.0, although the bootstrap
oy.

HARDWARE CONFIGURATIONS AND VERSIONS

Two jumpers are used to indicate the type of floppy disk drive present in the M20 system. The four floppy disk drive jumper configurations are as follows:

Jumper Settings		
X4 to X5	ZA to	System Interpretation
ON (0)	ZA1 (0)	Skip diagnostics, and query user for floppy drive type.
ON (0)	ZA2 (1)	Floppy disk drive configuration #1 -- 160 kbyte.
OFF (1)	ZA1 (0)	Floppy disk drive configuration #2 -- 320 kbyte.
OFF (1)	ZA2 (1)	Floppy disk drive configuration #3 -- 640 kbyte.

VALID OPERATIONS

The following chart shows the valid combinations of disk drive types, diskette types, and possible operations. The codes have the following meanings:

160 kbyte = 1,320 kbyte = 2,640 kbyte = 3,read = R,and write = W.

drive type	diskette type	support	
1	1	R, W	
1	2	NONE	(side 1 cannot be accessed)
1	3	NONE	(side 1 cannot be accessed) (too many tracks for stepping mech.)
2	1	R, W	
2	2	R, W	
2	3	NONE	(too many tracks for stepping mech.)
3	1	R ONLY	
3	2	R ONLY	
3	3	R, W	

drtype = dktype-->R, W;
drtype > dktype-->R ONLY (e. g., 320 kb diskette on 640 kb drive)
except (2, 1) --> R, W;
drtype < dktype-->NONE. (e. g., 640 kb diskette on 320 kb drive)

FILE MANAGEMENT OVERVIEW

File management routines use and maintain certain tables of information. File management capabilities and methods can be understood by describing these tables and explaining how certain entries are used. These descriptions and explanations are given below.

LOGICAL BLOCKS

The concept of logical blocks should be explained first. A logical block is 256 bytes of disk space. Each block is uniquely identified by a number starting with zero and ending with the capacity of the diskette or disk; that number requires four bytes for storage. A logical block number can be converted to a unique physical address. Finally, logical blocks are often linked in groups of associated logical blocks by placing the next block number in the last four bytes of each block. A nil pointer, hexadecimal FFFFFFFF, ends such a group.

CONTROL TRACK

The control track is in a central location on the diskette or hard disk. The track contains the first blocks of the directory and the bit-map which shows block allocation. File management often needs the information on this track, and its central location minimizes seek-time going between it and the other files.

Note:

The two diskette type numbers used by for MSDOS/CPM diskettes, 4 and 5, are equivalent to types 1 and 2 respectively. Type 4 is the 160KB diskette, and type 5 is the 320KB diskette.

For 160 Kb and 320 Kb diskettes, track 16 is the control track. For 640 Kb diskettes, track 32 is used. The control track is on side 0.

VOLUME DESCRIPTOR BLOCK

The Volume Descriptor Block (VDB) is the first block on the control track, and is logical block zero. Important locations within the block are described below. Locations are given in hexadecimal.

Locations	Contents
0-0D	Optional volume name, left-justified, zero filled.
0E-1B	Optional password, left-justified, zero filled.
1C-1D	Code used to detect whether diskette has changed since last disk I/O. Used to prevent damage to the bit-map.
1F	Diskette type code. 1=160 Kb, 2=320 Kb, 3=640 Kb.
38	Start of bit map.

The bit-map has one bit for each logical block on the diskette or disk, and the sequence of bits corresponds to the logical block numbers. Bits are 0 if the corresponding block is available, 1 if allocated. The bit-map is initialized with all zeros except for bytes 38 and 39, which are all ones because logical blocks 0-15 are the control track blocks. As blocks are allocated and de-allocated, the bit-map changes. File management routines looking for available disk space search the bit-map for free blocks.

Diskette bit-maps fit into logical block 0 with the VDB, except for 640 Kb diskettes, whose bit-maps extend into block 1.

ALLOCATION OF BLOCKS

Blocks are allocated by extents. An extent is a group of logically contiguous blocks on the diskette. The number of blocks in an extent is a Set System parameter setting, and starts out at 8.

Blocks can be chained together by using a 4-byte pointer at the end of each block containing the next block number to use. Therefore, allocated blocks do not have to be physically contiguous. Changing the extent value does not affect blocks allocated under the prior value. The extent value just tells file management how many blocks to allocate when creating or extending a file.

Extents are logically contiguous groups of blocks; that is, they are linked in succession with a 4-byte pointer at the end of a block pointing to the next block. They are often assigned in sequential logical block numbers, especially during early use of the volume. When a volume has seen much use, with files being created and deleted, the extent linkages will skip so that the logical block components of files seem to be intertwined. If this begins to slow file access, FCOPY can be used to make a more contiguous set of files on a new diskette.

The number of blocks in an extent is usually 8, but can be set larger or smaller. The theoretical upper limit is 65,535. When a file is created, the first block of the extent contains the File Descriptor Block and the following blocks contain file data. Extent handling is discussed later under "File Descriptor Block."

FILE DIRECTORY

The file directory starts in logical block 2, the third block of the control track. The remaining blocks in the track are allocated to it with pointers at the end of each block linking to the next. If additional directory space is needed eventually, more blocks can be linked without regard for their actual location. Directory blocks are initialized to hexadecimal FFs, or nil characters. Such a directory appears empty.

THE DIRECTORY ENTRY

The directory entry contains a filename of up to 14 characters and a 4-byte pointer to the File Descriptor Block. Each directory block can contain 14 entries and ends with a 4-byte link to the next directory block. The original directory blocks, 2-F, can contain up to 196 entries. Further blocks are linked as needed. A nil pointer, FFFFFFFF, indicates the last directory block allocated.

FILENAME HANDLING

The filename can be up to 14 characters, and is used as the reference to the file.

When a file is deleted (by FKILL), a nil character, FF, replaces the first character in the filename. That character goes to the end of the filename.

The associated logical blocks are de-allocated in the bit-map. The file can be restored, so long as those blocks have not been re-allocated. RKILL restores a file by checking the bit-map and, if all is well, replacing the first character in the filename. If there had been a 14th character, it is lost.

When a file is hidden, the first character of the filename is replaced with an ASCII /CR/, hexadecimal 01). That first character is saved in the FDB.

FILE DESCRIPTOR BLOCK

The File Descriptor Block (FDB) has the following information. Locations are given in hexadecimal.

Location	Contents
0-1	File size in bytes. The actual size, not the allocated size
2-3	The number of extents, usually 1
4	Reserved for "hiding" a file. The first character of the name of the hidden file goes here
5	Write protect flag; 00 means writable, FF means protected
6-9	Pointer 1. Points to the first extent
A-B	Length 1. The number of blocks in that extent

From this point, the FDB is available for additional pointers and lengths up to Pointer 37 and Length 37. These values are usually zero, and must be zero if the file has no further extents. The last four bytes of the block are available for a pointer to a linked extent descriptor block which could contain up to 42 more extent pointers and lengths, and another pointer to yet another block. There is no theoretical limit for having extents, except the limit of blocks available. In actual use, it is very unusual for an FDB to have a link to an extent continuation block, and this pointer is usually set to nil characters, FFFFFFFF.

OVERVIEW OF FILE MANAGEMENT UTILITIES

A brief overview of file and volume utility programs is given below. Where appropriate, concise information on the internal working of the utility is included.

FCOPY — Copies file by file. Can copy between diskettes of different capacities (unlike VCOPY). Copying from a diskette that has had file activity to a clean diskette will make the files more compact, and more efficiently accessible, because the destination space is allocated in contiguously.

FDEPASS — File delete password. Must know password.

FFREE — Frees unused blocks on diskette for use.

FKILL — Delete a file. De-allocates the sectors. The filename is flagged deleted in the directory, but the name is preserved. RKILL can be used to recover the file so long as the file space is still unused and the directory entry is not changed (by VALPHA, for example).

FLIST — Lists file contents on display screen.

FMOVE — The equivalent of FCOPY for single-drive systems. Holds files in memory while diskettes are swapped in drive.

FNEW	Pre-allocates space for a file, with file name.
FPASS	Set password for file.
FRENAME	Changes file name in directory.
FSAVE	For file transfer between systems. Uses RS-232 connection.
FUNPROTECT	Unprotect a protected file.
FWPROTECT	Write-protect a file.
RKILL	Restores killed file, if possible. FKILL replaces the first character of the filename with a nil character, hexadecimal FF, and places that character at the end of the name entry. RKILL checks to see if the file space has been re- allocated, and if not, restores the original filename. A 13-character filename can be restored unambiguously. The last character filename is lost.
VALPHA	Alphabetizes the directory and squeezes the entries together, compacting them. A killed file cannot be restored after VALPHA.
VCOPY	Copies between diskettes of the same capacity, placing logical blocks in the same locations without compacting. Copies control and boot information, but not serialization data. Determines the maximum size of system memory available, reads logical blocks into memory, writes from memory. Much faster than FCOPY. Cannot be used between diskettes of different capacities because of control track and boot track differences.
VDEPASS	Volume delete password. Must know password.
VFORMAT	Formats diskette or hard disk. Creates sectors and tracks with proper address information, handles interleave of sector address. For hard disk, flags bad sectors. When finished, calls VNEW to set up file system information.
VPASS	Volume assign password.
VNEW	Sets up clean file system by initializing the control information including the bit map. Requires formatted diskette or disk, replaces any existing file system information.
VLIST	Lists all files in volume, for diskette or hard disk, with size and allocation information.
VQUICK	Concise version of VLIST, gives only filenames and number of blocks left in volume.

VMOVE — Equivalent of volume copy (VCOPY) for a single-drive system. Allocates maximum memory available, and will overwrite PCOS. Reads logical blocks into memory, writes out when diskette swapped. Will do multiple passes as necessary, depending on size of diskette relative to memory capacity.

VRENAME — Names or renames a volume.

VVERIFY — Non-destructive test of diskette or disk.

SYSTEM CALL OVERVIEW

For the general user, actual disk input/output operations and file management operations are done using system calls. The calls are described below. For additional information on the bytestream calls, see the "System Calls" section in Part 2. For additional information on these calls and file management calls, see the Assembler User Guide.

DISK BYTESTREAM I/O CALLS

Disk input and output are all done by bytestream system calls. A stream structure for an open file maintains a 32-bit pointer to the current position in the file at which the next byte will be read or written. Files will be extended automatically as they are written, in increments specified by the Set System global parameter for extents.

The following calls are used for disk files. Those marked "ds" are disk specific, used only for disk files. The other calls can be also used for other devices (printer, console, or communication ports).

Close (19)	DGetLen (24)
OpenFile (22)	DGetPosition (25) ds
DSeek (23) ds	

The bytestream calls are described in the "System Calls" section, Part 2. The disk FIDs are:

1 - 15	BASIC files
20 - 24	PCOS files

PCOS can use FIDs 1-15, but BASIC cannot use FIDs 20-24.

FILE MANAGEMENT CALLS

The file management system calls do not have FIDs. They are used to handle file and volume names, to work with directories, and to handle disks and volumes.

DRemove (26)

Removes specified file name from disk directory.

Input:

R9	<-	length
RR10	<-	address

Output:

R5	->	error status

DRename (27)

Renames specified file.

Input:

RR6	<-	old address
R8	<-	old length
RR10	<-	new address
R9	<-	new length

Output:

R5	<-	error status

DDirectory (28)

Displays list of files from specified disk.

Input:

R9	<-	file identifier length
RR10	<-	file identifier address

Output:

R5	->	error status

DisectName (96)

Parses file or volume name.

```
    Input:
            R9    <-  string len
            RR10  <-  string addr
            RR12  <-  names record addr

    Output:
            @RR12 ->  names record
            R7    ->  volume number
            R5    ->  error status
```

CheckVolume (97)

Forces a check of disk volumes.

```
    Input:
            (there are no parameters)

    Output:
            R5    ->  error status
```

Search (98)

Searches on a specified disk for a file name supplied by user.

```
    Input:
            R6    <-  drive
            R7    <-  search mode
            R9    <-  length
            RR10  <-  pointer to buffer for output file name
            RR12  <-  'name pointer' (pointer to input file
                        name)

    Output:
            RR10  ->  pointer to file name
            RR12  ->  fdb logical block
            R5    ->  error status
            R9    ->  length of output file name
```

SetVol (102)

Sets the active volume for the next access.

```
    Input:
            R7    <-  vol number

    Output:
            R5    ->  error status
```

DiskFree (106)

Returns number of free sectors on disk.

```
    Input:
            R7      <-  volume number

    Output:
            RR10    ->  number of sectors
            R5      ->  error status
```

15. OTHER DRIVERS

ABOUT THIS CHAPTER

This chapter contains information on the RS-232-C driver, the IEEE-488 driver, and the system calls through which these drivers can be accessed.

CONTENTS

OVERVIEW

This section contains background information on two device drivers, the RS-232-C driver and the IEEE-488 driver. These drivers are not seen directly by the user. The RS232 driver supports the SCOMM package and the CI calling BASIC. The IEEE driver supports the IEEE commands in BASIC. The programmer could, if necessary, access these drivers through system calls. System call information is given in this section.

These drivers support the following devices in the PCOS device table:

Driver	Default Name	Description
RS-232-C	Com:	Standard RS-232-C communications port
	Com1:	First RS-232-C communications port on Twin Board
	Com2:	Second RS-232-C communications port on Twin Board
IEEE-488	ieee:	IEEE-488 communications port

Com1, Com2, and ieee require expansion input/output boards. These devices can all be used in device rerouting as a source or destination.

More information on these drivers is available in the "I/O with External Peripherals User Guide."

RS-232-C DEVICE DRIVER

USE

The user generally accesses this driver via the SCOMM command or by calling CI in BASIC. Before SCOMM or CF can be used, the RS-232-C driver must be loaded using the RS232.SAV command. Once loaded, the driver stays in memory until the end of the working session.

The driver can also be accessed using bytestream system calls.

DESCRIPTION

The RS-232-C device driver is a general purpose asynchronous communication package. Its implementation allows the user (by means of the SCOMM command or the CI call) to specify the baud rate, parity, stop bits, and data bits for the communication line. In addition, it also supports the standard XON/XOFF handshake, a variable-length input buffer, and both full (character echoing) and half (no character echoing) duplex.

All driver parameters are specified in the Device Parameter Table so that accessing the parameters is simplified. The receive mechanism is interrupt driven and maintains an input ring buffer. The output routine is not interrupt driven.

HANDSHAKE

Handshaking can be enabled or disabled. When handshaking is enabled, the RS-232 device driver implements the standard XON/XOFF serial handshake. When the input buffer is 75% full, the receiving routine will send an XOFF (DC3 = 13 hex) to the transmitting device. When the buffer becomes less than 50% full, the XON (DC1 = 11 hex) character is sent to the sending device.

The receive interrupt routine scans incoming characters for either the XON or XOFF characters, and sets or resets a flag to indicate the handshake status. The transmitting routine looks at this flag prior to transmitting a character and will wait until the XON character has been received before sending the character.

DEVICE PARAMETER TABLE

A table of values called the Device Parameter Table (DPT) is used by the driver to control the serial I/O port. This table contains the port status word, all the I/O port addresses and device commands, and the receive buffer control parameters. The SCOMM command will use the DPT to set the port parameters. A ResetByte call to the driver causes the hardware and input buffer to be reset based on the parameters in the DPT. An OpenFile command initializes the hardware and also allocates the input buffer on the heap. If handshaking is enabled, an OpenFile command will also transmit an initial XON.

INPUT ERROR HANDLING

Errors that occur when a character is input (a ring buffer overflow, or a hardware parity, overflow, or framing error) will cause flags to be set in the driver. The first operation that performs a read from the input buffer will return the error code for Disk I/O Error.

This allows the calling program running to know that an error has occurred, but not on which character the error has occurred. (This approach eliminates the need to store an extra byte in the buffer as a status for each character received).

SYSTEM CALLS

The RS232 driver is accessed by bytestream system calls. The FIDs are 19, 25, and 26 for Com, Com1, and Com2, respectively. The system calls are:

LookByte (9)	ResetByte (18)
GetByte (10)	Close (19)
PutByte (11)	SetControlByte (20)
ReadBytes (12)	GetStatusByte (21)
WriteBytes (13)	OpenFile (22)
Eof (16)	DGetLen (24)

For general information, see the description of bytestream calls in the "System Calls" section Part 2. For further details, see the Assembler User Guide.

IEEE-488 DEVICE DRIVER

USE

This driver package supports devices on the IEEE-488 channel. The command IEEE.SAV is used to load and initialize the package. Once loaded, the package stays in memory until the end of the working session. The functions supported by this package can be accessed using BASIC commands. A set of system calls also accesses the IEEE package. They are described below and in more detail in the Assembly Language manual.

DESCRIPTION

This command loads and initializes the IEEE-488 package -- a group of programs that execute the BASIC IEEE statements ISET, IRESET, ON SRQ GOSUB, POLL, PRINT@, WBYTE, RBYTE, INPUT@, and LINE INPUT@. These statements allow the user to perform the fol- lowing operations on an IEEE-488 bus:

a) Control the IFC (interface clear) and REN (remote enable) lines.

b) Receive a service request from another device on the bus, identify the requesting device through serial polling, and process the service request.

c) Write control bytes (e. g., "Device Clear", "Device Trigger", etc.) to other devices.

d) Address, write data to, and read data from other devices.

e) Allow the devices within an IEEE-488 network to transfer data on the bus (i. e., assigning "Talker" status to one device, and "Listener" status to one or more devices).

It should be noted that IE, when called, stays in memory. The display flag option is NOT used with this call.

IEEE MAILBOX

A mailbox area (9 bytes) is used by the IEEE driver to communicate with the BASIC interpreter.

Format of Mailbox Area

Bytes	Description
0-5	array "IEEE"; values set by IEEE driver for use by BASIC interpreter
6	flag "srq_488"; value set by IEEE interrupt service routine "ibsrq92", tested by BASIC interpreter. Indicates that service request has been received.
7	S1/S2 key depression flag, set by the keyboard driver. 1 = S1 depressed, 2 = S2 depressed. Zero is returned for ANY key except SR1 or SR2.
8	reserved for system use

For information on the array "IEEE" and the flag "srq-488" see the discussion of system calls 78 through 87 in the Assembly Language Manual.

When BASIC calls GRFINIT (45), it is passed the mailbox address in RR10.

IEEE SYSTEM CALLS

If the system does not have an IEEE option board, these system calls generate error 34, "IEEE: Board Not Present." For further information on these calls, see the Assembler User Guide.

IBSr00 (78)

Disables the service request (SRQ) interrupt.

```
Input:
        (no input parameters)

Output:
        R5    ->  error status
```

IBSr01 (79)

Enables the service request (SRQ) interrupt.

```
    Input:
            (no parameters)

    Output:
            R5    ->  error status
```

IBPoll (80)

Polls a specified device on an instrument bus.

```
    Input:
            R8    <-  talker addr

    Output:
            RR10  ->  ptr to status
            R5    ->  error status
```

IBISet (81)

Causes remote enable (REN) or interface clear (IFC) to be sent.

```
    Input:
            R8    <-  operand

    Output:
            R5    ->  error status
```

IBRSet (82)

Causes remote enable (REN) message to be sent false.

```
    Input:
            (no parameters)

    Output:
            R5    ->  error status
```

IBPrnt (83)

Checks address and then causes output of data bytes.

```
    Input:
            RR6   <-  buffer addr
            R8    <-  listener addr
            R9    <-  buffer len, in bytes
            R10   <-  delimiter

    Output:
            R5    ->  error status
```

IBWByt (84)

Outputs commands (optional) and writes data bytes (optional).

```
    Input:
            RR6   <-  numval addr
            R8    <-  comlist length
            R9    <-  numval length
            RR10  <-  comlist addr

    Output:
            R5    ->  error status
```

IBInpt (85)

Places bytes received, into a buffer.

```
    Input:
            R7    <-  buffer length
            R8    <-  talker addr
            R9    <-  listener addr
            RR10  <-  buffer addr

    Output:
            R5    ->  error status
            R7    ->  number of bytes not read
```

IBLinpt (86)

Places bytes received into a buffer as a single line of data.

```
    Input:
            R7    <-  buffer length
            R8    <-  talker addr
            R9    <-  listener addr
            RR10  <-  buffer addr

    Output:
            R5    ->  error status
            R7    ->  number of bytes not read
```

IBRByt (87)

Outputs commands (optional) and reads data bytes (optional).

```
Input:
        RR6   <-  buffer addr
        R8    <-  comlist length
        R9    <-  buffer len, in bytes
        RR10  <-  comlist addr

Output:
         R5    ->  error status
```

ERROR HANDLING

Possible IEEE call errors are:

1. Incomplete data handshake. The handshake was aborted by operator input.

2. No active device. An attempt was made to output data without any addressed listeners or to input data without an addressed talker.

3. An illegal function call. Bad parameters.

4. Bad file data. More parameters than data.

5. Communications buffer overflow. Too many bytes in line.

6. Type mismatch. Variable and data of different type.

16. THE PRINTER DRIVER AND PRINTER MANAGEMENT

ABOUT THIS CHAPTER

This chapter describes the capabilities of the printer driver and the use of its associated utilities. Included is a brief discussion on supporting two printers and on using the printer driver to drive special devices, such as plotters.

CONTENTS

OVERVIEW

This section describes the capabilities of the printer driver and the use of its associated utilities. The utilities include SFORM, which gives the user control over printer configuration parameters; LSCREEN, which allows the printing of display screen text; and SPRINT, which allows the printing of both text and graphics from the display screen contents. LSCREEN and SPRINT use system capabilities outside the printer driver, and their implementation is briefly discussed.

This section includes a brief discussion on supporting two printers and on using the printer driver to drive special devices, such as plotters.

PRINTER DRIVER DESCRIPTION

The printer driver supports printing both ASCII text and graphics (on certain printers) using either parallel or serial output, and provides a variety of printing options. The SFORM utility can be used to set or display the parameters which control the driver functions. Figure 16-1 below, shows these interrelationships.

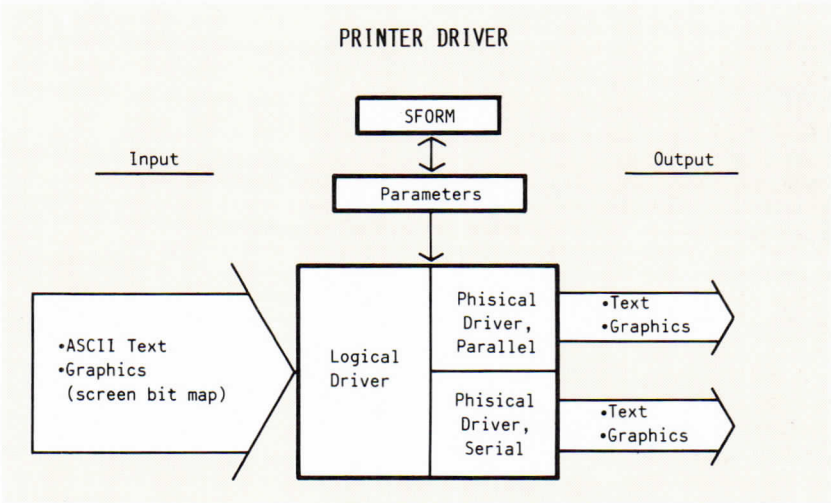


Fig. 16-1 Printer Driver

PRINTER OUTPUT

The standard M20 has both a parallel and a serial interface available for connecting printers. The parallel interface is Centronics compatible, and any compatible parallel printer may be connected. The serial interface uses the RS-232-C standard. Part 1 of this manual provides a list of available printers in the "Hardware Configuration Options" section. In general, dot matrix printers are connected via the parallel interface, and the daisy-wheel printers via the serial interface. Some printer models are available with either parallel or serial interface. Parity, when present, is handled by hardware.

PRINTING TEXT

The driver transmits text to the printer in a stream of ASCII bytes. The driver may receive information from the printer in the form of a status byte, which contains information on whether the printer is ready to receive data and on error conditions.

The printer receives ASCII codes, interprets them, and prints them as characters according to its internal control mechanism. Dot-matrix printers have internal fonts separate from the PCOS fonts, often in a 7 x 7 matrix. The ASCII code is used to look up the equivalent font. Daisy-wheel printers interpret the ASCII code as a position on the wheel. For further information on the printer's display of the ASCII codes, the documentation for the printer must be consulted.

The ASCII values are printed according to the ASCII assignments used by the video display. These assignments are set and modified by SLANG, CKEY, and PKEY. Non-ASCII fonts developed using RFONT cannot be printed as text. (The assigned code may print as some ASCII character.) Those printers that can print graphics print RFONT characters in the same manner as other graphics. Figure 16-2 below gives an overview of text printing.

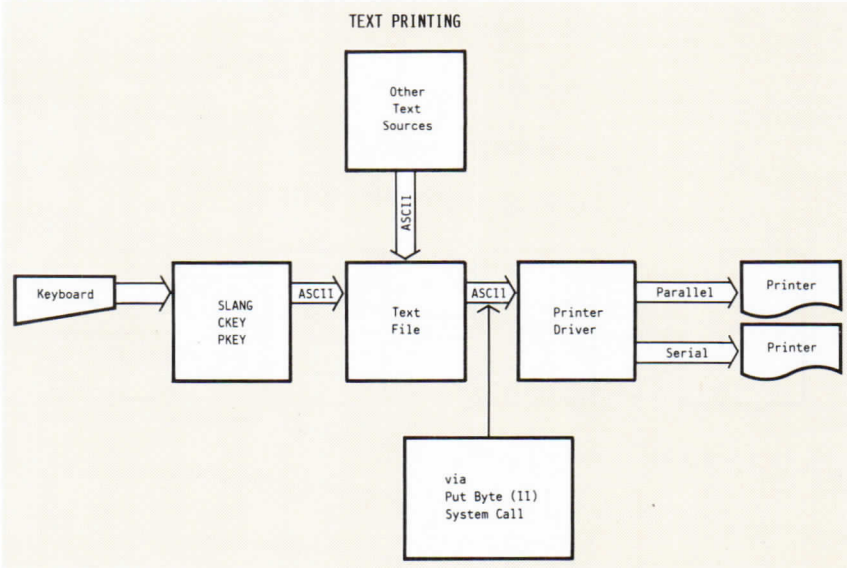


Fig. 16-2 Text Printing

PRINTING GRAPHICS

Graphics printing is handled according to the constraints of those printers which can print graphics. The driver builds a block of dots based on the contents of the screen bit-map in pixels and the requirements of the printer, and sends that block to the printer as a succession of bytes. Figure 16-3 below, gives an overview of graphics printing. More details are given in the discussion of SPRINT later in this section.

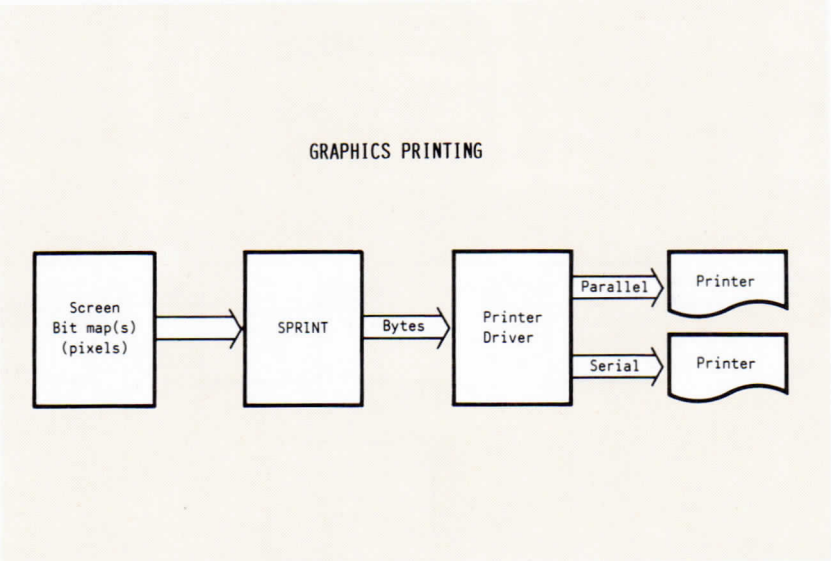


Fig. 16-3 Graphics Printing

USING SFORM TO SET THE PRINTING ENVIRONMENT

The SFORM command specifies the type of printer being used, the interface, and the printing format, and allows the user to change parameters in the printer driver. If the SFORM command is invoked without specifying parameters, it gives a display of the current values for the printing environment.

SFORM parameters are:

auto — The AUTO parameter shows the status of the SFORM parameters, and can be used to change that set of values. OFF indicates that the default values are in effect. ON indicates that new values (selected by SFORM) are in effect. This is true even if new values are PSAVED. Setting OFF returns to the default values, setting ON selects the new values.

ptype — The ptype parameter specifies the type of printer: PR1450 (the default value), PR1471, PR2400, PR1481, PR2300, ET-121, ET-231, PR-430, PR2835, PR320, or TRANSP (transparent mode). In transparent mode, file contents are printed exactly as specified in the file independent of the type of printer. No additional end-of-line characters or form feed characters are added.

lines — The lines parameter specifies the number of lines to be printed on each page before software-generated automatic form feed. Zero implies that no form feed will be issued. The default value is 60.

spacing — This parameter specifies the number of inter-line spaces between printed lines. Its value can be 1 (single spacing); 2 (double spacing), etc. The default is 1.

compress — This parameter specifies which of six styles of characters are to be used. It is made up of two characters, the first of which must be either w = wide (bold) or n = narrow width. ("Wide" printing is only supported on certain printers; for example, PR1450, PR1471, PR1481.) The second character specifies the pitch at narrow width and must be c = compressed; that is, 16.6 characters per inch; e = elite; that is, 12.5 characters per inch; or p = pica; that is 10 characters per inch. At "wide" width, the printer will print two horizontally adjacent dots for each one that would have been printed at narrow width. The default values are n,e.

interface — This parameter specifies whether the printer is to be connected to the serial (RS-232-C) or parallel (Centronics-like) interface. Its value must be either se = the serial (RS-232-C) interface or pa = the parallel (Centronics-like) interface. The default value is pa.

title — This parameter defines the title to be printed at the top of each page. It can comprise as many as 24 characters and must be enclosed in quotation marks. Entering a value of '}' deletes the current title. The default value is no title.

The SFORM settings take effect during later working sessions according to the current auto setting. The prior SFORM settings saved by PSAVE take effect when the current auto setting is ON. For more complete information on SFORM, see the PCOS Operating System User Guide.

SUPPORTING TWO PRINTERS

It may be desirable to support both a dot-matrix printer with graphic capabilities and a daisy-wheel printer, or some other combination of printers. PCOS assumes the presence of only one printer, FID 18. However, SFORM allows switching between printers and interfaces. One printer could be used on the serial interface and one on the parallel. Two PCOS versions can be configured, one for each, using SFORM and PSAVE. Then whichever system is desired can be loaded, either by booting or by using PRUN.

An alternative method allows using two printers without switching between them. A parallel printer is connected as FID 18 and controlled by the printer driver. A serial printer is driven using the RS232 driver. This approach requires special programming to support the serial printer and does not provide support for the printer utilities: SFORM, LSCREEN, and SPRINT.

CONNECTING OTHER DEVICES TO THE DRIVER

The printer driver can be used in transparent mode to drive special devices. For example, a plotter could be connected as FID 18. In transparent mode, (the SFORM setting of TRANSP for the ptype parameter), the plotter would receive exactly the characters sent to FID 18 without change, addition, or deletion. Using the printer driver in this fashion can be very useful for the programmer who must provide interface and control routines for a special device, so long as the device is appropriate.

PRINTING SCREEN TEXT WITH THE LSCREEN UTILITY

LSCREEN dumps the text contents of any screen window onto any alphanumeric printer supported by the M20 PCOS system. It works with the current system font, which may be any 95-character font set.

The utility is presently limited to 95-character font sets, although the fundamental approach could be extended for additional characters. It works by reading the screen bitmap and, in effect, comparing the screen bits with the current system font to determine the corresponding character codes.

USING LSCREEN

The command syntax is as follows:

```
ls [window_number] /CR/
```

The text in the specified window will be printed. If no window number is given, the text in the current window will be printed. Window 0 refers to the entire screen.

For example, type

```
ls /CR/
```

to display the current window. To display window 3, type

```
ls 3 /CR/
```

The following are some examples of BASIC calls to this utility:

```
100 EXEC "LS"                    ' print current window
200 EXEC "LS 5"                  ' print window 5
300 CALL "LS"(windnum)           ' print window %windnum
```

The specified window may have text with any spacing in it. Any screen data within the normal 5 by 10 character dot matrix not recognizable as text will be printed as blanks. On a color system, only screen plane 0 will be read; if the background color is even, the foreground color must be odd and vice versa.

The utility does not distinguish highlighted displays on the screen, but does recognize and print both 'normal' and 'inverse video' characters.

LSCREEN works with 95-character fonts, a limit which could be changed if necessary. The 95-character limit is based on the actual internal representation of fonts, which is horizontal across the entire set of font characters. Decoding any other length of font set with LSCREEN yields scrambled characters.

Therefore, LSCREEN cannot print the Greek or Katakana fonts. These fonts both have a full Roman character font set followed by the Greek or the Katakana characters. The full font tables extend past the 95-font limit. For the same reason, special RFONT character sets longer (or shorter) than 95 characters cannot be printed. Modified RFONT characters within a 95-character font set would be detected, but would print as ASCII characters.

IMPLEMENTATION OF LSCREEN

The utility accesses the font table via the font pointer data structure referenced in the master table. It also accesses the parameters for the specified window, which define the size, position, and character and line spacing of the window. Hence, the PCOS master table, font pointer table, and window data structures are used by the utility. Also, the character format (the exact positioning of characters within a window, given the horizontal and vertical spacing) is determined by reading screen memory to locate the placement of the character bitmaps.

A hash-code mechanism is used to speed the font table lookup. The hash table used is constructed at the beginning of execution of the program from the current system font. Font characters are evaluated into hash numbers which are used to look up a small set of corresponding values; typically, a has number has fewer than half a dozen corresponding values. The corresponding values are ASCII codes. The printer receives ASCII codes and prints them according to its own fonts or its daisy-wheel characters.

PRINTING TEXT AND GRAPHICS WITH THE SPRINT UTILITY

This utility prints an image of all text and graphics displayed on the screen or within a specified window. The display is transferred pixel by pixel. SPRINT can only be used with printers that have graphics capabilities. This command can specify the "polarity" of the printout; that is, white on black printout for white on black display, or black on white printout for white on black display. Printouts from color screens are printed in black and white, one value used for the foreground color and the other value for all other colors. A title can also be specified to appear at the top of the printout with or without date and time.

SPRINT PARAMETERS

The specific parameters for SPRINT are as follows:

window number — from 0 through 16, inclusive, representing the window to be printed. 0 indicates the entire screen, whether or not the screen is divided into windows. The default value is 0 (the whole screen).

polarity — n or p, describing the paper image in monochromatic terms. Positive (p) gives black on paper for black on screen, negative (n) gives black on paper for white on screen. The default value is "p."

title — is an optional alphanumeric title string, no longer than 24 characters, shown above the graphic output. The default is no title.

date/time — is specified by dt, or no, determining whether or not the current date and time are to be printed above the graphics output. The default is "no."

SPRINT IMPLEMENTATION

To dump the contents of the screen or window, SPRINT takes a sequence of bits from the screen bit-map as input and manipulates the information according to the need of the printer. SPRINT finds the printer model in the parameter information maintained by SFORM. There are two fundamentally different approaches used by Olivetti printers when printing graphics. The "raster-format" printers receive a line of pixels, from left to right or right to left, sent in bytes. The "column-format" or "band-format" printers receive a succession of vertical columns of pixels, from top to bottom and then from one side to the other, sent in bytes. Figure 16-4, below, is a general illustration of the two approaches.

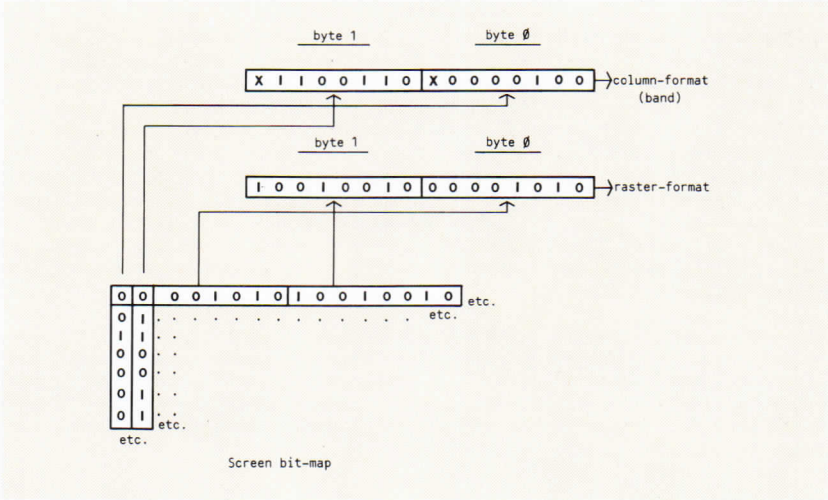


Fig. 16-4 Encoding Pixels

a) Raster-oriented printers receive a series of lines of dots, from one side to the other, shown here as left to right. In the example, the printer is being sent 8-bit bytes (as for the PR2300), but it could be sent fewer bits (7 bits for the PR2400).

b) Band-oriented printers receive a band of parallel columns of 6 or 7 dots, from top to bottom, then from one side to the other, shown here from left to right. The PR1450 uses 6-dot columns. The PR1471 and PR1481 use 7-dot columns.

c) Printers that use fewer than 8 bits receive 8-bit bytes with fill-bits, shown by x's.

Here are some examples of these two approaches. The thermal printer, PR2400, is raster-format, and writes a scan line of pixels at a time. The spark ink-jet printer, PR2300, is also raster-format but buffers several lines of pixels and writes a scan line at a time. It uses a print head which is a graphite cylinder that sweeps horizontally. The PR1450 is band-format, and prints 6-line bands. The PR1471/81, also band-format, prints 7-line bands.

In addition to encoding the pixels, it is necessary to fill out bytes for those printers that receive 6 or 7 bits. The driver always sends 8-bit bytes. SPRINT provides data bytes with a high-order 01 or 1, for 6-bit or 7-bit data. Control codes are given a high-order 00 or 0.

CORRECTION TO PRESERVE ASPECT RATIO

In the graphics printers supported, horizontal and vertical pixel density are the same. Therefore, in PR2300 output image, there are 108 pixels per inch horizontally and vertically; in a PR2400 or PR1481 output image, 70; etc. Consequently, to print a square, the number of horizontal and vertical pixels must be equal. However, in an M20 screen image, the horizontal pixel density is greater than the vertical print density, since 512 horizontal pixels are crowded into a length less than twice as great as the screen image height (which contains 256 pixels). A rectangle containing an equal number of pixels horizontally and vertically would be longer vertically than horizontally, not square; and the full screen (which has an aspect ratio of 3:2) would produce an output image with an aspect ratio of 4:2.

To produce output images without noticeable aspect ratio distortion, SPRINT adds a correction line after every 5 screen image lines. This stretches the image vertically, compensating for the horizontal decompression in the output image. The content of the correction line is calculated from those of the screen image lines immediately above and below it.

PRINTING COLOR GRAPHICS

Color values shown on the display are built by combining the pixel values of two or three screen bit-maps (two for four-color, three for eight-color). The value of the associated bits designates the color. In an eight color system, 111 specifies white, 101 yellow, etc., SPRINT combines bits also, by looking at the same pixel position in all bit maps and combining the two or three values. (The figure below illustrates this.) At present, it produces either a zero or one from the combination and reduces the color to white or black. Future printers will support color printing and so will SPRINT.

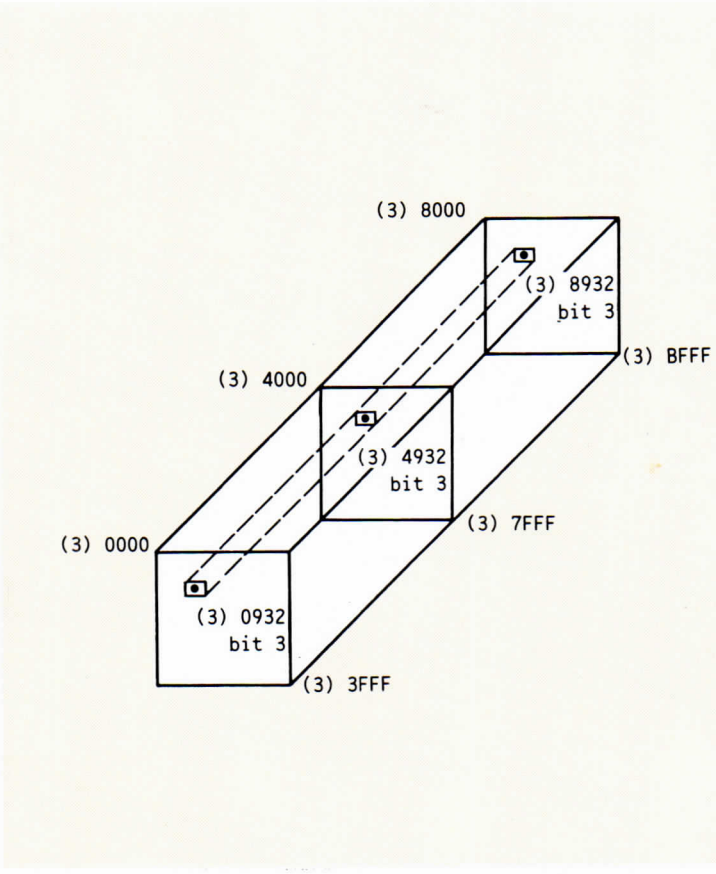


Fig. 16-5 Color Bit Plane Coding

PRINTER SYSTEM CALLS

There is only one printer system call, the bytestream call PutByte (11). The printer FID is 18.

17. GRAPHICS SUB-SYSTEM

ABOUT THIS CHAPTER

This chapter describes the PASCAL graphic routines that make up the graphics sub-system.

CONTENTS

OVERVIEW

This section describes an extensive set of library routines that are used in PASCAL to provide graphics. These routines can also be accessed via assembly language calls. In addition to describing the capabilities of these routines, the contents of this section provide an example of the use of the PCOS graphics system calls which underlie these routines.

The graphics system calls are described in the "Video Driver" section of Part 2, with accompanying background information on cursor use, blinkrate, and other such concepts.

In this section, preliminary discussion covers the basic methods used in applying the graphics program. Each of the functions is listed with a detailed description of its use, register assignments, possible error messages, and examples. Reference information at the end of this section gives development language binding and compare this package to the BASIC graphics package.

DESCRIPTION OF THE M20 GRAPHICS PACKAGE

The M20 Graphics Package is a graphics library for M20 development languages (currently PASCAL). The package preserves the functionality of the graphics developed for M20 BASIC, which means, for example, that a PASCAL user is able to achieve the same graphics results as a BASIC user. Directions contained in this section guide the programmer through the necessary steps of writing and compiling source code, linking the object code compiled from the source code, and finally running the linked, executable module. The executable module uses PCOS system calls to create screen graphics.

The graphics package can be used for both business and scientific applications. It is currently supported under the PASCAL language and can be accessed using assembly language. It can be used with black and white, four-color, and eight-color configurations.

The creation of screen graphics results from manipulation of compiled, executable modules which include routines from the graphics library (GLIB) as well as necessary PCOS system calls.

Even though M20 BASIC functionality is maintained, the syntax of the language 'bindings' reflects that of GKS, an emerging ISO and ANSI graphics standard. The M20 graphics library is not an implementation of GKS, but it does reflect GKS syntax and organization. M20 graphics library differs from GKS on the concept of color organization as well as other, more complex details. Some functional differences are summarized in the following chart:

	FUNCTION	Graphics Package	GKS
1.	Maintains color logic operators	YES	NO
2.	Direct pipeline from world coordinate space to viewing surface	YES	NO
3.	Permits single-element graphic entry as in LINEABS	YES	NO (Has only POLYLINE)

For a complete description of the GKS standard, see the GKS Draft International Standard document (DIS 7942), available from ANSI, 1430 Broadway, New York, N. Y. 10018. A quick overview of this standard is also given in IEEE Computer Graphics and Applications, July 1982: "GKS -- The First Graphics Standard," pp 9-23.

CONFIGURATIONS AND VERSIONS

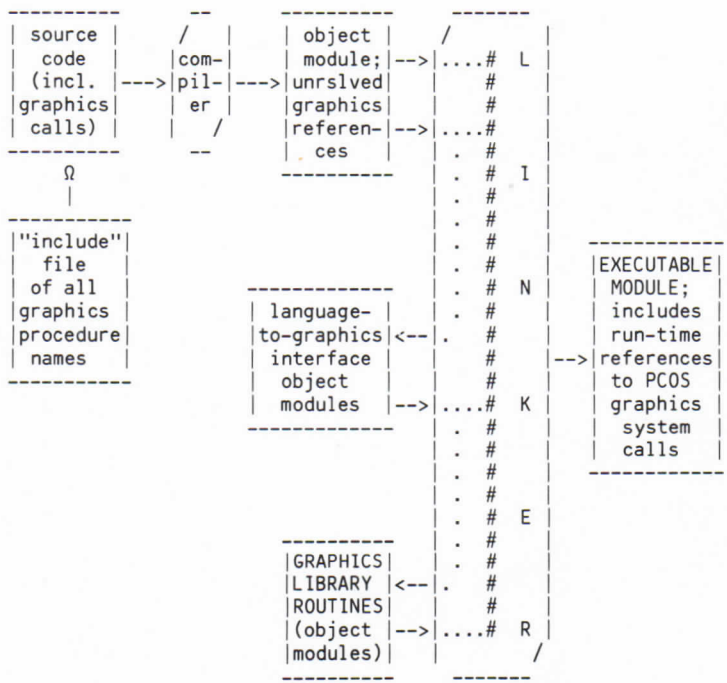
HARDWARE

The Graphics Package works across all M20 configurations: black-and-white, four-color, and eight-color. The minimum confi- guration may limit program size (and prohibit some compilers). Color systems will require their usual expansion-board configura- tions.

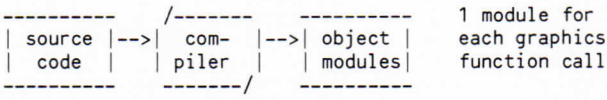
SOFTWARE

The initial level of this Graphics Package is "Version 3.0a," and is designed to run under PCOS 3.0 or later versions of PCOS. There are no special ROM requirements, except that the ROM must support PCOS 3.0 or later.

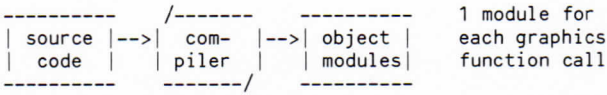
FUNCTIONAL FLOW DIAGRAMS



language-to-graphics interface:



GRAPHICS LIBRARY ROUTINES



IMPLEMENTATION LANGUAGE

The PCOS 3.0 Graphics Library Package is implemented in PLZ/ASM segmented code.

GENERAL APPLICATION INFORMATION

STEPS IN MODULE DEVELOPMENT

An applications programmer creates graphic output using his compiled development language program and the following steps:

1. Write the source code. Currently, the graphics package is supported under PASCAL and can also be accessed by using assembly language. For other development languages, if the language supports inclusion of assembly language subroutine calls, it may be possible to provide graphic routines in that manner. Should the language require it (e.g., PASCAL), insert, in each module which includes graphics calls, a section declaring the graphics routine names as "externals." A file of graphics routine names is provided for each language with graphics support, so that external declaration may be accomplished by using an "include" statement. (See the Reference information on language bindings.)

2. To minimize the size of the run-time requirements, a user may wish to edit this file so that it references only those routines actually used. Then create the appropriate procedures, subroutines, or other algorithm units, incorporating graphics calls that create the images desired.

3. Compile the source code. If the above-mentioned "include" statements are used for declaring the graphics calls as externals, the file containing that list must be included on the same diskette as the source code module.

4. Link the object code compiled from source code with the required external object modules, specifically including **both** a file which contains the source-language-to-graphics interface object modules **and** the GRAPHICS LIBRARY ROUTINES object modules.

5. RUN the linked, executable module under PCOS 3.0.

A graphics programmer has several types of routines available in this package to create graphic output. One class of routines is graphic output, which places geometric coordinate information. A second class is output attributes which modify the graphic output (color, for example). A third class is transformation & control, which largely have to do with the transformation (mathematical) of coordinates from the dimensions of the application program to the physical dimensions of the display device (more about these transformations in a moment). The fourth class is a set of inquiry routines, whereby the application program can consult tables and variables in the Graphics Package for various data such as color, current position, and dimensions of the coordinate space.

ENTERING THE GRAPHICS PROGRAM

Any application program using any Graphics Package routine must enter the "OPEN GRAPHICS" routine as the first Graphics Package routine to be processed. This latter call may be used to clear away previous graphics work and start afresh. The application may recover memory space and clear no longer needed graphics work with the CLOSE GRAPHICS routine.

DEFINING COORDINATES

A fundamental relationship is the one between the dimensions of the object in real space (WORLD COORDINATE SPACE, given typically in real numbers), and the dimensions of the object as it appears on the viewing surface (given in device coordinates* -- see footnote below for differentiation between viewing surface, window, and view area). In the Graphics Package, a defined rectangle in world coordinate space is "pipelined" to a defined rectangle on (all or part of) the view area. World coordinates are mathematically transformed into device coordinates as graphic output flows from the pipeline.

Experienced M20 users will recognize "view area" as equivalent to M20 Basic's "window." The term "window" is not used in this document, since graphics standards documents (such as GKS) use the term in a different sense. In GKS, for example, the model is that of a pipeline in which graphic output flows from within a frame in world coordinate space through the pipeline (a mathematical transformation) to a viewport. The term "window" is used to mean the input to the pipeline, the world-coordinate-space-frame, rather than the output of that pipeline (the viewport).

There is a lot of flexibility in mapping world coordinate space to view area. The view area can take up the full screen or some rectangular part of it. There can be up to 16 view areas, each with a mapping (or transformation) from a world coordinate space to that area. Each view area and its mapping are identified by transformation number.

A view area is formed by dividing an existing view area horizontally or vertically into two parts, and whereas view areas need not be the same size, they are not allowed to overlap. One part retains the transformation number of the original view area, and the other is assigned a new transformation number. The two views also inherit some of the characteristics of the pre-division view area, such as character spacing, line height, color attributes, graphics cursor shapes, and world coordinate space definition. Of these inherited characteristics, the last one requires further commentary.

The world coordinate space definition (see SET WORLD COORDINATE SPACE) is a set of real numbers that define a rectangle; it establishes the scaling basis on which the graphical output is described. However, it is the world coordinate space definition along with the shape of the view area which determines the proportions of the objects within that view (see DIVIDE VIEW AREA).

Note:

Note that world coordinate space defines a Cartesian plane, that is, a rectangular surface on which the scale of each of two axes (x and y) is determined. The plane may contain an origin, a point of crossing of the x- and y-axes at which the coordinates are (0.0,0.0).

The relative magnitude of the width units to the height units has NO bearing on the ultimate view area size or proportions; an x-axis could be defined as extending from 0.0 to 100,000.0 while a y-axis is from +0.023 +0.097, yet the result on the viewing surface could be a square.

POSITION LOCATORS

Current Position Locators and Cursors

The Graphics Package maintains two current positions, one each for text and graphics, and also two cursors, one each for text and graphics, and also two cursors, one each for text and graphics. One cursor or the other (or neither) displays at one time. While the text current position and text cursor are always at the same location (the point at which the next text output will appear), the graphics current position (the point at which the next graphics output will appear) and the graphics cursor are not coincident. This separation makes it possible to use the graphics cursor as a locator in interactive applications: use of the INQ GRAPHCURSOR routine can return graphics cursor coordinates with which the current graphics position can be updated. Note that the current graphics position will be used in many but not all graphics output routines (e.g., POLYLINE will establish its own starting point rather than use that of the current graph- ics position).

Absolute and Relative Coordinates

Some graphics routines use "absolute" coordinates, others use "relative" coordinates. The distinction is that absolute coordinates are distances along the x- or y-axis from the origin point of the Cartesian plane, while relative coordinates are distances along the x- or y-axis from the last coordinate (which last coordinate could be either absolute or relative). Thus, LINEABS draws a line from the current graphics position to an absolute point (a point referenced in terms of the Cartesian plane's origin), whereas LINEREL draws a line from the current graphics position to a relative point (a point referenced in terms of distances from the current graphics position). In both cases, the current graphics position is updated to the end of the new line.

Referencing Positions Outside View Area

The Graphics Package always references the current graphics position. It maintains this position itself instead of referencing the PCOS graphics accumulator. For this reason, it is possible to specify points that are outside the world coordinate space rectangle defined in SET WORLD

COORDINATE SPACE. When the current graphics position ends up outside the view area, the PCOS graphics accumulator is "undefined;" that is, its contents no longer reflect the Graphics Package's current graphics position. No problem arises, because the Graphics Package never relies on the PCOS graphics accumulator to re-establish the current graphics position. Note that in interactive applications, a user may have difficulty understanding what is happening when the current position is outside the view area, especially if required to make a response at that moment.

Color Attributes

Most of the output routines are affected both by the color attributes and by the current logic-operator attribute. The "foreground" color determines the color of the text output, the "background" color determines the color behind the letter, and the "graphics" color determines the color of the graphics output (lines, dots, etc.). Note that the background color is the color of choice when the view area is cleared. These attributes are selectable from the range of colors on any given M20 configuration. For further details, see SET COLOR REPRESENTATION, SELECT GRAPHICS COLOR, and SELECT TEXT COLOR.

Logic Operators

The logic-operator attribute determines which color will appear in the view area, considering the following: the type of graphic routine (text or graphic output); the setting of the foreground, background, or graphics color attribute; and the color of the targeted pixel(s) in the view area.

There are six logic operators; some use only one operand (either a color attribute or the targeted pixel), others use two (both the targeted pixel and a color attribute). The logic operator acts on a pixel-by-pixel basis in determining what the actual final color of each pixel shall be (see SET COLOR LOGIC for more details).

Exchanging Data with the Host Language

This Graphics Package uses three number types in exchanging data with the host language: integer, real, and address pointer. The integers are signed single-word integers in the range of -32768 through +32767. One exception to this format is the cursor arrays, in which a word stores a pair of eight-bit patterns, and thus is a 16-bit unsigned integer; in these arrays, the high-order byte must come first (i.e., have the lower index number). The real numbers are always IEEE single-precision two-word elements; the 32 bits are as follows, from most-significant to least-significant bit: 1 sign bit, eight bits of exponent, and 23 bits of mantissa. The mantissa is "dehydrated", in that it is the 23 low-order bits of a 24-bit mantissa whose most-significant-bit (always a "1") is missing. (The Graphics Package math package "reconstitutes" the mantissa by adding back the high-order bit.) In array storage, the high-order byte must precede and the high order word must precede (have the low index number). The address pointers are Z-8000 segmented addresses; consult

the standard Z-8000 literature for their format.

Errors

Error reporting is handled in two ways. For all routines, an error status is reported in register 5; the value "0" means "no error". For most routines, this status value is transferred to an error status variable maintained by the graphics package; the ERROR INQUIRY routine returns the current value of this variable (that is, the error status of the most recent Graphics Package routine other than the INQ... routines).

The INQ... group of routines handles error reports differently. These routines never touch the error status variable (except ERROR INQUIRY, which retrieves it). Rather, these routines report any error status directly through an "err" parameter. Thus, INQ... routines cannot "generate an error status". The "err" parameter is maintained in all INQ... routines as a matter of form (in keeping with the practice of the GKS standard), even though in this package there are several routines for which the "err" routine can return no other value than "0" ("no error").

FURTHER REFERENCES

Notice that in the routine descriptions, neither the routine headings nor the examples are drawn from any specific language. For the language bindings of these routines, see the reference section.

For those familiar with M20 BASIC graphics, the reference section also has a concordance linking BASIC graphics calls with the routines in this package.

GRAPHICS LIBRARY FUNCTIONS: SPECIFICATIONS

LIST OF ROUTINES

Output Generation Functions

```
LINEABS(x,y)
LINEREL(dx,dy)
POLYLINE(#points,Xarray,Yarray)
MARKERABS(x,y)
MARKERREL(dx,dy)
POLYMARKER(#points,Xarray,Yarray)
TEXTCURSOR(column,row)
GRAPHPOSABS(x,y)
GRAPHPOSREL(dx,dy)
GRAPHCURSORABS(x,y)
GRAPHCURSORREL(dx,dy)
PIXEL ARRAY(Xwdth,Yht,arrayname)
GDP(functionnmbr,numberofpoints,Xarray,Yarray,datarec)
   [Defined GDP's: 1=circle, 2=ellipse]
```

Output Attribute Setting Functions

```
SET LINE CLASS(classnmbr)
SET TEXTLINE(chrwdth,txtlineht)
SELECT CURSOR(selectnmbr)
SET TEXT CURSOR BLINKRATE(rate)
SET GRAPHICS CURSOR BLINKRATE(rate)
SET TEXT CURSOR SHAPE(arrayname)
SET GRAPHICS CURSOR SHAPE(arrayname)
SET COLOR REPRESENTATION(indx#,colr#)
SELECT GRAPHICS COLOR(nmbr)
SELECT TEXT COLOR(FGnmbr,BGnmbr)
SET COLOR LOGIC(operatornmbr)
```

Transformation & Control Functions

```
OPEN GRAPHICS
CLOSE GRAPHICS
SET WORLD COORDINATE SPACE(xform#,x0,y0,x1,y1)
DIVIDE VIEW AREA((div/orient,divpt,xform#)
SELECT VIEW TRANSFORMATION(xform#)
CLOSE VIEW TRANSFORMATION(xform#)
                          [(xform#=0)-->close 2--16]
CLEAR VIEW AREA(xform#,err)
ESCAPE(functionnmber, recordname)
      [Defined ESCAPE: 1=flood]
```

Inquiry Functions

```
INQ VIEW AREA
    (err,bytewdth,scanlineht,chrwdth,txtlineht)
INQ WORLD COORDINATE SPACE(err,x0,y0,x1,y1)
INQ CURRENT TRANSFORMATION NUMBER(err,xform#)
INQ ATTRIBUTES
    (err,GRcolr,FGcolr,BGcolr,logop,lineclass)
INQ TEXTCURSOR(err,column,row,blinkrate)
INQ GRAPHPOS(err,x,y)
INQ GRAPHCURSOR(err,x,y,blinkrate)
INQ PIXEL ARRAY(Xwdth,Yht,err,invalidvals,arrayname)
INQ PIXEL COORDINATES
    (Xworld,Yworld,err,Xpixcoord,Ypixcoord)
INQ PIXEL(x,y,err,pxlcolrnmbr)
ERROR INQUIRY(errorcode)
```

OUTPUT GENERATION FUNCTIONS

The following 13 routines have a geometric influence on the view area -- that is, they determine either directly, or indirectly, the placement and shape of output to the view area.

OUTPUT GENERATION FUNCTIONS

```
LINEABS(x,y)
LINEREL(dx,dy)
POLYLINE(#points,Xarray,Yarray)
MARKERABS(x,y)
MARKERREL(dx,dy)
POLYMARKER(#points,Xarray,Yarray)
TEXTCURSOR(column,row)
GRAPHPOSABS(x,y)
GRAPHPOSREL(dx,dy)
GRAPHCURSORABS(x,y)
GRAPHCURSORREL(dx,dy)
PIXEL ARRAY(Xwdth,Yht,arrayname)
GDP(functionnmbr,numberofpoints,Xarray,Yarray,
    datarec)
   [Defined GDP's: 1=circle, 2=ellipse]
```

Each of these functions is described in detail in the following pages, with register assignments, possible error messages and examples.

LINEABS(x,y)

Draws a line from the current graphic position to the absolute position (x,y). Several default conditions apply: coordinate space, color, logic operator, and line class conditions (see output attribute and transformation function calls).

Register assignment:

```
    Input           rr0     <-- x
                    rr2     <-- y

    Output          r5      <-- error code
```

Inputs are IEEE single-precision real; output is integer.

ERRORS

```
     0      (no error)
    38      "Parameter out of range"
```

EXAMPLE

```
    LineAbs(2.33,-6.8)
    LineAbs(xdisp,ydisp)
        [where xdisp and ydisp  have  been  assigned
        real-number values in a world-coordinate-space]
```

Each of these sample calls will create a line from the current graphics position to an absolute point in the view area. The first example would draw a line to a point that could be in any direction from the starting point, and might extend for any length. The length and direction of the line depend on the currently-defined world coordinate space.

If the coordinates imply a point outside the view area but are within the range of a single-precision floating-point number, a line will appear, drawn in the direction of the specified point but clipped at the edge of the view area. The specified point becomes the current graphics position.

LINEREL(dx,dy)

Draws a line from the current graphic position to a position displaced by the amount dx parallel to the x-axis and dy parallel to the y-axis. Several default conditions apply: co-ordinate space, color, logic operator, and line class conditions (see output attribute and transformation function calls).

Register assignment:

```
    Input           rr0     <-- dx
                    rr2     <-- dy

    Output          r5      <-- error code
```

Inputs are IEEE single-precision real; output is integer.

ERRORS

0	(no error)
38	"Parameter out of range"

EXAMPLE

```
LineRel(2.33,-6.8)
LineRel(xdisp,ydisp)
       [where xdisp and ydisp have been
        assigned real-number values in a
        world-coordinate-space]
```

Each of these sample calls will create a line from the current graphics position to a point relative to that current position. The first example would draw a line to a point to the right and below the starting point. The length of the line depends on the currently-defined world coordinate space.

If the coordinates imply a point outside the view area, yet are within the range of a single-precision floating-point number, a line will appear drawn in the direction of the specified point but clipped at the edge of the view area. The specified point becomes the current graphics position.

POLYLINE(#points, Xarray, Yarray)

Draws lines connecting the points specified by the two arrays (parameters two and three). "#points" is an integer specifying the number of points. The points are absolute locations in world coordinate space. Thus the two arrays are the same size and contain (single precision) real numbers. Correspondingly indexed values in the arrays (e.g., Xarray[32], Yarray[32]) constitute a number pair that specifies a point in world coordinate space. Several default conditions apply: coordinate space, color, and logic operator (see output attribute and transformation function calls).

Register assignment:

```
    Input          rr6     <-- Xarray pointer
                   rr2     <-- Yarray pointer
                   r4      <-- number of points, =>2

    Output         r5      <-- error code
```

Pointers are segmented addresses; other values are integer.

ERRORS

0	(no error)
9	"Subscript out of range"
38	"Parameter out of range"
76	"Error in parameter"

ARRAY STRUCTURE

The application program must declare and allocate the two coordinate arrays. Each array contains IEEE single-precision numbers; the high-order word must precede the low-order word. The size of each array must be at least large enough to store as many double-word numbers as there are points.

EXAMPLE

```
["pts" is type integer and equals 10]
["Xvals" and "Yvals" each are one-dimensional arrays
of size 10; each contains 10 single-precision real
numbers representing distances along the x- or y-axis
from the original Cartesian plane defined in world
coordinates.]

Polyline(pts,Xvals,Yvals)
Polyline(10,harry,varry)
```

Each of these sample calls will create nine lines contiguously connecting ten points. The figure will not be "closed" unless the first and last points specified by the arrays happen to coincide. The application program calculates coordinates and deposits them in two arrays, then calls "Polyline" once to draw the line series.

If the coordinates imply points outside the view area but are within the range of single-precision floating-point numbers, the figure will not be distorted but it will be clipped at the edge of the view area. When the last point is outside the view area, the current graphics position is set at that point.

If "pts" were to be less than the value "2", error #38 would be generated and no lines would be drawn.

MARKERABS(x,y)

Places a visible point at the absolute (world coordinate) position specified by parameters x and y. Several default conditions apply: coordinate space, color, and logic operator (see output attribute and transformation function calls).

Register assignment:

```
Input          rr0     <-- x
               rr2     <-- y

Output         r5      <-- error code
```

Inputs are IEEE single-precision real; output is integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
MarkerAbs(2.33,-6.8)
MarkerAbs(xdisp,ydisp)
    [where  xdisp  and  ydisp  have  been  assigned
    real-number values in a world-coordinate-space]
```

Each of these sample calls will display a point at the absolute locations specified by their parameters. If the coordinates imply a point outside the view area but are within the range of a single-precision floating-point number, no point will appear. However, the specified point becomes the current graphics position.

MARKERREL(dx,dy)

Places a visible point displaced in world coordinate space by the amounts dx parallel to the x-axis and dy parallel to the y-axis from the Graphics Package's current position, as specified by parameters dx and dy. Several default conditions apply: coordinate space, color, and logic operator (see output attribute and transformation function calls).

Register assignment:

```
Input           rr0     <-- dx
                rr2     <-- dy

Output          r5      <-- error code
```

Inputs are IEEE single-precision real; output is integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
MarkerRel(2.33,-6.8)
MarkerRel(xdisp,ydisp)
    [where  xdisp and  ydisp  have  been  assigned
    real-number values in a world-coordinate-space]
```

Each of these sample calls will display a point displaced relative to the current graphics position. The first example would place a point to the right and below the current graphics position. The length of the displacement depends on the currently-defined world coordinate space. If the coordinates imply a point outside the view area but are within the range of a single-precision floating-point number, no point will appear. However, the specified point becomes the current graphics position.

POLYMARKER(#points,Xarray,Yarray)

Places the visible points specified by the two arrays (parameters two and three). "#points" is an integer specifying the number of points. The points are absolute locations in world coordinate space. Thus the two arrays are the same size and contain (single precision) real numbers. Correspondingly indexed values in the arrays (e.g., Xarray[32], Yarray[32]) specify a point in world coordinate space. Several default conditions apply: co-ordinate space, color, and logic operator (see output attribute and transformation function calls).

Register assignment:

```
    Input          rr6     <-- Xarray pointer
                   rr2     <-- Yarray pointer
                   r4      <-- number of points, =>1

    Output         r5      <-- error code
```

Pointers are segmented addresses; other values are integer.

ERRORS

```
     0      (no error)
     9      "Subscript out of range"
    38      "Parameter out of range"
    76      "Error in parameter"
```

ARRAY STRUCTURE

The application program must declare and allocate the two coordinate arrays. Each array contains IEEE single-precision numbers; the high-order word must precede the low-order word. The size of each array must be at least large enough to store as many double-word numbers as there are points.

EXAMPLE

```
    ["pts" is type integer and equals 10]
    ["Xvals" and "Yvals" each are one-dimensional arrays
    of size 10;  each contains 10  single-precision real
    numbers representing distances along the x- or y-axis
    from the origin of a Cartesian plane defined in world
    coordinates.]

    Polymarker(pts,Xvals,Yvals)
    Polymarker(10,harry,varry)
```

Each of these sample calls will create ten visible points. The application program calculates coordinates and deposits them in two arrays, then calls "Polymarker" once to place the points.

Coordinates that imply points outside the view area but within the range of single-precision floating-point numbers will not appear, nor will an error message be generated. However, the current graphics position will

track these non-visible points. Should the last point be outside the view area, it nevertheless becomes the current graphics position.

If "pts" were to be less than the value "1", error #38 would be generated and no points would be drawn.

TEXTCURSOR(column,row)

Moves the text cursor and thereby determines the next screen position at which text will appear. Note that the text cursor will appear only if the latest cursor selection has set the text cursor to be displayed (see SELECT CURSOR).

Maximum ranges for a full-screen view area are the default text parameters currently active: 64 columns by 16 rows or 80 columns by 25 rows. If the current view area is smaller than full-screen, then the maximum text parameters are commensurately smaller.

Register assignment:

```
Input          r8      <-- text column (1..64 or 80)
               r9      <-- text row  (1..16 or 25)

Output         r5      <-- error code
```

All values are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
Textcursor(33,17)
Textcursor(X,Y)
```

The first of these calls will cause the next text output to appear in column 33, row 17, of the current view area. If the current view area is the full screeen and character spacing is 64 by 16, then text will start one character below and right of the mid-screen point. The second example assumes that X and Y are integers.

If the coordinates imply a point outside the current view area, an error message is generated in r5, and the current text cursor and position are unchanged.

GRAPHPOSABS(x,y)

Redefines the current graphics position, that is, the current position for subsequent graphics output (for text, see (TEXTCURSOR). Coordinates x and y define an absolute location in world coordinate space. Thus, any subsequent graphics output that uses the current graphics position as a starting point will use this point. Note that this position is not automatically associated with the position of the graphics cursor; the two positions coincide only when both are explicitly assigned the same coordinates (see GRAPHCURSORABS, GRAPHCURSORREL). The separation of current graphics position and graphics cursor permits the application program to use the graphics cursor as a locator in interactive applications.

Register assignment:

```
Input          rr0     <-- x
               rr2     <-- y

Output         r5      <-- error code
```

Inputs are IEEE single-precision real; output is integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
GraphPosAbs(2.33,-6.8)
GraphPosAbs(xloc,yloc)
    [where xloc and yloc have been assigned
    real-number values in a world-coordinate-space]
```

Each of these sample calls will redefine the location of the current graphics position to an absolute point in the view area.

If the coordinates specify a point outside the view area, that point nevertheless becomes the current graphics position.

GRAPHPOSREL(dx,dy)

Redefines the current graphics position, that is, the current position for subsequent graphics output (for text, see (TEXTCURSOR). The new position is displaced from the old position by adding the factors dx and dy (which are world coordinate space values) to the corresponding coordinates of the old current graphics position. Thus, any subsequent graphics output that uses the current graphics position as a starting point will use this new point. Note that this position is not automatically associated with the position of the graphics cursor; the two positions coincide only when both are explicitly assigned the same coordinates (see GRAPHCURSORABS, GRAPHCURSORREL).

The separation of current graphics position and graphics cursor permits the application program to use the graphics cursor as a locator in interactive applications.

Register assignment:

Input	rr0	<--	dx
	rr2	<--	dy
Output	r5	<--	error code

Inputs are IEEE single-precision real; output is integer.

ERRORS

0	(no error)
38	"Parameter out of range"

EXAMPLE

```
GraphPosRel(2.33,-6.8)
GraphPosRel(dxloc,dyloc)
    [where dxloc and dyloc have been assigned
    real-number values in a world-coordinate-space]
```

Each of these sample calls will redefine the location of the current graphics position to a new point in the view area, displaced from the old point. In the first example, the new graphics position will be to the right and below the old one.

If the coordinates result in a point outside the view area, that point nevertheless becomes the current graphics position.

GRAPHCURSORABS(x,y)

Moves the graphics cursor (but not the current graphics position, nor the text position) to a new absolute position (in world coordinates). Note that the graphics cursor will appear only if the latest cursor selection has set the graphics cursor to be displayed (see SELECT CURSOR). If the coordinates define a position outside of the view area, an error code is sent to the error status variable. The separation of current graphics position and graphics cursor permits the application program to use the graphics cursor as a locator in interactive applications.

Register assignment:

Input	rr0	<--	x
	rr2	<--	y
Output	r5	<--	error code

Inputs are IEEE single-precision real; output is integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
GraphcursorAbs(0.39,3.17)
GraphcursorAbs(X,Y)
```

The first of these calls would cause the graphics cursor, when it is turned on, to appear at the absolute location specified by the arguments (in world coordinate space). The second call, in which X and Y are real numbers, would behave similarly.

GRAPHCURSORREL(dx,dy)

Moves the graphics cursor (but not the current graphics position, nor the text position) to a new position (in world coordinates). This new position is displaced from the old position by the factors dx and dy, added to the coordinates of the old position. Note that the graphics cursor will appear only if the latest cursor selection has set the graphics cursor to be displayed (see SELECT CURSOR). If the coordinates result in a position outside of the view area, an error code is sent to the error status variable.

Register assignment:

```
Input           rr0     <-- dx
                rr2     <-- dy

Output          r5      <-- error code
```

Inputs are IEEE single-precision real; output is integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
GraphcursorRel(-0.39,3.17)
GraphcursorRel(X,Y)
```

The first of these sample calls would cause the graphics cursor, when it is turned on, to appear at a new location shifted leftward and up from the old location. The second call, in which X and Y are real numbers, would likewise shift the graphics cursor in some direction.

PIXEL ARRAY(Xwdth,Yht,arrayname)

Transfers a rectangular screen image to the screen. The M20 screen displays what is stored in one or more bit-planes (where each bit on a plane corresponds to a screen pixel). A rectangle from any part (or all) of a view area can thus be stored elsewhere in memory (see INQ PIXEL ARRAY), then re-displayed at any point in any view area by using PIXEL ARRAY. The rectangle size is "Xwd" wide by "Yht" high (world coordinates), and is placed with the rectangle's upper-left corner at the current graphics position.

The rectangle is retrieved from the one-dimensional array identified by the "arrayname" parameter. The x and y parameters need not correspond to the full size implied by the array. A "too-small" dimension clips the right or bottom edge; a "too-large" dimension yields that dimension's maximum (and no more). If the current graphics position is "too close" to the right and/or bottom edge, the rectangle will transfer anyway but be clipped at the screen edge. The current logic operator influences the color output to the screen on the usual pixel-by-pixel basis (see SET COLOR LOGIC).

Register assignment:

```
Input          rr0     <-- Xwidth
               rr2     <-- Yheight
               rr10    <-- array pointer

Output
Output         r5      <-- error code
```

Size inputs are IEEE single-precision real; pointer is Z8000 segmented address; xoutput is integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
[Assume a properly dimensioned array, PurtyPix,
with a previously-saved screen block -- not
necessarily the full size of the screen]
[Assume also a World Coordinate Space, 150.0 x 100.0]

GraphPosAbs(75.0,50.0)
PixelArray(30.,20.,PurtyPix)
```

The rectangular display saved in PurtyPix will appear on the screen, with its upper left corner at the middle of the screen. If the rectangle in PurtyPix is relatively large compared to PixelArray's arguments (30.,20.), then only part of the stored picture will appear; anything to the right of world coordinate 105.0 (75.0 + 30.0) will not appear, nor will anything below 30.0 (50.0 - 20.0).

If the rectangle in PurtyPix is relatively small compared to PixelArray's arguments, then the full picture will appear and not extend to the rectangle's implied right and bottom borders. And, of course, if PurtyPix differs in shape compared to PixelArray's parameters, then one but not the other of the right or bottom edges may be clipped.

GDP(functionnmbr,numberofpoints,Xarray,Yarray,datarec)

Generalized Drawing Primitive, for creating specialized geometric output. This Graphics Package has two such GDP's: a circle (functionnmbr=1) and an ellipse (functionnmber=2). Separate discussions of each follow.

CIRCLE

```
GDP(functionnmbr,numberofpoints,Xarray,Yarray,datarec)
    [functionnmbr-->1 (number, constant, or variable)]
    [numberofpoints-->2 (number, constant, or variable)]
```

Draws a circle according to the world coordinates in arrays Xarray and Yarray. The circle is centered at the point [Xarray(1),Yarray(1)] and has a radius determined by the distance from the circle center to absolute location [Xarray(2),Yarray(2)]. The second parameter (the value 2) indicates that the array parameters will specify two geometric points for this primitive. The dummy parameter "datarec" is required by the GDP call but not used for circle drawing. The shape is always a circle regardless of the coordinate space definition. Default conditions that apply are color and logic operator (see output attribute setting calls) and, with reference to placement of the specified center and perimeter points, the coordinate space. The current graphics position is unaffected.

Register assignment:

```
    Input       rr6     <-- Xarray pointer
                rr2     <-- Yarray pointer
                r4      <-- 1:  1 = "CIRCLE"

    Output      r5      <-- error code
```

Pointers are Z8000 segmented addresses; other values are integer.

ERRORS

0	(no error)
38	"Parameter out of range"
76	"Error in Parameter

EXAMPLE

[Assume arrays Xdata and Ydata,
with the following contents:]

	Xdata	Ydata
(1)	137.60	89.36
(2)	129.6	93.36

```
GDP(1,2,Xdata,Ydata,nuldata)
```

Generates a circle centered at absolute (world coordinate) position (137.6,89.36), with a radius of approximately 8.944.

If the coordinates generate a circle larger than the viewing surface can accommodate, the portions of the circle that lie outside the viewing surface are clipped. It is possible to specify a circle no part of which is visible.

ELLIPSE

```
GDP(functionnmbr,numberofpoints,Xarray,Yarray,datarec)
    [functionnmbr-->2 (number, constant, or variable)]
    [numberofpoints-->3 (number, constant, or variable)]
```

Draws an ellipse parallel to the x or y-axis. The coordinates are given in the arrays Xarray and Yarray. The second parameter (the value 3) indicates that there are three points, and thus three values in each coordinate array. The center of the ellipse is given by the point [Xarray(1),Yarray(1)]. The major and minor axis crossings (in either order) are given by points [Xarray(2),Yarray(2)] and [Xarray(3),Yarray(3). The dummy parameter "datarec" is required by the GDP call but not used for ellipse drawing. The exact shape may vary depending on the coordinate space definition. Default conditions that apply are color and logic operator (see output attribute setting calls) and, with reference to placement of the specified center and perimeter points, the coordinate space. The current graphics position is unaffected.

Register assignment:

```
Input       rr6     <-- Xarray pointer
            rr2     <-- Yarray pointer
            r4      <-- 2:  2 = "ELLIPSE"

Output      r5      <-- error code
```

Pointers are segmented addresses; other values are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
76      "Error in parameter"
```

EXAMPLE

```
[Assume FuncNo = 2]
[Assume arrays Xdata and Ydata,
 with the  following  contents:]

          Xdata       Ydata
(1)       200          150
(2)       190          150
(3)       200          175

GDP(FuncNo,3,Xdata,Ydata,nuldata)
```

Generates an ellipse centered at absolute (world coordinate) position (200,150), with a major axis parallel to the y-axis and of length 50, and a minor axis of width 20.

If the coordinates generate an ellipse larger than the viewing surface can accommodate, the non-visible portions are clipped at the viewing surface edge. It is possible to specify an ellipse no part of which is visible.

OUTPUT ATTRIBUTE SETTING FUNCTIONS

The following eleven routines influence various aspects of the appearance of the geometric output routines. Specifically, they set values in various tables that the geometric routines use when they create output. Each function is detailed in the following pages.

Output Attributes

```
SET LINE CLASS(classnmbr)
SET TEXTLINE(chrwdth,txtlineht)
SELECT CURSOR(selectnmbr)
SET TEXT CURSOR BLINKRATE(rate)
SET GRAPHICS CURSOR BLINKRATE(rate)
SET TEXT CURSOR SHAPE(arrayname)
SET GRAPHICS CURSOR SHAPE(arrayname)
SET COLOR REPRESENTATION(indx#,colr#)
SELECT GRAPHICS COLOR(nmbr)
SELECT TEXT COLOR(FGnmbr,BGnmbr)
SET COLOR LOGIC(operatornmbr)
```

SET LINE CLASS(classnmbr)

For the output functions LINEABS and LINEREL, determines whether the graphic output will be a line, a hollow rectangle, or a solid rectangle. In the latter two cases, the coordinates constitute opposite corners of the rectangle. The current graphics color is used for both lines and filling (see SELECT GRAPHICS COLOR).

Register assignment:

Input	r3	<-- 0..2:	0 = "LINE"
			1 = "BOX"
			2 = "BOXFILL"
Output	r5	<-- error code	

All values are integer.

ERRORS

0	(no error)
38	"Parameter out of range"

EXAMPLE

```
[Assume the integer variable ClsN = 2]

SetLineClass(0)
LineAbs(x2,y2)
SetLineClass(ClsN)
LineRel(dx,dy)
```

The screen will display a line between absolute points [x1,y1] and [x2,y2], and a solid rectangle having a diagonal from [x2,y2] to the point [x2+dx, y2+dy]. N.B.: The next LineAbs or LineRel function would also generate a rectangle, unless a "SetLineClass" call intervened.

SET TEXTLINE(chrwdth,txtlineht)

Sets the character width (in pixels) to 6 or 8 (no other legal values), and the text-line height (in scanlines) to any size from 10 to 16 (no other legal values). This definition holds for the current view area and all subdivisions of it (until a subsequent SET TEXTLINE call). Note that this setting influences the width of subsequently-defined view areas.

Register assignment:

Input	r10	<-- lineheight (10..16)
	r12	<-- character spacing (6 or 8)
Output	r5	<-- error code

All values are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
76      "Error in parameter"
```

EXAMPLE

```
SetTextline(6,12)
```

All subsequent text will have 6 pixels per column, 12 scanlines per text line; a full-screen view area would have 80 columns, 21 textlines. Note that the individual character size does not change, but rather the space around each character grows or shrinks.

SELECT CURSOR(selectnmbr)

Chooses which, if either, cursor is to be displayed.

0: OFF--neither cursor is displayed.

1: The GRAPHICS cursor is displayed. The cursor (default shape: a small rectangle) appears with its upper left corner at the current cursor coordinates. (N.B.: The graphic cursor and the current graphic position are

2: The TEXT cursor is displayed. The cursor (default shape: a rectangle 7 x 11 pixels) appears at the next position that text will be entered. not the same.)

Register assignment:

```
Input           r8      <-- 0..2:   0 = "OFF"
                                    1 = "GRAPHIC CURSOR"
                                    2 = "TEXT CURSOR"

Output          r5      <-- error code
```

All values are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
[Assume that the integer variable SelN = 2]

SelectCursor(0)
SelectCursor(1)
SelectCursor(SelN)
```

After the first example, no cursor will display. After the second example, the text cursor will display; subsequent text entry will begin at this point. After the third example, the graphics cursor will display. However, subsequent graphics output will not start from this point unless the current graphics position has been updated to this same location.

Note that the text and graphics cursors need not, and usually do not, occupy the same location. Note also that the two cursors cannot be displayed simultaneously.

SET TEXT CURSOR BLINKRATE(rate)

Sets the blinkrate for the text cursor, from 0 (no blink) to 20 per second, truncated to the nearest 50-millisecond increment. A zero value leaves the cursor on continuously. The blink rate is state-changes; a rate of 20 is 10 blinks per second. Note that this function does not affect which cursor, if any, is to be displayed currently; see SELECT CURSOR.

Register assignment:

```
Input          r8      <-- blinkrate (0..20)

Output         r5      <-- error code
```

All values are integer.

ERRORS

0	(no error)
38	"Parameter out of range"
76	"Error in parameter"

EXAMPLE

```
[Assume BkR=5]

SetTxCsrBlinkrate(Bkr)
```

The text cursor, when it is displayed, will change from "on" to "off" or from "off" to "on" 5 times per second.

SET GRAPHICS CURSOR BLINKRATE(rate)

Sets the blinkrate for the graphics cursor, from 0 (no blink) to 20 per second, truncated to the nearest 50-millisecond increment. A zero value leaves the cursor on continuously. The blink rate is state-changes; a rate of 20 is 10 blinks per second. Note that this function does not affect which cursor, if any, is to be displayed currently: see SELECT CURSOR.

Register assignment:

```
    Input           r8      <-- blinkrate (0..20)

    Output          r5      <-- error code
```

All values are integer.

ERRORS

```
     0      (no error)
    38      "Parameter out of range"
    76      "Error in parameter"
```

EXAMPLE

```
    [Assume BkR=5)

    SetGrCsrBlinkrate(BkR)
```

The graphics cursor, when it is displayed, will blink 2-1/2 times per second.

SET TEXT CURSOR SHAPE(arrayname)

Defines the text cursor shape according to the contents of the shape-array. Note that this function does not affect which cursor, if any, is to be displayed currently; see SELECT CURSOR.

The shape-array is byte-oriented; each of the 12 bytes represents the bit-pattern of a scanline, the first byte being the highest of 12 scan-lines. Warning: in the text cursor, if the most- significant-bit of each byte is used, the leftmost column of pixels will touch the previous char-acter.

Register assignment:

```
    Input           rr8     <-- shape-array pointer

    Output          r5      <-- error code
```

Pointer is a segmented address; error code is integer.

ERRORS

```
     0      (no error)
    38      "Parameter out of range"
    76      "Error in parameter"
```

ARRAY FORMAT

The array consists of 6 one-word elements, each containing a 16-bit unsigned integer. Each integer is, in fact, a pair of bytes; each byte is a bit-map of a scanline of the cursor. The first element's high-order

byte is the top scanline of the new cursor; the sixth element's low-order byte is the last scanline of the new cursor.

EXAMPLE

```
[assume a 6-word array "Arrow" holds 12 bytes of
 of pixel information]

STxCsrShape(Arrow)
```

The text cursor shape is redefined in accordance with the bit-by-bit specification in the array "Arrow". If that array holds the bit-patterns shown in the following table, the text-cursor will be shaped as an up-arrow.

element (word)	element content (binary bit-map)
1	00001000
	00011100
2	00111110
	01111111
3	00011100
	00011100
4	00011100
	00011100
5	00011100
	00011100
6	00011100
	00011100

(Note that each word contains a PAIR of 1-byte bit patterns, and that the higher byte is the high-order byte of the word.)

SET GRAPHICS CURSOR SHAPE(arrayname)

Defines the graphics cursor shape according to the contents of the shape-array. Note that this function does not affect which cursor, if any, is to be displayed currently; see SELECT CURSOR.

The shape-array is byte-oriented; each of the 12 bytes represents the bit-pattern of a scanline, the first byte being the highest of 12 scan-lines.

Register assignment:

```
Input          rr8     <-- shape-array pointer

Output         r5      <-- error code
```

Pointer is a segmented address; error code is integer.

ERRORS

0	(no error)
38	"Parameter out of range"
76	"Error in parameter"

ARRAY FORMAT

The array consists of 6 one-word elements, each containing a 16-bit unsigned integer. Each integer is, in fact, a pair of bytes; each byte is a bit-map of a scanline of the cursor. The first element's high-order byte is the top scanline of the new cursor; the sixth element's low-order byte is the last scanline of the new cursor.

EXAMPLE

```
[assume a 6-word array "Box" holds 12 bytes of
 pixel information]

STxCsrShape(Box)
```

The text cursor shape is redefined in accordance with the bit-by-bit specification in the array Box. If that array holds the bit-patterns shown in th following table, the graphics-cursor will be shaped as an upright rectangle.

element (word)	element content (binary bit-map)
1	11111111 10000001
2	10000001 10000001
3	10000001 10000001
4	10000001 10000001
5	10000001 10000001
6	10000001 11111111

(Note that each word contains a PAIR of 1-byte bit patterns, and that the higher byte is the high-order byte of the word.)

SET COLOR REPRESENTATION(indx#,colr#)

On four-color systems, sets one of the four color indices (indx#) to be one of the eight M20 colors (colr#). (On monochrome and 8-color systems, this command generates no effect and no error message.) Legal values for indx# are integers 0..3. Legal values for colr# are integers 0..7, with the following meanings: 0=black, 1=green, 2=blue, 3=cyan, 4=red, 5=yellow, 6=magenta, 7=white.

Register assignment:

```
Input           r1      <-- 0..3
                r2      <-- 0..7

Output          r5      <-- error code
```

All values are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
76      "Error in parameter
```

EXAMPLE

```
[Assume Indx=2,Clr=6]

SetColorRep(3,1)
SetColorRep(Indx,Clr)
```

In the first example, all screen output using color index #3 (lines, circles, points, filled areas, text) becomes green (both previously- and subsequently- created output). In the second example, all screen output using color index #2 becomes magenta.

SELECT GRAPHICS COLOR(nmbr)

Selects a color to be the current color attribute for graphics output (not text output; see SELECT TEXT COLOR). The color of subsequent graphic output is determined by the argument "nmbr". There are different effects on monochrome, four-color and eight-color systems. On monochrome systems, "nmbr=0" sets black as the graphics color attribute: any integer in the range 1..7 sets white as the color attribute. On four-color systems, "nmbr" selects the color attribute indirectly by acting as an index into a table of four colors preselected from the eight possible colors (see SET COLOR REPRESENTATION); integers 0..3 select colors directly, but bits 0 & 2 of the binary representation of 4..7 are logically OR'd before selection. On eight-color systems, "nmbr" is a color number and selects that color directly as the graphics color attribute.

Register assignment:

```
Input           r8      <-- color code or index (0..7)

Output          r5      <-- error code
```

All values are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
76      "Error in parameter
```

EXAMPLE

```
[Assume that SetColorRep has associated color #5
 with index #2 in the color table.]

SelectGraphClr(2)
```

Each configuration will interpret this example differently, according to how it uses the color code. For information on the color code, see the discussion of color in the "Video Display" section.

A monochrome system will ignore the SetColorRep statement and, since the number 2 falls within the range 1..7 inclusive, set the graphics color attribute to be WHITE.

A four-color system will use the argument to index the color table and thereby set the graphics color attribute to be color #5, YELLOW.

An eight-color system will ignore the SetColorRep statement and use the number directly to set the graphics color attribute to be color #2, BLUE. Had the argument been "6" rather than "2": the monochrome system would still have set WHITE, the eight-color system would have selected MAGENTA, and the four-color system would have selected the color for index #3.

SELECT TEXT COLOR(FGnmbr,BGnmbr)

Selects colors used in character output (not graphics output; see SELECT GRAPHICS COLOR). Characters appear in the foreground color (the first parameter); the backdrop for the characters is the background color (the second parameter). There are different effects on monochrome, four-color and eight-color systems. On monochrome systems, the number 0 selects black; any integer in the range 1..7 selects white. On four-color systems, each parameter selects a color indirectly by acting as an index into a table of four colors preselected from the eight possible colors (see SET COLOR REPRESENTATION); while integers 0..3 will select colors as one might expect, integers 4..7 select colors in a not-easily-predictable manner. On eight-color systems, each parameter is a color number and selects that color directly as the foreground or background color.

Register assignment:

```
Input           r8      <-- foreground color
                                code or index (0..7)
                r9      <-- background color
                                code or index (0..7)

Output          r5      <-- error code
```

All values are integer.

ERRORS

0	(no error)
38	"Parameter out of range"
76	"Error in parameter

EXAMPLE

```
[Assume that SetColorRep statements have associated
color #5 with index #2, and color #3 with index #0
in the color table.)

SelectTextClr(2,0)
```

Each system configuration will interpret this example differently, according to how it uses the color code. For information on the color code, see the discussion of color in the "Video Display" section.

A monochrome system will ignore the SetColorRep statement, notice that the first parameter, the number 2, falls within the range 1..7 inclusive, and thus set the foreground color to be WHITE; and since the second parameter is 0, it will set the background to be BLACK.

A four-color system will use the arguments to index the color table and thereby set the foreground color to be color #5, YELLOW, and the background to be color #3, CYAN. An eight-color system will ignore the SetColorRep statements and use the numbers directly to set the foreground color to be color #2, BLUE, and the background to be color #0, BLACK.

Had the first argument been "6" rather than "2": for the foreground color, the monochrome system would still have been set WHITE, the eight-color system would have selected MAGENTA, and the four-color system would have selected the color for index #3.

SET COLOR LOGIC(operatornmbr)

For all subsequent screen output except text, defines a logic operator that influences the output color on a pixel-by-pixel basis. Each operand is the color number (or, for four-color systems, one of the four color index numbers) for a pixel--a new-output pixel, a target-location pixel, or one of each. As the screen is updated with new input, the logic operation is applied one pixel at a time. The logic operations deal with the numbers 0..7 as three-bit binary quantities, so that, for example, (3 OR 4)-->7, and (3 AND 4)-->0. There are six logic operators, with the following effects:

```
NewPix# =   new pixel's color number or color index
ScrPix# =   screen pixel's color number or color index
            (at targeted location)

 number                    new number at pixel location
  code      operator       [=result of a logic operation]


    0       PSET           Pixel SET [number = NewPix#]

    1       XOR            result of (NewPix# XOR ScrPix#)

    2       AND            result of (NewPix# AND ScrPix#)

    3       NOT            complement of ScrPix#

    4       OR             result of (NewPix# OR ScrPix#)

    5       PRESET         Pixel RESET [number = background
                                        color number or index]
```

The specific results vary depending on system configuration. Monochrome systems transform the numbers 2..7 to 1; thus the only operands are 0 and 1, and are the colors black and white respectively, with corresponding results. Eight-color systems make no transformation at all, dealing with the numbers directly as colors, with corresponding results (1:green OR 2:blue --> 3:cyan). Four-color systems treat the numbers not directly as colors but as indices into the four-color palette table; predicting the final color result is possible but will take a little concentration.

Register assignment:

```
    Input           r10     <-- 0..5:   0 = "PSET"
                                        1 = "XOR"
                                        2 = "AND"
                                        3 = "NOT"
                                        4 = "OR"
                                        5 = "PRESET"

    Output          r5      <-- error code
```

All numbers are integer.

ERRORS

```
     0      (no error)
    38      "Parameter out of range"
```

EXAMPLE

```
    [Assume integer variable LogOp = 1]

    SetColorLogic(1)
    SetColorLogic(LogOp)
```

These two examples are equivalent. Subsequent graphics output will be compared for color on a pixel-by-pixel basis with the target screen location. The current-color-attribute number and the targeted pixel's color number (or color index number) will be logically XOR'd, and the result placed in the target screen location (to be exact, in the target location in the screen bit-plane or bit-planes). In monochrome and eight-color systems, the operands are color numbers; in four-color systems, the operands are color-table indices.

TRANSFORMATION AND CONTROL FUNCTIONS

These routines set values that have an effect across much or all of a view area. Largely, they have to do with the mathematical transformation of world coordinate values to device coordinate values (screen pixel locations) in the view area. The following is a list of these functions.

```
OPEN GRAPHICS
CLOSE GRAPHICS
SET WORLD COORDINATE SPACE(xform#,x0,y0,x1,y1)
DIVIDE VIEW AREA((div/orient,divpt,xform#)
SELECT VIEW TRANSFORMATION(xform#)
CLOSE VIEW TRANSFORMATION(xform#) [(xform#=0)-->close 2--16]
CLEAR VIEW AREA(xform#,err)
ESCAPE(functionnmber, recordname) [Defined ESCAPE: 1=flood]
```

OPEN GRAPHICS

Sets up the M20 for creating graphics, using the routines in this Graphics Package.

THIS ROUTINE MUST BE REFERENCED BEFORE ANY OTHER GRAPHICS PACKAGE ROUTINE IS REFERENCED. It may be used to "re-initialize" the graphics environment, in which case the effects of all preceding graphics routines are cleared away and the application program is prepared to handle subsequent graphics routines and their output as if starting afresh.

The starting condition for graphics is a single view area (labeled as transformation number 1), with the default world coordinates set to the normal display device coordinates (512 X 256), black as the background color amd white (for black-&-white systems) or green (for color systems) as the foreground color, with no cursor displayed.

Register assignment:

Input	none
Output	none

ERRORS

none

EXAMPLE

```
OpenGraphics
```

This MUST be the first Graphics Package call; it MAY be a later one as well. In either case, the Graphics Package tables and view area definitions are established in their default condition and graphics routines may be used.

CLOSE GRAPHICS

Clears the application program environment of the special setup for the Graphics Package, such as view area heap space and Graphics Package tables. All view areas are closed except view area #1 which reverts to the original default parameters (full screen, defined with 512 x 256 coordinates, startup foreground and background colors, no cursor displayed, blank screen). Graphics Package routines should not appear in the application program following this routine until "OPEN GRAPHICS" has been called again.

Register assignment:

Input	none
Output	none

ERRORS

none

EXAMPLE

```
CloseGraphics
```

If this routine appears, it must be the last Graphics Package call. The Graphics Package tables and view area definitions are cleared, and the default initial conditions are reinstated.

SET WORLD COORDINATE SPACE(xform#,x0,y0,x1,y1)

Redefines the world coordinate space (sometimes called the "problem space") for an existing view area of the screen. "xform#", the first parameter, identifies the transformation by number and must have been previously defined by a DIVIDE VIEW AREA call. All subsequent graphic coordinates in this viewspace (i.e., the transformation and viewspace identified by "xform#) will be scaled to the view area by a transformation routine using the coordinates (parameters 2 through 5) that define a diagonal of the entire rectangular area. The coordinates are single-precision real numbers.

Register assignment:

```
Input           rr0     <-- x0
                rr2     <-- y0
                r4      <-- xform# (1..16)
                rr6     <-- x1
                rr8     <-- y1

Output          r5      <-- error code
```

Coordinates are IEEE single-precision real; others are integer.

ERRORS

```
 0      (no error)
35      "View area not open"
38      "Parameter out of range"
```

EXAMPLE

```
[Assume DivideViewArea has created a view area it has
labelled as #6]


SetWCSpace(6,35,-1.5,-20,2.5)
```

The world coordinate space is defined along the x-axis from -20.0 to 35.0, and along the y-axis from -1.5 to 2.5. Note that this set of values does not define the proportions the view area rectangle! The view area size is determined entirely by the DIVIDE VIEW AREA function; these coordinates determine the scaling interpretation within that space. Thus, this view area now recognizes 55 whole-number divisions along its x-axis, and 4 whole-number divisions along its y-axis. Note that the coordinate ordering was deliberately perverse to demonstrate flexibility.

DIVIDE VIEW AREA(div/orient,divpt,xform#)

Creates a new view area by dividing the current one according to the first two parameters; the number of the new view area is returned in the third parameter. The new view area inherits many of the attributes of its "parent," such as text spacing, color defaults, and world coordinate space definition.

The first parameter, "div/orient", determines whether the current view area shall be divided horizontally or vertically, and on which side of that division the new view area shall be oriented (above/below or left/right:

"div/orient"	Division	Loc. of New View Area
0	HORIZONTAL	ABOVE division
1	HORIZONTAL	BELOW division
2	VERTICAL	LEFT of division
3	VERTICAL	RIGHT of division

The second parameter, "divpt", places the division point. For horizontal divisions (div/orient=0,1), the parameter is in scanlines from the top of the current view area (min.=1 scanline; max.=current view area height - 1 scanline). For vertical divisions (div/orient=2,3), the parameter is in character positions from the left edge of the current view area. The character width used is as currently defined, i.e., 6 or 8 pixels wide. However, all actual side edges are always on byte boundaries, i.e., at multiples of 8 pixels from the screen's leftmost column of pixels. When the character width is 8 pixels, the resultant divisions are intuitively predictable, but when characters are 6 pixels wide, prediction is more difficult. The actual view area width, in pixels, of the left-side viewing area may be determined by the following prediction formula:

```
WIDTH = truncate [ (nmbr-of-chars * char-width + 3) / 8 ] * 8
```

where "nmbr-of-chars" is the second parameter and "char-width" is the current pixel width (6 or 8) of characters. With 6-pixel characters, there is often a right-side margin, narrower than pixels, in which characters cannot appear; consequently, such view areas frequently allow one fewer characters per line than the "divpt" parameter would seem to imply. An alternate entry for either vertical or horizontal divisions is "-1", which divides the current view area as equally as possible.

The third parameter returns the "transformation number" by which this view area and all its attributes (in particular, the scaling transformation definition from world coordinates to the view area) is subsequently referenced.

Note that as a start-up condition, the full screen is defined as view area #1. This view area can, of course, be subdivided in multiple ways, and adjacent view areas can be closed and joined with it (as long as the resultant view area is rectangular), but view area #1 ALWAYS exists; unlike other view areas, view area #1 cannot be closed. There may be as many as 16 active view areas at any one time. A new view area is assigned the lowest available number in the range 2..16 (e.g., if 6 view areas are created, and then #3 is closed, the newly-created view area will be assigned the number 3).

Register assignment:

```
Input          r8      <-- div/orient (0..3)
               r9      <-- divpt [(1..63 or 79)
                                 or (0..255)]

Output         r5      <-- error code
               r7      <-- xform# (2..16)
```

All values are integers.

ERRORS

```
 0      (no error)
36      "Unable to create view area"
38      "Parameter out of range"
```

EXAMPLE

```
DivViewArea(0,127,xfi)
DivViewArea(1,-1,xfj)
DivViewArea(3,23,xfk)
```

The first example divides the current view area horizontally with the upper view area being the new view area, 127 scanlines high; its number is assigned to xfi. The second example divides the current view area horizontally at the midpoint; the new view area is below the division, and its number is assigned to xfj. The third example divides the current view area vertically; the view area left of the division is (theoretically) 23 characters wide, and retains the number of the current view area; the right side of the division is the new view area, and its number is assigned to xfk. Note that if character width = 6 pixels, the actual width of the left view area is 22 6-pixel characters plus a 4-pixel margin. Should the three examples follow successively as the first statements after startup (i.e., no view area subdivision yet), the result would be four view areas: the upper half of the screen (view area #2); the bottom quarter of the screen (#3); and two in the lower middle quarter of the screen, a smaller one on the left (#1) and a larger on the right (#4).

SELECT VIEW TRANSFORMATION(xform#)

Selects the view area identified by the value "xform" (as previously defined by DIVIDE VIEW AREA). All subsequent text and graphics output will go to this view area and enter in accordance with the attributes of this view area (color, world coordinate scaling, text line height and character width, and current text and graphics locations, for examples). Note especially that graphic objects described in world coordinates will be mapped to this view area by a transformation using the world coordinate space and view area definition of this transform number.

The value of "xform#" MUST be 1 or correspond to a view area defined by DIVIDE VIEW AREA; otherwise, an error code is sent to the error status variable.

Register assignment:

```
Input          r8      <-- xform# (1..16)

Output         r5      <-- error code
```

All values are integer.

ERRORS

```
 0      (no error)
35      "View area not open"
38      "Parameter out of range"
```

EXAMPLE

```
[Assume DivideViewArea has created a view area it has
labelled as #6 and assigned to Nwdw]


SelViewTrans(Nwdw)
```

All further text and graphic output will be directed to view area #6, in accordance to the coordinate, color, etc., definitions active for that view area. If view area #6 is not currently defined, an error is generated.

CLOSE VIEW TRANSFORMATION(xform#)

Closes the view area identified by the value "xform" (as previously defined by DIVIDE VIEW AREA). That view area is joined to a view area or view areas adjacent to it; the area is cleared to the background color of the area(s) to which it is joined, and the enlarged view area(s) have their coordinate definitions correspondingly adjusted to map to the new view area dimensions (note that the world coordinate values are not changed, but rather the mapping is). If the current view area is closed, then view area #1 becomes the current view area.

If the value of "xform#" is zero, then any and all currently defined view areas labeled in the range 2..16 are closed, and view area #1 is the current view area, filling the entire screen.

Note that this command can attempt to close view area #1 without generating an error; however, view area #1 cannot be closed, and such an attempt simply has no effect. Also, this command can attempt to close view areas that have not been opened (defined by a DIVIDE VIEW AREA call), as well as hypothetical view areas outside the range of 1..16, without generating an error condition or error message.

Register assignment:

```
Input           r8      <-- xform# (0,1..16, or any
                                other integer)

Output          none
```

ERRORS

none

EXAMPLE

```
CloseViewTrans(xform#)
```

The view area identified by the value of "xform#" is cleared to the background color of an adjacent view area or areas (it may be necessary to divide the area being closed, in order to join it with two different adjacent areas, in order to preserve rectangularity of the enlarged areas), and the closed area is joined to the adjacent area(s). If view area #6 was the current view area, view area #1 becomes the current view area.

CLEAR VIEW AREA(xform#,err)

Clears the view area identified by the value "xform" (as previously defined by DIVIDE VIEW AREA) to the background color (default is black; SELECT TEXT COLOR can set a different background color). Note that the view area is not closed; this call merely removes all current contents of this view area.

Register assignment:

```
Input           r4      <-- xform# (1..16)

Output          r5      <-- error code
```

Both values are integer.

ERRORS

```
 0      (no error)
35      "View area not open"
38      "Parameter out of range"
```

EXAMPLE

```
ClearViewArea(xform#,err)
```

The view area identified by the value of "xform#" is cleared to the background color of that view area. The view area is still open to receive new output in accordance with the coordinate transformation and color attributes currently defined for that view area.

ESCAPE(functionnmbr, recordname)

A routine that performs a special graphics function. In this Graphics Package, there is only one Escape routine defined, "flood".

```
FLOOD

ESCAPE(functionnmbr, recordname)
    [functionnmbr-->1 (number, constant, or variable)]
```

Floods an area (i.e., paints it) in accordance with the parameters in the record (data structure) "recordname". That data structure (array, record, etc.) contains the following information:

```
Nominal point coordinates x, y
    [identifies area to be flooded]
Color number  [color nmbr, or color index on 4-colr sys]
Bordercolor   [as with color number]
```

The area surrounding the point (x,y) is flooded with the color identified in the data record, within a contiguous border of the color identified by "bordercolor" in the data record. If the area-locator point happens to fall in a field of the color "bordercolor", no flooding will occur. Note that the area flooded is that bounded by a single color ; if, for example, blue is to flood an area bordered by green, a red line will not serve as a border--it will be flooded out.

Register assignment:

```
Input          r1      <-- 1:  1 = "FLOOD"
               rr2     <-- recordname pointer

Output         r5      <-- error code
```

Pointer is a segmented address; other values are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
76      "Error in parameter"
```

RECORD STRUCTURE

The "recordname" parameter is the name of a record having the following structure:

```
X coordinate:   2 16-bit words; IEEE single-prec. real
                  number, high-order word appears first
Y coordinate:   2 16-bit words; IEEE single-prec. real
                  number, high-order word appears first
Color number:   1 16-bit word, integer (high-order first)
Bordercolor:    1 16-bit word, integer (high-order first)
```

EXAMPLE

Assume the data structure "FRec",containing:

```
27.34   [x coord., area locator]
128.    [y coord., area locator]
2       [flood color]
3       [border color]
```

```
Esc(1,FRec)
```

An area in world-coordinate space that contains the point (27.34,128.) will be flooded. The result will be a filled polygon where the number 2 governs its color. The polygon border is determined by the screen contents at the moment the function is called; the flood will fill every contiguous nook and cranny that lies within a contiguous line whose color is governed by the number 3. Note that wherever such a line is not continuous, the flooding will "leak" through to a new area.

The language describing this function's relation to color numbers is obtuse because the example will respond differently on monochrome, four-color, and eight-color systems. Monochrome and eight-color systems deal directly with color numbers, but four-color systems deal indirectly via indices to a table of four pre-selected colors from a gamut of eight. However, integers in the range 0..7 for both color parameters will work without error generation in all color configurations (see discussion in SET COLOR REPRESENTATION).

INQUIRY FUNCTIONS

These routines retrieve data, mostly from tables or other variables in the Graphics Package. Unlike routines in the other categories, these routines do not generate errors -- that is, none of these 11 routines sends an error code to the Graphics Package error status variable. Rather, they report the detection of any errors directly by returning an error code number (or "0" for "no error") through an "err" parameter in each routine. Note that, for most of these inquiry routines, no error number is defined; in most cases, there is no error that can occur. The "err" parameter is maintained in these routines for format compatibility with the GKS graphics standard. Each function is detailed in the following pages.

```
INQ VIEW AREA(err,bytewdth,scanlineht,chrwdth,txtlineht)
INQ WORLD COORDINATE SPACE(err,x0,y0,x1,y1)
INQ CURRENT TRANSFORMATION NUMBER(err,xform#)
INQ ATTRIBUTES(err,GRcolr,FGcolr,BGcolr,logop,lineclass)
INQ TEXTCURSOR(err,column,row,blinkrate)
INQ GRAPHPOS(err,x,y)
INQ GRAPHCURSOR(err,x,y,blinkrate)
INQ PIXEL ARRAY(Xwdth,Yht,err,invalidvals,arrayname)
INQ PIXEL COORDINATES (Xworld,Yworld,err,Xpixcoord,Ypixcoord)
INQ PIXEL(x,y,err,pxlcolrnmbr)
ERROR INQUIRY(errorcode)
```

INQ VIEW AREA(err,bytewdth,scanlineht,chrwdth,txtlineht)

Returns for the current view area, the size definition and text parameters of this view area. The view area width is in bytes, the height is in scanlines, the current width of a character is in pixels (6 or 8), and the textline height is in scanlines (10..16).

Register assignment:

```
   Input     none

   Output    r5     <-- error code
             r8     <-- view area width (1..64 bytes)
             r9     <-- view area height (1..256 scanlines
             r10    <-- text character width (6,8 pixels)
             r11    <-- text line height (10..16 scanlines)
```

All values are integer.

ERRORS

```
        0      (no error)
```

EXAMPLE

```
    InqViewArea(ErVar,VAwd,VAht,TXwd,TXht)
```

Returns for the current view area, in the latter four parameters: the view area width, view area height, text character width, and text line height. ErVar=0.

INQ WORLD COORDINATE SPACE(err,x0,y0,x1,y1)

Returns the world coordinate space parameters for the current view area. Parameters 3 and 4, (x0,y0), give the world coordinates for the lower left corner of the space (which maps to the lower left corner of the view area). Parameters 5 and 6, (x1,y1), give the world coordinates for the upper right corner of the space (which maps to the upper right corner of the view area). Note that these coordinates do not determine the relative proportions of top and side of the view area, but rather determine how points in world coordinate space will map to that view area.

Register assignment:

```
Input          none

Output         r5      <-- error code
               rr6     <-- x0
               rr8     <-- y0
               rr10    <-- x1
               rr12    <-- y1
```

All x and y values are IEEE single-precision real; the error code is integer.

ERRORS

```
0      (no error)
```

EXAMPLE

```
InqWorldCoords(ErVar,Xloleft,Yloleft,Xhiright,Yright)
```

Returns for the current view area, in the latter four parameters: the world coordinates for the lower left corner, and the world coordinates for the upper right corner of the rectangular "problem" space (the space in which the application program's problem is defined). ErVar = 0.

INQ CURRENT TRANSFORMATION NUMBER(err,xform#)

Returns the identification number of the current view area number in "xform#". This number is used in selecting a different view area of the screen to which to move (SELECT VIEW TRANSFORMATION), to re-define the world coordinates (SET WORLD COORDINATE SPACE), to erase a view area's contents (CLEAR VIEW AREA), to close (i.e., undefine) a viewspace (CLOSE VIEW TRANSFORMATION), and to retrieve a host of color and coordinate information about a view space (see the various INQ ... functions).

Register assignment:

```
    Input           none

    Ouput           r5      <-- error code
                    r7      <-- view area transformation
                                number (1..16)
```

All values are integer.

ERRORS

```
        0       (no error)
```

EXAMPLE

```
    InqCurTransNmbr(ErVal,ViewArea)
```

Returns the identification number of the current view area in "ViewArea."
ErVal = 0.

INQ ATTRIBUTES(err,grcolr,fgcolr,bgcolr,logop,lineclass)

Returns the color, logic, and line attributes for the current view area.
Information returned is:

grcolr	graphics color (current) (see SET GRAPHICS COLOR)
fgcolr	foreground color for text (see SET TEXT COLOR)
bgcolr	background color for text, CLEAR VIEW AREA, and logic (see SET TEXT COLOR)
logop	logic operator (PSET|XOR|AND|NOT|OR|PRESET) for color(see SET COLOR LOGIC)
lineclass	interpretation of line coordinates (LINE|BOX|BOXFILL; see SET LINE CLASS)

Register assignment:

```
    Input           none

    Output          r5      <-- error code
                    r7      <-- logop (0..5)
                    r8      <-- lineclass (0..2)
                    r9      <-- grcolr (0..7)
                    r10     <-- fgcolr (0..7)
                    r11     <-- bgcolr (0..7)
```

All values are integer.

ERRORS

0	(no error)

EXAMPLE

```
InqAttributes(ErVar,Kgraph,Ktext,Kbackg,Klogic,LBF)
```

Returns for the current view area, in the latter five parameters: the current color for graphic output, the current color for text characters (note that these colors may be different), the current background color, the current logic operator used (pixel-by-pixel) between new output and target-area color numbers, and whether the coordinates of a line-function will be used to generate lines, boxes, or filled boxes. ErVar = 0.

INQ TEXTCURSOR(err,column,row,blinkrate)

Returns the next text entry point (which coincides with the location of the text cursor) and the text cursor blink rate for the current view area. This position is given in number-of-characters from the view area's left edge (column), text lines from the view area's top (row), and state changes per second, rounded to the nearest 50-millisecond increment (blinkrate). Note that this position is NOT where the next graphic output will appear (see INQ GRAPHPOS). Unlike the graphics case, the next point of entry for text and the text cursor position are identical. If the information is not available (e.g., view area too small for text), an error code is returned in "err" and the remaining parameters are left undefined, but no error code is sent to the error status variable.

Register assignment:

```
Input           none

Output          r5      <-- error code
                r7      <-- blinkrate (0..20)
                r8      <-- text column (1..64 or 80)
                r9      <-- text row (1..16 or 25)
```

All values are integer.

ERRORS

0	(no error)
151	"Information not available" (in this case: view area too small to contain text)

EXAMPLE

```
InqTextcursor(ErVar,TXcol,TXrow,TXblinkrate)
```

Returns for the current view area, in the latter three parameters: the current entry-point and cursor location by column (TXrow) and row (TXrow), and the blinkrate for the cursor. If the information is available, ErVar=0; otherwise, ErVar=151.

INQ GRAPHPOS(err,x,y)

Returns for the current view area the location at which new graphics output (note: not text output) will begin (e.g., a LINEREL function would generate a line with one end at this point). The parameters (x,y) define a point in world coordinate space. "err" is set to 0.

Register assignment:

```
Input           none

Output          rr2     <-- y
                r5      <-- error code
                rr6     <-- x
```

Values x and y are IEEE single-precision real; the error code is integer.

ERRORS

```
        0       (no error)
```

EXAMPLE

```
InqGraphPos(ErVar,Xpos,Ypos)
```

Returns for the current view area, in the latter two parameters: the world coordinates for the current graphics position, i.e., the position which relative-oriented new graphics output (e.g., LINEREL) will reference. ErVar = 0.

INQ GRAPHCURSOR(err,x,y,blinkrate)

Returns for the current view area the location of the graphics cursor, and its blinkrate. Note that the graphics cursor location and the graphics position are generally NOT the same location (unlike the case for text); the graphics cursor merely marks a place in the view area. An application program might wish to use these coordinates to update the current graphics position, or to set an absolute location starting point for LINEABS or some other geometric output routine. The parameters (x,y) define a point in world coordinate space; the graphics cursor will place the upper left corner of its 8 x 12 pixel shape at this location. The blinkrate is in state changes per second, rounded to the nearest 50-millisecond increment (blinkrate). Note that the graphics cursor location and the text cursor location are entirely independent, and that only one of these two cursors (but possibly neither, if so specified) may appear at one time.

Register assignment:

Input	none	
Output	rr0	<-- x
	rr2	<-- y
	r5	<-- error code
	r9	<-- blinkrate (0..20)

Values x & y are single precision real; others are integer.

ERRORS

0 (no error)

EXAMPLE

```
InqGraphCursor(ErVar,GrX,GrY,blinkrate)
```

Returns for the current view area, in the latter three parameters: the current graphics cursor location (GrX,GrY, world coordinates), and the blinkrate for the graphics cursor. ErVar=0.

INQ PIXEL ARRAY(Xwdth,Yht,err,invalidvals,arrayname)

Retrieves a rectangle from the view area screen and stores it (for later re-display) in "arrayname". The inverse function is PIXEL ARRAY, which outputs the rectangle stored in an array to the view area. The upper left corner of the rectangle to be retrieved from the screen is placed at the current graphics position. The rectangle size is specified in world coordinate space dimensions; "Xwdth" is in x-axis units, "Yht" is in y-axis units. These dimensions are transformed into view area dimensions (pixels).

It is the total pixel count, plus three housekeeping words, that determines the size of the storage array (recall that, for color systems, the pixel count effectively doubles or triples). The application program is responsible for knowing the required array size and for allotting (dimensioning) space for it.

The bulk of the array is bit images for the scanlines within the rectangle, packed 16 bits per array entry. Each scanline image will begin with the first bit for the scanline in bit 15 of the first array entry for the scanline (i.e., left-justified). The array must be allotted a size at least large enough to accommodate the rectangle; the size in words may be calculated according to the formula:

```
size = truncate[(pixelwidth+15)/16]*pixelheight*colrplanes+3
```

where "colrplanes" is the number of color planes in a system configuration (monochrome=1, four-color=2, eight-color=3); the final 3 words are for the three housekeeping words at the beginning of the array that store the rectangle width, height, and special codes. The maximum size is a full screen (assuming a configuration with sufficient memory to

accommodate that large an array). The array is one-dimensional.

The "err" parameter returns value 0 unless a problem occurs, in which case it returns an integer error code value. An error code will be generated if the combination of the current graphics position and the width and height parameters imply a point (i.e., a corner of the rectangle) that is outside the view area. When an out-of-bounds error is detected, the destination array is left untouched.

The "invalidvals" parameter reports discovery of invalid pixel color values. If for some reason the color of a pixel cannot be ascertained, the "invalidvals" parameter is set to indicate "PRESENT"; if all pixel values are valid, the "invalid- vals" parameter is set to "ABSENT". Note that the "err" parameter is not affected by the discovery of invalid pixel color values.

Register assignment:

```
Input          rr0     <-- X-width
               rr2     <-- Y-height
               rr6     <-- arrayname pointer

Output         r4      <-- 0,1:    invalid values
                                     0 = ABSENT
                                     1 = PRESENT
               r5      <-- error code
```

Dimensions are IEEE single-precision real; the pointer is a segmented address; others are integer.

ERRORS

```
 0      (no error)
 7      "Out of memory"
38      "Parameter out of range"
76      "Error in Parameter"
```

ARRAY FORMAT

The array is a single-dimension array of one-word integers. The first three words are housekeeping data; the remainder of the array stores 16-bit screen image words in each element.

The first word is a one-word integer that gives the width of the rectangle in pixels. The second word is a one-word integer that gives the height of the rectangle in scanlines. The third word holds codes that tell about how this array was generated. The low-order byte of this third word gives the number of color planes that were in use when the array was filled (monochrome, 1; four-color, 2; eight-color, 3); the high-order byte is reserved (and should be assigned a value of 0 when filling array values "from scratch").

The remainder of the array is one-word integer elements which contain bit-patterns (16 bits/word) that are retrieved from the screen bit-map color-planes. The exact structure of the array from this point on is

dependent on how many color planes were active when the array was filled, since bit-plane data is interleaved scanline-by-scanline. A first batch of one or more words (depending on view area width) gets pixels from the first color-plane's first scanline, transferring pixels left-to-right into one or more word(s), high-order bits filled first; the last word has from 0 to 15 low-order bits unused. If there is a second color-plane (i.e., it is a four-color system), that plane's first scanline of pixels is transferred (again with possible leftover unused bits in the last word); and so on until all colorplanes have had one scanline transferred. Then another scanline (if any) is transferred, again color-plane by colorplane; until finally all scanlines have been transferred to the array.

EXAMPLE

```
[Assume the default world coordinates (=view
 area pixel coordinates, 0..x..511, 0..y..255),
 view area #1 only is open:]

InqPixelArray(512,256,ErVar,bogeys,Holdpic)
```

Retrieves the entire screen (assuming only view area #1 is open) for storage in the array Holdpic. Holdpic must be large enough: for monochrome, 8195 words; for four-color, 16387 words; for eight-color, 24579 words. If there is an error (e.g., either rectangle-size parameter is too large), an error code is returned in ErVar, but no error code is sent to the error status variable. If, for some reason, the color of one or more pixels cannot be determined, "bogeys" returns the value 1; otherwise it is 0.

INQ PIXEL COORDINATES(Xworld,Yworld,err,Xpxlcoord,Ypxlcoord)

Given a point (Xworld,Yworld) in world coordinate space, returns the corresponding pixel coordinates with respect to the borders of the current view area, in Xpxlcoord and Ypxlcoord. If the function acts without problem, "err" returns value 0, otherwise it returns an error code. However, no error code is sent to the error status variable.

Register assignment:

```
Input          rr0     <-- Xworld
               rr6     <-- Yworld

Output         r5      <-- error code
               r6      <-- Xpxlcoord (0..511)
               r7      <-- Ypxlcoord (0..255)
```

Inputs are IEEE single-precision real; outputs are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
Assume a view area that is eight 8-bit characters wide
and 96 scanlines high, with world coordinates
(x-axis: -10,0 to 10.0) and (y-axis: -1.0 to 1.0).
Assume XX= 7.5, YY=-0.125

InqPixelCoords(XX,YY,ErVar,xpx,ypx)
```

For the given point (XX,YY), the function returns: xpx=56, ypx=42, ErVar=0. Had XX or YY been outside of their assigned ranges, then ErVal=38, and xpx and ypx would be undefined; however, no error condition would be generated.

INQ PIXEL(x,y,err,pxlcolrnmbr)

In the current view area, for the point (x,y) in world coordinate space, returns the color number of the nearest pixel when the point is mapped onto the view area space. If the function is successful, "err" returns 0; if not, "err" returns an error code.

The pixel color number is the direct number of a color in monochrome and eight-color systems, but in four-color systems, the value is an index into a table of pre-selected colors (four colors selected from eight available).

Register assignment:

```
Input           rr0     <-- x
                rr6     <-- y

Output          r3      <-- color number (0..7)
                r5      <-- error code
```

Values x & y are IEEE single precision real; others are integer.

ERRORS

```
 0      (no error)
38      "Parameter out of range"
```

EXAMPLE

```
InqPixel(XX,YY,ErVar,Colr)
```

Assuming (XX,YY) is a valid world-coordinate-space point, a corresponding point in the view area is calculated, and the nearest pixel is selected. "Colr" returns a value for that pixel that is interpreted as a color number or as an index into a color table which selects a color number

(see discussion above). "ErVar"=0. If (XX,YY) falls outside the defined world coordinate space, then "ErVar" returns an error code, and "Colr" is undefined.

ERROR INQUIRY(errorcode)

Returns the error status for the most recently called graphics routine other than an INQ ... routine. The INQ ... class of functions do not alter or reference the error status variable: rather, each routine of the INQ ... class has its own "error" parameter, through which it reports any problems (or "OK"). Each of the non-INQ... routines clears the error status variable prior to execution of its assigned task, so that upon completion of the routine, the error status variable reflects the error status of that routine. If the status value is "0", then no error has occurred.

Register assignment:

```
Input     none

Output    r5      <-- error code (0..255)
                                    0 = no error,
                                    1..255 = some
                                       error code
```

All output is integer.

ERRORS

none (this routine does not itself generate errors or write to the error status variable)

EXAMPLE

```
[Assume an application program has already defined 16
 view areas]
DivViewArea(0,23,TransNmbr)
ErrorInq(ErVal)
if (ErVal>0) then exit
```

DivViewArea attempts to create one more view area than is possible, thereby placing an error code,(#36, "unable to create view area") in the error status variable. "ErrorInq" returns that value in "ErVal"; since it is indeed greater than zero, the "if" test is "true" and the consequent statement is executed (e.g., if "exit" implies a jump to a procedure which ends the application program, then the program is terminated).

REFERENCES

Reference material gives language bindings PASCAL and assembly language, and provides a concordance between BASIC and PCOS 3.0 that compares M20 BASIC graphics with this Graphics Package.

Additional reference material lists graphics system calls and graphics-caused error messages.

LANGUAGE BINDINGS

Pascal Language Binding

The following identifiers are the procedure names for PASCAL. They are listed in the order that they appear in the functional description. Parameters are as specified in the functional descriptions: integer numbers or variables, real numbers or variables, or array identifiers.

```
LineAbs
LineRel
Polyline
MarkerAbs
MarkerRel
Polymarker
TextCursor
GraphPosAbs
GraphPosRel
GraphCursorAbs
GraphCursorRel
PixelArray
GDP

SetLineClass
SetTextline
SelectCursor
SetTxCsrBlinkrate
SetGrCsrBlinkrate
SetTxCsrShape
SetGrCsrShape
SetColourRep
SelectGrColour
SelectTxColour
SetColourLogic

OpenGraphics
CloseGraphics
SetWorldCoordSpace
DivideViewArea
SelectViewTrans
CloseViewTrans
ClearViewArea
Escape

InqViewArea
InqWorldCoordSpace
InqCurTransNumber
InqAttributes
InqTextCursor
InqGraphPos
InqGraphCursor
InqPixelArray
InqPixelCoords
InqPixel
ErrorInquiry
```

Assembly Language Binding

The following identifiers are the procedure names for PLZ/ASM and M20 assembler. They are listed in the order that they appear in the functional description. Parameters are as specified in the functional descriptions: integer numbers or variables, real numbers or variables, or array identifiers. Consult the functional description for parameter register assignments and structures for arrays and records.

LineAbs
LineRel
Polyline
MarkerAbs
MarkerRel
Polymarker
TextCursor
GraphPosAbs
GraphPosRel
GraphCursorAbs
GraphCursorRel
PixelArray
GDP

SetLineClass
SetTextline
SelectCursor
SetTxCsrBlnkrate
SetGrCsrBlnkrate
SetTxCsrShape
SetGrCsrShape
SetColourRep
SelectGrColour
SelectTxColour
SetColourLogic

OpenGraphics
CloseGraphics
SetWorldCoordSp
DivideViewArea
SelectViewTrans
CloseViewTrans
ClearViewArea
Escape

InqViewArea
InqWorldCoordSp
InqCurTransNmbr
InqAttributes
InqTextCursor
InqGraphPos
InqGraphCursor
InqPixelArray
InqPixelCoords
InqPixel
ErrorInquiry

CONCORDANCE BETWEEN BASIC AND PCOS 3.0 GRAPHICS PACKAGE

BASIC		GRAPHICS PACKAGE
WINDOWS:	Define New Window var=WINDOW (Q,P,[,V] [,h]	DIVIDE VIEW AREA (div/ orient, divpt, xform#) See also SET TEXTLINE
	Change to Different Window WINDOW %num_or_var	SELECT VIEW TRANFORMATION (xform#)
	Retrieve Current window Number var=WINDOW (0,0)	INQ CURRENT TRANSFORMATION NUMBER (err, xform#)

	Define Text Spacing var=WINDOW (0,0,[,v] [,h])	SET TEXTLINE (chrwdth, txtlineht)
	Clear Window CLS [%wndw]	CLEARVIEW AREA (xform#, err)
	Release Window Definition CLOSE WINDOW [%wndw]	CLOSE VIEW TRANSFORMATION (xform#) [(xform#=0)-->close view areas 2-16]
COLORS:	Select Palette COLOR= n1, n2, n3, n4	SET COLOR REPRESENTATION (indx#, colr#)
	Set Foreground, Background Color COLOR [%wndw] n [,n]	SELECT GRAPHICS COLOR (nmbr) SELECT TEXT COLOR (fgnmbr, BGnmbr)
	Fill Area with Color PAINT [%wndw] (x,y) [,clr [,borderclr]]	ESCAPE (1,recordname) [1st param.: 1="flood"] [recordname: x,y,color, bordercolor]
	Define Coordinate System (WC&NDC&DC) SCALE [%wndw] x1,xh,y1, yh	SET WORLD COORDINATE SPACE (xform#,x0,y0,x1,y1)

	Retrieve Pixel Coordinate given SCALE SCALEX (x_scale_coord) SCALEY (x_scale_coord)	INQ PIXEL COORDINATES (xworld,Yworld,err, Xpixcoord,Ypixcoord)
CURSOR:	Move Text Cursor CURSOR [(x,y)] [0:1 (=off:on)] [,rate [,shape]]	TEXTCURSOR (column,row) See also SET TEXT CURSOR BLINKRATE SET GRAPHICS CURSOR BLINK-RATE SET TEXT CURSOR SHAPE SET GRAPHICS CURSOR SHAPE
	Move Graphics Cursor CURSOR POINT [[same arg list as CURSOR]]	GRAPHCURSORABS (x,y) GRAPHCURSORREL (dx,dy) See also SET TEXT CURSOR BLINKRATE SET GRAPHICS CURSOR BLINK-RATE SET TEXT CURSOR SHAPE SET GRAPHICS CURSOR SHAPE
	Turn Off Cursor See CURSOR and CURSOR POINT	SELECT CURSOR (selectnmbr)
	Define Cursor Attributes See CURSOR and CURSOR POINT	SET TEXT CURSOR BLINKRATE (rate) SET GRAPHICS CURSOR BLINK-RATE (rate) SET TEXT CURSOR SHAPE (arrayname) SET GRAPHICS CURSOR SHAPE (arrayname)

	Retrieve Cursor position re: current window var=POS (0:1) [[=row:col]] (above for text only)	INQ TEXTCURSOR (err, column, row, blinkrate) INQ GRAPHCURSOR (err,x,y,blinkrate)
DRAW:	Set Current Graphics Position See DRAW	GRAPHOSABS (x,y) GRAPHOSREL (dx,dy)
	Draw Line LINE [%wndw,] [[STEP] (x1,y1)-] [STEP](x2,y2) [,[clr][,[B[F]][[,vb]	LINEABS (x,y) LINEREL (dx,dy) POLYLINE(#points,Xarray, Yarray) See also GRAPHOSABS, GRAPHOSREL, SET LINE CLASS, SELECT FOREGROUND COLOR SET COLOR LOGIC
	Draw Circle (or Ellipse) CIRCLE [%wndw,] (x,y),r [,[dr][,[asp][,vb]]	GDP(1,2,Xarray,Yarray, datarec) (1=Circle) GDP(2,3,Xarray,Yarray, datarec) (2=Ellipse) [arrays contain defining coordinates; "datarec" is unused] See also: SELECT GRAPHICS COLOR SET COLOR LOGIC

	Move (& Draw?) to Point(s) DRAW [%WNDW] str. const.:var M dx,dy U dy J x,y D dy L dx C clr R dx B- [prefix: don't draw] verb postfixes: -A [and] -O [or] -N [not] -X [xor] -P [=PSET; default] -R [=PRESET, bkgnd clr]	See above Geometric Primitives
PIXEL:	Set Pixel to Foreground (or Color) PSET [%windw](x,y) [,clr]	MARKERABS (x,y) MARKERREL (dx,dy) POLYMARKER(#points, Xarray,Yarray) See also: SELECT GRAPHICS COLOR
	Set Mearest Pixel to Background Color PRESET[%wndw])(x,y)	Set graphics color to BACKGROUND; use MARKER...? Change graphics color
	Retrieve Color Number, Nearest Pixel var=POINT (x,y)	INQ PIXEL(x,y,err, pxlcolrnmbr)
BITMAP:	Store Displayed Bitmap Segment GET [%wndw,] (x1,y1)-(x2,y2), 1st array elem.	INQ PIXEL ARRAY(Xwdth,Yht, err,invalidvals, arrayname) See also GRAPHOSABS, GRAPHOSREL, SELECT VIEW TRANSFORMATION

	Display Stored Bitmap Segment PUT [%wndw,] (x1,y1)[-(x2,y2)], 1st array elem. [,vb]	PIXEL ARRAY(Xwdth,Yht, arrayname See also: GRAPHOSABS, GRAPHOSREL, SET COLOR LOGIC
MISC.	Open, Re-initialize, and Close Graphics (Call in BASIC) CLEAR	OPEN GRAPHICS CLOSE GRAPHICS
	Preset Primitive Attributes	SET COLOUR LOGIC (operatornmbr) SET LINE CLASS (classnmbr)
	Retrieve Status Data	INQ GRAPHOS(err,x,y) INQ ATTRIBUTES(err,GRcolr, FGcolr,BGcolr,logop, lineclass) INQVIEW AREA(err, bytewdth,scanlineht, chrwdth,txtlineht) INQ WORLD COORDINATE SPACE (err,x0,y0,x1,y1) ERROR INQUIRY(errorcode)

PART III

18. OVERVIEW

ABOUT THIS CHAPTER

This chapter describes the contents of Part 3, which contains practical information for programmer use and reference material.

CONTENTS

INFORMATION IN PART 3

Part 2 of the manual contains an extended functional description of PCOS architecture. Part 3 supplements Part 2 with two kinds of information:

- Practical information on how to enhance or configure PCOS, such as "Creating M20 System Utilities" or "Customizing a PCOS System"

- Reference material such as "Data Passing Mechanism" or "PCOS Error Codes." Some reference material is included in order to provide a concise reference to information covered in more detail elsewhere, but scattered in several locations. Other reference material is unique.

BRIEF DESCRIPTION OF CONTENTS

A brief description of the contents of Part 3 is given below.

CREATING M20 SYSTEM UTILITIES

This section gives information on how to develop a utility program that can be executed as a PCOS command. The command routine must be capable of being loaded and executed by the command line interpreter, and properly returning control to the system after its task is completed. It may also need to make use of parameter information entered on the command line and passed to it by the command line interpreter.

SYSTEM CONFIGURATION

This section and the next two form a set that discusses the concepts of system configuration, environment, and customization. This first section relates system configuration to operational environment, and also contains some examples of hardware system configuration changes that require software support using PCOS utilities or custom software.

PCOS ENVIRONMENT AND GLOBAL COMMANDS

This section explains the relationship between PCOS and the operational environments that provide particular capabilities to the user. An overview is given of all the global commands that collectively define the PCOS and BASIC environments.

CUSTOMIZING A PCOS SYSTEM

This section reviews the software resources and utilities available to the system programmer for customizing PCOS to the requirements of particular installations and applications.

DATA PASSING MECHANISM

This section describes the general data passing mechanism PCOS uses to pass information among its internal elements and between PCOS and its supported languages. The method uses the stack, and can pass numbers, strings, null values, and a return address.

LANGUAGE SUPPORT

This section provides a fundamental overview of the support PCOS provides for high-level languages, some of which can be used for assembly language programming. Topics include data passing, calling on internal PCOS resources, memory allocation, and internal representation of numbers.

INSTALLING PCOS ON A HARD DISK

This section gives information on installing or updating PCOS on a hard disk. Information is included on how to preserve prior system reconfigurations.

ASCII

This section describes the ASCII standard for information interchange as used in the PCOS system and gives information on its general use and modification.

PCOS ERROR CODES

This section lists error codes and their meanings for BASIC and PCOS. Information is provided on changes and differences between PCOS 1.X and PCOS 3.X error codes. Suggestions to the programmer on the use of error codes are also given.

GLOSSARY

The glossary contains definitions of terms together with conceptual explanations. Definitions are oriented towards PCOS and the M20 system.

19. CREATING M20 SYSTEM UTILITIES

ABOUT THIS CHAPTER

This chapter gives information on how to develop an assembly language utility program that can be executed as a PCOS command.

CONTENTS

OVERVIEW

This section gives information on how to develop an assembly language utility program that can be executed as a PCOS command. When the command routine is developed and tested, it can be given an appropriate name and extension and used as a transient command or PSAVED and made part of a customized operating system. For information on names and extensions, see Part 2, "Commands and Utilities."

The command routine must be capable of being loaded and executed by the command line interpreter and properly returning control to the system after its task is completed. It may also need to make use of parameter information made available to it by the command line interpreter. The remainder of this section explains these matters.

OBJECT CODE FORMAT

Here is some information on the object code format for a transient utility and how to create the source code for the required object codes. The subjects covered are codefile formats, banners, external referencing, parameter passing, and error handling.

CODEFILE FORMAT

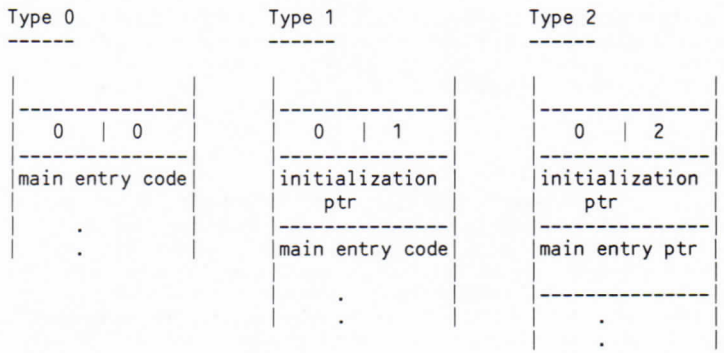
There is an Olivetti - Microsoft codefile standard for all M20 relocatable utilities. The format is very simple; the codefile begins with a 2-byte code, the codefile configuration type, which defines the format further. The information which follows the configuration code tells the Command Line Interpreter or PLOAD where the initialization code is and where the main program is. This free-format allows for some useful features. It automates the execution of initialization code, if any, required by the utility. It also allows the execution of the utility to start at a specified entry point.

The initialization code is the code which should be executed only once, upon the loading of the utility. The main program is the code which should be executed only at run time and not at load time.

The configuration type MUST be the first word in the code file. This is accomplished by entering "wval" statements at the beginning of the source file, and by making sure that at TLOC time the segment which contains this code is imaged first.

Configuration type is an integer which may be 0, 1, or 2. This is easy to remember because it always indicates the number of entry pointers that follow.

Configuration Type	Explanation
0	There are no pointers to either the main program or the initialization program. The next location is the main entry of the utility and there is no initialization code.
1	The next location is a pointer to the initialization code. What follows this pointer is the main entry point of the utility.
2	Two pointers follow. The first is a pointer to initialization code, and the second is a pointer to the main entry point of the utility.



Type 0 is the most common type used by M20 utilities. However, if a utility has initialization code, then either type 1 or 2 must be used. If the entry point of the main line program is not at the beginning of a utility, then type 2 must be used.

Type 9 wie Type 1 aber Entladeverbot

BANNERS

The banner, as used for PCOS utilities, has a very rigid format which was adhered to, because many items depended upon it. PLOAD depends on the name string to be at a fixed location so it can be displayed. DSTRING depends on the null terminator so it knows the length of the string. Labels and comments allow keeping track of versions. The user expects all utilities to behave in a predictable and consistent manner. However, the user may have more allowance in his or her choices (and may not need a banner at all).

The rules followed by PCOS software development were as follows:

1. In the object file, the banner must begin at the fourth byte within the main procedure (code segment). It must end with a zero byte.

2. In the source file, the banner must be preceded by the label str: and terminated with the standard comment.

3. The name of the utility must be spelled out in full, with exactly two capital letters; those used to invoke the util- ity. The name should end with at least one space, to separate it from the revision code.

4. The example below has a revision code. By definition, is it kept the same for all utilities for a particular revision. In the example, this information is included via an "include" file called revnum.

5. In the example, a development code is used to indicate the development version. It is lower case letter that should start out as "a", and increment once with each modification. For release, the development code is changed to a blank.

The example which follows is for the FCOPY utility. The object code would be placed on disk as a transient command under the name "fcopy.cmd".

```
str:    array [* byte]  := 'File Copy  '
  array [* byte] := 'a %r%00'  // INCR DEV. LEVEL FOR EACH MODIF.
```

EXTERNAL REFERENCING

Generally speaking, relocatable utilities cannot reference external procedures or variables, because they must run regardless of location. In order for the programmer to reference necessary external values, PCOS provides appropriate system calls. All information contained in PCOS system variables that is required for use when writing a PCOS utility is available through system calls, directly or indirectly.

PCOS internal routines can also access external variables through the "master table." However, these table entries are subject to change of location on every release of PCOS, while system calls are independent of location. Therefore, command routines cannot use the master table.

System calls are discussed in Part 2 of this manual. A detailed reference for each call is available in the Assembler User Guide.

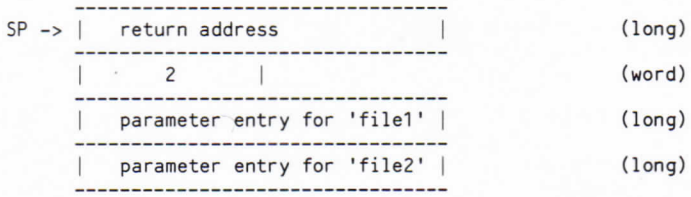
PARAMETER PASSING

Most utilities expect to process one or more parameters. The command line interpreter pushes all parameters onto the stack according to a particular format, so they must be popped by the utility according to the same format. When the command line interpreter calls the routine, its return address goes on top of the stack. Returning via this address allows the system to take control properly. The address must be safeguarded.

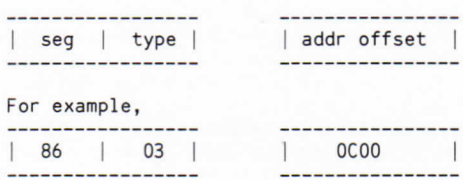
Then, the utility pops off one word, which is "n", the number of parameters. The command line interpreter limits the number of parameters, so this number will never be negative and will never be larger than 20 (hexadecimal 14). Each parameter entry is a long word, so the utility must next pop "n" number of long words off the stack. (The first one popped is the first parameter the user gave, and so on.) The utility must be careful to pop exactly "n" long words, and not rely on the number which is expected, because the user can enter an unexpected command line.

Here is an example of the stack upon entry to the FCOPY procedure. The command line was:

```
fc file1 file2
```



The format of the parameter entry is complex. It is an address pointer where the segment portion of the long word is "OR"ed with the parameter type. As an illustration,



Where the pointer is <<6>>0C00 and the parameter type is 3.

The parameter type must be extracted from the entry, by clearing the low-byte of the segment, before the pointer can be used. In this example, the parameter type is 3 and the pointer is 8600 0C00.

The format of item pointed to depends upon the parameter type. The command line interpreter passes three types of parameters to command routines:

0	null
2	integer
3	string

There are other types of parameters used within PCOS, especially by the languages supported by PCOS. For information on these other types, see the "Data Passing Mechanism" section in Part 3.

For convenience, a brief discussion of format of these three parameter types is given below along with notes on the usage of these types by command routines. More details are given in the "Data Passing Mechanism" section.

Null Parameters and Default Values

Null (or nil) parameters are put on the stack when the user enters a delimiter without a parameter entry. (The parameter entry is blank or null.) The command routine is responsible for supplying default values for null parameters. Using the SBASIC command for an example:

```
sb ,,5,512/CR/
```

The SBASIC command has four possible parameters. In this case, the first two entries supplied by the CLI are null. The command routine must supply default values.

Missing Parameters

Default values must also be supplied for missing parameters when appropriate. For example:

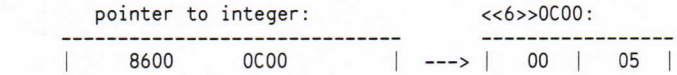
```
ps /CR/

sb ,,5/CR/
```

In the case of PSAVE, an optional file identifier could have been supplied. The command routine is given zero parameters and supplies the default file identifier. In the case of SBASIC, three parameters are supplied (two in null form). The command routine will supply the fourth.

Integer Parameters

If the parameter is an integer, then the pointer points to a 2-byte array containing the value. This array may or may not be at an even boundary address, so the value must be loaded into registers one byte at a time. For example, if the pointer entry is 8602 0C00, then after the type is

extracted the pointer is 8600 0C00.

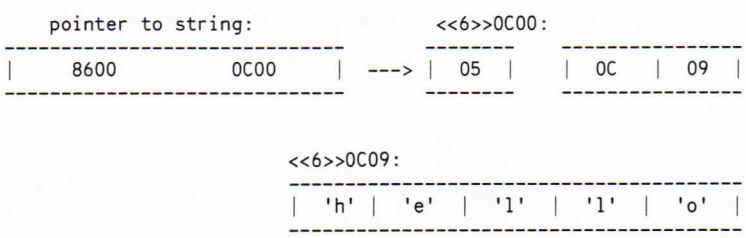


In this example, the integer value is 5.

String Parameters

If the parameter is a string, then the pointer points to a 3-byte array where the first byte contains the length of the string, and the other two are the integer offset of the pointer to the string. (The segment is assumed to be the same as before.) Again, this offset may or may not be at an even boundary address, so it must be loaded into registers one byte at a time.

For example, if the pointer entry is 8603 0C00, after the type is extracted the pointer is 8600 0C00.



In this example, the string length is 5.

ERROR HANDLING

If only we could assume no errors will occur, our code could be reduced in size by 90%. However, we should still check for errors. There is a standard method for doing this in PCOS transient utilities.

All the PCOS errors have an error code between 0 and 127. The "PCOS Error Codes" section in this part of the manual describes these errors. The section contains a sample include file defining all errors as constants. When referencing an error code in your utility, you should use the constant name rather than the number. For example, use "file_exists_err" rather than "58".

All PCOS components and utilities, as well as BASIC, assume the error code is in register r5. If r5 is zero, then there is no error. Because all error codes fit into the lower byte of r5, the high byte can be used for reporting a parameter number, if desired. Parameter numbers are optional, and if not reported, then rh5 should be zero.

With regard to error handling, there are two responsibilities prior to exit. First, the message must be displayed. This is done by system call #88, error message display. PCOS will display the appropriate message for the error in r5.

The second responsibility is to retain the error code in r5 upon exit.

Here is an example:

```
        clr     r5                      // no errors to report
        jr      normal_return
    error_hand_routine:
        ld      r5,error_code_num       // errors to report
        sc      #Error
    normal_return:
        ldl     rr14,return_address
        ret
```

EXAMPLE UTILITY

On the following pages is an example utility, FCOPY, which may be used as an example for Z8000 assembly language programming demonstrating configuration code setup, banners, system calls, parameter processing, and error handling.

```
fcopy MODULE
$SEGMENTED

!=================================================================
      edit history
      who     date                    description
      Ken     4/13/82         Added copy protection schemes.
=================================================================!

#include  <errcons.i>            //error code names
#include  <constants.i>          //system call names

CONSTANT
       STRING := 3                      // parameter type
GLOBAL
fc PROCEDURE
ENTRY
fcstart:  wval 0
       jr     start
internal
str:    array [* byte]:= 'File Copy '              //program id
#include <revnum.i>
   array [* byte] := 'c %r%00'  // INCR DEV. LEVEL EACH MODIF.
internal
   retadr      long
   param_count byte
start:
       lda    rr12,str         //display program id
       sc     #Dstring
```

```
        clrb    param_count
        pop     r0,@rr14        //number of parameters passed
        clr     r2
        ld      r3,r0
        sll     r3,#2
        addl    rr2,rr14        //compute return address
        ldl     retadr,rr2      //save return address
        test    r0
        jp      z,fc_param_bad


        incb    param_count     // FIRST FILE PARAMETER
        popl    rr2, @rr14      // rr2 = ptr to first file
        cpb     RL2, #STRING    // check for string type
        jp      ne,fc_param_bad
        clrb    RL2
        clrb    RH6
        ldb     RL6, @rr2       // r6 = filename len of file 1
        inc     r3
        ldb     RH1, @rr2       // get first half of ptr offset
        inc     r3


        ldb     RL1, @rr2       // second half, now offset in r1
        ld      r9, r1
        ld      r8, r2          // rr8 is real ptr for file 1
        incb    param_count     // SECOND FILE PARAMETER
        dec     r0              // one less parameter
        jp      z,fc_param_bad


        popl    rr2, @rr14      // rr2 = ptr to second file
        cpb     RL2, #STRING    // check for string type
        jp      ne,fc_param_bad
        clrb    RL2
        clrb    RH7
        ldb     RL7, @rr2       // r7 = filename len of file 2
        inc     r3
        ldb     RH1, @rr2       // get first half of ptr offset
        inc     r3
        ldb     RL1, @rr2       // second half now offset in r1
        ld      r11, r1
        ld      r10, r2         // rr10 is real ptr for file 2
        dec     r0              // one less parameter

            ! rest of utility code goes here !


fc_param_bad:
        ldb     RH5,param_count
        ldb     RL5,#param_err
        jr      fc_quit
nofile_error:
```

```
        ld      r5,#file_not_found_err

fc_quit:
        ldl     rr14,retadr
        ret

END fc


                    ! internal procedures go here !


END fcopy
```

20. SYSTEM CONFIGURATION

ABOUT THIS CHAPTER

This chapter describes the relationship between hardware system configuration and the operational environments. The chapter also contains some examples of hardware system configuration changes that require software support using PCOS utilities or custom software.

CONTENTS

OVERVIEW

This section describes the relationship between hardware system configuration and the operational environments, which consist of the system capabilities available to the user. The section also contains some examples of hardware system configuration changes that require software support using PCOS utilities or custom software.

This section together with the next two make a complete set. "PCOS Environment and Global Commands" gives an overview of the utilities available to define the BASIC and PCOS environments. "Customizing a PCOS System" concludes the set of three sections with an overview of the methods available in PCOS to configure a system for a particular installation or environment.

RELATIONSHIP OF CONFIGURATION AND ENVIRONMENT

The figure below shows the fundamental relationships among the M20 hardware configuration, the Professional Computer Operating System (PCOS), BASIC, the PCOS utilities and commands, and application programs.

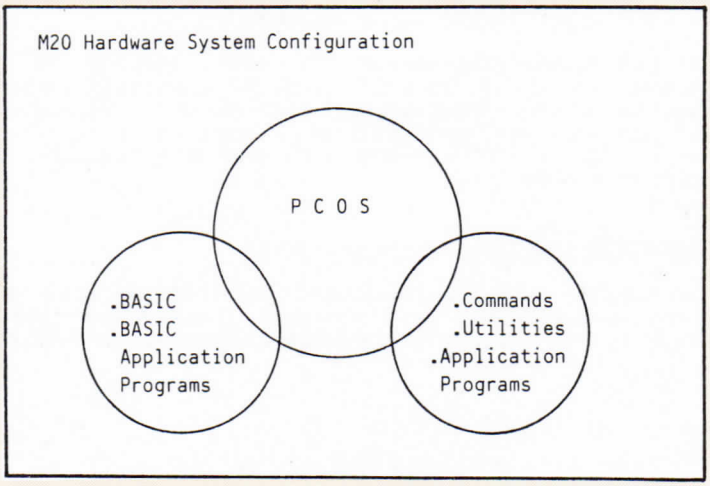


Fig. 20-1 Configuration and Environment

PCOS

The hardware configuration provides the fundamental universe of possibilities for use of the operating system and the other software resources. PCOS (or another operating system) provides functionality to the potentials of the hardware. The PCOS kernel controls the system peripherals and provides input/output capability. The kernel manages system resources such as system memory and the real-time clock, communicates with the user via the command line interpreter, and schedules internal activities. PCOS contains many other routines which are brought into action by the kernel as needed. Some of these software resources are simply part of PCOS but not part of the kernel. Others have names and are visible to the user as PCOS commands and utilities.

BASIC

BASIC uses PCOS resources and supplies resources of its own to the application programmer. When BASIC is running, PCOS is hidden. The user interacts with the BASIC operational environment which provides its own services and capabilities.

When an application program written in BASIC is running, it defines the user environment within the possibilities of BASIC. However, PCOS utilities can be called from BASIC.

OTHER LANGUAGES

Commands, utilities, and application programs that are written in assembly language or a compiled language such as PASCAL or C, all present PCOS resources to the user directly. Some of these may enhance the PCOS environment to the degree that they present a new operational environment. The Video File Editor is an example.

MODIFYING THE PCOS ENVIRONMENT

PCOS makes available to the user several utilities that allow modification of the PCOS and BASIC environment. They are called the Set System global commands. The next section "PCOS Environment and Global Commands" provides a general discussion of this important topic.

SOFTWARE RE-CONFIGURATION OF HARDWARE

The possibilities of M20 hardware configuration are presented in Part 1. They include adding expansion memory, using a color display, and providing optional peripherals and communications methods.

In addition, there are possibilities of hardware configuration that require software support using PCOS utilities or custom routines. Some of these are presented below.

PRINTERS

It is possible to use both a serial and a parallel printer although PCOS supports only one printer as FID 18. The easiest approach is to use SFORM and configure two versions of PCOS, one for each printer. Then either printer can be used alternately as desired.

Another method is to configure the parallel printer using SFORM, and then to use the RS232 interface to support a serial printer. This requires work, but allows both printers to be used together.

In some cases, non-standard printers can be supported using SFORM with the TRANSP (transparent) setting for the printer type. This setting causes all values in the text file to be sent without modification, which allows sending of special control sequences.

However, using a non-standard printer loses many of the design benefits of PCOS which so closely integrate the printer with other system resources, such as font definition and graphics.

DISK FORMATS

The "Disk Driver" section of Part 2 mentions that the driver can support certain non-Olivetti formats. These include ECMA and MS-DOS. The key design point for PCOS support of other formats is that the order of sectors within a track is independent of the disk driver, which does not have internal tables with this information. The order of sectors is set by the format utility, VFORMAT.

The order can be altered by developing a format utility that uses a different order. Therefore, so long as the desired format fits within the general range that PCOS supports for sector size and number of tracks, it can be used. Of course, a different order of sectors may degrade performance by causing the driver to miss a sector and have to access it on the next rotation.

21. PCOS ENVIRONMENT AND GLOBAL COMMANDS

ABOUT THIS CHAPTER

This chapter explains the relationship between PCOS and the operational environments that provide particular capabilities to the user. An overview is given of all the global commands that collectively define the PCOS and BASIC environments.

CONTENTS

PCOS ENVIRONMENT

PCOS support for the M20 system provides three operational environments: PCOS itself, BASIC, and the Video File Editor. Each operational environment provides particular capabilities to the user. PCOS is fundamental to the other two: BASIC and the editor require support services from PCOS.

GLOBAL COMMANDS

PCOS provides global commands which are utility programs that allow the user to change "global parameters." These parameters are internal values that collectively define a PCOS environment.

The global commands can be used by non-programmers to define the PCOS environment and the BASIC environment. The commands are documented individually in the User Manual. An overview of the global commands follows.

GLOBAL COMMAND OVERVIEW

Like other PCOS commands, the global commands have standard or default values. However, once a global parameter has been set by a global command, that setting remains in the system during successive working sessions until it is reset. There are a few exceptions to the rule that the most recent setting is the default value. Exceptions are noted below.

1. SBASIC (sb) sets the BASIC programming environment.

Files	The number from 0 to 15 that can be opened concurrently.
Bytes	Amount available within 57K.
Windows	Preallocated memory space from 1 to 16.
Record size	The maximum record size from 1 to 4096 bytes available for random files.

   The effect of parameter settings on user memory allocation is as follows:

   - Each window after the first requires 108 bytes.
   - The file and record settings interact to require storage according to the formula:

     829 + F(578 + R)

     where F is the number of files that can be open and R is the maximum record size. Note that the maximum record is allocated for every file.

2. SCOMM (sc) sets the transmission environment for an RS-232-C communications port. For more details refer to the "I/O with External Peripherals User Guide."

3. SDEVICE (sd) displays the names of devices in the system and permits renaming of devices. A new device name may be assigned using a name consisting of 13 characters or less.

    Default Device Names

    ```
    prt:  -- PCOS Printer Driver
    cons: -- PCOS Console Driver (video and keyboard)
    com:  -- Standard RS232-C communication port
    com1: -- First RS232-C communication port on
             Twin Board
    com2: -- Second RS232-C communication port on
             Twin Board
    ieee: -- IEEE-488 driver
    ```

    Com1, com2, and ieee require optional hardware boards for implementation.

4. SFORM (sf) specifies type of printer, printer interface (serial or parallel), and printing format.

    The SFORM command is used to set the printing environment. It specifies the type of printer being used and the printing format, and allows the user to change parameters in the printer driver.

    SFORM parameters are:

    auto — This parameter specifies whether default values are used, or new values specified by SFORM (PSAVED or not). Auto OFF always returns to the default values.

    ptype — The ptype parameter specifies the type of printer or TRANSP (transparent mode). In transparent mode file contents are printed exactly as specified in the file irrespective of the type of printer.

    lines — The lines parameter specifies the number of lines to be printed on each page before automatic form feed. Zero implies that no form feed will be issued.

    spacing — This parameter specifies the number of inter-line spaces between printed lines.

    compress — This parameter specifies the style of the character. The width of the character can be specified and whether it is to be normal print or bold print.

    interface — This parameter specifies whether the printer is to be connected to the serial or parallel interface.

title This parameter defines an optional title to be printed at the top of each page.

5. SLANG (sl) selects the current keyboard from the national keyboards for various languages. This command can be used either to directly select one of the keyboards, or to display the menu of the available country configurations. The new keyboard can be changed using another SLANG command, or it can be made permanent using the PSAVE command.

6. SSYS (ss) sets the following system parameters:

   Date — Set date. Form depends on national keyboard.

   Time — Set time, hh:mm:ss.

   Disk Verify — Verify on, verify off. Verify on causes data that is written to diskette or the hard disk to be read back and checked.

   Extent Size — The number of sectors to be allocated to a file when more space is required, range 1 to 1087.

   Display — Select 16 lines of 64 characters each, or 25 lines of 80 characters each.

   Disk Time — Select the number of seconds the motor remains on following the last access to a diskette in a particular drive, range 1 to 30.

   Date and Time parameters are incremented until the system is physically reset or switched off, then revert to the default values. Changes to other parameters are valid until respecified or until the working session ends. With the exception of Disk Time, modified parameters can be permanently retained using PSAVE.

For additional information on the Set System global commands, refer to the PCOS Operating System User Guide.

PSAVE AND DEFAULT OPTIONS

Different versions of PCOS can be configured by using different settings of these global parameters and PSAVing the setting (with certain exception noted above). The configuration selected and PSAVED then becomes the current system configuration when that PCOS is booted. (PSAVE copies PCOS, and the new settings, to a new PCOS.file.) The user can, thus, have several system disks, each configured for a certain task, and containing the system settings needed, any PLOADED utilities which will be used in the work session.

INTERACTION OF BASIC AND GLOBAL COMMANDS

BASIC has the capability of executing PCOS commands, including the global commands, by using the EXEC or CALL verbs. Therefore, some questions arise which are answered below.

SBASIC

Executing SBASIC while in BASIC has no effect on the current BASIC parameter settings, which cannot be changed dynamically. The new BASIC parameters will take effect the next time BASIC is loaded.

SSYS (SET SYSTEM) AND DISPLAY MODE

Executing SSYS within BASIC to change the display mode causes problems. When BASIC is initialized, it reads the current display mode setting (16 lines of 64 characters or 25 lines of 80 characters). BASIC uses that information to control text and graphic display. If the display mode is changed while BASIC is operating, BASIC will attempt to operate in the prior mode while PCOS supports the changed mode. The results are unpredictable and sometimes unsatisfactory. If it is unavoidably necessary to change character spacing while in BASIC, use the following special case of the window statement:

```
W = WINDOW (0,0,vertical_spacing,horizontal_spacing)
```

22. CUSTOMIZING A PCOS SYSTEM

ABOUT THIS CHAPTER

This chapter reviews the software resources and utilities available to the system programmer for customizing PCOS to the requirements of particular installations and applications.

CONTENTS

SOFTWARE CONFIGURATION

PCOS is unique among operating systems because of the broad flexibility accorded the system programmer in configuring the system for particular installations and applications. Software configuration involves the elements discussed below. A summary at the end reviews the software resources and utilities available to the system programmer.

STANDARD INITIALIZATION

Standard initialization begins once the M20 system is powered on. Diagnostics are run, then a search for a bootable file begins. The first place checked for this file is on the hard disk drive 10, if available. If not found there, the drive 0 diskette is searched, then the drive 1 diskette. The bootable file must be the first file on a diskette. Once the file has been found and PCOS is in effect, the system proceeds in search of an optional initialization file in the following order:

NAME	DESCRIPTION
INIT.CMD	Any program in machine language, such as a PCOS command. The system loads the file into system memory, executes it, then purges it. The system would remain in the PCOS environment, unless the init routine brought up BASIC by using a Call User Command.
INIT.SAV	A program with the same characteristics as an INIT.CMD program and loaded only if INIT.CMD does not exist. The system retains it after execution for the duration of the working session.
INIT.BAS	Any BASIC program, loaded if neither INIT.CMD nor INIT.SAV exists. The system loads appropriate utilities (BASIC.CMD and BASIC.ABS), enters the BASIC environment and executes the program. The system would remain in the BASIC environment, unless the init routine returned to PCOS with a SYSTEM command.

The standard initialization process also happens after a logical reset of the system and after execution of PSAVE or PRUN command. With PSAVE, PCOS saves the current configuration of the operating system on a file, and then reboots the system, using that file (if on drive 0:). Following a PRUN command, the system searches the drives for the file (in any location) specified by the PRUN command. In these cases, the startup diagnostics are not run and therefore the non-standard interventions described next cannot be done. PRUN, however, can cause a different version of PCOS to be initialized. PSAVE and PRUN are discussed later in this section.

NON-STANDARD INITIALIZATION

Non-standard initialization, which permits operation in unconventional modes or for special purposes, is substituted for the standard procedure by pressing one of five keys while the diagnostic routines are being performed. /L/ and /D/ are used to cause looping during startup diagnostics. /F/, /B/, and /S/ cause initialization to proceed on an alternate path. They are keyed during startup diagnostics, then take effect later. They have the following effect:

/F/ First examine the diskette drive, rather than the hard-disk drive, for a bootable file.

/B/ Enter the BASIC command mode without execution of an initialization file.

/S/ Enter the PCOS command mode without execution of an initialization file.

CUSTOMIZING THE KEYBOARD

CKEY

The CKEY command is used to change the value of a key or to set the shift lock for the alphanumeric and/or numeric keypads. Use /CTRL/ or /COMMAND/ in conjunction with assignment of the new value to avoid cancelling the original function of the key. The new value is retained until changed by another CKEY command, replaced as part of the conversion table by the SLANG command, or disabled by the end of the working session; it can be retained permanently by using the PSAVE command. For further information, see the discussion in the "Keyboard Driver" section of Part 2 or the PCOS Operating System User Guide.

PKEY

PKEY can be used to assign a string of characters to a single key. In this way, single keystroke commands can be implemented, or data strings that are often used can be entered with a keystroke. Values of keys (except /SHIFT/, /CTRL/, /COMMAND/, /RESET/, /S1/, /S2/, and /CR/ can changed. Use /CTRL/ or /COMMAND/ in conjunction with assignment of the new value to avoid cancelling the original function of the key. Note that the new value is retained for the duration of the working session (or permanently by using PSAVE) and reduces the amount of user memory.

GENERAL

Values assigned using CKEY and PKEY remain effective when another environment or application program is entered. Care must be taken, therefore, to avoid disabling functions that may be required in the other environment or program.

CKEY and PKEY are discussed in the "Keyboard Driver" section of Part 2. For more detailed information, see the PCOS Operating System User Guide.

CUSTOMIZING FONT CHARACTERS

To change shapes of existing characters and/or add characters for display, RFONT and WFONT are used. This feature is useful for incorporating non-Roman characters and/or small graphic symbols for text or games, but full-screen illustrations are not accommodated. The specialized font sets can be printed by all the dot-matrix or dot-matrix-type printers, but not by the daisy-wheel printer. The font sets may be stored on diskettes for use each time particular characters or symbols are required, or may be incorporated in the system by using PSAVE.

RFONT and WFONT are discussed in the "Keyboard Driver" section of Part 2. For further information, see the PCOS Operating System User Guide.

SET SYSTEM GLOBAL COMMANDS

The Set System global commands can be used by non-programmers to define the PCOS environment and the BASIC environment. These global commands are discussed in the "PCOS Environment and Global Commands" section of Part 3. They are described briefly below:

SBASIC sets the BASIC programming environment.

SCOMM sets the transmission environment for an RS232-C communications port.

SDEVICE permits renaming system device names.

SFORM specifies the printer interface, type of printer, and printing format.

SLANG selects the national keyboard.

SSYS specifies fundamental PCOS parameters, including date and time settings, display mode, and disk control parameters.

INCORPORATING TRANSIENT COMMANDS

PLOAD and PSAVE are used to tailor the operating system to particular installations or applications. Transient commands (all commands except PLOAD, PUNLOAD, and LTERM) with the extension .CMD are loaded into memory only long enough for execution. Transient commands with the extension .SAV are loaded into memory, executed, and retained for the duration of the working session. The PLOAD command is used to retain transient commands in memory, without initial execution, even after the system diskette is removed, for the duration of the working session.

SAVING THE RECONFIGURED SYSTEM

If the reconfigured operating system is required for future use, incorporating desired PLOADED commands, PKEYED definitions, and global-command parameters it can be stored permanently on diskette or hard disk by using the PSAVE command.

PSAVE

The PSAVE utility permits customizing of the operating system under a specific environment. The customized PCOS system configuration is saved on disk and can be used at any time by using the PRUN command or the bootstrap procedure. The PSAVE utility can save the system under a special name to be used by a PRUN command or under the default name PCOS.SAV.

If the system is saved without a name, the file will be made a bootable file and will be accessed by its location rather than name. It cannot be password protected. If the system is saved by name, it can be password protected. PRUN will be used to access it, and it can be located anywhere.

THE PCOS.SAV STANDARD FILE

On a distributed PCOS diskette, the operating system is stored in the first file, PCOS.SAV. (The filename is not important for the boot process.) This standard file is the minimum operating system required to run PCOS. New utilities can be added and some system parameters can be changed using PSAVE, but no portion of the standard PCOS.SAV file can be deleted.

THE PSAVE PROCEDURE

The action of the PSAVE command is as follows: all memory is checked and, when a block contains information, data are stored in a file with the block address and the bytes count. PSAVE creates a bootable file including all memory, parameters, configuration, and utilities loaded in memory and saved on disk.

PSAVE AND MEMORY EXPANSION

A PCOS system that is PSAVED on an M20 with expansion memory may not boot up on an M20 with less memory. During PCOS startup, the PCOS kernel looks at memory configuration information supplied by the ROM diagnostics and then configures memory according to the requirements of the fundamental PCOS and the additional saved material. If memory capacity is not sufficient, PCOS will give an error message and die.

BOOT BLOCK UPDATING

In addition to writing the PCOS file on disk, PSAVE updates the boot block to specify the address of the file. In this way the new file is automatically loaded as the PCOS file.

It is important to note the following:

- if a disk contains several PCOS files, only the file last PSAVED is automatically loaded by the bootstrap
- copying a file using the FCOPY utility does not update the boot block and the file just copied is not automatically booted unless it is the first file on a new diskette
- on hard disk, PSAVE must always be used to create a bootable file because the first file is not automatically booted if bad sectors are present.

The user can save an automatically booted PCOS file with PSAVE or copy the file to another disk, saving a copy of a PCOS file that will not be automatically booted.

For more information about using this command, refer to the PCOS Operating System User Guide.

A PCOS BOOTABLE FILE

The structure of PCOS files on disk permits the bootstrap to load any configuration and even a different operating system. A bootable file is divided into records, each comprising a header and a block of data. The header contains information such as the address in which data must be loaded and a count of data bytes. After the last record is loaded, the bootstrap begins execution of the code just loaded in a particular memory location. The bootstrap is a generalized procedure, and there are no constraints about the contents of the PCOS file.

BOOTSTRAP BACKGROUND INFORMATION

The bootstrap, a small program contained in ROM, is started automatically after a reset. The bootstrap examines the drive for a bootable file and, if found, loads it and starts execution.

BOOT ROM 1.0

In Boot ROM 1.0, the first release, the bootstrap searches the first sector of track 0, side 1, for a bootable file. This search is made first for the drive 0 and then for the drive 1. If a nonbootable file is discovered in this position, an error message is returned. Note that PCOS must always occupy the first position on a system diskette.

BOOT ROM 2.0

The bootstrap release Boot ROM 2.0 uses a specific location on the boot block as the address of the first sector of a bootable file. Therefore, under the correct conditions, PCOS can reside on any part of the disk. Only PCOS 2.0 and subsequent releases update or read the boot-block address.

THE PRUN COMMAND

The PRUN command can be used to boot a PCOS file that would not be automatically loaded by the bootstrap. This command is equivalent to pressing the reset key except that a filename is specified. PRUN looks for this filename on disk and starts the standard boot procedure with this file. The file can be anywhere, but must be bootable.

To boot the PSAVED PCOS file, PRUN opens the file in the standard way, and therefore the file can be protected with the standard PCOS features (volume password, file password, etc.). This is not possible for an automatically booted PCOS file because the bootstrap is a direct routine bypassing the file system.

If PRUN is used without specifying a filename, it will boot PCOS from either diskette drive or the hard disk, so long as PCOS is accessible through the boot block in the usual manner.

SUMMARY

The simplest tools available to the analyst or programmer for customizing versions of PCOS are the PSAVE and PRUN commands. These can be used with PLOAD and the Set System global commands to quickly and easily configure versions of PCOS for particular applications. Within a particular installation, versions of PCOS and associated commands can be made for the use of a data-entry group, an accounting group, and so on.

The meanings of keyboard keys can be changed or enhanced by use of the CKEY and PKEY utilities. PKEY can be used to develop one-key command strings or to provide an easy way to enter often-used strings. The RFONT and WFONT utilities allow development of new fonts for display and for printing.

The keyboard and font utilities, used in conjunction, allow the development of new keyboards including non-Roman keyboards. They also allow the development of special characters for display or printing that could be useful for particular applications.

The keyboard and font customizations can be saved using PSAVE so that they are available either throughout all systems in an installation or only for particular applications according to the requirements of the installation.

The discussion of bootable files, details of PSAVE actions, and the standard and non-standard initialization process shows how alternate paths can be made available, so that different versions of PCOS can be brought up for standard use or special use.

Finally, the use of init files gives the system programmer many possibilities for configuring a specialized PCOS. Init files in BASIC or assembly language allow bringing the system up in either the BASIC or PCOS environment, and if desired, within a particular program. Such init files and associated programs could be customized for use in particular application environments, such as data entry, accounting, graphic displays for interactive reviewing, etc.

Remember that BASIC allows calling PCOS commands using the EXEC or CALL verbs. Similarly, in assembly language the Call User (77) system call can invoke a PCOS command. Either of the methods can be used to set global parameters with the Set System commands.

Some further suggestions on the use of init commands:

Administration — The init routine can read a text file (which is updated regularly) and display that information for any user who starts up the system. The init file could also ask for user identification, perhaps even requiring a password, and then bring the user up in a particular version of PCOS or a particular program depending on the response.

IEEE	Bring a system up with the appropriate IEEE commands and do data-logging of devices on the IEEE bus. The use of an appropriate init file provides consistency in data logging at various times by different people.
Communications	Bring up two systems set for RS232 communications with the appropriate device rewriting parameters. One unit can send commands to the other and receive data from it. The use of pre-set and tested init files simplifies co-ordination between the sites.

23. DATA PASSING MECHANISM

ABOUT THIS CHAPTER

This chapter describes how information is passed among the PCOS internal elements and between PCOS and its supported languages. Included is information on the stack, the passing of integer and floating point numbers, strings, null values, and return address.

CONTENTS

OVERVIEW

PCOS uses one general method for passing information among its internal elements and between PCOS and its supported languages. The method uses the stack, and can pass numbers, both integer and floating point, strings, null values, and a return address. The method is that used by Microsoft for BASIC and other languages, and has been generalized for use by PCOS and its supported languages.

USE OF THE STACK

Data items, also called parameters, are pushed onto the stack by the sending routine and popped from the stack by the receiving routine. They are received in reverse order. If the sending routines pushes items 1, 2, and 3 the receiving routine will pop 3, 2, 1. The top of the stack (last pushed, first popped) is a count of data items on the stack. After the items, if appropriate, is a return address (first pushed, last popped). The count is a word, and the other entries are long words.

The return address is not included in the count. Its presence or absence is a matter of convention agreed upon for the sending routine and receiving routine. If it is present, the receiving routine will finish its processing by issuing a return instruction using that address. If the sending routine calls the receiving routine, the return address will be placed on top of the stack by the call. If the sending routine passes control directly, without using a call, there would be no automatic return address.

The receiving routine pops off one word, which is "n", the number of entries. Each entry is a long word, so the receiving routine must next pop "n" long words off the stack. The receiving routine must be careful to pop exactly "n" long words, and not rely on an expected number, because the sending routine may push an incorrect or unexpected number of entries.

There is no inherent limit to the number of data items that can be placed on the stack for transmission, except practical usage. The command line interpreter enforces a limit of 20, and therefore command routines never receive more than 20 items.

Here is an example of the stack upon entry to a receiving routine.

```
        ---------------------------------
SP ->   |   return address              |       (long)
        ---------------------------------
        |       2       |               |       (word)
        ---------------------------------
        |   second item pushed          |       (long)
        ---------------------------------
        |   first item pushed           |       (long)
        ---------------------------------
```

FORMAT OF DATA ITEMS

The long-word data item entry contains an address segment pointer, an address offset, and a parameter type descriptor. The type descriptor is ORed with the segment pointer. As an illustration,

```
-----------------       ----------------
| seg  |  type |        | addr offset  |
-----------------       ----------------
```

For example,

```
-----------------       ----------------
|  86  |   03  |        |    0C00      |
-----------------       ----------------
```

Where the pointer is <<6>>0C00 and the parameter type is 3.

The parameter type must be extracted from the entry, by clearing the low-byte of the segment, before the pointer can be used. In this example, the pointer is 8600 0C00.

The format of data item depends upon the parameter type. Parameter types are:

0	null
2	integer
x	long integer*
3	string
4	single-precision floating point
8	double-precision floating point

* Long integer is not currently supported.

NULL PARAMETERS AND DEFAULT VALUES

Null (or nil) parameters are put on the stack when the convention followed by the calling routine and receiving routine allows the receiving routine to substitute standard or default values.

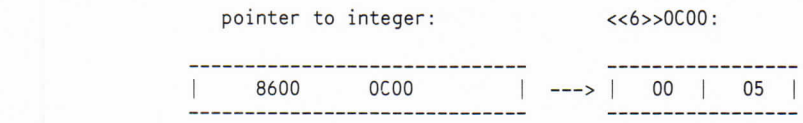
The null parameter keeps the same two-word format of the other entry types. Only the type byte matters. The other three bytes are filled with FFs by the sending routine and ignored by the receiving routine.

null parameter:

```
-----------          -----------
| FF | 00 |          | FF | FF |
-----------          -----------
```

INTEGER PARAMETERS

The pointer points to a 2-byte array containing the value. This array may or may not be at an even boundary address, so the value must be loaded into registers one byte at a time. For example, if the pointer entry is 8602 0C00, then after the type is extracted the pointer is 8600 0C00.

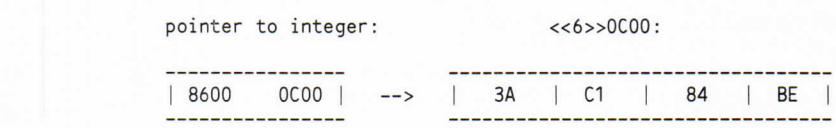


In this example, the integer value is 5.

LONG INTEGER PARAMETER

This parameter is not currently supported, and the value is used only internally in PASCAL programs. The following information is given for planning purposes. The data passing mechanism could be used if a parameter type number were assigned for long integer.

The pointer points to a 4-byte array containing the value. This array may or may not be at an even boundary address, so the value must be loaded into registers one byte at a time. For example, if the pointer entry is 860x 0C00, then after the type is extracted the pointer is 8600 0C00.

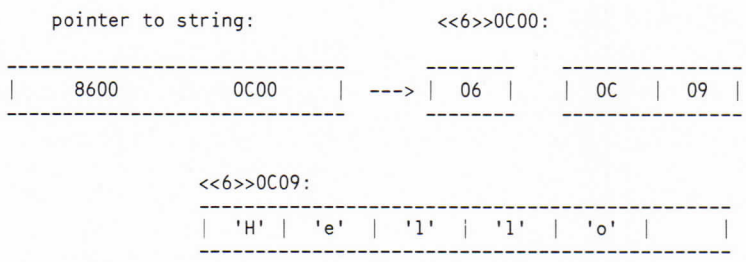


In this example, the long integer value is %3AC184BE.

STRING PARAMETER

The pointer points to a 3-byte array where the first byte contains the length of the string, and the other two are the integer offset of the pointer to the string. (The memory segment is assumed to be the same as before. If not, a rather obscure error will result.) This offset may or may not be at an even boundary address, so it must be loaded into registers one byte at a time.

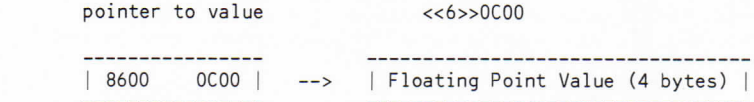
For example, if the pointer entry is 8603 0C00, after the type is extracted the pointer is 8600 0C00.



In this example, the string length is 6.

SINGLE-PRECISION FLOATING POINT PARAMETER

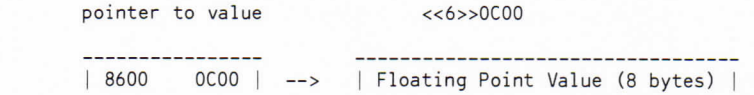
The pointer points to a four-byte floating point value. The value is not necessarily on an even boundary, and should be loaded a byte at a time.



The representation of a single-precision floating point number is explained in the "Language Support" section.

DOUBLE-PRECISION FLOATING POINT PARAMETER

The pointer points to an eight-byte floating point value. The value is not necessarily on an even boundary, and should be loaded a byte at a time.



The representation of a double-precision floating point number is explained in the "Language Support" section.

SEGMENT BOUNDARIES AND POINTERS

The numerical parameters consist of a pointer to a value, and the string parameter consists of a pointer to a descriptor which includes a pointer to the actual string. In all these cases, it is assumed that the value is in the same memory segment as the pointer. As a practical matter, only the string parameter is likely to present difficulties because of its three-part structure. When constructing the string parameter, it may be necessary to move the string into a new segment in order to have all three parts of its description in the target segment.

24. LANGUAGE SUPPORT

ABOUT THIS CHAPTER

This chapter provides a fundamental overview of the support PCOS provides for high-level languages, some of which can be used for assembly-language programming. Topics include data passing, calling on internal PCOS resources, memory allocation, and internal representation of numbers.

CONTENTS

OVERVIEW

This section provides a fundamental overview of the support PCOS provides for high-level languages, some of which can be used for assembly-language programming. High-level languages currently supported are BASIC and PASCAL. Topics include data passing, calling on internal PCOS resources, memory allocation, and internal representation of numbers.

DATA PASSING

All data passing between PCOS and the languages it supports, in both directions, is done using the scheme described in the "Data Passing" section. This general approach is used throughout the PCOS system.

The data passing scheme allows passing of numbers, strings, null values, and a return address. A null value can be passed where a convention between the sending and the receiving routine allows the receiving routine to substitute a standard or default value. Numbers can be integers or floating point values. The internal representation of numbers is described in this section.

Within the language implementations, and within many PCOS elements, 32-bit pointers are used so that all of memory can be treated alike. The data passing mechanism uses 16-bit pointers and has the implied understanding that the items pointed to are in the same segment as the pointer.

AVAILABLE REPRESENTATIONS

The following table shows the internal data representations available in the languages supported by PCOS. "X" means available, "-" means not available.

Data Representation Usage

	Null	Integer	Long Integer	Single Float	Double Float	String
BASIC	x	x	-	x	x	x
PASCAL	x	x	x	x	x	x

Assembly Language can use all data representations, but requires the support of appropriate mathematics routines.

LONG INTEGER EXCEPTION

Internally, PASCAL and many other commercial compilers pass numerical parameters as ASCII strings. Compilers used with PCOS receive parameters from PCOS, and send them, using the formats and mechanism explained in the "Data Passing" section. Most programmers never see the internal format. The one effect this difference has on PCOS is that the data passing mechanism does not currently support the passing of long integers, which are used only in PASCAL. The PCOS data passing mechanism could accommodate long integers by passing a pointer to the four-byte value. All that would be necessary is to assign a type number for long integers.

SYSTEM CALLS vs MASTER TABLE

All use of system resources by a supported language is done via system calls. Languages do not use the PCOS Master Table, which is subject to change. By using system calls, languages remain independent of changes in PCOS releases, and so do application programs written in those languages. In particular, languages must use appropriate system calls for using internal system resources, such as system memory, and for input/output operations. The system calls for data manipulation, string handling, etc., are available for use but not required.

INTERNAL SYSTEM RESOURCES

Supported languages use the storage allocation calls to obtain system memory from the heap and to release it back to the system. Once the language has obtained memory space, the language internal routines may manipulate and configure that memory as desired.

Interaction with PCOS itself is done using the system management calls. This includes setting or reading the real time clock using the time and date calls.

INPUT/OUTPUT

All input/output operations are done using system calls. These calls include the bytestream I/O group, file management and disk I/O, and various special I/O control commands.

PCOS AND LANGUAGE MEMORY ALLOCATION

Conceptual overviews of memory handling for BASIC and for compiled languages are given below. BASIC is of special interest, because the implementations of BASIC and PCOS are very closely coupled, and also because BASIC, being an interpreted language, must be able to call upon system resources in a dynamic and interactive manner. BASIC may need to allocate memory dynamically.

PASCAL and assembled assembly language are treated alike as "compiled languages." After PASCAL has been compiled, it is equivalent to assembled assembly language.

In theory, PASCAL and assembly language programs could be allocated fixed memory space at execution time, because their memory requirements should be known. In practice, this is not quite true. They can begin with a fixed allocation, but if the routines interact dynamically with a user, the routines may need to dynamically request and release memory space.

BASIC

BASIC was originally designed to work in 64 Kb of memory, with approximately 36 Kb for the interpreter and 28 Kb for user memory. In the Olivetti enhancement of BASIC for PCOS, the BASIC interpreter is loaded into CS1 and the user area is in DS2. Code execution takes place in the interpreter area (CS1) and the user program and its associated variables can be treated as data. The user area contains some overhead tables and other necessary information, so the user has available approximately 57 Kb (the actual amount is set by the Set BASIC (SBASIC) global command.

The fundamental BASIC program, that is, the interpreter and its supporting routines, are loaded into CS1. They then use the rest of the memory assigned for BASIC, allocating space as required to support user interaction or the running of a BASIC program.

For example, when a user is programming in BASIC, the interpreter has three areas reserved in DS2: one for program statements, one for program variables, and one for strings. Each statement the user enters is stored in the statement area. Inserting, deleting, or modifying statements is done using this area for storage. Whenever a statement creates a variable, space is allocated in the variable area for it, with overhead information so that references from the statement area can find it. Strings are created and handled similarly. Deleting or modifying variables and strings affects these areas.

The size of these areas is dynamically modified when necessary. When a defined BASIC program is loaded, the associated variable and string space is created at the same time. When a BASIC program is run, the assigned space is used, and when necessary, BASIC internally allocates space to interact with user requests and actions at run-time. The allocations of space in the user area for program statements and variables start at the upper and lower limits and grow towards the middle of the area. Therefore, the user area size cannot be changed while BASIC is active.

This data management is done by BASIC, and the role of PCOS is merely to provide heap space originally, to supply more if requested by BASIC, and to de-allocate heap space when BASIC is finished.

COMPILED LANGUAGES

Compiled PASCAL programs, other compiled languages, and assembled assembly-language programs are equivalent. Modules of these languages are processed by the linking loader and assigned to memory locations. Space required by a module is based on its actual code size and the data and buffer space it has defined.

Generally, compiled code is relocatable, and the linker can assign space based on its own calculations. Code and data are first assigned within segment 6. Other segments can be assigned as necessary. Segment 2 is assigned only as a last resort.

In many cases, the exact memory requirements are known and assigned before run-time. The program makes use of its assigned space independent of PCOS memory management. However, a program may have need to dynamically allocate space based on user interaction or on processing requirements that are dependent on external factors, such as handling a variable number of files. In such cases, the language support routines call on PCOS memory management functions to allocate and de-allocate space.

NUMERICAL REPRESENTATION

The numerical representation for integers and floating point values, which is described below, was designed and implemented by Microsoft. It is used in mathematical routines for BASIC, and has been extended to other high-level languages. Each language uses a different mathematics package, but all packages have a common numerical representation.

The floating point representation is not an exact implementation of the proposed IEEE standard, but functions in a similar fashion and can interface with IEEE-standard routines with proper safeguards. The differences are explained in this discussion.

Four types of numbers are represented: integer, long integer, single-precision floating point, and double-precision floating point. The supported range for each type is as follows:

Integer:	-32,768 to +32,767
Long Integer:	-2,147,483,647 to +2,147,483,647
Single:	+/- 1.1754944E -38 to +/- 3.4028237E +38 and 0
Double:	+/- 2.2250738585072D -308 to +/- 1.7976931348623 1D +308 and 0

In theory, the long integer would have a negative limit that is one greater in absolute value than its positive limit. However, PASCAL has the negative limit of -2,147,483,647.

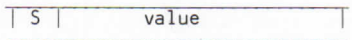
INTERNAL REPRESENTATION

The floating point representation described here is standard, but some mathematics packages expand the format during calculation which allows higher precision. Among the PCOS Languages, BASIC does this but not PASCAL.

For floating point, the exponent is biased by half its total range. Negative values are in the bottom half of the exponent range, positive values in the top half. The fractional part is normalized to have a leading 1 bit with an implied decimal point following it. Because the one and its decimal point are always present in the value, they are implied but not represented. (The value of zero is the only exception, and consists of all zeros.)

REPRESENTATION LAYOUTS

Integer: (2 bytes)



S = Sign Bit
Value = 15 Bits

Negative numbers are in two's complement form.

Long Integer: (4 bytes)

S = Sign Bit
Value = 31 Bits

Negative numbers are in two's complement form. The two words are treated as one 32-bit word, using the Z8000 32-bit instructions.

Single Precision: (4 bytes)

S = Sign Bit
E = 8 bit exponent biased by 127
M0 = Most significant 7 bits of Mantissa
M1 = Least significant 16 bits of Mantissa

M0 and M1 combined are a 23 bit mantissa which follows an implied leading 1 and decimal point.

Double Precision: (8 bytes)

S = Sign Bit
E = 11 bit exponent biased by 1023
M0 = Most significant 4 bits of Mantissa
M1 = Next significant 16 bits of Mantissa
M2 = Next significant 16 bits of Mantissa
M3 = Least significant 16 bits of Mantissa

M0, M1, M2 and M3 combined are a 52 bit mantissa which follows an implied leading 1 and decimal point.

EXPONENT BIASING

In single-precision, an exponent of 127 equals an unbiased zero exponent. The negative exponents (unbiased) from -128 through -1 are represented by 0 through 126. In double-precision, 1023 represents zero, and the exponents from 0 through 1022 represent -1024 through -1. (However, there are limits on actual use of the largest and smallest exponents. See the discussion on IEEE limitations, below.)

ROUNDING

In floating point formats the low order bits of the fraction serve only as guard bits in calculations and are not intended to be used in the final result. Though these bits may be kept in intermediate results, rounding of at least the low-order four bits should take place before assigning the final number or printing.

PRECISION

The precision of calculation varies among mathematics packages. Some packages extend the numerical representations shown with additional bits during calculation and then round to a lesser number of bits after. Here is a table of floating-point precision in bits and digits (bits/digits), as implemented for Z8000 BASIC:

Variable	Actual	Effective	IEEE stored	Printed
Single	32/9.6	28/8.4	24/7.2	19.9/6
Double	64/19.3	60/18.1	53/15.95	49.8/15

Input numeric constants are rounded to 28 and 58 bits. Output is rounded to 6 and 15 digits. PAK and UNPAK routines convert between internal and external format; for IEEE, this involves rounding to 24 and 53 bits. The scientific functions are calculated to a precision in excess of 8.25 digits, except for ATN at 7.69.

IEEE STANDARD LIMITATIONS

The math package routines are meant to interface to IEEE standard routines. However, some IEEE features such as -0, +/- infinity, and not-a-number values are NOT supported. To meet these constraints, the internal form must obey certain restrictions:

1. The maximum exponent value (FF or 7FF hexadecimal) is never used. Machine infinity is an exponent of FE or 7FE and a mantissa filled with binary ones.

2. A value of zero is represented, by convention, as all zeros. When the exponent is zero, the sign and mantissa are also zero.

The BASIC mathematics package preserves these assumptions, if initially true, but the UNPAK routines do not check the validity of incoming numbers. The PASCAL mathematics package also preserves these assumptions. The only place where problems can arise for either language is when reading values from external files written in another form.

25. INSTALLING PCOS ON A HARD DISK

ABOUT THIS CHAPTER

This chapter describes the procedure for installing PCOS or updating PCOS on a hard disk M20 system. Information is given on how to maintain prior PCOS reconfigurations in the new PCOS.

CONTENTS

OVERVIEW

This section describes the procedure for installing PCOS on a hard disk M20 system. Both the first installation and updating the hard disk drive with a new version of PCOS are covered. Information is given on how to maintain prior PCOS reconfigurations in the new PCOS.

Hardware considerations for installing the hard disk are not discussed. Instructions are provided with the drive.

The hard disk comes with a PCOS system diskette which has all routines necessary to make use of it. The installation procedure is simple and makes use of existing PCOS utilities.

Before installing, make a note of the current CKEY, PKEY, and BKEYBOARD values that you wish to have on the new PCOS. They will need to be re-entered.

NEW INSTALLATION

The following procedure installs PCOS on a new hard disk system. The hard disk identifier is 10.

LOADING PCOS INTO THE SYSTEM

To format the hard disk you must first load PCOS from the diskette you received with the system. Do this by inserting the PCOS system diskette in the floppy disk drive and turning on the power. Press the "f" key on the keyboard before the two beeps are heard, to cause the Bootstrap ROM to load PCOS from the floppy disk rather than from the hard disk.

FORMATTING THE HARD DISK DRIVE

After PCOS is loaded, type

vf 10:

This tells PCOS to run the VFORMAT program and to format drive number 10. VFORMAT will display each cylinder on the hard disk that is being formatted.

LOADING PCOS ONTO THE HARD DISK

When the VFORMAT program is complete (as signified by the message, Formatting Complete), the next step is to put a bootstrap file on the hard disk. Do this by typing

ps 10:

which invokes the PSAVE utility to install the current version of PCOS on the hard disk drive. When the PSAVE utility is complete, it will

automatically re-boot the PCOS on the hard disk.

COPY PCOS COMMANDS ONTO HARD DISK

The next step is to to copy all the PCOS commands to the hard disk drive. This is done by using the PCOS FCOPY command. To copy all the floppy disk files to the hard disk, type

```
fc 0:* 10:
```

UPDATE INSTALLATION

If the hard disk drive has been through the initial installation described above, all that is required is to place the latest version of PCOS and the utilities on the hard disk. For this the following procedure should be used.

Boot the hard disk system and then place the new PCOS diskette in the floppy disk drive. Type

```
fc %f 0:* 10:
```

FCOPY will copy all the files from the floppy diskette to the hard disk. The "%f" (force) option will cause FCOPY to copy the files to the hard disk even though the files may already exist on the hard disk. When FCOPY is done, so is the installation.

Notice that the hard disk is NOT formatted in this procedure. Formatting would destroy all files on the hard disk. In this update procedure, the only files affected are those commands and utilities that may have existed on the hard disk until replaced by new files copied from the diskette.

CONFIGURING THE NEW PCOS

The newly installed PCOS will have its own settings for the Set System global commands. It is a good idea to use these utilities to review the current settings and make any appropriate changes. Current font changes made using RFONT should remain untouched in their files, but will need to be attached to the new PCOS with WFONT. CKEY, PKEY, and BKEYBOARD change values will have to be reassigned.

26. ASCII

ABOUT THIS CHAPTER

This chapter describes the ASCII standard for information interchange as used in the PCOS system and gives information on its general use and modification.

CONTENTS

OVERVIEW

This section describes the ASCII standard for information interchange as used in the PCOS system and gives information on its general use and modification.

BACKGROUND

ASCII, American Standard Code for Information Interchange, was a national standard code in the United States and is now an international standard code. Originally comprised of 128 seven-bit values, ASCII is now generally encountered in eight-bit form, having 256 possible values.

The high-order eight-bit is not defined in standard ASCII. It can be used for parity or can be set to zero or to one for all characters. In eight-bit form, the 128 ASCII characters each have two codes. However, the PCOS utility CKEY may be used to assign other values for the range of codes from 128 - 255, (or for standard ASCII codes).

ASCII and PCOS

The values for 0 through 127 are shown. Unless modified, values for 128 through 255 correspond in the same order. That is, 128 is Null, 255 is Delete.

The displayable portion of the ASCII table shows the M20 keyboard values used for the United States. Certain displayable characters are different on other national keyboards, and are noted in the table.

PCOS allows reconfigurating the keyboard with CKEY and PKEY and developing new fonts for display and printing with RFONT and WFONT. These utilities allow the development of non-ASCII and non-Roman character sets. However, the internal codes for characters are in the ASCII range. ASCII is widely used as a standard for interchange among computer systems, and the M20 system can exchange ASCII with other systems. For reconfigured PCOS systems certain practical difficulties arise. These difficulties can be ameliorated by use of PSAVE, which allows the retention of standard ASCII systems while using non-ASCII systems. Also, care in redefining systems can help preserve interchange difficulties, especially if the character codes in the range 0 - 127 are not changed.

ASCII CONTROL CHARACTERS

These codes are used for device and telecommunications control and for providing device functions.

Decimal	Hexadecimal	Code	Comment
0	0	NUL	Null
1	1	SOH	Start of Header
2	2	STX	Start of Text
3	3	ETX	End of Text
4	4	EOT	End of Transmission
5	5	ENQ	Enquiry
6	6	ACK	Acknowledge
7	7	BEL	Bell
8	8	BS	Backspace
9	9	HT	Horizontal Tab
10	A	LF	Linefeed
11	B	VT	Vertical Tab
12	C	FF	Formfeed
13	D	CR	Carriage Return
14	E	SO	Shift Out
15	F	SI	Shift In
16	10	DLE	Data Link Escape
17	11	DC1	Device Control 1
18	12	DC2	Device Control 2
19	13	DC3	Device Control 3
20	14	DC4	Device Control 4
21	15	NAK	Negative Acknowledge
22	16	SYN	Synchronous Idle
23	17	ETB	End of Transmission Block
24	18	CAN	Cancel
25	19	EM	End of Message
26	1A	SUB	Substitute
27	1B	ESC	Escape
28	1C	FS	Field Separator
29	1D	GS	Group Separator
30	1E	RS	Record Separator
31	1F	US	Unit Separator

Table 26-1 ASCII Control Characters

DISPLAYABLE ASCII CHARACTERS

Ninety-five of the following characters are displayable, that is, can be shown on the display screen or printed. The last character is the Delete character, and is not usually displayed.

Decimal	Hexadecimal	Code	Comment
32	20	SPACE	Blank
33	21	!	
34	22	"	
35	23	#	Different on National Keyboards
36	24	$	Different on National Keyboards
37	25	%	
38	26	&	
39	27	'	Single quote
40	28	(	
41	29	)	
42	2A	*	
43	2B	+	
44	2C	,	Comma
45	2D	-	
46	2E	.	
47	2F	/	
48	30	0	
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	
57	39	9	
58	3A	:	
59	3B	;	
60	3C	<	
61	3D	=	
62	3E	>	
63	3F	?	
64	40	@	Different on National Keyboards
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	

72	48	H	
73	49	I	
74	4A	J	
75	4B	K	
76	4C	L	
77	4D	M	
78	4E	N	
79	4F	O	
80	50	P	
81	51	Q	
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	Different on National Keyboards
92	5C		Different on National Keyboards
93	5D	]	Different on National Keyboards
94	5E	^	
95	5F		
96	60	`	Different on National Keyboards
97	61	a	
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	
103	67	g	
104	68	h	
105	69	i	
106	6A	j	
107	6B	k	
108	6C	l	
109	6D	m	
110	6E	n	
111	6F	o	
112	70	p	
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	

119	77	w	
120	78	x	
121	79	y	
122	7A	z	
123	7B	{	Different on National Keyboards
124	7C	|	Different on National Keyboards
125	7D	}	Different on National Keyboards
126	7E	±	Different on National Keyboards
127	7F	DEL	

Table 26-2 Displayable ASCII Characters

27. PCOS ERROR CODES

ABOUT THIS CHAPTER

This chapter provides a comprehensive table of error codes with an indication of which are used by PCOS, by BASIC, and by both. There is also a cross-reference table showing the differences between the 3.0 error codes and those of earlier versions.

CONTENTS

OVERVIEW

The error codes listed in the first table are for PCOS 3.0 and later versions. The table provides an indication of which codes are used by PCOS, by BASIC, and by both.

BASIC and PCOS display errors differently. BASIC gives the description only and PCOS the number only, unless the EPRINT utility is active. EPRINT causes PCOS to display both number and description.

Error codes were somewhat different in earlier versions of PCOS. A cross-reference table follows showing these differences.

Suggestions to the programmer on the use of error codes are given at the end of this section.

COMPREHENSIVE PCOS 3.0 ERRORS

Code	Meaning	BASIC/PCOS
0	No error	B P
1	NEXT without For	B
2	Syntax error	B P
3	RETURN without GOSUB/Illegal Function	B
4	Out of DATA	B
5	Illegal function call	B P
6	Overflow	B P
7	Out of memory	B P
8	Undefined line number	B
9	Subscript out of range	B P
10	Duplicate Definition	P
11	Division by zero	B
12	Illegal direct	B
13	Type mismatch	B P
14	Out of string space	B
15	String too long	B P
16	String formula too complex	B
17	Can't continue	B
18	Undefined user function	B P
19	No RESUME	B
20	RESUME without error	B
21	Unprintable error	B
22	Missing operand	B P
23	Line buffer overflow	B P
26	FOR Without NEXT	B
29	WHILE without WEND	B
30	WEND without WHILE	B
31	IEEE: Invalid talker/listener address	B

32	IEEE: talker = listener address	B	
33	IEEE: Unprintable error	B	
34	IEEE: Board not present	B	
35	Window not open	B	P
36	Unable to create window	B	P
37	Invalid action-verb	B	
38	Parameter out of range	B	P
39	Too many dimensions	B	
50	FIELD overflow	B	
51	Internal error	B	
52	Bad file number	B	
53	File not found	B	P
54	Bad file mode	B	P
55	File already open	B	P
57	Disk I/O error	B	P
58	File already exists	B	P
59	Disk type mismatch		P
60	Disk not initialized		P
61	Disk filled	B	P
62	End of file	B	P
63	Invalid record number	B	P
64	Invalid file name	B	P
66	Direct statement in file	B	
67	Too many files	B	P
68	Internal error	B	P
69	Volume name not found	B	P
70	Rename error	B	P
71	Invalid volume number	B	P
72	Volume not enabled	B	P
73	Invalid Password	B	P
74	Illegal disk change	B	P
75	Write Protected File	B	P
76	Error in Parameter	B	P
77	Invalid number of parameters	B	P
78	File not OPEN	B	P
79	Printer error	B	P
80	Copy Protected File		P
81	Paper Empty		P
82	Printer Fault		P
92	Command not found		P
93	Control C from console		P
99	Bad load file		P
101	Error in time or date		P
108	Call User error		P
110	Time Out		P
111	Invalid Device		P
129	Missing Transporter Board (LAN)		P
130	Reserved for LAN		P
131	Server Address error (LAN)		P
132	Illegal Op on Satellite (LAN)		P

133	Reserved for LAN	P
134	Reserved for LAN	P
135	Protection Violation (LAN)	P
136	Protection Violation (LAN)	P
137	Reserved for LAN	P
138	Reserved for LAN	P
139	Reserved for LAN	P
140	Non existent Directory (LAN)	P
255	Invalid PCOS System Call	P

Table 27-1 PCOS Error Messages

CROSS-REFERENCE ERROR TABLES

This table cross references differences in the use of error codes between PCOS 1.x and 3.x. In most cases the error code functions are the same, and are not shown. In some cases PCOS 3.x provides new codes, these are flagged with an asterisk (*). For seven errors the code has changed. Those seven cases are flagged with (**) and repeated in a supplementary table.

PCOS 1.X	PCOS 3.X	Error Messages (1.X/3.X)
--	2	*/syntax error
--	3	*/illegal function call
--	6	*/overflow
--	9	*/out of range
--	10	*/duplicate definition
13	13	bad data type/type mismatch
--	15	*/string too long
--	18	*/undefined function
--	22	*/missing operand
--	23	*/line buffer overflow
--	35	*/window not open
--	36	*/unable to create window
54	54	bad file open mode/bad file mode
--	59	*/disk type mismatch
63	63	bad record number/invalid record number
64	64	bad filename/invalid filename
--	67	*/too many files
--	68	*/internal error
--	69	*/volume name not found
71**	71	volume name not found/invalid volume number
--	72	*/volume not enabled
73**	73	invalid volume number/invalid password
--	74	*/illegal disk change

75**	75	volume not enabled/write protected file
76**	76	password not valid/error in parameter
77**	77	illegal disk change/invalid number of parameters
78**	78	write protected file/file not open
79**	79	copy protected file/printer error
--	80	*/copy protected file
--	81	*/paper empty
--	82	*/printer fault
90	--	error in parameter
91	--	too many parameters
--	93	*/control C from console
96	--	file not open
101	101	time or date/error in time or date
106	--	function key already exists

Table 27-2 PCOS 1.X to PCOS 3.X Error Messages Cross Referenced

* Error code used in PCOS 3.x only. All codes greater than 111 appear only in PCOS 3.x.
** Change in function for same error number. See the following table.

ERROR CODE CHANGES

In the following cases different error codes are used for the same error.

PCOS 1.X	PCOS 3.X	Error Message
71	69	volume name not found
73	71	invalid volume number
75	72	volume not enabled
76	73	invalid password
77	74	illegal disk change
78	75	write protected file
79	80	copy protected file

Table 27-3 PCOS 1.X to PCOS 3.X Error Code Changes

SUGGESTIONS TO THE PROGRAMMER

SETTING AND DISPLAYING ERRORS

Error codes, values from 0 - 127, are placed in the low byte of R5. The high byte can be used to hold a number identifying the parameter that caused the error, if desired. Otherwise, the high byte must be zero. A system call, number (88), is used to display PCOS errors.

ERROR CODE SYMBOLIC NAMES

Although error codes are passed to PCOS in R5 as a numeric value, the preferred approach is to use symbolic names. A file of symbolic names and the corresponding codes can be developed for all programmers in an installation to use as an include file. The name is used to load the corresponding code into R5. This makes source code more readable and protects against changes in error code assignments. When it is necessary to change existing error codes, only the one include file needs to be changed. The affected programs are then reassembled.

An example include file for PCOS 3.0 error codes is shown on the following pages.

```
//
//  Sample Error Code Include File
//

    CONSTANT
    nxt_wo_for              := 1    // BASIC next without for
    syntax_err              := 2
    ret_wo_gosub            := 3    // BASIC return without gosub
    out_of_data             := 4
    illegal_funcall         := 5    // BASIC illegal function call
    overflow                := 6
    mem_full_err            := 7    // exceeded memory limit
    undef_line              := 8    // BASIC undefined line number
    out_of_range            := 9
    dupl_def                := 10   // duplicate definition
    div_by_0                := 11
    illegal_direct          := 12
    bad_data_type_err       := 13   // type mismatch
    end_of_string           := 14   // out of string space
    str_leng_err            := 15   // string too long
    str_complx_err          := 16   // string too complex
    continue_err            := 17   // can't continue
    undef_function          := 18   // undefined user function
    no_resume               := 19
    resume_wo_err           := 20   // BASIC resume without error
    unprintable_err         := 21   // ??
    missing_oper            := 22   // missing operand
    line_buf_ovflw          := 23   // line buffer overflow

    for_wo_nxt              := 26   // BASIC for without next
```

```
while_wo_wend          := 29   // BASIC while without wend
wend_wo_while          := 30   // BASIC wend without while
ieee_inv_adr           := 31   // IEEE invalid talk/listen adr
ieee_t_l_same          := 32   // IEEE talk=listem adr
ieee_unprtable_err     := 33   // IEEE unprintable error
ieee_no_board          := 34   // IEEE board not present
wind_not_open_err      := 35   // nonexist window selected
wind_create_err        := 36   // unable to create window
invalid_averb          := 37   // invalid action verb
param_range_err        := 38   // parameter out of range
too_many_dim           := 39   // too many dimensions
field_ovflw            := 50
int_err                := 51   // internal error
inv_file_number        := 52   // invalid file number
file_not_found_err     := 53   // file not found
bad_mode_err           := 54   // bad file open mode
file_open_err          := 55   // file already open

disk_io_err            := 57   // disk h/w i/o error
file_exists_err        := 58   // file already exists
vol_mismatch           := 59   // src/dst are diff disk type
bad_disk_err           := 60   // disk not initialized
disk_full_err          := 61   // disk is full
eof_err                := 62   // end of file
bad_rec_num_err        := 63   // invalid record number
bad_filnam_err         := 64   // invalid filename

direct_in_file         := 66   // direct statement in file
too_many_files         := 67
internal_err           := 68   // internal error
volnam_not_found_err   := 69   // volume name not found
rename_err             := 70   // fname exists/ across volume
volnum_err             := 71   // volume number invalid
vol_not_enab_err       := 72   // volume not enabled
invalid_string_err     := 73   // invalid password
illegal_disk_chng_err  := 74   // disk not verified same with
                               // open files
err_wr_prot            := 75   // file is write-protected
param_err              := 76   // error in parameter
too_many_param_err     := 77   // wrong number of parameters
file_not_open_err      := 78   // file not open
printer_err            := 79
err_cp_prot            := 80   // file is copy-protected
paper_empty_err        := 81   // paper empty on printer
printer_fault_err      := 82   // printer fault error

cmnd_not_found_err     := 92   // command not found
ctrl_c_hit             := 93   // control c from console

bad_ld_file_err        := 99   // invalid load file
time_date_err          := 101  // bad time or date

calluser_err           := 108  // error in calluser interface

time_out_err           := 110  // time out error
```

```
    invalid_device          := 111  // invalid device

!   errors 129 - 140 reserved for LAN   !

    network_err             := 129  // missing transporter
    protection_err          := 130  // network protection violation
    file_locked_err         := 131  // file locked
    fs_hw_err               := 132  // file server hardware error
    fs_sw_err               := 133  // file server software error
    not_pcos_file           := 134  // not PCOS compatible
    files_in_dir_err        := 135  // directory contains files
    local_op_err            := 136  // illegal operation
                                    //    in local mode
    fs_pcos_err             := 137  // file server PCOS error
    net_hw_missing_err      := 138  // missing net hardware
    net_sw_missing_err      := 139  // missing net software
    illegal_op_err          := 140  // illegal operation


    invalid_sys_call        := 255
```

28. GLOSSARY

ABOUT THIS CHAPTER

This chapter contains definitions of terms used in this manual. Some of these terms have more general meanings in general data processing use.

CONTENTS

GLOSSARY OF TERMS

The following terms are defined as used in this manual. Some of these terms have more general meanings in general data processing use.

ASCII

American Standard Code for Information Interchange. An international standard code for data representation used by the M20 system.

There are 128 defined ASCII characters. They include control codes, such as Backspace or Carriage Return, and displayable characters. Displayable characters include the digits 0 through 9, upper- and lowercase alphabetic characters (A through Z, a through z), and special characters, such as # and %. All displayable ASCII characters are found on the USA keyboard for the M20.

assembler

A program that translates assembly language statements into executable form. See assembly language.

assembly language

A programming language that uses symbolic statements for machine instructions, constants, addresses, and work space allocations. In the M20, the assembly language used is that of the Z8000 CPU family. The programmer works directly with details of the CPU functioning. Assembly language cannot be run on a different type of CPU, but has advantages in efficiency due to close control of the CPU resources. Compare to high-level language.

block

When referring to M20 system memory, a block is 16 Kb. In the context of disk space, a block is 256 bytes.

boot

"Boot" or "boot the system" means to start the system. See bootstrap loader.

bootable file

A file of a specific format that the bootstrap loader can load into memory to initialize the system.

bootstrap loader

A routine available in ROM that initializes the system by loading a bootstrap file and turning control over to it. That routine then loads other routines to initialize the system. ("Bootstrap" comes from the expression "to lift oneself by one's own bootstraps.")

byte	Eight bits of data. The fundamental cell size in system memory. Memory addresses refer to bytes. A byte can hold one ASCII character, such as the letter "A" or the decimal number "3." It can hold binary values ranging from 0 to 255 decimal.
character font	A 5 x 7 matrix of dots or pixels. Characters are delineated by selecting a pattern of dots within this matrix to be displayed or printed. The character font is contained within the display font. (See also display font.)
CLI	See command line interpreter.
command line interpreter	The PCOS system routine that interfaces most closely with the user. The user enters the name of a command and any other desired or required information (parameters) and the command line interpreter translates the request into the appropriate form and calls the command routine which performs the requested function.
command routine	A program that executes a function required by the PCOS system or the user. Usually command routines are written in assembly language. The routine must be capable of being called by the command line interpreter, of accepting any necessary information (parameters) from the CLI, and of returning control to the CLI when finished.
commands (1)	Functions useful to the user or to PCOS that can be called by name and executed. See command line interpreter, command routine.
commands (2)	In the context of assembly language or machine code, the word refers to commands which are issued to peripheral devices using output or control instructions. A command may be a special purpose instruction or a "command code" sent by an output instruction.
commands (3)	In BASIC, a command is a BASIC verb.
compiler	A program which translates high-level language statements (PASCAL or C, for example) into groups of machine instructions and into calls for support routines.

	Compare to interpreter, interpreter language.
compiler language	A high-level language, such as PASCAL or C, that is processed by a compiler to produce executable code, similar to assembled assembly language.
configuration (1)	Hardware configuration refers to the hardware resources available for a particular M20 system, such as black and white or color display, memory extension boards, input/output options, peripherals, etc.
configuration (2)	Configuration can also refer to software resources, such as languages and application programs.
configuration (3)	PCOS configuration refers to the selection of global parameters defining a particular PCOS system.
CPU	"Central processing unit." In the M20, the Z8001 chip. The CPU provides central system control and arithmetical and logical operations for the M20.
CPU board	See motherboard.
daisy-wheel printer	Printer in which characters are struck by fully-formed typefaces. The type faces are on the spokes of a wheel. Compare to dot-matrix printer.
default	A default value or default name is a standard value or name used when no value or name is specified.
default name	See default.
default value	See default.
device	See peripheral device.
device rerouting	A system facility that enables input to be accepted from devices or files other than the keyboard, and output to be directed to devices and files other than the video display.
diskette	In the M20 system, a single or double-sided 5 1/4 inch floppy disk. In other systems, diskettes of other sizes are used as well.

display font

A bit pattern describing the shape of the character or graphic pattern to be displayed on the video screen. The M20 font pattern is an 8 by 10 matrix of dots. The character font is a 5 x 7 matrix within the display font.

dot-matrix printer

Printer in which characters are formed by striking a pattern of dots. Compare to daisy-wheel printer.

double word

Same as long word; four bytes.

drive number

An integer referring to a diskette drive or the hard disk drive.

```
  0 -- right-hand diskette drive
  1 -- left-hand diskette drive
 10 -- hard disk
```

environment (1)

An operational environment in which the M20 provides particular capabilities to the user. PCOS support for the M20 system has three distinct environments.

```
PCOS
BASIC
Video File Editor
```

PCOS is the fundamental environment. BASIC and the editor are supported by PCOS services.

environment (2)

In this manual, "environment" also refers to the user environment for which PCOS is being configured. The combination of user application needs and the M20 hardware and software resources available constitute an environment for which PCOS can be adapted and enhanced.

extension

See filename extension.

file extension

See filename extension.

FID

See FID number

FID number

"File identifier" number. An integer used by PCOS to identify peripherals.

```
0-15        BASIC files

16, 20-24   Reserved system files
```

17	Console (keyboard or display)
18	Printer
19, 25, 26	RS232 communication (Com, Com1, Com2)

System calls that do input or output with disk files or with more than one device use an FID number.

filename

The name of a file, which may have an extension, "filename.ext", for example. The filename must start with a letter. The remaining characters can be letters or numbers, up to a total of 14 characters. The extension follows a ".".

filename extension

An optional portion of a filename, which follows a "." and can be up to 14 characters in length (although usually it is 1 to 3 characters). Extensions to the filenames of commands have a special meaning:

xxxxx.cmd	Indicates an ordinary transient command.
xxxxx.sav	Indicates a transient command which it stays in memory after being loaded.
xxxxx.bas	Indicates a routine written in BASIC.

font

See display font.

global, globally

A modifier (adjective or adverb) meaning system-wide. In this manual, means affecting PCOS. See global command, global parameter. Contrasted with local, locally.

global command

A PCOS command that allows the user to change the global parameters that define the PCOS environment.

global parameter

A parameter that defines a feature of the PCOS environment.

hard disk

In the M20 system, a 5 1/4 inch Winchester disk unit. The recording surfaces and the read/write access heads and arms are sealed within a container that provides security from contamination. Hard disks of other sizes are in general use.

high-level language

A language which is translated into executable form by an interpreter or compiler. For example, BASIC, PASCAL, and C. A high-level language is somewhat independent of the system on which it runs, and allows transporting programs across systems and types of CPU. To be transportable, a routine or program written in high-level language (HLL) must avoid using system-dependent features. Compare to assembly language. See also interpreter, compiler.

initialization

The process of starting up PCOS. When the M20 diagnostics have successfully completed, the PCOS' nucleus is loaded. It initializes itself according to the memory available and its global parameter settings. At the end of the initialization process, it reads and executes an initialization file, if present.

initialization file

A file writen in either assembly language or BASIC that is automatically loaded and executed on system initialization. It may have one of the following names:

```
-- INIT.CMD
-- INIT.SAV
-- INIT.BAS
```

instruction

A machine instruction is a binary bit pattern (or "machine code") recognized by the microprocessor that causes a defined action to occur. "Instruction" is also used for an assembly language statement which is translated to machine code.

interpreter

A language translator which interprets statements (BASIC statements, for example) into calls on supporting routines and parameters to be passed to those supporting routines. Compare to compiler, compiler language.

interpreter language

A high-level language, such as BASIC, that requires an interpreter in order to be executed.

kb

Abbreviation of kilobyte, or sometimes of keyboard.

kernel

Another term for PCOS nucleus.

kilobyte

1024 bytes of data.

letter-quality printer	A printer suitable for use in sending letters or doing finished work; for example, a daisy-wheel printer.
local, locally	A modifier (adjective or adverb) meaning "of limited, immediate effect." Contrasted with global, globally. For example, device rerouting can be of local effect, applying to only one command.
logical reset	A reset of all global parameters (except those controlled by the real-time clock) and re-initialization of the system (without performing diagnostic tests). It is caused by pressing /CTRL/ /RESET/, simultaneously.
long word	Four bytes; 32 bits. Accessed in memory on an even address boundary.
machine code	See instruction.
memory block	16 Kb of system memory. A memory block has a starting address that is a multiple of 400 hexadecimal.
motherboard	The fundamental M20 board which contains the Z8001 CPU, 128 Kb of system memory, the start up ROM, and Input/Output control. System expansion boards plug into slots on this board. Also called the CPU board.
nil parameter	Same as null parameter.
non-standard initialization	A system initialization where /L/, /D/, /F/, /B/, or /S/ is pressed during power-up diagnostics, or following a PRUN command.
nucleus	A term used interchangeably with kernel. See PCOS nucleus.
null parameter	An unspecified parameter. The receiving routine substitutes a default or standard value. In the case of global commands, the command routine uses the value last specified.
parameter	A data item passed between routines. In high-level languages, a function may have parameters (constants or variables). These parameters are passed by the language translator to the called routine which implements the function. In system programming the term is used for all data items passed between system elements or the

operating system and its supported languages.

PCOS nucleus

The PCOS nucleus, or kernel, is a fundamental part of PCOS required to handle input/output for system peripherals (keyboard, display, printer, and disks), to decode command lines and execute commands, and to manage system memory. It is loaded into memory when the system is initialized and remains there until the working session is terminated. Other system software modules are loaded by the kernel when needed.

peripheral

See peripheral device.

peripheral device

A hardware resource controlled by the system, or at least communicated with by the system. A peripheral device may provide input, such as the keyboard, or output, such as the display or printer, or storage, such as disks. In addition to these devices (called the system peripherals), an M20 may use a great variety of other peripherals such as plotters, laboratory instruments, machine tools, magnetic tape drives, etc.

permanent memory area

That part of memory occupied by the PCOS nucleus, and by those command routines, assembler programs, programmed key definitions and user defined fonts made permanent by a PSAVE command.

physical reset

A system re-initialization caused by pressing the physical reset button. It is equivalent to powering on the system. The subsequent initialization includes diagnostic tests and a reset of all global parameters (including those controlled by the real-time clock).

pixel

"Picture element." The fundamental unit of screen display. It is a dot capable of being set to black or white, or to a color on color display screens.

program

A sequence of instructions coded by the programmer directing the computer system to carry out a set of functions. See also assembly language, high-level languages.

programmed key

A key that has either had its associated ASCII code changed by means of a CKEY

	command, or had a string assigned to it by means of the PKEY command.
RAM	"Random access memory." Refers to the read/write memory chips in system memory. Information in M20 RAM is lost when power is turned off. Compare to ROM.
raster	The grid of pixels (dots) used for the display screen, organized as rows (scan-lines) and columns. Each pixel can be addressed and set individually by its grid location. See also screen bit-map.
raw key code	The immediate code generated by a key (or the key in combination with /Control/ or /Shift/) corresponding to the physical position of the key on the keyboard, independent of system tables.
resident command	A command always available in PCOS system memory. PCOS comes with three resident commands (PLOAD, PUNLOAD, and LTERM), and the user can make other commands temporarily or permanently resident using PLOAD or PSAVE. Compare to transient command.
ROM	"Read only memory." Memory chips which store a pattern permanently, even though power is off. Information in ROM cannot be written over or changed by M20 commands. In the M20, ROM holds the initializing routines such as start up diagnostics and the bootstrap loader.
scanline	A row of pixels (dots) across the display screen.
screen bit-map	A section of system memory that holds an image of the display screen; the source of the display. The display can be modified by changing the bit-map. Monochrome displays have one bit-map of 16K; color displays have 2 or 3 bit-maps of 16K each (for 4-color or 8-color). Each pixel on the screen is represented by a one-bit location in the bit-map or bit-maps. Monochrome displays set a pixel to white or black based on the presence of zero or one in the location. Color displays combine the 2 or 3 values for each location and produce the appropriate color.

segment	When referring to M20 system memory, a segment is potentially 64 Kb. Segments are built using 16 Kb blocks, and may actually contain 16 Kb, 32 Kb, 48 Kb, or 64 Kb.
semi-permanent memory area	That part of memory occupied by loaded commands and assembler programs, PKEYed strings and user-defined fonts that will be released on termination of the current working session.
standard initialization	Initialization following switch-on physical reset, or logical reset: not having /L/, /D/, /F/, /B/, or /S/ pressed during power-up diagnostics.
standard PCOS	The PCOS configuration supplied by Olivetti on the system diskette.
system call	A PCOS procedure used to handle input/output and to manage system resources such as memory and the system clock. System calls can be accessed by assembly language programs via the Z8000 System Call instruction.
system peripherals	The M20 keyboard, display, disk drives, and printer.
text file	An ASCII file whose records are separated either by CR/LF, or by record separator (RS) characters.
thermal printer	A printer that uses specially-treated heat-sensitive paper. Printed images are developed on the paper by applying minute amounts of heat rather than by impact, which results in especially quiet printing. Heat is applied in dots, similar to the operation of a dot-matrix impact printer.
transient command	A command that is not loaded into memory at initialization, but is available on a diskette or the hard disk. This includes commands that are loaded and purged (those with CMD extension), and those that are loaded, but remain in memory after execution (those with SAV extension). Compare to resident command.
utility	A useful special-purpose routine or program. May be a transient command.
volume	The entire contents of a diskette or hard disk.

volume name	An optional name assigned to identify a volume.
wild-card character	A special symbol used for referring to filenames in groups. The ? symbol represents any single character and * represents any string of characters.
word	Two bytes; 16 bits. Accessed in memory on an even address boundary.
working session	The time between booting PCOS and the next boot of PCOS or turning power off.

NOTICE

Ing. C. Olivetti & C. S.p.A. reserves the right to make improvements in the product described in this manual at any time and without notice.

This material was prepared for the benefit of Olivetti customers. It is recommended that the package be test run before actual use.

Anything in the standard form of the Olivetti Sales Contract to the contrary not withstanding, all software being licensed to Customer is licensed "as is". THERE ARE NO WARRANTIES EXPRESS OR IMPLIED INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTY OF FITNESS FOR PURPOSE AND OLIVETTI SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES IN CONNECTION WITH SUCH SOFTWARE.

Code 3985100 F (0)
Printed in Italy

olivetti

Code 3985100 F (0)
Printed in Italy

olivetti