PolyDos

INTRODUCTION

PolyData
microcenter

Strandboulevarden 63, DK-2100 Copenhagen O
Tel 01-420705, Telex DK-FOTEX 16600 att POLYDATA

The manuals

Six manuals are supplied with your PolyDos disk operating system. These are:

PolyDos Users Guide

> The Users Guide describes to you how to operate PolyDos, e.g. the power-up procedure, the concept of a file, and the commands recognized by PolyDos. It is suggested that you read this manual before using the system.

PolyDos System Programmers Guide

> The System Programmers Guide describes all programming aspects of PolyDos. It assumes that you are familiar with the system, and that you have read the Users Guide. The following subjects are discussed: The system workspace, the file system, system subroutines, the overlay mechanism, file formats, and printer interfacing. In addition the System Programmers Guide includes assembly listings of some essential system programs. The System Programmers Guide is meant as a reference guide to assembly language programmers.

PolyDos Utilities Guide

> The Utilities Guide is a manual to the utility programs included on your system disk. The utility programs are FORMAT, BACKUP, and SuperZap.

PolyEdit Users Guide

> This manual describes to you how to operate the system editor. It is recommended that you read this manual before approaching PolyEdit.

PolyZap Users Guide

> This manual describes the PolyZap disk assembler. The syntactical rules of assembly language programming are discussed, as well as the pseudo operations supported by PolyZap, and the assembly options you may use.

PolyDos DISK BASIC Guide

> The DISK BASIC Guide describes the DISK BASIC supplied with PolyDos.

The PolyDos documentation was created on a NASCOM 2 computer, using the PolyText word processing system running under PolyDos, and printed on a NEC 3515 Spinwriter.

Installing the Controller EPROMs

The PolyDos Controller EPROMs are supplied as two 2708s, suitable for installation on the NASCOM 2 main PCB, on a NASCOM RAM A card, or on a Gemini G813 EPROM card. Before installing the EPROMs check that they match your hardware configuration: EPROMs marked G809 should be used in connection with a Gemini G809 floppy disk controller card, and EPROMs marked G805 should be used in connection with a Gemini G805 floppy disk unit. The EPROMs should be origined in memory at address 0D000H (PD2A EPROM) and 0D400H (PD2B EPROM). Below is shown some examples of installation:

NASCOM 2 main PCB, block A

> On LKB1 and LKB2 connect pins 8-12, pins 7-11, pins 6-10, and pins 5-9. On LKS1 connect pins 4-7 (BLOCK A - XROM), and pins 4-10 (BLOCK A - D000-DFFF). Insert EPROM marked PD2A in socket A1, and PD2B in socket A2.

NASCOM 2 main PCB, block B

> On LKB5 and LKB6 connect pins 8-12, pins 7-11, pins 6-10, and pins 5-9. On LKS1 connect pins 6-7 (BLOCK B - XROM), and pins 6-10 (BLOCK B - D000-DFFF). Insert EPROM marked PD2A in socket B5, and PD2B in socket B6.

NASCOM RAM A card

> On the decode pad connect P5-10. Insert EPROM marked PD2A in socket IC27, and PD2B in socket IC28.

Gemini G813 EPROM card, bank 1

> Fit links for 2708 type EPROMs on decode pads for IC13 and IC14. On SKT1 connect pins 23-14 (BANK 1 SELECT - D000-DFFF). Insert EPROM marked PD2A in socket IC13, and PD2B in socket IC14.

Once installed, check that the EPROMs are addressed properly, e.g. executing 'TD000 D008' in NAS-SYS, which should display:

```
D000 C3 03 D0 31 00 10 CD 0D
```

The final step is to set the RESET address. LSW1/1, LSW1/3, and LSW1/4 should be in position UP, and LSW1/2 should be in position DOWN. This completes the installation.

PolyDos

USERS GUIDE

PolyData
microcenter

TABLE OF CONTENTS

TABLE OF CONTENTS

Section 1

Introduction to PolyDos

PolyDos is a high-level disk operating system designed specially for the NASCOM 1 and 2 with NAS-SYS 1 or NAS-SYS 3 monitor. The basic concept of PolyDos is that it is totally compatible with exsiting software written for NAS-SYS and the NASCOM ROM BASIC. The PolyDos package includes the PolyDos controller ROM, the PolyDos system files, the PolyEdit on-screen editor, the PolyZap disk assembler, the PolyDos DISK BASIC expansion to the NASCOM ROM BASIC, and a number of utility programs for formatting, editing and copying disks.

1.1 The manual

This manual describes how to operate PolyDos. In programming matters you should refer to the PolyDos System Programmers Guide. Section 2 contains some general system information. Section 3 discusses the concept of a file, how to name a file, and what kinds of files the system will handle. Section 4 describes the commands recognized by PolyDos, and tells you how to execute files. Section 5 is a detailled description of what happens when you boot the system. Section 6 describes the system files. Section 7 lists all error messages along with a description.

1.2 Notations

Throughout this manual the following notations are used to describe syntactical elements (e.g. commands and file names):

[...] Contains an optional element. If the element is selected it may only be specified once.

{...} Contains an optional element. If the element is selected it may be specified any number of times.

<...> Contains an element name. The meaning of the element is explained in the text.

As an example of these notations, consider the following line, which describes the format of a command line using the COPY command:

```
$COPY <fs1> <fs2>{,<fs1> <fs2>}[;[Y][S]]
```

The command line starts with the command word COPY, which must be specified exactly as is, i.e. using upper case letters. The command word is followed by a blank and two file specifiers separated by a blank. Optionally more file specifiers may be given in pairs, each pair separated from the others by commas. The last element on the line is an option list consisting of any combination of the letters 'Y' and 'S'. If selected, the option list must be preceded by a semicolon.

Section 2

General System Information

2.1 PolyDos memory organization

When operating under PolyDos your memory is organized in the following manner:

0000-07FF	NAS-SYS 1 or NAS-SYS 3 monitor
0800-0BFF	Video RAM
0C00-1000	System stack and NAS-SYS workspace
1000-BFFF	44K of user RAM
C000-C3FF	PolyDos workspace
C400-C7FF	Disk directory buffer
C800-CFFF	PolyDos overlay area
D000-D7FF	PolyDos controller ROM
D800-DFFF	User RAM/ROM
E000-FFFF	NASCOM ROM BASIC

2.2 Disk formats

The PolyDos G809/G815 version and the PolyDos G805 version both support single density format. In addition the G809/G815 version supports double density format. Both formats are double sided (35 tracks per side), with a sector length of 256 bytes. To minimize head movements and to increase system performance the software accesses first side 0 of the disk on a particular track, and then side 1 before stepping to the next track.

2.2.1 Single density format

The single density format is supported by both versions of PolyDos and is therefore suitable for data transfers between the two systems. Each track is divided into 10 sectors, giving a total storage capacity of 175K bytes. To access a single density disk you should refer to drive numbers 0-3 in the G805 version, and drive numbers 4-7 in the G809/G815 version.

2.2.2 Double density format

The double density format is only supported by the G809/G815 version. Each track of the disk holds 18 sectors, giving a total storage capacity of 315K bytes. To access a double density disk you should refer to drive numbers 0-3. Note that drive 0 and drive 4 are physically the same drive, but with different formats.

2.3 Replacing disks

To allow fast command processing the directory of a disk is not read into memory each time the drive is accessed, but only at the first access. Each time the memory copy of the disk directory is updated, it is written to the disk, thus allowing you to remove the from the drive, whenever it stops, without loosing any information. However, inserting a new disk without telling PolyDos about it, can cause strange things to happen and may very well cause irreparable damage to the directory of that disk, as PolyDos continues to use the directory of the disk you have replaced. Therefore, the <u>only</u> times you are allowed to replace a disk in one of the drives are:

1) At power-up, or when RESET has been pressed.
2) Just prior to executing a NEW or a BOOT command.
3) When PolyDos asks you to insert/replace a disk.

2.4 TAB characters

In addition to the usual NAS-SYS control characters (e.g. BS, ENTER, ESC), PolyDos supports TAB characters. TAB has the ASCII value 09, and can be produced from the keyboard by pressing CTRL/I. When printed, a TAB character moves the cursor to the next character column which is a multiple of 8. Thus, it expands into between 1 and 8 spaces, depending on the cursor position. Printing a TAB when the cursor is in columns 0-7, will move the it to column 8. When the cursor is in columns 8-15, it will move to column 16, etc. TAB characters are especially useful for setting up assembly language programs.

2.5 The BREAK function

At any time when a program is scanning the keyboard you may BREAK by pressing CTRL/SHIFT/@, which will interrupt whatever is going on and return you to the command level.

Section 3

PolyDos disk files

Disk files are groups of data. The data can be anything you want it to be - words, numbers, programs, etc. Each file has a name which enables you to recignize it, and an extension which tells you and PolyDos the type of the file.

The name/extension of each file on a disk is recorded in the disk directory. The directory holds up to 50 file entries, theoretically allowing you to create 50 files on each disk. However, as the storage capacity of a disk is limited (please refer to section 2.2 for the exact figures), 50 files of an average size will usually consume more sectors than available on a single disk.

When a file is deleted it is not removed from the directory until the disk is packed, using the PACK command. Often this a an invaluable advantage, as a file can be recovered even if it has been deleted. The DIR command with an E option will tell you the number of files in use, deleted, and free in the selected directory.

3.1 File specifiers

Files are accessed through file specifiers. A file specifier consists of a file name, which enables you and PolyDos to recognize the file, an extension, which defines the type of the file, and a drive number, arranged in the following manner:

```
<name>.<extension>:<drive>
```

3.1.1 File names

File names may contain upper and lower case letters, digits, and special symbols. The symbols that a file name may not contain are control characters, graphic characters, a comma, a space, a period, a colon, or a semicolon. A file name can be from 1 to 8 characters in length. Some examples of legal file names:

```
MYFILE
TFOR2
letter
2001
X&Y-5
```

Some examples of illegal file names:

```
STOCK-CONTROL           (file name too long)
data:99                 (colon in file name)
```

You cannot have more than one active (undeleted) file with the same name and extension on a disk. If you try to save a file under a file name that already exists on that disk, the old file will be deleted, or an error message will be produced, depending on the type of command.

3.1.2 File extensions

When you display a disk directory you will notice that all file names end with a two-character extension after a period, e.g. ACCOUNTS.TX. These extensions give additional information to PolyDos and to you about file contents. Standard file types (extensions) are:

.TX	Text file
.GO	Machine code program file
.OV	Overlay file
.BS	BASIC program file
.DT	BASIC data file
.IN	Information file

Whenever you create one of the above types of files, PolyDos automatically affixes the proper extension to the file name. However, PolyDos does not enforce the use of the above extensions. Actually any two character extension of letters and/or numbers seperated from the file name by a period is acceptable.

3.1.3 Drive numbers

If you don't specify a drive number when you are working on a file, PolyDos normally assumes that the file resides on the master drive, i.e. the drive that was booted at power-up. To access other drives than the master drive, you must add a drive specification to the file name. A drive specification consists of a colon followed by the drive number. Some examples of file specifiers with drive numbers:

```
TEST:1
Game.BS:0
O&X:5
```

3.2 File attributes

Apart from its name and extension, each file has 10 bytes of attributes, which holds some 'technical information' on the file. The file attributes are:

System flags (1 byte)
User flags (1 byte)
Sector address (2 bytes)
Length in sectors (2 bytes)
Load address (2 bytes)
Execute address (2 bytes)

The system flags byte holds two one-bit flags indicating the status of the file. If bit 0 is set the file is locked, and if bit 1 is set the file is deleted. The user flags byte is never used by the system, and may contain any one-byte value. The sector address is the number of the first sector the file occupies. The length in sectors gives the number of sectors occupied by the file. The load address holds the memory address at which the file is to be loaded. The execute address holds the entry point address of the file. The execute address is only used by PolyDos when executing a machine code program file (extension .GO).

Files are always stored sequentially, i.e. as one contiguous block of sectors. When a file is locked it cannot be deleted, renamed, or edited, and it will not be displayed in a directory list unless you request it (using the 'L' option). Normally, the system files are locked to prevent accidental deletion.

3.3 Family file specifiers

Some PolyDos commands supports family file specifiers. A family file specifier is constructed as any other file specifier, except that it has its name and/or extension missing. Instead of producing an error the command (or program) will include all files in the specified directory that matches the family file name. Some examples:

TEST	will include all files on the master drive that has the name TEST, regardless of their extension.
.GO:1	will include all machine code program files on drive 1.
:0	will include all files on drive 0.

Note that commands that do not support family names will often allow you to omit the extension, and instead supply the extension of the first file found in the directory.

Section 4

PolyDos operation

4.1 Power-up

Upon power-up, or when RESET is pressed, PolyDos prompts:

```
Boot which drive?
```

Insert a system disk in one of the drives, and type the number of that drive, or, to return to NAS-SYS, type 'N'. The number you typed now becomes the number of the master drive, i.e. the drive that is selected if you don't specify anything else, and the drive from which system files are loaded. Normally the master drive is drive 0. Assuming that the disk is of a correct format and that it contains the system files, the screen will be cleared, and a prompt message will be output:

```
PolyDos x.x [yyyy]
Copyright (C) 1981
PolyData microcenter
```

where x.x is the version number, and yyyy is the implementation name ([G809/G815] for Gemini G809 FDC card with G815 floppy disk unit, and [G805] for Gemini G805 floppy disk system). If something goes wrong, PolyDos outputs:

```
(Error ee)
```

and transfers control to NAS-SYS. ee is one of the error codes explained in section 7.

4.2 Command lines

Whenever you see the PolyDos prompt '$', you are talking to the part of the operating system called the executive (Exec). Exec handles all communications between PolyDos and you - it processes your input and responds to it with either the appropriate action or an error message.

Command lines are entered using NAS-SYS editing facilities. An entry is terminated by pressing <ENTER>. If you enter a line with no '$' as the first character it is considered a NAS-SYS command line (see section 4.2.2) and normal NAS-SYS syntactical rules apply to it. If the '$' is followed by one or more spaces, the line is considered a comment line, and thus ignored.

If the input line is not a NAS-SYS command line or a comment line, Exec it looks at the first word typed (the command word) and compares it to the list of legal commands in its command table. If a match occurs, Exec takes the appropriate action in responce to the command. If no match occurs, Exec decides that you are trying to execute a file, an it moves on to looking up the file specifier in the directory. If Exec does not find a file of the given name on the disk specified, it outputs an error message. If such a file does exist, Exec tries to execute it. The process of executing a file is described in section 4.5. Some examples of command lines:

```
$DIR
```

Show on the screen an unextended directory listing of the master drive.

```
$LIST LETTER:1;P
```

List the file called LETTER on drive one to the printer.

```
$STARTREK
```

Execute the program called STARTREK on the master drive. Note that as STARTREK is not a PolyDos command, Exec automatically assumes that it is a file to be executed. What actually happens when STARTREK is executed, depends on the type of the file as well as the file itself.

```
$Hello.BS:1
```

Execute the BASIC program file called Hello on drive one.

4.2.1 Command options

Most PolyDos commands will respond to one or more command options. Command options are always the last element of a command line, and must preceded by a semicolon ';'. The semicolon need not be preceded by a space. Each option consists of a single upper case character. Some examples:

```
DIR;ELD
LIST GAME.BS;P
```

In the description of each command you can see what options the command will allow, and how it will respond to them.

4.2.2 NAS-SYS commands

If you delete the '$' prompt output by Exec your input line will be handled as a NAS-SYS command line. Once you execute a NAS-SYS command Exec stops outputting '$' prompts until you type one yourself. Note that some NAS-SYS commands are not available using this method. These are the 'B' and the 'S' command. If you try execute one of them, you will get an error. The only way to make 'B' and 'S' response properly is to enter 'E0', which restores normal NAS-SYS operation.

4.3 PolyDos commands

On the following pages the PolyDos commands are described. Note that all commands must be typed in upper case letters. Each description is headed by a line defining the syntax of the command.

4.3.1 DIR - List directory

```
$DIR <fspec>[;[E][L][D][P]]
```

The DIR command will display the all files that match the family file specifier <fspec> given on the command line. If the drive number is not specified, the master drive will be selected. The 'E' option specifies that the directory display should be in its extended form, which means that all data relating to the disk should be displayed, as well as the attributes of each file. 'L' requests that locked files be included in the display. Similary, 'D' requests that deleted files be included. 'P' requests that the directory display be sent to the printer instead of the screen. Some examples:

```
$DIR
```

Display the name/extension of all active and unlocked files on the master drive.

```
$DIR .GO:1;L
```

Display the names of all machine code files on drive one, including the ones that are locked.

```
$DIR TEST;P
```

List to the printer all files that are active and undeleted and named TEST, regardless of their extensions.

```
$DIR ;ELD
```

Display an extended directory list of all files on the master drive.

A normal directory list, i.e. a non-extended list, will display three file specifiers on each line, e.g.:

```
Exec.OV        Emsg.OV        Dfun.OV
Ecmd.OV        Edit.OV        Info.IN
GAME.BS        LETTER.TX      FORMAT.GO
BACKUP.GO      ZYPT.X1
```

An extended directory list will display one line for each file, giving its sector address, its length in sectors, its load address, its execute address, its flags, and its file specifier. A flag value of 'D' means that the file is deleted, and a flag value of 'L' means that it is locked. If nothing is displayed in the flag column, the file is neither locked nor deleted. In addition, an extended directory display will output the name of the selected drive, and the number of files/sectors in use, deleted, and free:

```
Drive 0: PolyDos 2.0 SYSTEM
6 files in use, 1 deleted, 43 free.
34 sectors in use, 2 deleted, 1224 free.
Sect Nsct Load Exec F Name
0004 0008 0000 0000 L Exec.OV
000C 0004 0000 0000 L Emsg.OV
0010 0008 0000 0000 L Dfun.OV
```

```
0018 0001 C200 0000 L Info.IN
0019 0006 1000 107A   Invader.GO
001F 0002 0000 0000 D LETTER.TX
0021 0003 0000 0000   LETTER.TX
```

If the screen is used for output, the DIR command will stop and blink the cursor each time 15 lines has been written. Pressing CTRL/SHIFT/@ aborts the command, any other key continues the command.

4.3.2 COPY - Copy files

```
$COPY <fs1> <fs2>{,<fs1> <fs2>}[;[Y][S]]
```

The COPY command will copy the contents of a file into a new file. <fs1> is a family file specifier giving the name/extension of the source file(s). <fs2> defines the name/extension of the new file(s) to be created. Elements omitted from <fs2> will be taken from <fs1>. If a drive number is not specified in <fs1>, the master drive is assumed. When <fs1> is a family file specifier, i.e. the name and/or extension is missing, COPY will prompt you each time a match is found, e.g:

```
Copy DELTA.BS:0 to GAMMA.TX:1?
```

Typing 'Y' causes COPY to copy the file. The 'Y' option supresses prompting. If you are running on a single drive system, the 'S' option will be of use to you when you want to copy files from one disk to another. Instead of creating a new file on the same disk as the source file, COPY will ask you to swap disks during the duplication. The COPY command will always include locked files, and always exclude deleted files. Some examples of COPY command lines:

```
$COPY TEST1.TX TEST2,Wakeup.GO :1
```

Copy the file called TEST1.TX on the master drive into a new file called TEST2.TX also on the master drive, and copy the file called Wakeup.GO on the master drive into a file of the same name/extension on drive 1.

```
$COPY :1 :0;Y
```

Copy all files on drive one to drive zero. As neither name nor extension of the destination files are specified, the they will have the same names and extensions as the source files. The 'Y' option causes COPY to copy all files with no prompting.

```
$COPY TEST.GO;S
```

Copy the file called TEST.GO on the master drive, to a new file, also called TEST.GO, on another disk. The 'S' option causes COPY to ask you to insert a new disk before it creates the destination file.

4.3.3 REN - Rename files

```
$REN <fs1> <fs2>{,<fs1> <fs2>}[;Y]
```

The REN command will change name and/or extension of the files selected. <fs1> is a family file specifier giving the names/extensions of the files to be renamed. <fs2> defines the new names/extensions. Elements omitted from <fs2> will be taken from <fs1>. If a drive number is not specified in <fs1>, the master drive is selected. You should never specify a drive number in <fs2>, as the REN command cannot rename across drives. If the name and/or the extension is omitted from <fs1>, it is considered a family file specifier, and all files matching the elements given are taken into account. When <fs1> is a family file specifier, the REN command will prompt you each time a match is found, e.g.:

```
Rename ZYPT.TX:0 to ZOT.BS:0?
```

Typing 'Y' causes REN to rename the file. The 'Y' option will supress prompting. REN only includes files that are active (undeleted) and unlocked. Some examples of REN command lines:

```
$REN Alhpa.SY Beta,GAME.TX:1 .BS
```

Rename the file called Alpha.SY on the master drive to Beta.SY, and rename the file called GAME.TX on drive one to GAME.BS.

```
$REN APPLE:1 PEAR;Y
```

Rename all files on drive one called APPLE to PEAR, without changing their extensions, and without asking you before each rename.

4.3.4 DEL - Delete files

```
$DEL <fspec>{,<fspec>}[;Y]
```

The DEL command will delete all files that matches one of the file specifiers given on the command line. Family file specifiers will cause DEL to prompt you each time a match is found, for example:

```
Delete INTRO.GO:1?
```

Typing 'Y' causes DEL to delete the file. The 'Y' option will supress prompting. DEL only includes files that are active (undeleted) and unlocked. Some examples of DEL command lines:

```
$DEL MYSTERY.XY,Invader.GO,FIFO:1
```

Delete the files called MYSTERY.XY and Invader.GO on the master drive, as well as all files called FIFO on drive one. When the FIFO files are processed, DEL will prompt you each time a file is found, as FIFO:1 is a family file specifier.

```
$DEL :1;Y
```

Delete all files on drive one with no prompting.

4.3.5 UNDEL - Undelete file

```
$UNDEL <fspec>{,<fspec>}
```

UNDEL will undelete (recover) files. Note that you cannot undelete a file which has the same name and extension as an already undeleted file. Also note that if there are more deleted files with the name you specify, the last file will be undeleted. UNDEL does not support family file specifiers. If the extension is omitted from <fspec>, the last file of the name given will be undeleted. Some examples of UNDEL command lines:

```
$UNDEL TEXT.TX,BYTE.BS
```

Undelete the file called TEST.TX and the file called BYTE.BS on the master drive.

```
$UNDEL FlipFlop:1
```

Undelete the last file called FlipFlop on drive one, regardless of its extension.

4.3.6 LOCK - Lock files

```
$LOCK <fspec>{,<fspec>}[;Y]
```

The LOCK command is identical to the DEL command (see section 4.3.4), except that the files specified are locked.

4.3.7 UNLOCK - Unlock files

```
$UNLOCK <fspec>{,<fspec>}[;Y]
```

The UNLOCK command is identical to the LOCK and the DEL command (see section 4.3.4), except that the files specified are unlocked.

4.3.8 PACK - Pack disk

```
$PACK <drive>
```

The PACK command will physically remove all deleted files from the drive specified. This is done by erasing the deleted files and moving the rest of the files 'up', i.e. moving them towards the beginning of the disk, so that no empty areas are left between the files. If the drive number is omitted, the master drive is selected.

4.3.9 SAVE - Save file

```
$SAVE <fspec> <from> <to>[ <load>[ <exec>]]
```

The SAVE command will save the memory block starting at address <from> up to, but not including, address <to> under the file name <fspec>. <load> and <exec> are the load and execute addresses of the file. If <load> and <exec> are omitted, the value of <from> is used. If <exec> is omitted, the value of <load> is used. <fspec> must define both name and extension of the file. However, the drive number may be omitted, in which case the master drive is assumed. An Example:

```
$SAVE PingPong.GO 1000 1F56 1000 1321
```

The above command line will create a file called PingPong.GO on the master drive, and save in it the memory block between 1000H and 1F56H. When executed, PingPong will be loaded into address 1000H and runned at address 1321H.

4.3.10 LOAD - Load file

```
$LOAD <fspec>[ <addr>]
```

The LOAD command will load into memory, starting at address <addr>, the file given by <fspec>. If <addr> is omitted, the load address of the file will be used. If the extension is omitted from <fspec>, the first file with a matching name is loaded.

4.3.11 ATTRIB - Change file attributes

```
$ATTRIB <fspec> <load> <exec>
```

The ATTRIB command will change the attributes of the file given by <fspec>. <load> is a hexadecimal number giving the new load address, and <exec> is a hexadecimal number giving the new execute address. If the extension is omitted from <fspec> the first file with a matching name is used. An example:

```
$ATTRIB EXTRA.GO 1000 1E74
```

The above command line will change the load address of the file called EXTRA.GO on the master drive to 1000H and the execute address to 1E74H.

4.3.12 LIST - List file

```
$LIST <fspec>[;P]
```

The LIST command will list the file specified. If the screen is used for output (i.e. if the 'P' option is not present), 15 lines are output at a time, whereafter LIST blinks the cursor awaiting a key to be pressed. Pressing CTRL/SHIFT/@ aborts the list, any other key continues. If the 'P' option is present, the printer is used for output. If the extension is omitted from <fspec>, the first file with a matching name is listed.

4.3.13 SKIP - Print blank lines

```
$SKIP[ <lines>][;P]
```

The SKIP command will print <lines> blank lines (i.e. <lines> carriage returns), or, if <lines> is omitted, a form-feed. The 'P' option causes the printer to be used for output. Normally SKIP is only used in connection with the 'P' option.

4.3.14 BUFFER - Define RAM buffer

```
$BUFFER <start> <length>
```

The BUFFER command will redefine the parameters of the RAM buffer used by the COPY, PACK, and LIST commands. <start> and <length> are hex numbers, <start> giving the start address of the buffer, and <length> giving the length in sectors, i.e. in 100H-byte blocks. At power-up PolyDos deafults to the largest buffer possible, i.e. a buffer starting at address 1000H of length B0H bytes.

4.3.15 NAME - Rename disk

```
$NAME <drive>
```

The NAME command will change the name of the disk specified. When activated, NAME prompts:

```
New disk name?
```

Type the new name (max. 24 characters) and press <ENTER>, whereafter the new name is written to the disk. If <drive> is omitted, the master drive is selected.

4.3.16 READ - Read sectors

```
$READ <addr> <sector> <numsec>[ <drive>]
```

The READ command will read <numsec> sectors starting at sector <sector> on drive <drive> into memory starting at address <addr>. If <drive> is omitted, the master drive is selected. <addr>, <sector>, and <numsec> are hex numbers.

4.3.17 WRITE - Write sectors

```
$WRITE <addr> <sector> <numsec>[ <drive>]
```

The WRITE command will write <numsec> sectors starting at sector <sector> on drive <drive> from memory starting at address <addr>. If <drive> is omitted, the master drive is selected. <addr>, <sector>, and <numsec> are hex numbers.

WARNING: Do not use the WRITE command unless you are absolutely sure of what you are doing, or otherwise you may cause irreparable damage to the data on the disk.

4.3.18 NEW - New disk(s) inserted

```
$NEW
```

The NEW command informs PolyDos that you have inserted one or more new disk(s) in the drive(s), thus making it necessary to reread the directory. Always use this command when a new disk is inserted.

4.3.19 BOOT - Reboot PolyDos

```
$BOOT <drive>
```

The BOOT command may be compared to a 'soft' RESET. It reboots PolyDos, making the drive you specify the master drive. If <drive> is omitted, the master drive is rebooted. Read more about the boot process in section 5.

4.4 Special commands

Apart from the system commands described in section 4.3, PolyDos has two special commands, one which will inkove PolyEdit (the system editor), and one which will invoke the DISK BASIC.

4.4.1 EDIT - Invoke PolyEdit

The EDIT command will invoke PolyEdit, the system editor, if it is present on the master drive. Read more about this in the editor manual.

4.4.2 BASIC - Invoke DISK BASIC

The BASIC command will invoke DISK BASIC if it is present on the master drive. Read more about this in the DISK BASIC manual.

4.5 Executing files

A file is executed by entering its file specifier when PolyDos wants a command. The actions taken when a file is executed is entirely defined by the type (extension) of the file.

4.5.1 Standard file types

Machine code program files (extension .GO) and textfiles (extension .TX) are immediately recognized by PolyDos when they are executed.

4.5.1.1 Machine code program files

When a mahcine code program file is executed, it is read into memory starting at its load address, and executed at its execution address.

4.5.1.2 Text files

When a text file is executed, PolyDos enters command file mode. In this mode, the system will obtain its input from a text file instead of the keyboard. What actually happens is that the NAS-SYS routine called BLINK, which normally provides a blinking cursor during input, will fetch input characters from the command file. Assume that the following text file has been created using the editor, and saved under the name CMDFILE.TX:

```
             Now creating system disk in drive one

         COPY Exec.OV :1,Dfun.OV :1,Emsg.OV :1
         COPY Ecmd.OV :1,Edit.OV :1,Info.IN :1
         COPY BSfh.OV :1,BSdr.BR :1

                 ******** Copy Complete ********
```

If you execute the above file, by entering its name on the command line, the following happens:

```
$CMDFILE
$    Now creating system disk in drive one
$
$COPY Exec.OV :1,Dfun.OV :1,Emsg.OV :1
Copying Exec.OV:0 to Exec.OV:1.
Copying Dfun.OV:0 to Dfun.OV:1.
Copying Emsg.OV:0 to Emsg.OV:1.
$COPY Ecmd.OV :1,Edit.OV :1,Info.IN :1
Copying Ecmd.OV:0 to Ecmd.OV:1.
Copying Edit.OV:0 to Edit.OV:1.
Copying Info.IN:0 to Info.IN:1.
$COPY BSfh.OV :1,BSdr.BR :1
Copying BSfh.OV:0 to BSfh.OV:1.
Copying BSdr.BR:0 to BSdr.BR:1.
$
$        ******** Copy Complete ********
$
```

Note that a line is considered a comment line if it starts with a blank. This may be used to provide comments to the operator when a command file is executing. The command file mode remains in effect until one of the following events occur:

1) End-of-file is reached.
2) CTRL/SHIFT/@ is pressed on the keyboard.
3) An error occurs.
4) A PACK, NEW, or BOOT command is executed.

If the command file mode is aborted before the command file ends, the message:

```
(Cmdf abort)
```

is displayed. Command files cannot be nested. If a command file executes another command file, the first command file is not reactivated when the second command file ends.

4.5.2 User defined file types

Other file types than machine code program files and textfiles cannot be executed immediately, as PolyDos does not know what to do with such files. However, the system will not just output an error message if you try executing a file of non-standard type. Instead it will try locate a file handler for that specific file type. A file handler is an overlay file (extension .OV) which contains the code to be executed when a file of its associated type is executed from the command level. The name of the file handler overlay tells PolyDos what type of files it will handle. For instance, a file handler overlay capable of executing files of extension .AB would be named ABfh.OV. The first two characters of the file handler name defines the extension of its associated file type. The next two characters are always 'fh' to indicate that it is a file handler, and the extension is .OV to indicate that it is an overlay.

An example of a file handler overlay is the overlay file called BSfh.OV on your system disk. This overlay is activated whenever you execute a file of extension .BS. As you know from a discussion earlier in this manual, .BS files are BASIC program files. So what actaully goes on, when you execute a BASIC program, is that PolyDos loads the BASIC file handler overlay (BSfh.OV) into the overlay area (C800H-CFFFH) and executes it. The actions taken hereafter is entirely defined by the file handler overlay. In this specific case, BSfh colstarts the ROM BASIC, loads the DISK BASIC routines file, loads your BASIC program file, and starts executing it.

If you try to execute an existing file of non-standard type, and the disk does not contain a file handler overlay for that specific file type, PolyDos responds:

```
I can't find that file
```

The file that PolyDos cannot find is not the file you tried to execute but its file handler. The process of creating a file handler overlay is described in the System Programmers Guide.

Section 5

The boot process

When PolyDos is booted, it prompts you for the number of the master drive. After this, and until you see the sign-on message on the screen, several things happen.

First, the controller ROM initializes the system workspace, loads the directory of the master drive, and loads the file called Exec.OV into the overlay area. If no errors occur, control is then transferred to Exec.

When Exec is invoked, it is told to continue the boot process. To do this, it looks up a file called Info.IN on the master drive. Info is the system information file which holds all data relevant to the printer attached to your system, as well as a definition of the cursor character, the cursor blink rate, and the keyboard repeat rates. If Info is present, it is loaded into the slot reserved for it in the system workspace. If Info is not present, some default values are inserted in the proper locations to satisfy the above system parameters. Read more about the information file in the System Programmers Guide.

The next thing Exec does is to look up a file called Init on the master drive. If Init is there, it is executed, just as if you typed Init as a command line. If Init is not present, Exec outputs the sign-on message, and enters the command processing loop, which prints a '$' prompt and awaits input.

5.1 Creating a turn-key system

Creating a turn-key system as actually very simple - just rename the file you want executed at power-up to Init. Init can be of any type (extension) you wish - a command file (extension .TX), a machine code program file (extension .GO), a BASIC program file (extension .BS), etc. Just remember that every time you boot a disk with a file called Init on it, Init is executed automatically.

Section 6

The system files

From earlier discussions you already know some of the system files (Exec.OV, Info.IN, etc.). On a system disk the following files are normally present:

Exec.OV The system executive, which gains control when the system is booted and when you exit a program or a command. Exec evaluates your command lines and decides which actions to take in response. In addition, Exec contains the code for the following commands: DIR, DEL, UNDEL, SAVE, LOAD, LIST, SKIP, READ, WRITE, NEW, and BOOT.

Dfun.OV The Dfun overlay contains the code for a number of PolyDos commands. These are: COPY, REN, LOCK, UNLOCK, ATTRIB, PACK, BUFFER, and NAME.

Emsg.OV The system error message writer. Each time an error occurs, Emsg is invoked to print an error message.

Info.IN The system information file. Info contains all parameters relevant to the printer attached to your system, as well as the cursor character, the cursor blink rate, and the keyboard repeat delays. If Info is not present, PolyDos will supply some suitable values for the above paremeters.

Ecmd.OV The Ecmd overlay handles the EDIT command. It is of no use unless the Edit overlay is present as well.

Edit.OV The PolyEdit editor. The Edit overlay is normally invoked by the Ecmd overlay, but may also be invoked from elsewhere, i.e. from one of your own programs. Read more about this in the PolyEdit manual.

BSfh.OV The BASIC program file handler. Apart from handling execution of BASIC program files, BSfh also handles the BASIC command. It is of no use unless the BSdr file is present as well.

BSdr.BR The DISK BASIC routines file. This file is loaded by BSfh before control is transferred to the ROM BASIC. It contains the code for the DISK BASIC commands.

FORMAT.GO The PolyDos Disk Format Program. This program is used to format new disks. Read more about it in the PolyDos Utilities Guide.

BACKUP.GO The PolyDos Disk Backup Program. This program is used to make backup copies of disks. Read more about it in the PolyDos Utilities Guide.

SZAP.GO The PolyDos SuperZap Program. SuperZap is used to edit disk sectors and may be used by experienced programmers to recover crashed disks. Read more about it in the PolyDos Utilities Guide.

PZAP.GO The PolyZap Z-80 Disk Assembler. PolyZap is used to translate assembly language source files into executeable Z-80 machine code. Read more about it in the PolyZap Users Guide.

SYSEQU.SY The PolyDos Equate File. This files contains an assembled symbol table giving symbolic names of all PolyDos and NAS-SYS routines, etc. SYSEQU is of no use unless PolyZap is present as well. Read more about SYSEQU in the System Programmers Guide.

From the above list of system files you can construct system disks to suit special purposes. A minimum system disk need only include Exec and Emsg. The facilities provided by such a disk are however very restricted: The commands supported by Dfun are not available, printer communications has been cut off, and you cannot EDIT files, FORMAT disks, BACKUP disks, translate assembly langauge programs, nor run BASIC programs.

Section 7

Error messages

Error messages are output by the error message writer overlay called Emsg.OV. Internally, each error message is identified by a two-digit error code. Normally you don't have to bother with these error codes, but in some extreme error conditions, when PolyDos is unable to invoke Emsg, they will appear. All error codes are listed below along with their associated error messages and a description. The error codes 20 through 25 will only occur when 8 retries has proven useless.

01 Syntax error

The command line contains a syntactical error, e.g. an invalid hex constant.

02 Too many/few parameters

You are specifying too few or too many command parameters.

03 Bad parameters

The command parameters passed to the command are symtactically correct, but conflicting, e.g. a start address is higher than an end address.

10 Illegal character in filename

The following characters are not allowed in file names and extensions: Graphic characters, control characters, a period, a comma, a colon, a semicolon, a blank or a TAB character.

11 Filename too long

A filename may not be more than 8 characters in length.

12 Bad drive identifier

The drive identifier is not a valid drive number.

13 Filename missing

The filename is missing from a file specifier.

14 Extension missing

The extension is missing from a file specifier.

15 Drive number missing

The drive number is missing from a file spevifier.

20 Drive not ready

You are trying to access a drive with no disk in it or with the door open. The drive not ready error will only occur if PolyDos has previously accessed the drive.

21 Disk write protected

You are trying to write to a write protected disk. Remove the write protect tab.

22 Write fault

This message is caused by a signal from the disk drive itself, and should never occur where Pertec FD250 drives are used.

23 Record not found

The disk controller is unable it locate an error free sector header or an error free data block. If this error occurs it is strongly advisable that you copy the disk to another one using the BACKUP program. If the error persists the information in that sector will have been lost. Provided you have an idea of the original contents of the secter it can however be reconstructed on the new disk using the SuperZap program. Once copied, reformat the disk that caused the error.

24 Checksum error

A checksum error occurred when reading a sector. For comments on this error see above.

25 Lost data error

This error should not occur. If it does it implies that the CPU clock rate is too slow. The minimum clock rate PolyDos can run with is 2MHz without any wait states.

26 Bad disk address

The sector address passed to the low-level sector I/O routines is out of range.

27 No disk or wrong format

You are trying to access a drive with no disk in it or with the door open or the disk in the drive is of a wrong format.

28 Illegal drive number

You are trying to access a non-existing drive.

29 Disk is full

During a block read/write the sector I/O routine was requested to access a sector beyond the end of the disk. If this error occur it indicates that there is no more room on the disk. Pack the disk and retry.

30 I can't find that file

The file you are trying to access does not exist on the disk specified.

31 That file already exists

You are trying to create a file with the same name and extension as an already existing file.

32 Directory is full

There is not enough room in the directory to create new files. Pack the disk and retry.

33 I can't do that to a locked file

You are trying to delete or rename a locked file. Unlock the file or use another name.

40 I can't rename across drives

The drive numbers of the current file specifier and the new file specifier does not agree.

PolyDos

SYSTEM PROGRAMMERS GUIDE

PolyData
microcenter

TABLE OF CONTENTS

TABLE OF CONTENTS

Section 1

Introduction

This manual describes all programming aspects of the PolyDos disk operating system. The manual assumes that you are familiar with the system and that you have read the PolyDos Users Guide. Furthermore it is required that you have some knowledge of assembly language programming.

Section 2 describes the system workspace and each of the sections it is divided into. Section 3 describes the PolyDos file system. Section 4 describes the system subroutines available to the system programmer. Section 5 discusses the overlay mechanism, and how to create overlays. Section 6 describes the internal format of standard file types. Section 7 discusses the command file mode, and provides a method of activating it. Section 8 describes the information file.

Throughout the manual a lot of symbolic names are introduced as identifiers for various system locations and subroutines. The SYSEQU file, which is listed in appendix A, provides a way of referencing these symbols. It is included as a symbol table file (SYSEQU.SY) on your system disk. SYSEQU.SY contains an assembled symbol table which can be referenced from your assembly language source programs using the REFS and REF pseudo-ops supported by the PolyZap assembler (for further details on REFS and REF, please refer to the PolyZap Users Guide).

Section 2

PolyDos workspace

PolyDos uses addresses C000H through CFFFH as workspace. The workspace area is divided into 6 sections:

Addresses	Name	
C000H-C0FFH	WORKSP	System variables
C100H-C1FFH	SCTB	SCAL address table
C200H-C2FFH	INFOFA	Information file area
C300H-C3FFH	SECBUF	Sector buffer
C400H-C7FFH	DIRBUF	Directory buffer
C800H-CFFFH	OVAREA	Overlay area

2.1 System variables

The system variables area may be compared to an extension of the NAS-SYS workspace. The descriptions that follow gives the address and symbolic name of each system variable.

Name	Addr	Size	Description
MDRV	C000	1	Master drive number.
DDRV	C001	1	Directory drive number. Contains the number of the drive whose directory is currently held within the directory buffer. A value of 0FFH indicates that no directory is currently within the buffer.
DRVCOD	C002	1	Drive code. Contains a drive code for the currently selected drive. Writing 0FFH to this location deselects all drives.
FIRST	C003	1	Power-up flag. A value of zero indicates that the system is being booted.
ERRFLG	C004	1	Error flag. A non-zero value indicates that the CKER routine is in the process of calling the Emsg overlay.
ERRCOD	C005	1	Error code. Contains the error code of the most recent error.
BREAK	C006	2	Break address. Contains the address of the routine to jump to when CTRL/SHIFT/@ is detected from the keyboard by either CKBRK or RKBD.
BRAM	C008	2	RAM buffer start address. The RAM buffer is used by the COPY, PACK, and LIST commands.
BNSC	C00A	1	RAM buffer length in sectors. Contains the length of the RAM buffer in 100H-byte blocks.
CFFLG	C00B	1	Command file flag. A non-zero value indicates the the command file mode is active.
CFDRV	C00C	1	Command file drive. Contains the drive number of the command file.
CFSEC	C00D	2	Command file sector address. The disk address of the next sector to be loaded from the command file.

CFNSC	C00F	1	Command file sector counter. The number of sectors remaining to be loaded from the command file.
CFSBP	C010	1	Command file sector buffer pointer. Points to the next character to be loaded from the command file sector buffer (SECBUF, address C300H-C3FFH).
RKROW	C011	1	Keyboard row number (1-8) of the currently repeating key. Zero indicates that no key is repeating.
RKBIT	C012	1	Keyboard bit mask for the currently repeating key.
RKVAL	C013	1	ASCII value of the currently repeating key.
RKCNT	C014	2	Delay counter for repeat keyboard routine.
BLINKF	C016	1	Blink routine flag. Contains the ASCII value of the character overlayed by the cursor. The BLINK routine sets this flag. It is checked by the CKBRK routine when a break occurs to see if a character is to be restored. Zero indicates that no cursor is on the screen.
PLCT	C017	1	Printer line counter. Contains the number of lines printed on the current page. The first line has the value 0.
PPOS	C018	1	Print head position. Contains the print head position of the printer, i.e. the number of characters printed on the current line. The first position has the value 0. By OR-ing the contents of PPOS with the contents of PLCT you can determine if the printer is at the top of a form.
CLINP	C019	2	Command line pointer. CLINP points to the next non-blank character in the command line buffer when a command or a program is invoked.
CLIN	C01B	48	Command line buffer. When a command line is input it is copied to this buffer. The '$' prompt is not included. The command line is ended by 0.
OVFCB	C04B	10	Overlay file controller block. This FCB is by the routines COV and COVR to look up overlay files.
S1FCB	C055	20	First system file controller block. S1FCB is used by the system commands to look up files. You are allowed to use it from your own programs.
S2FCB	C069	20	Second system file controller block. S2FCB is used by some system commands to look up files. You are allowed to use it from your own programs.
DSKWSP	C07D	6	Disk I/O routines workspace.
SYSWSP	C083	61	Miscellaneous system workspace. This area is used by some system command handlers.
USRWSP	C0C0	64	User workspace. This area is not used by PolyDos.

2.2 SCAL address table

The SCAL address table contains the addresses of the SCAL routines. Upon power-up PolyDos copies the NAS-SYS SCAL table (routines 41H to 7CH) and the PolyDos SCAL table (routines 7DH to 8FH) to this area, and loads the logical start address into STAB (0C71H-0C72H) in the NAS-SYS workspace. The first address contained in the table is the address of routine number 41H (NAS-SYS 'A' command). Thus, the logical start address is SCTB less 82H bytes. The size of the SCAL address table far exceeds the number of routines defined by NAS-SYS and PolyDos (128 routines are possible, numbered from 41H to C0H). You may wish to take advantage from this by defining new SCALs. If you do so, you should not use routines 90H-9FH, as these might be defined in future versions of PolyDos.

2.3 Information file area

This chapter only defines the memory layout of the information file area. For a functional description, please refer to section 8.

Name	Addr	Size	Description
CURCHR	C200	1	Cursor character. Contains the ASCII value of the character used to provide a blinking cursor.
CURBLR	C201	1	Cursor blink rate. The value contained in this location defines the number of times the BLINK routine should scan the keyboard before blinking the cursor.
RKLON	C202	2	Keyboard initial repeat delay.
RKSHO	C204	2	Keyboard repeat speed.
PLPP	C210	1	Lines per page on printer.
PBMG	C211	1	Bottom margin on printer. PBMG is included in PLPP.
PCPL	C212	1	Characters per line on printer.
PLMG	C213	1	Left margin on printer. PLMG is included in PCPL.
INSLEN	C214	1	Length of initialization string (maximum is 43 characters).
INSTR	C215	43	Initialization string.
PCHR	C240	192	Entry point of routine to output A to the printer.

2.4 Sector buffer

The sector buffer is used by PolyDos only when the command file mode is active. Should you wish to use this area from one of your programs, call the CFMA routine to make sure that no command file is executing.

2.5 Directory buffer

The directory buffer contains a memory image of the directory of drive DDRV. For more details on directories, pelase refer to section 3.3.

2.6 Overlay area

The overlay area is the area into which overlay files are loaded when they are invoked. The first four bytes of an overlay (C800H-C803H) contains the overlay name. An overlay is always invoked at address C804H. For more details on overlays, please refer to section 5.

Section 3

The PolyDos file system

3.1 Disk formats

The G809/G815 and the G805 versions of PolyDos both support single density format. In addition the G809/G815 version supports double density format. Both formats are double sided (35 tracks per side) with a sector length of 256 (100H) bytes. Sectors are accessed through 16-bit sector addresses, starting with address 0000H. PolyDos automatically translates sector addresses into track/sector numbers.

Single density disks divide each track into 10 sectors, giving a total storage capacity of 700 sectors. Thus, sector addresses should be within the range 0000H-02BBH. To access a single density disk you should refer to drives 0-3 in the G805 version and drives 4-7 in the G809/G815 version.

Double density disks divide each track into 18 sectors, giving a total storage capacity of 1260 sectors. Thus, sector addresses should be within the range 0000H-04EBH. To access a double density disk you should refer to drives 0-3.

3.2 Files

A file is a group of contiguous sectors on a disk. It must be totally contained on a single disk, and files may not overlap or share sectors. The internal format of the file is determined by the file extension and by the programs that read and write the file.

A file is defined by a File Controller Block (FCB) in the disk directory. The FCB contains all information required to locate, access, and delimit the file data on the disk. An FCB consumes 20 bytes, arranged in the following manner:

Name	Offset	Contents
FNAM	0	File name (8 bytes).
FEXT	8	File extension (2 bytes).
FSFL	10	System flags (1 byte).
FUFL	11	User flags (1 byte).
FSEC	12	Sector address (2 bytes).
FNSC	14	Length in sectors (2 bytes).
FLDA	16	Load address (2 bytes).
FEXA	18	Execute address (2 bytes).

where offset is the offset from the start address of the FCB.

3.2.1 FNAM - File name

The FNAM slot contains the file name. The maximum length is 8 characters. The characters are stored in the same order as they are typed, and unused characters are blank filled, i.e. set to

20H. A file name should not contain graphic characters, control characters, blanks, colons, semicolons, periods, or commas.

3.2.2 FEXT - File extension

The file extension is a two-byte field following the file name. The characters in the extension field are stored in the same order as they are typed. An extension should not contain graphic characters, control characters, blanks, colons, semicolons, periods, or commas.

3.2.3 FSFL - System flags

The system flags byte is used to store two one bit flags defining the status of the file:

Bit 0	Lock flag.
Bit 1	Delete flag.

If bit 0 is set the file is considered locked. If bit 1 is set the file is considered deleted. Bits 2-7 are reserved for future expansion.

3.2.4 FUFL - User flags

The user flags byte is never accessed by PolyDos, except when a file is created, which stores a zero in FUFL.

3.2.5 FSEC - Sector address

FSEC contains the 16-bit sector address of the first sector occupied by the file.

3.2.6 FNSC - Length in sectors

FNSC contains a 16-bit value giving the length in sectors of the file.

3.2.7 FLDA - Load address

For machine code program files (extension .GO) FLDA defines the 16-bit memory load address. For other file types this field is normally zeroed, but any value is allowed.

3.2.8 FEXA - Execute address

For machine code program files (extension .GO) FEXA defines the 16-bit memory execution address. For other file types this FCB field is not used, and may contain any value.

3.3 The disk directory

The disk directory is a collection of FCBs and control data used to allocate and retrieve files. The directory is always stored in sectors 0000H to 0003H of a disk. Since the directory is a fixed 1024 bytes in length, the number of FCBs it may contain is limited to 50. The disk directory consists of the following fields:

Name	Addr	Size	Description
DNAME	C400	20	20 character disk name.
NXTSEC	C414	2	Next free sector address.
NXTFCB	C416	2	Next free FCB address.
FCBS	C418	1000	FCB list.

The addresses referred to above are the addresses at which the related field will reside when the directory is read into the directory buffer (DIRBUF, address C400H-C7FFH).

3.3.1 DNAME - Disk name

The disk name is a twenty-character field located at the beginning of the directory. If the disk name is less than 20 characters in length, the remaining bytes are blank filled, i.e. set to 20H.

3.3.2 NXTSEC - Next free sector address

NXTSEC contains the sixteen-bit disk address of the next free sector on the disk. Since files are allocated sequentially NXTSEC is also the number of sectors in use on the disk. When a disk is formatted NXTSEC is set to 0004H, thus reserving 4 sectors for the directory.

3.3.3 NXTFCB - Next free FCB address

NXTFCB contains the sixteen-bit memory address of the first unused FCB in the directory. Note that NXTFCB points to a location within DIRBUF. If the directory is not loaded into DIRBUF you must add an offset to obtain the correct address. When a disk is formatted NXTFCB is set to point at FCBS (C418H).

3.4 Allocating file and directory space

PolyDos allocates space on the disk sequentially for files and FCBs. NXTSEC always points to the first free sector past the used area of the disk. NXTFCB always points past the end of the last FCB in use in the directory. When a file is written to the disk, the data is written starting at the disk address contained in NXTSEC, and NXTSEC is changed to point beyond the last sector of the file. When the FCB is entered into the directory, it is stored at NXTFCB, and NXTFCB is updated to point past the new entry.

Files may not overlap or share sectors, and the order of FCBs in the directory must correspond to the order of the files on the disk. When files are deleted, the corresponding FCB is marked deleted, but the space in the directory, and the data on the disk, is not reclaimed until a PACK command is executed.

3.4.1 Accessing the directory

Accessing the disk directory in memory (in the DIRBUF area) involves the system cell DDRV, which is the drive number of the directory currently in DIRBUF. To access the directory you should follow these steps:

1) Read the directory into DIRBUF by calling the RDIR routine. To force a read even if the directory is already contained within DIRBUF, load a 0FFH into DDRV before calling RDIR.

2) Access the directory, preferrably using the system routines LOOK and ENTER. Note that ENTER automatically writes the updated directory to the disk.

Section 4

System subroutines

4.1 PolyDos routines

PolyDos provides an extensive set of system subroutines to the assembly language programmer. All routines are called using SCALs. Thus, a system routine call only consumes 2 bytes: A RST 18H instruction (DFH) followed by the routine number. PolyDos routines are numbered from 80H to 8FH. None of the system routines uses the alternate register set (AF', HL', DE', and BC') or the index registers (IX and IY). The only registers used are AF, HL, DE, and BC. Errors are reported using the zero-flag and the accumulator (A). If no errors occurred, the zero-flag is set (Z) and the accumulator is zero. Otherwise the zero-flag is clear (NZ), and the accumulator contains a two-digit error code.

4.1.1 DSIZE

Routine number:	80H
Purpose:	Return disk size

Entry:	C:	Drive number
Exit:	HL:	Disk size in sectors
	DE:	Unchanged
	BC:	Unchanged
	AF:	Status

DSIZE will check that C contains a valid drive number, and return the disk size in sectors in HL. Keep your programs implementation independant by using this routine. If the drive number is invalid, a 28 error code will be returned.

4.1.2 DRD

Routine number:	81H
Purpose:	Read sectors

Entry:	HL:	Memory address
	DE:	Disk address
	B:	Number of sectors
	C:	Drive number
Exit:	HL:	Unchanged
	DE:	Unchanged
	BC:	Unchanged
	AF:	Status

DRD will read B sectors from drive C starting at sector DE into memory starting at address HL. Possible error codes are 20, and 23-29.

4.1.3 DWR

Routine number: 82H
Purpose: Write sectors

Entry:	HL:	Memory address
	DE:	Disk address
	B:	Number of sectors
	C:	Drive number
Exit:	HL:	Unchanged
	DE:	Unchanged
	BC:	Unchanged
	AF:	Status

DWR will write B sectors to drive C starting at sector DE from memory starting at address HL. Possible error codes are 20-29.

4.1.4 RDIR

Routine number: 83H
Purpose: Read directory

Entry:	C:	Drive number
Exit:	HL:	Unchanged
	DE:	Unchanged
	BC:	Unchanged
	AF:	Status

RDIR will read the directory of drive C into the directory buffer (DIRBUF) and store the drive number in DDRV. However, RDIR first checks to see if the directory is already in DIRBUF, by comparing C to the contents of DDRV. If so, RDIR returns without accessing the disk. To force a read, load 0FFH into DDRV. Possible error codes are 20, and 23-29.

4.1.5 WDIR

Routine number: 84H
Purpose: Write directory

Entry:	No parameters required	
Exit:	HL:	Unchanged
	DE:	Unchanged
	BC:	Unchanged
	AF:	Status

WDIR writes the directory contained in DIRBUF to the disk directory sectors (0000H-0003H) on drive DDRV. WDIR should only be called when changes has been made to the directory. Possible error codes are 20-29.

4.1.6 CFS

Routine number: 85H
Purpose: Convert file specifier

Entry:	HL:	Address of FCB

	DE:	Address of text buffer
	B:	Flags:
		B0=1: Name optional
		B1=1: Extension optional
		B2=1: Drive number optional
Exit:	HL:	Unchanged
	DE:	Address of next character in text buffer
	B:	Flags:
		B0=1: No name
		B1=1: No extension
		B2=1: No drive number
	C:	Drive number
	AF:	Status

CFS converts a file specifier to FCB format. It is called with HL pointing to an FCB and DE pointing to the first character in the file specifier in the text buffer. CFS will only load values into FNAM and FEXT of the FCB. Hence, the FCB need only be 10 bytes long. Upon entry B contains three flags: If bit 0 is set, the file name is optional. If bit 1 is set, the extension is optional, and if bit 2 is set, the drive number is optional. If elements are missing from the file specifier which are not optional, an error code will be returned. If no drive number is specified, and the drive number is optional, the master drive number (MDRV) will be returned in C. If no name and/or no extension is specified, FNAM and/or FEXT will remain unchanged, allowing you to load default values into these slots before calling CFS. The following characters are considered delimiters: A blank, a comma, a semicolon, a carriage return, a TAB, and a zero. Upon exit, DE points to the next non-blank character in the text buffer following the file specifier, and B contains three flags: If bit 0 is set, no file name was given. If bit 1 is set, no extension was given, and if bit 2 is set, no drive number was given, in which case the master drive number has been loaded into C. Possible error codes are 10-15.

Below is shown the code needed to input a file name and convert it to FCB format:

```
START:  RST     PRS                 ;Prompt user
        DB      'File name? ',0
        SCAL    ZINLIN              ;Read input line
        LD      HL,11               ;Point to first character
        ADD     HL,DE
        EX      DE,HL               ;Pointer to DE
        LD      HL,'T'+'X'*256      ;Insert default extension
        LD      (S1FCB+FEXT),HL
        LD      HL,S1FCB            ;Point to FCB
        LD      B,110B              ;Extension/drive optional
        SCAL    ZCFS                ;Convert file specifier
        SCAL    ZCKER               ;Check for error
```

If no errors occur C contains the drive number and S1FCB contains file specifier converted into FCB format.

4.1.7 LOOK

Routine number: 86H
Purpose: Lookup file in directory

```
Entry:  HL:     Lookup FCB address
        DE:     Previous directory FCB address
        B:      Flags:
                B0=1: Don't match name
                B1=1: Don't match extension
                B4=1: Copy directory FCB to lookup FCB
                B5=1: Include locked files
                B6=1: Include deleted files
                B7=1: Not first look
Exit:   HL:     Unchanged
        DE:     Directory FCB address
        B:      Bit 7 is set to 1
        C:      Unchanged
        AF:     Status
```

LOOK will look up a file in the directory currently contained in DIRBUF. Upon entry HL contains the address of a lookup FCB with FNAM and FEXT initialized to the name and extension of the file you want to look up. B contains six one-bit flags:

Bit 0 If set, LOOK will not attempt to match the file name.
Bit 1 If set, LOOK will not attempt to match the file extension.
Bit 4 If set, LOOK will copy the matching FCB from the directory to the lookup FCB. In this case 20 bytes should be reserved for the lookup FCB (otherwise 10 will do).
Bit 5 If this bit is set it indicates that LOOK should include locked files.
Bit 6 If this bit is set it indicates that LOOK should include deleted files.
Bit 7 If this bit is clear LOOK will start the lookup from the first FCB in the directory. If not, LOOK will start at the FCB following the one pointed to by DE. This bit is always set to one by LOOK before it returns.

If a matching FCB is found in the directory, DE is set to point at the first byte of that FCB. Bit 7 in B provides a way of looking up family file specifiers through multiple calls to LOOK. At the first call bit 7 should be cleared, telling LOOK to start at the beginning of the directory. Before returning LOOK sets to one bit 7 in B. Provided that B and DE are left unchanged the next call to LOOK will continue from the next FCB instead of the first FCB. When LOOK returns an error, all files matching your input parameters have been processed, and the calls should be discontinued. Note that if bit 0 in B as well as bit 1 are set to one LOOK will include all files in the directory. The only possible error code returned by LOOK is 30.

Below in shown a program which will input a file specifier, look it up in the disk directory of the drive specified, and, if no errors occur, read it into memory starting at its load address:

```
    START:  RST     PRS             ;Prompt user
            DB      'Load which file? ',0
            SCAL    ZINLIN          ;Read input line
            LD      HL,17           ;Point to first character
            ADD     HL,DE
            EX      DE,HL           ;Pointer to DE
```

```
          LD      HL,S1FCB         ;Point to FCB
          LD      B,110B           ;Extension/drive optional
          SCAL    ZCFS             ;Convert file specifier
          SCAL    ZCKER            ;Check for error
          SCAL    ZRDIR            ;Read directory
          SCAL    ZCKER            ;Check for error
          SET     4,B              ;Copy directory FCB
          SET     5,B              ;Include locked files
          SCAL    ZLOOK            ;Lookup
          SCAL    ZCKER            ;Check for error
          LD      HL,(S1FCB+FLDA)  ;Pick up load address
          LD      DE,(S1FCB+FSEC)  ;Pick up sector address
          LD      A,(S1FCB+FNSC)   ;Get number of sectors
          LD      B,A              ;Put in B
          SCAL    ZDRD             ;Read the file
          SCAL    ZCKER            ;Check for error
```

Here is antoher program that will input a drive number and count the number of deleted files on that disk.

```
START:    RST     PRS              ;Prompt user
          DB      'Which drive? ',0
          SCAL    ZINLIN           ;Read input line
          LD      HL,13            ;Point to drive number
          ADD     HL,DE
          LD      A,(HL)           ;Get drive number
          SUB     '0'              ;Adjust
          LD      C,A              ;Put in C
          SCAL    ZRDIR            ;Read directory
          SCAL    ZCKER            ;Check for error
          LD      B,01100011B      ;Initialize flags
          LD      C,0              ;Clear counter
COUNT:    SCAL    ZLOOK            ;Lookup
          JR      NZ,DONE          ;Error => done
          LD      HL,FSFL          ;Point to FSFL
          ADD     HL,DE
          BIT     1,(HL)           ;Deleted file?
          JR      Z,COUNT          ;No => skip
          LD      A,C              ;Increment counter
          INC     A
          DAA
          LD      C,A
          JR      COUNT            ;Try next
DONE:     RST     PRS              ;Now print result
          DB      'Files deleted: ',0
          LD      A,C
          SCAL    ZB2HEX
          SCAL    ZCRLF
```

Note that as LOOK is requested to include all files (bit 1 and bit 0 in B are ones), HL need not point to an FCB upon entry (the name and the extension are never checked anyway).

4.1.8 ENTER

Routine number: 87H
Purpose: Enter FCB into directory

Entry: HL: FCB address

Exit:	HL:	Unchanged
	DE:	Directory FCB address
	BC:	Unchanged
	AF:	Status

Call ENTER to enter a new FCB into the directory currently in DIRBUF. At the time of the call HL should point to a copy of the FCB to be entered. ENTER first calls LOOK to see if the file already exists. If so, it returns with DE pointing to the existing FCB in the directory and an error code 31 in A. Your program may now decide to print an error message, or to delete the file, by setting high bit 1 of FSFL in the FCB pointed to by DE, and call ENTER once more. Once ENTER's call to LOOK results in an error (indicating that there are no active files of the name you specify within the directory) ENTER moves on to entering the FCB in the directory. If the directory is full ENTER reports an error. Otherwise it copies your FCB to the next free directory FCB. Next it picks up the value in FNSC and adds it to NXTSEC, making NXTSEC point to the next free sector on the disk. ENTER then calls WDIR to write the updated directory to the disk, and returns.

When you create a file it should always be written to the disk starting at the sector address contained in NXTSEC in the directory of that disk.

Below is shown a subroutine which will enter into the directory the FCB pointed to by HL. If active files exist of the same name and extension they will be deleted, unless they are locked, in which case an error 33 is returned.

```
ENTR:   SCAL    ZENTER          ;Try enter the file
        RET     Z               ;Ok => return
        CP      31H             ;Existing file error?
        RET     NZ              ;No => return
        PUSH    HL              ;Save FCB address
        LD      HL,FSFL         ;Point to system flag
        ADD     HL,DE           ;byte of directory FCB
        BIT     0,(HL)          ;Locked file?
        LD      A,33H           ;Error 33 if so
        JR      NZ,SKIP         ;Yes => return
        SET     1,(HL)          ;Delete the file
        POP     HL              ;Restore FCB address
        JR      ENTR            ;Go retry
SKIP:   POP     HL              ;Restore FCB address
        RET                     ;Return
```

The program shown below will save the contents of memory between 1000H and 2000H (10H sectors) in a file using a file name input by the user. The above routine is used to enter the file in the disk directory.

```
START:  RST     PRS             ;Prompt user
        DB      'File name? ',0
        SCAL    ZINLIN          ;Read input line
        LD      HL,11           ;Point to first character
        ADD     HL,DE
        EX      DE,HL           ;Pointer to DE
        LD      HL,S1FCB        ;Point to FCB
        LD      B,100B          ;Drive number optional
        SCAL    ZCFS            ;Convert file specifier
        SCAL    ZCKER           ;Check for error
        SCAL    ZRDIR           ;Read directory
        SCAL    ZCKER           ;Check for error
        LD      HL,0            ;Clear flag bytes
        LD      (S1FCB+FSFL),HL
        LD      HL,(NXTSEC)     ;Get next free sector
        LD      (S1FCB+FSEC),HL ;Store as sector address
        EX      DE,HL           ;Put into DE
        LD      HL,10H          ;Initialize file length
        LD      (S1FCB+FNSC),HL
        LD      HL,1000H        ;Initialize load and
        LD      (S1FCB+FLDA),HL ;execute addresses
        LD      (S1FCB+FEXA),HL
        LD      B,10H           ;Write 16 sectors
        SCAL    ZDWR
        SCAL    ZCKER           ;Check for error
        LD      HL,S1FCB        ;Point to FCB
        CALL    ENTR            ;Enter FCB in directory
```

4.1.9 COV

Routine number: 88H
Purpose: Call an overlay

COV and COVR provide the mechanisms for invokind overlay subroutines. These facilities are the cornerstones on which the PolyDos operating system is built. The overlay you invoke may or may not be in memory before you call it. Both the entering and the exiting register contents are defined by the overlay. Commom system conventions for overlays that process more than one function suggest that the function code be passed in A. The invokation of an overlay takes the form of the example below (assuming that registers and other entry parameters have already been set up to hold the proper contents):

```
        SCAL    COV
        DB      'Emsg'
```

Overlay names are defined to be four characters long, and the overlay name must follow the call to COV or COVR. If the overlay is not currently in memory it is read into memory from the master drive. The overlay is always entered at OVRLY (C804H). When the overlay executes a return instruction (RET) control is transferred to the code immediately following the overlay name in the call.

If you try to invoke a non-existing overlay, PolyDos will call Emsg to report the error, and return to the command mode.

Both COV and COVR invoke a function in an overlay, which may no be in memory at the time, and both return control to the calling program just after the overlay name following the call to COV or COVR. The only difference between COV and COVR is that COVR "remembers" the overlay currently in the overlay area (by pushing its name onto the stack) and restores that overlay before returning to the caller, while COV does not. Hence, COVR can be used within one overlay to call a function in another overlay, since the original overlay is restored when the called overlay returns. As COV does not restore the overlay currently in the overlay area it should only be used from programs outside the overlay area.

4.1.10 COVR

Routine number: 89H
Purpose: Call an overlay and restore

See COV for a description of this system service and how it differs from COV. Also see section 5 on overlays.

4.1.11 CKER

Routine number: 8AH
Purpose: Check for error

Entry: A: Error status
Exit: All registers unchanged

CKER is called with an error status in A. First A is checked to be zero. If so CKER returns immediately, as zero indicates no error. Otherwise the error code is stored in ERRCOD, CFMA is called to abort the command file mode, and the Emsg overlay is invoked to output an error message, whereafter control is transferred to the MRET routine. If PolyDos cannot invoke Emsg for some reason, it outputs:

```
(Error xx)
```

where xx is the error code, and returns control to NAS-SYS.

4.1.12 CKBRK

Routine number: 8BH
Purpose: Check for break

Entry: No parameters required
Exit: All registers but A unchanged

CKBRK does a fast scan of the keyboard to see if CTRL/SHIFT/@ are held down. If not, it returns immediately with all registers but A unchanged. If CTRL/SHIFT/@ are held down CKBRK calls CFMA to abort the command file mode and transfers control to the address contained in the system variable BREAK.

4.1.13 CFMA

Routine number: 8CH
Purpose: Abort command file mode

Entry: No parameters required
Exit: All registers but A unchanged

CFMA examines CFFLG to determine if PolyDos is in the command file mode. If CFFLG is zero, CFMA returns immediately. If not, a zero is loaded into CFFLG to abort the command file mode, and the message:

(Cmdf abort)

is displayed followed by a carriage return.

4.1.14 SSCV

```
Routine number: 8DH
Purpose:        Set SCAL vector

Entry:  HL:     New SCAL address
Exit:   HL:     Previous SCAL address
        DE:     Junk
        BC:     Junk
        AF:     Junk
```

Call SSCV to modify a jump vector in the SCAL address table. The call must be followed by one byte giving the number of the routine. Upon entry HL should hold the new routine address. Upon exit HL contains the address that was replaced. Below is shown an example of SSCV use:

```
LD      HL,XMRET        ;Get new MRET address
SCAL    ZSSCV           ;Insert in SCAL table
DB      ZMRET
LD      (MRETA),HL      ;Save previous address
```

when the above code is executed all calls to the MRET routine will be directed to XMRET.

4.1.15 JUMP

```
Routine number: 8EH
Purpose:        Execute jump table

Entry:  A:      Jump table key
Exit:   All registers unchanged
```

The call to JUMP should be followed by a list of addresses (words). The accumulator holds the number of the routine to jump to, zero corresponding to the first address. Consider the following example:

```
SCAL    ZJUMP
DW      START
DW      LBL1
DW      STOP
```

If A contains 0 JUMP will transfer control to START. If A contains 1 control is given to LBL1, and if A is 2 JUMP jumps to STOP. In this example A should never hold other values than 0, 1, or 2, as the table only defines addresses for these values.

4.1.16 POUT

Routine number:		8FH
Purpose:		Output A to printer
Entry:	A:	Character to be output
Exit:	HL:	Junk
	DE:	Junk
	BC:	Junk
	AF:	Unchanged

POUT outputs the character in A to the printer. It provides extensive forms handling, through the user defined forms handling parameters given in the information file (PLPP, PBMG, PCPL, and PLMG, see section 8.1.3). To output a character POUT calls the low level printer output routine (PCHR, see section 8.1.5) also contained in the information file. This enables you to define the interfacing characteristics of your printer. POUT will automatically supply a line-feed (LF=0AH) whenever a carriage return (CR=0DH) is output. If printing a CR/LF sequence causes the print head to be positioned at the bottom of a form, a bottom margin, consisting of BMRG CR/LFs will be output to PCHR. If a form-feed (FF=0CH) is output, POUT translates it into a suitable number of CR/LFs, depending on the number of lines already printed on that page. Tabulator characters (TAB=09H) will be converted into enough blanks to move the print head to the next multiple of 8 column. Other characters will be transmitted directly to PCHR, unless PCPL characters have already been printed on that line, in which case the character is ignored. However, if the print head is at the first column of a line when a character is to be output, LMRG blanks are output at first to provide a left margin. POUT maintains two counters in the workspace area giving the exact position of the print head. PLCT (location C017H) holds the current line number, zero being the first line on a page. PPOS (location C018H) holds the column number, zero being the first column on a line.

4.2 NAS-SYS routines

In addition to the routines described in section 4.1 three routines has been added to make NAS-SYS 1 compatible with NAS-SYS 3. These are RKBD, SP2, and SCALI. Furthermore, the routines MRET, CRT, BLINK, and NNIM will function slightly different as described in this section.

4.2.1 MRET

A call to MRET (routine number 5BH) transfers control to the system executive (the Exec overlay). If Exec is not in the overlay area when MRET is called, it is read from the master drive.

4.2.2 CRT

The CRT routine (routine number 65H) has been modified to support TAB characters.

4.2.3 NNIM

The input table activated by NNIM (routine number 78H) will call RKBD instead of KBD to provide a repeating keyboard.

4.2.4 BLINK

BLINK (routine number 7BH) has been modified to support the command file mode. Read more about this in section 7.

4.2.5 RKBD

Routine number: 7DH
Purpose: Input from keyboard with repeat

Entry: No parameters required
Exit: HL: Junk
DE: Junk
BC: Junk
A: If carry set, input character
F: Carry set if character

RKBD scans the keyboard once. If a key has been pressed since the last scan, or if the delay counter times out, RKBD returns with carry set and a character in A. The initial delay and the repeat delay can be adjusted by modifying RKLON and RKSHO in the information file area.

4.2.6 SP2

Routine number: 7EH
Purpose: Print two spaces

Entry: No parameters required
Exit: HL: Unchanged
DE: Unchanged
BC: Unchanged
A: 20H (ASCII space)

SP2 outputs two spaces by calling the SPACE routine twice.

4.2.7 SCALI

Routine number: 7FH
Purpose: SCAL indirect

Entry: E: Subroutine number
Exit: Defined by subroutine

Call SCALI to execute an indirect call to a system subroutine. The number of the routine you want to invoke should be contained in the E register.

Section 5

Overlays

The internal structure and flexibility of the PolyDos disk operating system is based on the overlay machanism.

The overlay area resides from C800H to CFFFH. Overlays should be assembled for this area, and may not exceed 2K bytes in size. Overlay names are defined to be four characters long. The first four bytes (C800H-C803H) of an overlay should contain its name, which must match the file name. An overlay may use portions of the overlay area itself for buffers or data. Remember, however, that such data is lost if another overlay is invoked.

As an example of an overlay the assembly listing of the system error message writer overlay (Emsg) is given in appendix C.

5.1 File handler overlays

File handler overlays are a special type of overlays. They serve to perform the functions that need to be done when a file of their associated type is executed. File handler overlays are invoked by Exec when you try to execute a file of an unknown type (extension). The first two characters of the overlay name are the two characters forming the extension of its associated file type, thus defining which type of files the overlay will handle. The last two characters of the name must be 'fh', indicating that the overlay is a file handler. The extension is OV indicating that the file is an overlay.

Let us assume that you have a file on your disk called GRONK.CM. When you attempt to execute it, Exec does not know what to do with it, as its extension is not among the standard file types (TX and GO). Instead of giving an error message Exec tries to locate an overlay called CMfh.OV. If CMfh exists on the master drive it is loaded into the overlay area an executed.

When Exec invokes the execute file function in a file handler overlay, the A register is zero, CLINP points to the next non-blank character following the file specifier, and S1FCB contains a copy of the directory FCB of the file. The drive number of the file in stored in the first byte of S2FCB. As the accumulator is always cleared when the overlay is invoked to execute a file, the accumulator should be use to distinguish between the execute file function and other overlay functions.

Below is shown an example of a file handler overlay for files of extension CM, thus called CMfh. Before loading and executing the file you specify CMfh will load into memory a file called CMfun.OB. In this case CMfun might be a collection of runtime routines that need be present in memory to run files of type CM.

```
        REFS    SYSEQU          ;Get symbols from SYSEQU
        REF                     ;Load all symbols

        ORG     OVAREA          ;Define origin
        IDNT    $,0                 ;Define load address

        DB      'CMfh'              ;Overlay name

        LD      HL,CMFCB            ;Point to CMFCB
        LD      B,00110000B         ;Copy FCB from directory
        SCAL    ZLOOK               ;Look on master drive
        SCAL    ZCKER               ;Check for error
        LD      HL,(CMFCB+FLDA)     ;Pick up load address
        LD      DE,(CMFCB+FSEC)     ;Pick up sector address
        LD      A,(CMFCB+FNSC)      ;Pick up length
        LD      B,A                 ;Put in B
        LD      A,(MDRV)            ;Read from master drive
        LD      C,A
        SCAL    ZDRD                ;Load CMfun
        SCAL    ZCKER               ;Check for error
        LD      HL,(S1FCB+FLDA)     ;Pick up load address
        LD      DE,(S1FCB+FSEC)     ;Pick up sector address
        LD      A,(S1FCB+FNSC)      ;Pick up length
        LD      B,A                 ;Put in B
        LD      A,(S2FCB)           ;Pick up drive number
        LD      C,A                 ;Put in C
        SCAL    ZDRD                ;Load the file
        SCAL    ZCKER               ;Check for error
        LD      HL,(S1FCB+FEXA)     ;Pick up execute address
        JP      (HL)                ;Go there

CMFCB:  DB      'CMfun   OB'        ;Name and extension
        DS      10                  ;Attributes buffer

        END
```

Note that as PolyDos always looks up overlays on the master drive, the directory of the master drive is contained in DIRBUF whenever an overlay is invoked. Therefore, the CMfh file handler shown above need not call RDIR before calling LOOK when it is to look up CMfun.

Section 6

File formats

This section defines the format of the following standard file types:

GO	Machine code program files
TX	Text fikes
OV	Overlay files

The above file types are 'known' to the system (and therefore defineable in this manual), i.e. they need no file handlers to be executed (remember though that you cannot execute an overlay file).

6.1 Machine code program files

The length of a machine code program file is given by the number of sectors required to hold all of the code forming the program. If the length of a program is 890H bytes the machine code program file will be nine sectors long. The first eight sectors and the 90H first bytes of the ninth sector contains the actual code. The remaining bytes of the last sector are undefined (and uninteresting).

6.2 Text files

The length of a text file is given by the number of sectors required to hold all of the text. Remaining bytes of the last sector are set to zero. These fillers must be stripped of when the file is processed.

6.3 Overlay files

The format of an overlay file is the same as that of a machine code program file. Remember that overlay files may not exceed 2K bytes in size, and that the build-in name (contained in the first four bytes of the overlay) should always match the file name.

Section 7

The command file mode

When PolyDos is in the command file mode all input, normally entered from the keyboard, will be taken from a text file instead. The command file mode only affects the BLINK routine, i.e. it only applies where you would normally see a blinking cursor. When BLINK is called to input a character it tests the value of CFFLG to see if the command file mode is active. If CFFLG is zero, BLINK acts as usual, blinking the cursor until a key is pressed. If CFFLG is not zero, BLINK will obtain its input character from a text file on the disk. To obtain the character BLINK uses the following procedure:

1) If CFSBP equals zero, thus indicating that the command file sector buffer is empty, the sector counter CFNSC is loaded and checked to be zero, in which case the command file mode is terminated by loading zero into CFFLG. If CFNSC is not zero it is decremented and a sector is loaded from drive CFDRV sector CFSEC into the sector buffer SECBUF, whereafter CFSEC is incremented.

2) The character pointed to by CFSBP (CFSBP is a one-byte pointer within SECBUF) is loaded into the accumulator and CFSBP is incremented. If the character is zero, it is considered a filler and skipped by repeating (1) and (2).

As you see from the above discussion PolyDos only knows the sector address and the drive number of the command file being executed. It does not know the name of the file and is tehrefore unable to detect external events such as insertion of another disk or overwriting of the file. It is up to you to make sure that these events does not occur or to deactivate the command file mode before they do. Below is shown the code needed to activate the command file mode using a file called CMDFILE.TX on the master drive:

```
CFCB:   DB      'CMDFILE TX'     ;Name and extension
        DS      10               ;Attributes buffer

START:  LD      A,(MDRV)         ;Get master drive number
        LD      (CFDRV),A        ;Put in CFDRV
        LD      C,A              ;Put in C
        SCAL    ZRDIR            ;Read directory
        SCAL    ZCKER            ;Check for error
        LD      HL,CFCB          ;Point to FCB
        LD      B,00110000B      ;Copy FCB from directory
        SCAL    ZLOOK            ;Lookup
        SCAL    ZCKER            ;ChecK for error
        LD      HL,(CFCB+FSEC)   ;Pick up sector address
        LD      (CFSEC),HL       ;Put in CFSEC
        LD      A,(CFCB+FNSC)    ;Pick up length
        LD      (CFNSC),A        ;Put in CFNSC
        XOR     A                ;Indicate that the sector
        LD      (CFSBP),A        ;buffer is empty
        DEC     A                ;Activate the command
        LD      (CFFLG),A        ;file mode
```

Section 8

The information file

As you have learned from the PolyDos Users Guide, a file called Info.IN os brought into memory (i.e. the information file area, addresses C200H-C2FFH) by Exec when PolyDos is booted.

8.1 Information file parameters

The information file contains various informations likely to vary between different systems. These are:

Cursor characteristics
Repeat keyboard keyboard delays
Printer forms parameters
Printer initialization string
Low level printer output routine

The above parameters are described in the following sections, which also define the values selected by default, i.e. the values loaded into the variables if Info.IN is not present on the master drive. For a quick reference refer to section 2.3.

8.1.1 Cursor characteristics

Two information file variables define the cursor characteristics. CURCHR (location C200H) holds the ASCII value of the cursor character, and CURBLR (location C201H) holds the cursor blink rate. The default values are CURCHR=5FH and CURBLR=C0H.

8.1.2 Repeat keyboard delays

Two information file variables define the repeat keyboard delays. RKLON (locations C202H-C203H) hold the initial delay, and RKSHO (C204H-C205) hold the repeat delay. Both values are 16-bit stored in standard byte reversed format. The default values are RKLON=0200H, and RKSHO=0080H.

8.1.3 Printer forms parameters

Four one-byte information file variables define the printer forms. PLPP (location C210H) gives the overall forms length in lines. PBMG (location C211H) gives the bottom margin, i.e. the number of blank lines to print to skip perforations on fan-fold paper. PBMG is included in PLPP. Thus, PLPP-PBMG lines of text will be printed on each page, before skipping to the next page. PCPL (location C212H) gives the overall line length in characters. PLMG (location C213H) gives the number of blanks to print at the beginning of each line to provide a left margin. PLMG is included in PCPL. Thus, PCPL-PLMG characters can be printed on each line. The default values are PLPP=255, PBMG=0, PCPL=255, and PLMG=0.

8.1.4 Printer initialization string

When the information file has been loaded off the disk, the initialization string is output to the printer. INSLEN (location C214H) defines the length of the initialization string (maximum is 43 characters), and INSTR (locations C215H-C23FH) contain the actual string. Each character (if any) is output by a call to the PCHR routine, which starts in location C240H. If INSTR contains any characters they normally form a control sequence to put the printer into another mode than its default.

8.1.5 Low level printer output routine

The low level printer output routine has its entry point at PCHR (location C240H). PCHR should contain the code needed to output the accumulator to the printer. The routine should end with a return (RET) instruction, and it need not save any registers (except for the alternative registers and the index registers which are never touched by PolyDos). The default value is a return instruction.

8.2 A sample information file

Below is shown an example of an information file. The low level printer output routine will control a serial printer with a BUSY (active HIGH) line connected to TP3 on the NASCOM 2 main PCB (TP3 is bit 7 in port 0).

```
        REFS    SYSEQU          ;Get symbols from SYSEQU
        REF                     ;Get all symbols

        ORG     INFOFA          ;Define origin
        IDNT    $,0             ;Define load address

        DB      5FH,0C0H        ;CURCHR,CURBLR
        DW      200H,80H        ;RKLON,RKSHO

        ORG     INFOFA+10H
                  6 [illegible] 2   [illegible]
        DB      72,8,122,10     ;PLPP,PBMG,PCPL,PLMG
        DB      2,ESC,14H       ;INSLEN,INSTR

        ORG     INFOFA+40H      ;PCHR

        PUSH    AF              ;Save char
BUSY:   IN      A,(0)           ;Read port 0
        RLA                     ;Bit 7 high?
        JR      C,BUSY          ;Yes => busy
        POP     AF              ;Restore char
        SCAL    ZSRLX           ;Print it
        RET

        END
```

```
                ;-----------------------------------------------
                ;
                ;       PolyDos 2.0
                ;
                ;       SYSEQU
                ;       The system equate file
                ;
                ;       By Anders Hejlsberg
                ;       Copyright (C) 1981
                ;       PolyData microcenter ApS
                ;
                ;-----------------------------------------------


                ;Memory organization equates

080A            VRAM:   EQU     0080AH  ;Video RAM addr
1000            STACK:  EQU     01000H  ;Addr of program stack
1000            RAM:    EQU     01000H  ;Addr of program RAM
C000            TOP:    EQU     0C000H  ;Highest RAM addr
D000            PDCROM: EQU     0D000H  ;Addr of PolyDos Controller

                ;NAS-SYS restarts

0000            RESET:  EQU     00H     ;System RESET
0008            RIN:    EQU     08H     ;Input A
0010            RCALH:  EQU     10H     ;Relative call
0018            SCALH:  EQU     18H     ;Subroutine call
0020            BRKPT:  EQU     20H     ;Breakpoint
0028            PRS:    EQU     28H     ;Print string
0030            ROUT:   EQU     30H     ;Output A
0038            RDEL:   EQU     38H     ;Delay

                ;NAS-SYS subroutines

000D            STMON:  EQU     000DH

                ;NAS-SYS SCAL subroutines

005B            ZMRET:  EQU     5BH     ;Return to system
005C            ZSCALJ: EQU     5CH     ;SCAL routine nbr A
005D            ZTDEL:  EQU     5DH     ;Delay apx 2 seconds
005E            ZFFLP:  EQU     5EH     ;Flip/flop bits in port 0
005F            ZMFLP:  EQU     5FH     ;Flip motor bit
0060            ZARGS:  EQU     60H     ;Get arguments
0061            ZKBD:   EQU     61H     ;Scan keyboard
0062            ZIN:    EQU     62H     ;Scan input devices
0063            ZINLIN: EQU     63H     ;Input a line
0064            ZNUM:   EQU     64H     ;Convert hexnumber
0065            ZCRT:   EQU     65H     ;Output to CRT
0066            ZTBCD3: EQU     66H     ;Output HL in hex with cksm
0067            ZTBCD2: EQU     67H     ;Output A in hex with cksm
0068            ZB2HEX: EQU     68H     ;Output A in hex
0069            ZSPACE: EQU     69H     ;Output space
006A            ZCRLF:  EQU     6AH     ;Output CR
006B            ZERRM:  EQU     6BH     ;Write error message
006C            ZTX1:   EQU     6CH     ;Output HL and DE in hex
006D            ZSOUT:  EQU     6DH     ;Output string to serial
006E            ZXOUT:  EQU     6EH     ;Output to external
006F            ZSRLX:  EQU     6FH     ;Output to serial
0070            ZSRLIN: EQU     70H     ;Input from serial
0071            ZNOM:   EQU     71H     ;New output table
```

```
0072           ZNIM:   EQU     72H     ;New input table
0073           ZATE:   EQU     73H     ;Execute routine table
0074           ZXKBD:  EQU     74H     ;Input from external
0075           ZUOUT:  EQU     75H     ;Output to user routine
0076           ZUIN:   EQU     76H     ;Input from user routine
0077           ZNNOM:  EQU     77H     ;Normal output table
0078           ZNNIM:  EQU     78H     ;Normal input table
0079           ZRLIN:  EQU     79H     ;Read and convert a line
007A           ZB1HEX: EQU     7AH     ;Output hexdigit
007B           ZBLINK: EQU     7BH     ;Input w. blinking cursor
007C           ZCPOS:  EQU     7CH     ;Calculate cursor pos
007D           ZRKBD:  EQU     7DH     ;Scan keyboard with repeat
007E           ZSP2:   EQU     7EH     ;Print two spaces
007F           ZSCALI: EQU     7FH     ;Call subroutine <E>

               ;PolyDos SCAL routines

0080           ZDSIZE: EQU     80H     ;Disk size
0081           ZDRD:   EQU     81H     ;Disk read
0082           ZDWR:   EQU     82H     ;Disk write
0083           ZRDIR:  EQU     83H     ;Read directory
0084           ZWDIR:  EQU     84H     ;Write directory
0085           ZCFS:   EQU     85H     ;Convert file specifier
0086           ZLOOK:  EQU     86H     ;Lookup file in directory
0087           ZENTER: EQU     87H     ;Enter file in directory
0088           ZCOV:   EQU     88H     ;Call overlay
0089           ZCOVR:  EQU     89H     ;Call overlay and restore
008A           ZCKER:  EQU     8AH     ;Check for error
008B           ZCKBRK: EQU     8BH     ;Ckeck for break
008C           ZCFMA:  EQU     8CH     ;Command file mode abort
008D           ZSSCV:  EQU     8DH     ;Set SCAL vector
008E           ZJUMP:  EQU     8EH     ;Jump table execution
008F           ZPOUT:  EQU     8FH     ;Printer output

               ;ASCII control characters

0008           BS:     EQU     08H     ;Backspace
0009           TAB:    EQU     09H     ;Tabulate
000A           LF:     EQU     0AH     ;Linefeed
000C           FF:     EQU     0CH     ;Formfeed
000D           CR:     EQU     0DH     ;Carriage return
0011           CUL:    EQU     11H     ;Cursor left
0012           CUR:    EQU     12H     ;Cursor right
0013           CUU:    EQU     13H     ;Cursor up
0014           CUD:    EQU     14H     ;Cursor down
0015           CSL:    EQU     15H     ;Delete character
0016           CSR:    EQU     16H     ;Insert character
0017           CH:     EQU     17H     ;Cursor home
0018           CCR:    EQU     18H     ;Newline
001B           ESC:    EQU     1BH     ;Clear line

               ;FCB offsets
                                                         S1FCB     S2FCB
0000           FNAM:   EQU     0       ;File name          C055      C069
0008           FEXT:   EQU     8       ;Extension          C05D      C071
000A           FSFL:   EQU     10      ;System flags       C05F      C073
000B           FUFL:   EQU     11      ;User flags         C060      C074
000C           FSEC:   EQU     12      ;Sector address     C061      C075
000E           FNSC:   EQU     14      ;Number of sectors  C063      C077
0010           FLDA:   EQU     16      ;Load address       C065      C079
0012           FEXA:   EQU     18      ;Execute address    C067      C07B
```

```
                          ;NAS-SYS workspace

0C00                              ORG      0C00H

0C00 + 0001         PORTO:   DS       1          ;State of output port 0
0C01 + 0009         KMAP:    DS       9          ;State of keyboard
0C0A + 0001         ARGC:    DS       1          ;Last processed routine
0C0B + 0001         ARGN:    DS       1          ;Number of arguments
0C0C + 0002         ARG1:    DS       2          ;Argument 1
0C0E + 0002         ARG2:    DS       2          ;Argument 2
0C10 + 0002         ARG3:    DS       2          ;Argument 3
0C12 + 0002         ARG4:    DS       2          ;Argument 4
0C14 + 0002         ARG5:    DS       2          ;Argument 5
0C16 + 0002         ARG6:    DS       2          ;Argument 6
0C18 + 0002         ARG7:    DS       2          ;Argument 7
0C1A + 0002         ARG8:    DS       2          ;Argument 8
0C1C + 0002         ARG9:    DS       2          ;Argument 9
0C1E + 0002         ARG10:   DS       2          ;Argument 10
0C20 + 0001         NUMN:    DS       1          ;Nbr of chars in value
0C21 + 0002         NUMV:    DS       2          ;Converted value
0C23 + 0002         BRKADR:  DS       2          ;Breakpoint address
0C25 + 0001         BRKVAL:  DS       1          ;Breakpoint value
0C26 + 0001         CONFLG:  DS       1          ;-1 if E command used
0C27 + 0001         KOPT:    DS       1          ;Keyboard options
0C28 + 0001         XOPT:    DS       1          ;External options
0C29 + 0002         CURSOR:  DS       2          ;Cursor address
0C2B + 0001         ARGX:    DS       1          ;Last command letter
0C2C + 0035                  DS       53         ;NAS-SYS stack
0C61                MONSTK:  EQU      $
0C61 + 0002         RBC:     DS       2          ;Register BC save area
0C63 + 0002         RDE:     DS       2          ;Register DE save area
0C65 + 0002         RHL:     DS       2          ;Register HL save area
0C67 + 0002         RAF:     DS       2          ;Register AF save area
0C69 + 0002         RPC:     DS       2          ;Program counter save area
0C6B + 0002         RSP:     DS       2          ;Stack pointer save area
0C6D + 0002         KTABL:   DS       2          ;Length of keyboard table
0C6F + 0002         KTAB:    DS       2          ;Address of keyboard table
0C71 + 0002         STAB:    DS       2          ;Start of routine table
0C73 + 0002         OUTTA:   DS       2          ;Start of output table
0C75 + 0002         INTA:    DS       2          ;Start of input table
0C77 + 0001         UOUTJ:   DS       1          ;Jump instruction
0C78 + 0002         UOUTA:   DS       2          ;User output routine addr
0C7A + 0001         UINJ:    DS       1          ;Jump instruction
0C7B + 0002         UINA:    DS       2          ;User input routine addr
0C7D + 0001         NMIJ:    DS       1          ;Jump instruction
0C7E + 0002         NMIA:    DS       2          ;NMI handler routine addr

                          ;PolyDos workspace

C000                WORKSP:  ORG      TOP+000H      ;WORKSPACE

C000 + 0001         MDRV:    DS       1          ;Master drive
C001 + 0001         DDRV:    DS       1          ;Directory drive
C002 + 0001         DRVCOD:  DS       1          ;Drive code
C003 + 0001         FIRST:   DS       1          ;Cold boot flag
C004 + 0001         ERRFLG:  DS       1          ;Error process flag
C005 + 0001         ERRCOD:  DS       1          ;Error code
C006 + 0002         BREAK:   DS       2          ;Break handler address
C008 + 0002         BRAM:    DS       2          ;RAM buffer address
C00A + 0001         BNSC:    DS       1          ;RAM buffer size in sectors
C00B + 0001         CFFLG:   DS       1          ;Command file flag
C00C + 0001         CFDRV:   DS       1          ;Command file drive
```

```
C00D + 0002    CFSEC:  DS      2        ;Command file sector addr
C00F + 0001    CFNSC:  DS      1        ;Command file sector count
C010 + 0001    CFSBP:  DS      1        ;Command file buffer ptr
C011 + 0001    RKROW:  DS      1        ;KBD row of repeat char
C012 + 0001    RKBIT:  DS      1        ;KBD bit of repeat char
C013 + 0001    RKVAL:  DS      1        ;ASCII value of rpt char
C014 + 0002    RKCNT:  DS      2        ;Repeat KBD counter
C016 + 0001    BLINKF: DS      1        ;BLINK routine flag
C017 + 0001    PLCT:   DS      1        ;Printer line counter
C018 + 0001    PPOS:   DS      1        ;Print head position
C019 + 0002    CLINP:  DS      2        ;Command line pointer
C01B + 0030    CLIN:   DS      48       ;Command line buffer
C04B + 000A    OVFCB:  DS      10       ;Overlay FCB
C055 + 0014    S1FCB:  DS      20       ;System FCB number 1
C069 + 0014    S2FCB:  DS      20       ;System FCB number 2
C07D + 0006    DSKWSP: DS      6        ;Disk routines workspace
C083           SYSWSP: EQU     $        ;Misc system workspace

C0C0           USRWSP: ORG     TOP+0C0H    ;USER WORKSPACE

C100           SCTB:   ORG     TOP+100H    ;SCAL TABLE

C07E           SCTBS:  EQU     SCTB-2*'A'  ;Actual start address

C200           INFOFA: ORG     TOP+200H    ;INFO FILE AREA

C200 + 0001    CURCHR: DS      1        ;Cursor character
C201 + 0001    CURBLR: DS      1        ;Cursor blink rate
C202 + 0002    RKLON:  DS      2        ;Keyboard long delay
C204 + 0002    RKSHO:  DS      2        ;Keyboard short delay
C206 + 000A            DS      10       ;Reserved
C210 + 0001    PLPP:   DS      1        ;Lines per page
C211 + 0001    PBMG:   DS      1        ;Bottom margin
C212 + 0001    PCPL:   DS      1        ;Characters per line
C213 + 0001    PLMG:   DS      1        ;Left margin
C214 + 0001    INSLEN: DS      1        ;Length of init string
C215 + 002B    INSTR:  DS      43       ;Init string
C240           PCHR:   EQU     $        ;Output routine

C300           SECBUF: ORG     TOP+300H    ;SECTOR BUFFER

C400           DIRBUF: ORG     TOP+400H    ;DIRECTORY BUFFER

C400 + 0014    DNAME:  DS      20       ;Disk name
C414 + 0002    NXTSEC: DS      2        ;Next sector address
C416 + 0002    NXTFCB: DS      2        ;Next FCB address
C418 + 03E8    FCBS:   DS      50*20    ;FCBs

C800           OVAREA: ORG     TOP+800H    ;OVERLAY AREA

C800 + 0004    OVNAM:  DS      4        ;Overlay name
C804           OVRLY:  EQU     $        ;Overlay entry point

C804                   END
```

```
ARG1    0C0C    ARG10   0C1E    ARG2    0C0E
ARG3    0C10    ARG4    0C12    ARG5    0C14
ARG6    0C16    ARG7    0C18    ARG8    0C1A
ARG9    0C1C    ARGC    0C0A    ARGN    0C0B
ARGX    0C2B    BLINKF  C016    BNSC    C00A
BRAM    C008    BREAK   C006    BRKADR  0C23
BRKPT   0020    BRKVAL  0C25    BS      0008
CCR     0018    CFDRV   C00C    CFFLG   C00B
CFNSC   C00F    CFSBP   C010    CFSEC   C00D
CH      0017    CLIN    C01B    CLINP   C019
CONFLG  0C26    CR      000D    CSL     0015
CSR     0016    CUD     0014    CUL     0011
CUR     0012    CURBLR  C201    CURCHR  C200
CURSOR  0C29    CUU     0013    DDRV    C001
DIRBUF  C400    DNAME   C400    DRVCOD  C002
DSKWSP  C07D    ERRCOD  C005    ERRFLG  C004
ESC     001B    FCBS    C418    FEXA    0012
FEXT    0008    FF      000C    FIRST   C003
FLDA    0010    FNAM    0000    FNSC    000E
FSEC    000C    FSFL    000A    FUFL    000B
INFOFA  C200    INSLEN  C214    INSTR   C215
INTA    0C75    KMAP    0C01    KOPT    0C27
KTAB    0C6F    KTABL   0C6D    LF      000A
MDRV    C000    MONSTK  0C61    NMIA    0C7E
NMIJ    0C7D    NUMN    0C20    NUMV    0C21
NXTFCB  C416    NXTSEC  C414    OUTTA   0C73
OVAREA  C800    OVFCB   C04B    OVNAM   C800
OVRLY   C804    PBMG    C211    PCHR    C240
PCPL    C212    PDCROM  D000    PLCT    C017
PLMG    C213    PLPP    C210    PORT0   0C00
PPOS    C018    PRS     0028    RAF     0C67
RAM     1000    RBC     0C61    RCALH   0010
RDE     0C63    RDEL    0038    RESET   0000
RHL     0C65    RIN     0008    RKBIT   C012
RKCNT   C014    RKLON   C202    RKROW   C011
RKSHO   C204    RKVAL   C013    ROUT    0030
RPC     0C69    RSP     0C6B    S1FCB   C055
S2FCB   C069    SCALH   0018    SCTB    C100
SCTBS   C07E    SECBUF  C300    STAB    0C71
STACK   1000    STMON   000D    SYSWSP  C083
TAB     0009    TOP     C000    UINA    0C7B
UINJ    0C7A    UOUTA   0C78    UOUTJ   0C77
USRWSP  C0C0    VRAM    080A    WORKSP  C000
XOPT    0C28    ZARGS   0060    ZATE    0073
ZB1HEX  007A    ZB2HEX  0068    ZBLINK  007B
ZCFMA   008C    ZCFS    0085    ZCKBRK  008B
ZCKER   008A    ZCOV    0088    ZCOVR   0089
ZCPOS   007C    ZCRLF   006A    ZCRT    0065
ZDRD    0081    ZDSIZE  0080    ZDWR    0082
ZENTER  0087    ZERRM   006B    ZFFLP   005E
ZIN     0062    ZINLIN  0063    ZJUMP   008E
ZKBD    0061    ZLOOK   0086    ZMFLP   005F
ZMRET   005B    ZNIM    0072    ZNNIM   0078
ZNNOM   0077    ZNOM    0071    ZNUM    0064
ZPOUT   008F    ZRDIR   0083    ZRKBD   007D
ZRLIN   0079    ZSCALI  007F    ZSCALJ  005C
ZSOUT   006D    ZSP2    007E    ZSPACE  0069
ZSRLIN  0070    ZSRLX   006F    ZSSCV   008D
ZTBCD2  0067    ZTBCD3  0066    ZTDEL   005D
ZTX1    006C    ZUIN    0076    ZUOUT   0075
ZWDIR   0084    ZXKBD   0074    ZXOUT   006E
```

```
                ;----------------------------------------------
                ;
                ;       PolyDos 2.0 R1
                ;       PolyDos Controller ROM
                ;
                ;       By Anders Hejlsberg
                ;       Copyright (C) 1981
                ;       PolyData microcenter ApS
                ;
                ;----------------------------------------------


                        REFS    SYSEQU
                        REF

0007            MAXDRV: EQU     7
0045            FFLP:   EQU     0045H


D000                    ORG     PDCROM
D000                    IDNT    $,$


                ;----------------------------------------------
                ; Here on power-up or RESET
                ;----------------------------------------------

D000 C303D0             JP      $+3             ;RESET jump
D003 310010             LD      SP,STACK        ;Set SP
D006 CD0D00             CALL    STMON           ;Initialize NAS-SYS
D009 EF                 RST     PRS             ;Prompt user
D00A 426F6F74           DB      'Boot which drive? ',0
D01D DF7B       PDC1:   SCAL    ZBLINK          ;Get drive number
D01F FE4E               CP      'N'             ;NAS-SYS?
D021 2006               JR      NZ,PDC2         ;No => skip
D023 EF                 RST     PRS             ;Clear screen
D024 1B00               DB      ESC,0
D026 C30500             JP      5               ;Go to NAS-SYS
D029 FE30       PDC2:   CP      '0'             ;Test drive number
D02B 38F0               JR      C,PDC1
D02D FE38               CP      MAXDRV+'0'+1
D02F 30EC               JR      NC,PDC1
D031 F7                 RST     ROUT            ;Print it
D032 D630               SUB     '0'             ;Adjust
D034 F5                 PUSH    AF              ;Save on stack
D035 2100C0             LD      HL,TOP          ;Initialize workspace
D038 0600               LD      B,0
D03A 3600       PDC3:   LD      (HL),0
D03C 23                 INC     HL
D03D 10FB               DJNZ    PDC3
D03F 3EFF               LD      A,-1
D041 3201C0             LD      (DDRV),A        ;No directory
D044 3202C0             LD      (DRVCOD),A      ;No drive selected
D047 3200C8             LD      (OVNAM),A       ;No overlay
D04A 2A710C             LD      HL,(STAB)       ;Get start addr of
D04D 118200             LD      DE,82H          ;NAS-SYS SCAL table
D050 19                 ADD     HL,DE
D051 1100C1             LD      DE,SCTB         ;Copy to SCTB
D054 017800             LD      BC,3CH*2
D057 EDB0               LDIR
D059 211BD5             LD      HL,PDSCTB       ;Get start addr of
```

```
D05C 012600             LD      BC,13H*2        ;PolyDos SCAL table
D05F EDB0               LDIR                    ;Copy to SCTB
D061 217EC0             LD      HL,SCTBS        ;Activate new SCAL table
D064 22710C             LD      (STAB),HL
D067 219DD0             LD      HL,PDOSW        ;Modify MRET vector
D06A DF8D               SCAL    ZSSCV
D06C 5B                 DB      ZMRET
D06D 21C7D3             LD      HL,CRT          ;Modify CRT vector
D070 DF8D               SCAL    ZSSCV
D072 65                 DB      ZCRT
D073 2119D4             LD      HL,BLINK        ;Modify BLINK vector
D076 DF8D               SCAL    ZSSCV
D078 7B                 DB      ZBLINK
D079 2110D4             LD      HL,DNNIM        ;Modify NNIM vector
D07C DF8D               SCAL    ZSSCV
D07E 78                 DB      ZNNIM
D07F DF78               SCAL    ZNNIM           ;Activate new input table
D081 2138D3             LD      HL,POUT         ;Make printer user output
D084 22780C             LD      (UOUTA),HL      ;device
D087 21C4D2             LD      HL,DBREAK       ;Initialize BREAK jump
D08A 2206C0             LD      (BREAK),HL      ;vector
D08D F1                 POP     AF              ;Restore drive number
D08E 3200C0             LD      (MDRV),A        ;Make master drive
D091 4F                 LD      C,A             ;Put in C
D092 CD41D5             CALL    INIT            ;Initialize controller
D095 2806               JR      Z,PDOSW         ;Skip if no error
D097 3205C0             LD      (ERRCOD),A      ;Save error code
D09A C3C8D2             JP      ABORT           ;Abort PolyDos


                ;-----------------------------------------------
                ; MRET routine entry point
                ;-----------------------------------------------

D09D 310010     PDOSW:  LD      SP,STACK        ;Set SP
D0A0 AF                 XOR     A               ;Clear A
D0A1 DF88               SCAL    ZCOV            ;Invoke Exec
D0A3 45786563           DB      'Exec'
D0A7 18F4               JR      PDOSW           ;Loop if Exec returns


                ; Disk read
                ;-----------------------------------------------
                ; Entry: HL:   Memory address
                ;        DE:   Disk address
                ;        B:    Number of sectors
                ;        C:    Drive
                ; Exit:  HL:   Unchanged
                ;        DE:   Unchanged
                ;        BC:   Unchanged
                ;        AF:   Status
                ;-----------------------------------------------

D0A9 AF         DRD:    XOR     A               ;A=0 => read
D0AA 1802               JR      DRW


                ; Disk write
                ;-----------------------------------------------
                ; Entry: HL:   Memory address
                ;        DE:   Disk address
                ;        B:    Number of sectors
```

```
                  ;        C:    Drive
                  ; Exit:  HL:   Unchanged
                  ;        DE:   Unchanged
                  ;        BC:   Unchanged
                  ;        AF:   Status
                  ;-----------------------------------------------

D0AC 3EFF         DWR:    LD      A,-1             ;A=-1 => write
D0AE D5           DRW:    PUSH    DE               ;Save
D0AF C5                   PUSH    BC
D0B0 E5                   PUSH    HL
D0B1 CD64D5               CALL    RWSCTS           ;Do read/write
D0B4 E1                   POP     HL               ;Restore
D0B5 C1                   POP     BC
D0B6 D1                   POP     DE
D0B7 C9                   RET


                  ; Read directory
                  ;-----------------------------------------------
                  ; Entry: C:    Drive number
                  ; Exit:  HL:   Unchanged
                  ;        DE:   Unchanged
                  ;        BC:   Unchanged
                  ;-----------------------------------------------

D0B8 3A01C0       RDIR:   LD      A,(DDRV)         ;Is directory already
D0BB 91                   SUB     C                ;there?
D0BC C8                   RET     Z                ;Yes => return
D0BD 79                   LD      A,C              ;Save as new directory
D0BE 3201C0               LD      (DDRV),A         ;drive number
D0C1 C5                   PUSH    BC               ;Save
D0C2 D5                   PUSH    DE
D0C3 E5                   PUSH    HL
D0C4 2100C4               LD      HL,DIRBUF        ;Read into DIRBUF
D0C7 110000               LD      DE,0             ;From sector 0
D0CA 0604                 LD      B,4              ;4 sectors
D0CC DF81                 SCAL    ZDRD             ;Do the read
D0CE E1                   POP     HL               ;Restore
D0CF D1                   POP     DE
D0D0 C1                   POP     BC
D0D1 C8                   RET     Z                ;No error => return
D0D2 E5                   PUSH    HL               ;Save
D0D3 2101C0               LD      HL,DDRV          ;Make directory invalid
D0D6 36FF                 LD      (HL),-1
D0D8 E1                   POP     HL               ;Restore
D0D9 C9                   RET


                  ; Write directory
                  ;-----------------------------------------------
                  ; Entry: No parameters required
                  ; Exit:  HL:   Unchanged
                  ;        DE:   Unchanged
                  ;        BC:   Unchanged
                  ;-----------------------------------------------

D0DA C5           WDIR:   PUSH    BC               ;Save
D0DB D5                   PUSH    DE
D0DC E5                   PUSH    HL
D0DD 2100C4               LD      HL,DIRBUF        ;Write from DIRBUF
D0E0 110000               LD      DE,0             ;To sector 0
```

```
D0E3 0604                 LD      B,4             ;4 sectors
D0E5 3A01C0               LD      A,(DDRV)        ;On drive DDRV
D0E8 4F                   LD      C,A
D0E9 DF82                 SCAL    ZDWR            ;Do the write
D0EB E1                   POP     HL              ;Restore
D0EC D1                   POP     DE
D0ED C1                   POP     BC
D0EE C9                   RET


                  ; Convert a file specifier
                  ;---------------------------------------------
                  ; Entry: HL:  FCB address
                  ;        DE:  Line buffer address
                  ;        B:   B0=1  Name optional
                  ;             B1=1  Extension optional
                  ;             B2=1  Drive optional
                  ; Exit:  HL:  Unchanged
                  ;        DE:  Next line buffer address
                  ;        B:   B0=1  No name
                  ;             B1=1  No extension
                  ;             B2=1  No drive
                  ;        C:   Drive number (MDRV if B.B2=1)
                  ;---------------------------------------------

D0EF E5           CFS:    PUSH    HL              ;Save FCB addr
D0F0 78                   LD      A,B             ;Compute flag mask
D0F1 2F                   CPL
D0F2 E607                 AND     111B
D0F4 F5                   PUSH    AF              ;Save on stack
D0F5 010907               LD      BC,709H         ;Init flags and counter
D0F8 1A           CFS1:   LD      A,(DE)          ;Get character
D0F9 FE20                 CP      ' '             ;Jump to CFS3 if it is
D0FB 282B                 JR      Z,CFS3          ;a delimiter
D0FD FE2E                 CP      '.'
D0FF 2827                 JR      Z,CFS3
D101 FE3A                 CP      ':'
D103 2823                 JR      Z,CFS3
D105 FE2C                 CP      ','
D107 281F                 JR      Z,CFS3
D109 FE3B                 CP      ';'
D10B 281B                 JR      Z,CFS3
D10D FE0D                 CP      CR
D10F 2817                 JR      Z,CFS3
D111 FE09                 CP      TAB
D113 2813                 JR      Z,CFS3
D115 B7                   OR      A
D116 2810                 JR      Z,CFS3
D118 D75F                 RCAL    TSTCH           ;Test character
D11A 0D                   DEC     C               ;8 characters done?
D11B 2807                 JR      Z,CFS2          ;Yes => skip
D11D 77                   LD      (HL),A          ;Save in FCB
D11E 23                   INC     HL              ;Point to next
D11F 13                   INC     DE
D120 CB80                 RES     0,B             ;Name specified
D122 18D4                 JR      CFS1
D124 3E11         CFS2:   LD      A,11H           ;Error 11
D126 184B                 JR      CFS9
D128 79           CFS3:   LD      A,C             ;Get counter
D129 0D           CFS4:   DEC     C               ;Filling done?
D12A 2809                 JR      Z,CFS11         ;Yes => skip
D12C FE09                 CP      9               ;Was name specified?
```

```
D12E 2802               JR      Z,CFS12         ;No => skip
D130 3620               LD      (HL),' '        ;Blank fill
D132 23         CFS12:  INC     HL              ;Point to next
D133 18F4               JR      CFS4            ;Repeat
D135 1A         CFS11:  LD      A,(DE)          ;Get character
D136 FE2E               CP      '.'             ;Period?
D138 200B               JR      NZ,CFS5         ;No => skip
D13A 13                 INC     DE              ;Point to next
D13B D73A               RCAL    GETCH           ;Get and test
D13D 77                 LD      (HL),A          ;Save in FEXT
D13E 23                 INC     HL              ;Point to next
D13F D736               RCAL    GETCH           ;Get and test
D141 77                 LD      (HL),A          ;Save in FEXT
D142 23                 INC     HL              ;Point to next
D143 CB88               RES     1,B             ;Extension specified
D145 3A00C0     CFS5:   LD      A,(MDRV)        ;Default is MDRV
D148 4F                 LD      C,A
D149 1A                 LD      A,(DE)          ;Get character
D14A FE3A               CP      ':'             ;Colon?
D14C 200E               JR      NZ,CFS6         ;No => skip
D14E 13                 INC     DE              ;Point to next
D14F 1A                 LD      A,(DE)          ;Get character
D150 13                 INC     DE              ;Point to next
D151 D630               SUB     '0'             ;Adjust
D153 381C               JR      C,CFS8          ;Error => skip
D155 FE08               CP      MAXDRV+1        ;Too big?
D157 3018               JR      NC,CFS8         ;Yes => skip
D159 4F                 LD      C,A             ;Put drive number in C
D15A CB90               RES     2,B             ;Drive specified
D15C 1A         CFS6:   LD      A,(DE)          ;Skip blanks
D15D FE20               CP      ' '
D15F 2003               JR      NZ,CFS7
D161 13                 INC     DE
D162 18F8               JR      CFS6
D164 F1         CFS7:   POP     AF              ;Get flag mask
D165 E1                 POP     HL              ;Get FCB addr
D166 A0                 AND     B               ;Flags ok?
D167 C8                 RET     Z               ;Yes => return
D168 0612               LD      B,12H           ;Compute error code
D16A 04         CFS10:  INC     B
D16B 1F                 RRA
D16C 30FC               JR      NC,CFS10
D16E 78                 LD      A,B             ;Put in A
D16F B7                 OR      A               ;Indicate error
D170 C9                 RET
D171 3E12       CFS8:   LD      A,12H           ;Error 12
D173 E1         CFS9:   POP     HL              ;Adjust
D174 E1                 POP     HL              ;Get FCB addr
D175 B7                 OR      A               ;Indicate error
D176 C9                 RET

D177 1A         GETCH:  LD      A,(DE)          ;Get character
D178 13                 INC     DE              ;Point to next
D179 FE21       TSTCH:  CP      21H             ;Control character?
D17B 3803               JR      C,TCH1          ;Yes => skip
D17D FE80               CP      80H             ;Graphic character
D17F D8                 RET     C               ;No => return
D180 E1         TCH1:   POP     HL              ;Adjust
D181 3E10               LD      A,10H           ;Error 10
D183 18EE               JR      CFS9

                ; Lookup file in current directory
```

```
                      ;----------------------------------------------
                      ; Entry: HL:  Lookup FCB address
                      ;        DE:  Previous directory FCB address
                      ;         B:  B0=1  Don't match file name
                      ;             B1=1  Don't match extension
                      ;             B4=1  Copy dir FCB to look FCB
                      ;             B5=1  Include locked files
                      ;             B6=1  Include deleted files
                      ;             B7=1  Not first look
                      ; Exit:  HL:  Unchanged
                      ;        DE:  Directory FCB address
                      ;         B:  B7 set, B6-B0 unchanged
                      ;         C:  Unchanged
                      ;----------------------------------------------

D185 CB78             LOOK:   BIT     7,B             ;First look?
D187 2005                     JR      NZ,LK1          ;No => skip
D189 1104C4                   LD      DE,FCBS-20      ;Start with first FCB
D18C CBF8                     SET     7,B             ;Next time not first
D18E E5               LK1:    PUSH    HL              ;Save FCB addr
D18F 211400           LK2:    LD      HL,20           ;Point to next directory
D192 19                       ADD     HL,DE           ;FCB
D193 EB                       EX      DE,HL           ;Put in DE
D194 2A16C4                   LD      HL,(NXTFCB)     ;Done all FCBs?
D197 37                       SCF
D198 ED52                     SBC     HL,DE
D19A E1                       POP     HL              ;(restore FCB addr)
D19B 3004                     JR      NC,LK3          ;No => skip
D19D 3E30                     LD      A,30H           ;Error 30
D19F B7                       OR      A
D1A0 C9                       RET
D1A1 E5               LK3:    PUSH    HL              ;Save lookup FCB addr
D1A2 D5                       PUSH    DE              ;Save directory FCB addr
D1A3 3E08                     LD      A,8             ;Compare names
D1A5 D738                     RCAL    CMPS
D1A7 2804                     JR      Z,LK4           ;Match => skip
D1A9 CB40                     BIT     0,B             ;Should they match?
D1AB 280A                     JR      Z,LK5           ;Yes => skip
D1AD 3E02             LK4:    LD      A,2             ;Compare extensions
D1AF D72E                     RCAL    CMPS
D1B1 2807                     JR      Z,LK6           ;Match => skip
D1B3 CB48                     BIT     1,B             ;Should thay match?
D1B5 2003                     JR      NZ,LK6          ;No => skip
D1B7 D1               LK5:    POP     DE              ;Restore dir FCB addr
D1B8 18D5                     JR      LK2             ;Try next
D1BA 1A               LK6:    LD      A,(DE)          ;Locked?
D1BB CB47                     BIT     0,A
D1BD 2804                     JR      Z,LK7           ;No => skip
D1BF CB68                     BIT     5,B             ;Include locked files?
D1C1 28F4                     JR      Z,LK5           ;No => try next
D1C3 CB4F             LK7:    BIT     1,A             ;Deleted?
D1C5 2804                     JR      Z,LK8           ;No => skip
D1C7 CB70                     BIT     6,B             ;Include deleted files?
D1C9 28EC                     JR      Z,LK5           ;No => try next
D1CB D1               LK8:    POP     DE              ;Restore dir FCB addr
D1CC E1                       POP     HL              ;Restore lookup FCB addr
D1CD CB60                     BIT     4,B             ;Copy directory FCB?
D1CF 280C                     JR      Z,LK9           ;No => skip
D1D1 C5                       PUSH    BC              ;Save
D1D2 D5                       PUSH    DE
D1D3 E5                       PUSH    HL
D1D4 EB                       EX      DE,HL           ;Copy FCB
```

```
D1D5 011400             LD      BC,20
D1D8 EDB0               LDIR
D1DA E1                 POP     HL              ;Restore
D1DB D1                 POP     DE
D1DC C1                 POP     BC
D1DD AF         LK9:    XOR     A               ;No error
D1DE C9                 RET

                ; Compare string at DE to string at HL for
                ; A characters

D1DF C5         CMPS:   PUSH    BC              ;Save BC
D1E0 47                 LD      B,A             ;Put length in B
D1E1 0E00               LD      C,0             ;Clear C
D1E3 1A         CPS1:   LD      A,(DE)          ;Get character
D1E4 BE                 CP      (HL)            ;Match?
D1E5 2801               JR      Z,CPS2          ;Yes => skip
D1E7 0D                 DEC     C               ;No match
D1E8 23         CPS2:   INC     HL              ;Point to next
D1E9 13                 INC     DE
D1EA 10F7               DJNZ    CPS1            ;Fall thru when done
D1EC 0C                 INC     C               ;Status to Z flag
D1ED 0D                 DEC     C
D1EE C1                 POP     BC              ;Restore BC
D1EF C9                 RET


                ; Enter file in current directory
                ;-----------------------------------------------
                ; Entry: HL:  Address of FCB to be entered
                ; Exit:  HL:  Unchanged
                ;        DE:  Directory FCB address
                ;        BC:  Unchanged
                ;-----------------------------------------------

D1F0 C5         ENTER:  PUSH    BC              ;Save
D1F1 E5                 PUSH    HL
D1F2 0620               LD      B,00100000B     ;Look it up
D1F4 DF86               SCAL    ZLOOK
D1F6 2004               JR      NZ,ENT1         ;Non-existing => skip
D1F8 3E31               LD      A,31H           ;Error 31
D1FA 1829               JR      ENT2
D1FC ED5B16C4   ENT1:   LD      DE,(NXTFCB)     ;Is directory full?
D200 2100C8             LD      HL,FCBS+50*20
D203 37                 SCF
D204 ED52               SBC     HL,DE
D206 3E32               LD      A,32H           ;(Error 32 if so)
D208 381B               JR      C,ENT2          ;Yes => skip
D20A E1                 POP     HL              ;Restore FCB addr
D20B E5                 PUSH    HL
D20C 011400             LD      BC,20           ;Copy 20 bytes
D20F EDB0               LDIR
D211 ED5316C4           LD      (NXTFCB),DE     ;Save new end addr
D215 11FAFF             LD      DE,FNSC-20      ;Get FNSC into DE
D218 19                 ADD     HL,DE
D219 5E                 LD      E,(HL)
D21A 23                 INC     HL
D21B 56                 LD      D,(HL)
D21C 2A14C4             LD      HL,(NXTSEC)     ;Add FNSC to NXTSEC
D21F 19                 ADD     HL,DE
D220 2214C4             LD      (NXTSEC),HL
D223 DF84               SCAL    ZWDIR           ;Write directory to disk
```

```
D225 E1         ENT2:   POP     HL              ;Restore
D226 C1                 POP     BC
D227 B7                 OR      A               ;Status to Z flag
D228 C9                 RET


                ; Call an overlay
                ;----------------------------------------------
                ; Entry: Registers defined by overlay
                ; Exit:  Registers defined by overlay
                ;----------------------------------------------

D229 E3         COV:    EX      (SP),HL         ;Get overlay name
D22A CD96D2             CALL    TROVN
D22D E3                 EX      (SP),HL
D22E CD53D2             CALL    GETOV           ;Read overlay
D231 C304C8             JP      OVRLY           ;Go to it


                ; Call an overlay and restore current overlay
                ;----------------------------------------------
                ; Entry: Registers defined by overlay
                ; Exit:  Registers defined by overlay
                ;----------------------------------------------

D234 E3         COVR:   EX      (SP),HL         ;Get overlay name
D235 CD96D2             CALL    TROVN
D238 E3                 EX      (SP),HL
D239 E5                 PUSH    HL              ;Save return addr
D23A 2A00C8             LD      HL,(OVNAM)      ;Push name of current
D23D E3                 EX      (SP),HL         ;overlay onto stack
D23E E5                 PUSH    HL
D23F 2A02C8             LD      HL,(OVNAM+2)
D242 E3                 EX      (SP),HL
D243 CD53D2             CALL    GETOV           ;Read new overlay
D246 CD04C8             CALL    OVRLY           ;Call it
D249 E3                 EX      (SP),HL         ;Get previous overlay
D24A 224DC0             LD      (OVFCB+2),HL    ;name
D24D E1                 POP     HL
D24E E3                 EX      (SP),HL
D24F 224BC0             LD      (OVFCB),HL
D252 E1                 POP     HL

                ; Read overlay in OVFCB into memory

D253 F5         GETOV:  PUSH    AF              ;Save all
D254 C5                 PUSH    BC
D255 D5                 PUSH    DE
D256 E5                 PUSH    HL
D257 214BC0             LD      HL,OVFCB+FNAM   ;Is it there already?
D25A 1100C8             LD      DE,OVNAM
D25D 3E04               LD      A,4
D25F CDDFD1             CALL    CMPS
D262 282D               JR      Z,GOV2          ;Yes => don't read
D264 0604               LD      B,4             ;Blank fill rest of name
D266 3620       GOV1:   LD      (HL),' '
D268 23                 INC     HL
D269 10FB               DJNZ    GOV1
D26B 364F               LD      (HL),'O'        ;Insert extension
D26D 23                 INC     HL
D26E 3656               LD      (HL),'V'
D270 3A00C0             LD      A,(MDRV)        ;Read from MDRV
```

```
D273 4F                 LD      C,A
D274 DF83               SCAL    ZRDIR           ;Read directory
D276 DF8A               SCAL    ZCKER           ;Check for error
D278 214BC0             LD      HL,OVFCB        ;Look it up
D27B 0620               LD      B,00100000B     ;Include locked files
D27D DF86               SCAL    ZLOOK
D27F DF8A               SCAL    ZCKER           ;Check for error
D281 210C00             LD      HL,FSEC         ;Point to FSEC slot
D284 19                 ADD     HL,DE
D285 5E                 LD      E,(HL)          ;Get FSEC into DE
D286 23                 INC     HL
D287 56                 LD      D,(HL)
D288 23                 INC     HL
D289 46                 LD      B,(HL)          ;Get FNSC into B
D28A 2100C8             LD      HL,OVAREA       ;Read into OVAREA
D28D DF81               SCAL    ZDRD            ;Do the read
D28F DF8A               SCAL    ZCKER           ;Check for error
D291 E1         GOV2:   POP     HL              ;Restore all
D292 D1                 POP     DE
D293 C1                 POP     BC
D294 F1                 POP     AF
D295 C9                 RET

                ; Transfer overlay name to OVFCB

D296 F5         TROVN:  PUSH    AF
D297 C5                 PUSH    BC
D298 D5                 PUSH    DE
D299 114BC0             LD      DE,OVFCB+FNAM
D29C 010400             LD      BC,4
D29F EDB0               LDIR
D2A1 D1                 POP     DE
D2A2 C1                 POP     BC
D2A3 F1                 POP     AF
D2A4 C9                 RET


                ; Check for error
                ;----------------------------------------------
                ; Entry: A:    Error code (0 => no error)
                ; Exit:   If no error, all registers unchanged
                ;         otherwise CKER never returns
                ;----------------------------------------------

D2A5 B7         CKER:   OR      A               ;Error?
D2A6 C8                 RET     Z               ;No => bye
D2A7 47                 LD      B,A             ;Put code in B
D2A8 DF77               SCAL    ZNNOM           ;Normal output
D2AA 3A04C0             LD      A,(ERRFLG)      ;Second error?
D2AD B7                 OR      A
D2AE 2018               JR      NZ,ABORT        ;Yes => trouble
D2B0 3D                 DEC     A               ;Set error flag
D2B1 3204C0             LD      (ERRFLG),A
D2B4 78                 LD      A,B             ;Save error code
D2B5 3205C0             LD      (ERRCOD),A
D2B8 DF88               SCAL    ZCOV            ;Call Emsg to print the
D2BA 456D7367           DB      'Emsg'          ;error message
D2BE DF6A               SCAL    ZCRLF
D2C0 AF                 XOR     A               ;Clear error flag
D2C1 3204C0             LD      (ERRFLG),A
D2C4 DF8C       DBREAK: SCAL    ZCFMA           ;Abort command file mode
D2C6 DF5B               SCAL    ZMRET           ;Back to Exec
```

```
                    ; Abort PolyDos, print error code, and return
                    ; control to NAS-SYS

D2C8 CD0D00         ABORT:  CALL    STMON           ;Initialize NAS-SYS
D2CB EF                     RST     PRS             ;Print error message
D2CC 28457272               DB      '(Error ',0
D2D4 3A05C0                 LD      A,(ERRCOD)
D2D7 DF68                   SCAL    ZB2HEX
D2D9 EF                     RST     PRS
D2DA 290D00                 DB      ')',CR,0
D2DD DF5B                   SCAL    ZMRET           ;Back to NAS-SYS


                    ; Check for break
                    ;----------------------------------------------
                    ; If CTRL/SHIFT/@ is pressed, abort any
                    ; operation, and return to via MRET
                    ;----------------------------------------------

D2DF 3E02           CKBRK:  LD      A,2             ;Reset KBD pointer
D2E1 CD4500                 CALL    FFLP
D2E4 DB00                   IN      A,(0)           ;Read first row
D2E6 F680                   OR      80H             ;Ignore bit 7
D2E8 FEC7                   CP      -1-38H          ;CTRL/SHIFT/@?
D2EA C0                     RET     NZ              ;No => bye
D2EB 3A16C0                 LD      A,(BLINKF)      ;Aborted from BLINK?
D2EE B7                     OR      A
D2EF 2808                   JR      Z,CKB1          ;No => skip
D2F1 2A290C                 LD      HL,(CURSOR)     ;Reinsert character
D2F4 77                     LD      (HL),A          ;at cursor
D2F5 AF                     XOR     A               ;Clear BLINK flag
D2F6 3216C0                 LD      (BLINKF),A
D2F9 2A06C0         CKB1:   LD      HL,(BREAK)      ;Go to BREAK handler
D2FC E9                     JP      (HL)


                    ; Abort command file mode
                    ;----------------------------------------------
                    ; If command file mode is active, abort it and
                    ; display (Cmdf abort)
                    ;----------------------------------------------

D2FD 210BC0         CFMA:   LD      HL,CFFLG        ;Is CFFLG set?
D300 AF                     XOR     A
D301 BE                     CP      (HL)
D302 C8                     RET     Z               ;No => bye
D303 77                     LD      (HL),A          ;Clear it
D304 EF                     RST     PRS             ;Display message
D305 28436D64               DB      '(Cmdf abort)',CR,0
D313 C9                     RET


                    ; Set SCAL vector
                    ;----------------------------------------------
                    ; Entry: HL:  New jump vector address
                    ;             Call is followed by routine number
                    ; Exit:  HL:  Previous jump vector address
                    ;        DE:  Junk
                    ;        BC:  Junk
                    ;----------------------------------------------
```

```
D314 E3         SSCV:   EX      (SP),HL         ;Get routine number
D315 5E                 LD      E,(HL)
D316 23                 INC     HL
D317 E3                 EX      (SP),HL
D318 E5                 PUSH    HL              ;Save HL
D319 1600               LD      D,0             ;Clear D
D31B 2A710C             LD      HL,(STAB)       ;Calculate addr in
D31E 19                 ADD     HL,DE           ;SCAL table
D31F 19                 ADD     HL,DE
D320 C1                 POP     BC              ;Get new vector
D321 5E                 LD      E,(HL)          ;Read old
D322 71                 LD      (HL),C          ;Save new
D323 23                 INC     HL              ;Point to next byte
D324 56                 LD      D,(HL)          ;Read old
D325 70                 LD      (HL),B          ;Save new
D326 EB                 EX      DE,HL           ;Put old vector into HL
D327 C9                 RET


                ;Execute jump table
                ;-----------------------------------------------
                ; Entry: A:   Jump vector number
                ;             Jump vectors follow call as DW's
                ; Exit:       Jumps to selected routine with all
                ;             registers intact
                ;-----------------------------------------------

D328 E3         JUMP:   EX      (SP),HL         ;Point to jump table
D329 D5                 PUSH    DE              ;Save
D32A F5                 PUSH    AF
D32B 5F                 LD      E,A             ;Calculate vector addr
D32C 1600               LD      D,0
D32E 19                 ADD     HL,DE
D32F 19                 ADD     HL,DE
D330 5E                 LD      E,(HL)          ;Get vector into DE
D331 23                 INC     HL
D332 56                 LD      D,(HL)
D333 EB                 EX      DE,HL           ;Put into HL
D334 F1                 POP     AF              ;Restore
D335 D1                 POP     DE
D336 E3                 EX      (SP),HL
D337 C9                 RET                     ;Go there


                ; Output character to printer
                ;-----------------------------------------------
                ; Entry: A:   Holds character to be printed
                ; Exit:  HL:  Junk
                ;        DE:  Junk
                ;        BC:  Junk
                ;        AF:  Unchanged
                ;-----------------------------------------------

D338 F5         POUT:   PUSH    AF              ;Save char
D339 2118C0             LD      HL,PPOS         ;Point to PPOS
D33C FE0D               CP      CR              ;Is it CR?
D33E 2021               JR      NZ,PO4          ;No => skip
D340 CDB7D3             CALL    PRCH            ;Print it
D343 3600               LD      (HL),0          ;Clear PPOS
D345 2B                 DEC     HL              ;Point to PLCT
D346 34                 INC     (HL)            ;Increment it
D347 3A11C2             LD      A,(PBMG)        ;Get PBMG
```

```
D34A 47                 LD      B,A             ;Put into B
D34B 3A10C2             LD      A,(PLPP)        ;Get PLPP
D34E 90                 SUB     B               ;Subtract PBMG
D34F 96                 SUB     (HL)            ;Subtract PLCT
D350 2057               JR      NZ,PO11         ;Not zero => skip
D352 04         PO1:    INC     B               ;Adjust B
D353 05         PO2:    DEC     B               ;Decrement count
D354 2808               JR      Z,PO3           ;Zero => skip
D356 3E0D               LD      A,CR            ;Print CR/LF
D358 CDB7D3             CALL    PRCH
D35B 34                 INC     (HL)            ;Increment PLCT
D35C 18F5               JR      PO2
D35E 70         PO3:    LD      (HL),B          ;Clear PLCT
D35F 1848               JR      PO11            ;Done
D361 FE0C       PO4:    CP      FF              ;Is it FF?
D363 200A               JR      NZ,PO5          ;No => skip
D365 3600               LD      (HL),0          ;Clear PPOS
D367 2B                 DEC     HL              ;Point to PLCT
D368 3A10C2             LD      A,(PLPP)        ;Calculate number of
D36B 96                 SUB     (HL)            ;CR/LFs to print
D36C 47                 LD      B,A             ;Put in B
D36D 18E3               JR      PO1             ;Go print them
D36F 3A12C2     PO5:    LD      A,(PCPL)        ;Are we at right margin?
D372 BE                 CP      (HL)
D373 2009               JR      NZ,PO6          ;No => skip
D375 C5                 PUSH    BC
D376 E5                 PUSH    HL
D377 3E0D               LD      A,CR            ;Move to next line
D379 CD38D3             CALL    POUT
D37C E1                 POP     HL
D37D C1                 POP     BC
D37E 7E         PO6:    LD      A,(HL)          ;Is PPOS zero?
D37F B7                 OR      A
D380 200F               JR      NZ,PO8          ;No => skip
D382 3A13C2             LD      A,(PLMG)        ;Get PLMG
D385 47                 LD      B,A             ;Put in B
D386 04                 INC     B               ;Adjust
D387 05         PO7:    DEC     B               ;Decrement count
D388 2807               JR      Z,PO8           ;Zero => skip
D38A 3E20               LD      A,' '           ;Print blank
D38C CDABD3             CALL    PRCHT
D38F 18F6               JR      PO7
D391 F1         PO8:    POP     AF              ;Restore char
D392 F5                 PUSH    AF
D393 FE09               CP      TAB             ;Is it TAB?
D395 0601               LD      B,1             ;(Print 1 char if not)
D397 200B               JR      NZ,PO10         ;No => skip
D399 3A13C2             LD      A,(PLMG)        ;Calculate number of
D39C 96                 SUB     (HL)            ;blanks to expand the
D39D 3D                 DEC     A               ;TAB into
D39E E607               AND     7
D3A0 3C                 INC     A
D3A1 47                 LD      B,A             ;Put in B
D3A2 3E20       PO9:    LD      A,' '           ;Print blank(s)
D3A4 CDABD3     PO10:   CALL    PRCHT           ;Print character
D3A7 10F9               DJNZ    PO9             ;Fall thru when done
D3A9 F1         PO11:   POP     AF              ;Restore char
D3AA C9                 RET

                ; Print character with right margin test

D3AB 4F         PRCHT:  LD      C,A             ;Put char in C
```

```
D3AC 3A12C2            LD      A,(PCPL)         ;Still room on line?
D3AF BE                CP      (HL)
D3B0 C8                RET     Z                ;No => return
D3B1 79                LD      A,C              ;Get char
D3B2 CDB7D3            CALL    PRCH             ;Print it
D3B5 34                INC     (HL)             ;Increment PPOS
D3B6 C9                RET

                ; Transfer character to user defined output
                ; routine, and add a LF in case of CR

D3B7 C5         PRCH:  PUSH    BC               ;Save
D3B8 E5                PUSH    HL
D3B9 F5                PUSH    AF
D3BA CD40C2            CALL    PCHR             ;Call user routine
D3BD F1                POP     AF               ;Restore
D3BE E1                POP     HL
D3BF C1                POP     BC
D3C0 FE0D              CP      CR               ;Was it CR?
D3C2 C0                RET     NZ               ;No => return
D3C3 3E0A              LD      A,LF             ;Supply LF
D3C5 18F0              JR      PRCH


                ; Output to CRT
                ;-----------------------------------------------
                ; Output character in A to the CRT. TAB chars
                ; are expanded into one or more spaces
                ;-----------------------------------------------

D3C7 FE20       CRT:   CP      ' '              ;Control char
D3C9 302B              JR      NC,CRTC          ;No => go print
D3CB B7                OR      A                ;Zero?
D3CC C8                RET     Z                ;Yes => bye
D3CD F5                PUSH    AF               ;Save char
D3CE FE09              CP      TAB              ;Is it TAB?
D3D0 280D              JR      Z,CRT1           ;Yes => skip
D3D2 47                LD      B,A              ;Put char in B
D3D3 3A0600            LD      A,(6)            ;Get NAS-SYS byte
D3D6 FEFE              CP      0FEH             ;NAS-SYS 3?
D3D8 78                LD      A,B              ;(Restore char)
D3D9 C25201            JP      NZ,152H          ;Yes => jump
D3DC C39301            JP      193H             ;Must be NAS-SYS 1
D3DF 3A290C     CRT1:  LD      A,(CURSOR)       ;Expand TAB
D3E2 E63F              AND     3FH
D3E4 2F                CPL
D3E5 C60A              ADD     A,10
D3E7 E607              AND     7
D3E9 3C                INC     A
D3EA 47                LD      B,A              ;Put count in B
D3EB C5         CRT2:  PUSH    BC               ;Save BC
D3EC 3E20              LD      A,' '            ;Print blank
D3EE CDF6D3            CALL    CRTC
D3F1 C1                POP     BC               ;Restore BC
D3F2 10F7              DJNZ    CRT2             ;Fall thru when done
D3F4 F1                POP     AF               ;Restore char
D3F5 C9                RET
D3F6 F5         CRTC:  PUSH    AF               ;Save char
D3F7 2A290C            LD      HL,(CURSOR)      ;Store at cursor
D3FA 77                LD      (HL),A
D3FB 23                INC     HL               ;Move cursor right
D3FC 7E                LD      A,(HL)           ;Is there a margin?
```

```
D3FD B7                 OR      A
D3FE 2805               JR      Z,CRTC1         ;Yes => skip
D400 22290C             LD      (CURSOR),HL     ;Save new cursor
D403 F1                 POP     AF              ;Restore char
D404 C9                 RET
D405 3A0600     CRTC1:  LD      A,(6)           ;NAS-SYS 3?
D408 FEFE               CP      0FEH
D40A C20E02             JP      NZ,20EH         ;Yes => jump
D40D C34F02             JP      24FH            ;Must be NAS-SYS 1?


                ; Normalize input table
                ;----------------------------------------------
                ; Restores normal input channels, i.e. routines
                ; RKBD and SRLIN. On exit HL contains address
                ; of previous input table
                ;----------------------------------------------

D410 2116D4     DNNIM:  LD      HL,INTBL
D413 DF72               SCAL    ZNIM
D415 C9                 RET

D416 7D7000     INTBL:  DB      ZRKBD,ZSRLIN,0


                ; Input from keyboard or command file
                ;----------------------------------------------
                ; If command file mode is active, get the
                ; character from the command file, else input
                ; it with a blinking cursor as normally.
                ; Pressing CTRL/SHIFT/@ will warm-boot the
                ; system
                ;----------------------------------------------

D419 3A0BC0     BLINK:  LD      A,(CFFLG)       ;Command file mode?
D41C B7                 OR      A
D41D 2028               JR      NZ,BL3          ;Yes => skip
D41F 2A290C     BL1:    LD      HL,(CURSOR)     ;Get character at cursor
D422 7E                 LD      A,(HL)
D423 3216C0             LD      (BLINKF),A      ;Save in BLINKF
D426 3A00C2             LD      A,(CURCHR)      ;Put cursor on screen
D429 77                 LD      (HL),A
D42A D710               RCAL    BIN             ;Scan KBD
D42C F5                 PUSH    AF              ;Save char
D42D 3A16C0             LD      A,(BLINKF)      ;Restore char at cursor
D430 77                 LD      (HL),A
D431 AF                 XOR     A               ;Clear BLINK flag
D432 3216C0             LD      (BLINKF),A
D435 F1                 POP     AF              ;Restore input char
D436 D8                 RET     C               ;Character => return
D437 D703               RCAL    BIN             ;Scan KBD
D439 30DE               JR      NC,BLINK        ;No char => repeat
D43B C9                 RET
D43C 3A01C2     BIN:    LD      A,(CURBLR)      ;Get blink rate
D43F 5F                 LD      E,A             ;Put in E
D440 DF62       BIN1:   SCAL    ZIN             ;Scan inputs
D442 D8                 RET     C               ;Char => return
D443 1D                 DEC     E               ;Decrement count
D444 20FA               JR      NZ,BIN1         ;Loop until done
D446 C9                 RET
D447 CDDFD2     BL3:    CALL    CKBRK           ;Check for break
D44A 3A10C0             LD      A,(CFSBP)       ;Get sector buffer ptr
```

```
D44D B7                  OR      A                 ;Buffer empty?
D44E 2025                JR      NZ,BL4            ;No => skip
D450 3A0FC0              LD      A,(CFNSC)         ;Get sector count
D453 320BC0              LD      (CFFLG),A         ;Save as flag
D456 B7                  OR      A                 ;Zero?
D457 28C6                JR      Z,BL1             ;Yes => skip
D459 3D                  DEC     A                 ;Decrement count
D45A 320FC0              LD      (CFNSC),A         ;Save it
D45D 2100C3              LD      HL,SECBUF         ;Read into SECBUF
D460 ED5B0DC0            LD      DE,(CFSEC)        ;From CFSEC
D464 C5                  PUSH    BC                ;Save BC
D465 0601                LD      B,1               ;Read one sector
D467 3A0CC0              LD      A,(CFDRV)         ;From CFDRV
D46A 4F                  LD      C,A
D46B DF81                SCAL    ZDRD              ;Do the read
D46D DF8A                SCAL    ZCKER             ;Check for error
D46F C1                  POP     BC                ;Restore BC
D470 13                  INC     DE                ;Increment sector addr
D471 ED530DC0            LD      (CFSEC),DE        ;Save it
D475 26C3       BL4:     LD      H,HIGH(SECBUF)    ;Set MSB of address
D477 6F                  LD      L,A               ;Set LSB
D478 3C                  INC     A                 ;Increment pointer
D479 3210C0              LD      (CFSBP),A         ;Save it
D47C 7E                  LD      A,(HL)            ;Get char
D47D B7                  OR      A                 ;Filler?
D47E 28C7                JR      Z,BL3             ;Yes => repeat
D480 C9                  RET


                ; Scan keyboard with repeat
                ;-----------------------------------------
                ; If character is available it is returned in A
                ; with carry set. Otherwise carry is cleared.
                ; Registers HL, DE, and BC are modified.
                ; Pressing CTRL/SHIFT/@ warm-boots system.
                ;-----------------------------------------

D481 CDDFD2     RKBD:    CALL    CKBRK             ;Check for break
D484 2A11C0              LD      HL,(RKROW)        ;Get bit/row into HL
D487 2C                  INC     L                 ;Is row zero?
D488 2D                  DEC     L
D489 2817                JR      Z,RK3             ;Yes => no repeat char
D48B 0608                LD      B,8               ;Do all 8 rows
D48D 3E01       RK1:     LD      A,1               ;Move to next row
D48F CD4500              CALL    FFLP
D492 F5                  PUSH    AF                ;Delay
D493 F1                  POP     AF
D494 7D                  LD      A,L               ;Repeat key row?
D495 B8                  CP      B
D496 2004                JR      NZ,RK2            ;No => skip
D498 DB00                IN      A,(0)             ;Read row status
D49A 2F                  CPL                       ;Complement
D49B 4F                  LD      C,A               ;Put in C
D49C 10EF       RK2:     DJNZ    RK1               ;Fall thru when done
D49E 7C                  LD      A,H               ;Is repeat key down?
D49F A1                  AND     C
D4A0 204F                JR      NZ,RK11           ;Yes => skip
D4A2 21010C     RK3:     LD      HL,KMAP           ;Point to KMAP
D4A5 DB00                IN      A,(0)             ;Read first row
D4A7 2F                  CPL                       ;Complement
D4A8 77                  LD      (HL),A            ;Store in KMAP
D4A9 0608                LD      B,8               ;Do 8 rows
```

```
D4AB 3E01       RK4:    LD      A,1             ;Move to next row
D4AD CD4500             CALL    FFLP
D4B0 23                 INC     HL              ;Increment KMAP pointer
D4B1 DB00               IN      A,(0)           ;Read row status
D4B3 2F                 CPL                     ;Complement
D4B4 E67F               AND     7FH             ;Ignore bit 7
D4B6 AE                 XOR     (HL)            ;Same as last time?
D4B7 2007               JR      NZ,RK7          ;No => find out why
D4B9 10F0       RK5:    DJNZ    RK4             ;Fall thru when done
D4BB AF         RK6:    XOR     A               ;Clear carry
D4BC 3211C0             LD      (RKROW),A       ;No repeat key
D4BF C9                 RET
D4C0 0EFF       RK7:    LD      C,-1            ;Compute bit mask and
D4C2 1600               LD      D,0             ;column number
D4C4 37                 SCF
D4C5 CB12       RK8:    RL      D
D4C7 0C                 INC     C
D4C8 1F                 RRA
D4C9 30FA               JR      NC,RK8
D4CB 7A                 LD      A,D             ;Get bit mask
D4CC AE                 XOR     (HL)            ;Update map
D4CD 77                 LD      (HL),A
D4CE 7A                 LD      A,D             ;Get bit mask
D4CF A6                 AND     (HL)            ;Key released?
D4D0 28E7               JR      Z,RK5           ;Yes => ignore
D4D2 2111C0             LD      HL,RKROW        ;Point to KBD data
D4D5 70                 LD      (HL),B          ;Save row number
D4D6 23                 INC     HL
D4D7 72                 LD      (HL),D          ;Save bit mask
D4D8 3A0600             LD      A,(6)           ;NAS-SYS 3?
D4DB FEFE               CP      0FEH
D4DD 2005               JR      NZ,RK9          ;No => skip
D4DF CD1301             CALL    113H            ;Call NAS-SYS 3
D4E2 1803               JR      RK10
D4E4 CDC900     RK9:    CALL    0C9H            ;Call NAS-SYS 1
D4E7 30D2       RK10:   JR      NC,RK6          ;Undefined key => skip
D4E9 3213C0             LD      (RKVAL),A       ;Save ASCII value
D4EC 2A02C2             LD      HL,(RKLON)      ;Long delay
D4EF 180B               JR      RK12
D4F1 2A14C0     RK11:   LD      HL,(RKCNT)      ;Get counter
D4F4 2B                 DEC     HL              ;Decrement
D4F5 7C                 LD      A,H             ;Zero?
D4F6 B5                 OR      L
D4F7 2007               JR      NZ,RK13         ;No => skip
D4F9 2A04C2             LD      HL,(RKSHO)      ;Short delay
D4FC 3A13C0     RK12:   LD      A,(RKVAL)       ;Get ASCII value
D4FF 37                 SCF                     ;Indicate char
D500 2214C0     RK13:   LD      (RKCNT),HL      ;Save counter
D503 C9                 RET


                ; Print 2 spaces
                ;-----------------------------------------------
                ; Print 2 spaces using the SPACE routine
                ;-----------------------------------------------

D504 DF69       SP2:    SCAL    ZSPACE
D506 DF69               SCAL    ZSPACE
D508 C9                 RET


                ; Call routine number E
                ;-----------------------------------------------
                ; Call SCAL routine number E
                ;-----------------------------------------------

D509 E5         SCALI:  PUSH    HL
D50A D5                 PUSH    DE
D50B F5                 PUSH    AF
D50C 1600               LD      D,0
D50E 2A710C             LD      HL,(STAB)
D511 19                 ADD     HL,DE
D512 19                 ADD     HL,DE
D513 5E                 LD      E,(HL)
D514 23                 INC     HL
D515 56                 LD      D,(HL)
D516 EB                 EX      DE,HL
D517 F1                 POP     AF
D518 D1                 POP     DE
D519 E3                 EX      (SP),HL
D51A C9                 RET


                ;-----------------------------------------------
                ; PolyDos SCAL table (routines 7DH to 8FH)
                ;-----------------------------------------------

D51B 81D4       PDSCTB: DW      RKBD            ;7DH
D51D 04D5               DW      SP2             ;7EH
D51F 09D5               DW      SCALI           ;7FH
D521 52D5               DW      DSIZE           ;80H
D523 A9D0               DW      DRD             ;81H
D525 ACD0               DW      DWR             ;82H
D527 B8D0               DW      RDIR            ;83H
D529 DAD0               DW      WDIR            ;84H
D52B EFD0               DW      CFS             ;85H
D52D 85D1               DW      LOOK            ;86H
D52F F0D1               DW      ENTER           ;87H
D531 29D2               DW      COV             ;88H
D533 34D2               DW      COVR            ;89H
D535 A5D2               DW      CKER            ;8AH
D537 DFD2               DW      CKBRK           ;8BH
D539 FDD2               DW      CFMA            ;8CH
D53B 14D3               DW      SSCV            ;8DH
D53D 28D3               DW      JUMP            ;8EH
D53F 38D3               DW      POUT            ;8FH
```

```
                         ;-----------------------------------------------
                         ;
                         ;       PolyDos 2.0 R1 (G809/G815)
                         ;       Disk Driver Routines Section
                         ;
                         ;       By Anders Hejlsberg
                         ;       Copyright (C) 1981
                         ;       PolyData microcenter ApS
                         ;
                         ;       Routines will control a Gemini G809
                         ;       FDC card (Western Digital 1797 floppy
                         ;       disk controller chip) with up to four
                         ;       Pertec FD250 5.25" floppy disk drives
                         ;
                         ;-----------------------------------------------


                         ; Port definitions

00E0                     CMDREG: EQU     0E0H    ;1797 command register
00E0                     STSREG: EQU     0E0H    ;1797 status register
00E1                     TRKREG: EQU     0E1H    ;1797 track register
00E2                     SECREG: EQU     0E2H    ;1797 sector register
00E3                     DATREG: EQU     0E3H    ;1797 data register
00E4                     STPORT: EQU     0E4H    ;G809 status port
00E4                     DRPORT: EQU     0E4H    ;G809 drive select port

                         ; 1797 commands

000B                     CRSTOR: EQU     00BH    ;Restore
001B                     CSEEK:  EQU     01BH    ;Seek track
003B                     CSTEP:  EQU     03BH    ;Step one track
0088                     CRDSEC: EQU     088H    ;Read sectors
00A8                     CWRSEC: EQU     0A8H    ;Write sectors
00C0                     CRDADR: EQU     0C0H    ;Read address
00D0                     CCLEAR: EQU     0D0H    ;Force interrupt

                         ; Workspace

C07D                     IDHEAD: EQU     DSKWSP


                         ; Initialize disk drivers and select drive C

D541 CD9AD5              INIT:   CALL    CNVCOD          ;Convert drive code
D544 CDE1D5                      CALL    CLEAR           ;Clear 1797
D547 CDF0D5                      CALL    MOTON           ;Start motors
D54A 3E0B                        LD      A,CRSTOR        ;Restore R/W head
D54C CDE3D5                      CALL    C1797
D54F C3FBD5                      JP      TSTDSK          ;Test for disk

                         ; Return disk size of drive C in HL

D552 3E07                DSIZE:  LD      A,MAXDRV        ;Too big?
D554 B9                          CP      C
D555 3E28                        LD      A,28H           ;(Error 28 if so)
D557 D8                          RET     C               ;Yes => return
D558 AF                          XOR     A               ;No error
D559 CB51                        BIT     2,C             ;Double density?
D55B 21EC04                      LD      HL,35*18*2      ;(Double density size)
D55E C8                          RET     Z               ;Yes => return
D55F 21BC02                      LD      HL,35*10*2      ;Single density size
```

```
D562 AF                 XOR     A               ;No error
D563 C9                 RET

                  ; Read or write B sectors starting at sector DE
                  ; on drive C to or from memory starting at HL.
                  ; A=0 indicates read, A=-1 indicates write

D564 F5         RWSCTS: PUSH    AF              ;Save R/W flag
D565 CD28D6             CALL    DRSEL           ;Select drive
D568 202D               JR      NZ,RWS3         ;Error => skip
D56A CDB2D5             CALL    CNVSAD          ;Convert sector addr
D56D 2028               JR      NZ,RWS3         ;Error => skip
D56F CDF0D5             CALL    MOTON           ;Start motors
D572 F1         RWS1:   POP     AF              ;Restore R/W flag
D573 F5                 PUSH    AF
D574 CDC9D6             CALL    RWSR            ;Read/Write one sector
D577 201E               JR      NZ,RWS3         ;Error => skip
D579 05                 DEC     B               ;Decrement count
D57A 281B               JR      Z,RWS3          ;Done => skip
D57C 24                 INC     H               ;Calculate next addr
D57D 1C                 INC     E               ;Increment sector nbr
D57E 3A02C0             LD      A,(DRVCOD)      ;Double density?
D581 CB67               BIT     4,A
D583 0E24               LD      C,18*2          ;(Double density size)
D585 2802               JR      Z,RWS2          ;Yes => skip
D587 0E14               LD      C,10*2          ;Single density size
D589 7B         RWS2:   LD      A,E             ;Get sector nbr
D58A B9                 CP      C               ;Too big?
D58B 38E5               JR      C,RWS1          ;No => skip
D58D 1E00               LD      E,0             ;Clear sector nbr
D58F 14                 INC     D               ;Increment track nbr
D590 7A                 LD      A,D             ;Get track nbr
D591 FE23               CP      35              ;Too big?
D593 38DD               JR      C,RWS1          ;No => skip
D595 3E29               LD      A,29H           ;Error 29
D597 E1         RWS3:   POP     HL              ;Adjust
D598 B7                 OR      A               ;Status to Z flag
D599 C9                 RET

                  ; Convert drive number in C to a drive code

D59A C5         CNVCOD: PUSH    BC              ;Save BC
D59B 41                 LD      B,C             ;Drive number to B
D59C 79                 LD      A,C             ;and to C
D59D E604               AND     4               ;Isolate density
D59F 07                 RLCA                    ;Move to bit 4
D5A0 07                 RLCA
D5A1 4F                 LD      C,A             ;Put in C
D5A2 78                 LD      A,B             ;Isolate drive number
D5A3 E603               AND     3
D5A5 3C                 INC     A               ;Make 1-4
D5A6 47                 LD      B,A             ;Put in B
D5A7 AF                 XOR     A               ;Set bit B in A
D5A8 37                 SCF
D5A9 17         CC1:    RLA
D5AA 10FD               DJNZ    CC1
D5AC B1                 OR      C               ;Include density
D5AD 3202C0             LD      (DRVCOD),A      ;Save as drive code
D5B0 C1                 POP     BC              ;Restore BC
D5B1 C9                 RET

                  ; Convert a sector address in DE into a track
```

```
                ; number in D and a sector number in E

D5B2 E5         CNVSAD: PUSH    HL              ;Save
D5B3 C5                 PUSH    BC
D5B4 62                 LD      H,D             ;Put sector addr in HL
D5B5 6B                 LD      L,E
D5B6 3A02C0             LD      A,(DRVCOD)      ;Get drive code
D5B9 CB67               BIT     4,A             ;Double density?
D5BB 012400             LD      BC,18*2         ;(Double density size)
D5BE 2803               JR      Z,CSA1          ;Yes => skip
D5C0 011400             LD      BC,10*2         ;Single density size
D5C3 3EFF       CSA1:   LD      A,-1            ;Track counter
D5C5 3C         CSA2:   INC     A               ;Increment track nbr
D5C6 FE23               CP      34+1            ;Overflow?
D5C8 300B               JR      NC,CSA3         ;Yes => skip
D5CA B7                 OR      A               ;Subtract track size
D5CB ED42               SBC     HL,BC
D5CD 30F6               JR      NC,CSA2         ;No carry => repeat
D5CF 09                 ADD     HL,BC           ;Adjust
D5D0 57                 LD      D,A             ;Pick up track
D5D1 5D                 LD      E,L             ;Pick up sector
D5D2 AF                 XOR     A               ;No error
D5D3 1802               JR      CSA4
D5D5 3E26       CSA3:   LD      A,26H           ;Error 26
D5D7 C1         CSA4:   POP     BC              ;Restore
D5D8 E1                 POP     HL
D5D9 B7                 OR      A               ;Status to Z flag
D5DA C9                 RET

                ; Delay for B milliseconds. Set up for 4MHz
                ; clock without wait states. The delay value
                ; need not be modified for slower clock rates.
                ; Note, however, that the minimum clock rate
                ; is 2MHz without wait states.

D5DB 3E5E       DELAY:  LD      A,94
D5DD FF                 RST     RDEL
D5DE 10FB               DJNZ    DELAY
D5E0 C9                 RET

                ; Clear the 1797

D5E1 3ED0       CLEAR:  LD      A,CCLEAR

                ; Do a 1797 type I command

D5E3 D3E0       C1797:  OUT     (CMDREG),A      ;Output command
D5E5 3E0A               LD      A,10            ;Small delay
D5E7 3D         C1A:    DEC     A
D5E8 20FD               JR      NZ,C1A
D5EA DBE0       C1B:    IN      A,(STSREG)      ;Done?
D5EC 1F                 RRA
D5ED 38FB               JR      C,C1B           ;No => wait
D5EF C9                 RET

                ; Keep drive motors running

D5F0 3A02C0     MOTON:  LD      A,(DRVCOD)      ;Get drive code
D5F3 D3E4               OUT     (DRPORT),A      ;Start drive
D5F5 DBE0       MO1:    IN      A,(STSREG)      ;Running?
D5F7 17                 RLA
D5F8 38FB               JR      C,MO1           ;No => wait
```

```
D5FA C9                  RET

                 ; Test that a disk is present in selected drive

D5FB C5          TSTDSK: PUSH    BC              ;Save BC
D5FC CDF0D5              CALL    MOTON           ;Start motors
D5FF 0664                LD      B,100           ;In case head loading
D601 CDDBD5              CALL    DELAY
D604 3EC0                LD      A,CRDADR        ;Do a read address
D606 D3E0                OUT     (CMDREG),A
D608 0E96                LD      C,150           ;Must complete in 150ms
D60A 0601        TD1:    LD      B,1             ;Delay one ms
D60C CDDBD5              CALL    DELAY
D60F DBE0                IN      A,(STSREG)      ;Done?
D611 CB47                BIT     0,A
D613 2803                JR      Z,TD2           ;Yes => skip
D615 0D                  DEC     C               ;Timeout?
D616 20F2                JR      NZ,TD1          ;No => retry
D618 CDE1D5      TD2:    CALL    CLEAR           ;Clear 1797
D61B AF                  XOR     A               ;No error
D61C 0C                  INC     C               ;Timeout?
D61D 0D                  DEC     C
D61E 2005                JR      NZ,TD3          ;No => skip
D620 3202C0              LD      (DRVCOD),A      ;No drive selected
D623 3E27                LD      A,27H           ;Error 27
D625 C1          TD3:    POP     BC              ;Restore BC
D626 B7                  OR      A               ;Status to Z flag
D627 C9                  RET

                 ; Select drive C

D628 3E07        DRSEL:  LD      A,MAXDRV        ;Too big?
D62A B9                  CP      C
D62B 3E28                LD      A,28H           ;(Error 28 if so)
D62D D8                  RET     C               ;Yes => return
D62E C5                  PUSH    BC              ;Save BC
D62F 3A02C0              LD      A,(DRVCOD)      ;Get current drive code
D632 47                  LD      B,A             ;Put in B
D633 CD9AD5              CALL    CNVCOD          ;Convert new drive code
D636 4F                  LD      C,A             ;Put in C
D637 CDF0D5              CALL    MOTON           ;Start motors
D63A 78                  LD      A,B             ;Drive already selected?
D63B 91                  SUB     C
D63C 2820                JR      Z,DRS1          ;Yes => bye
D63E CDFBD5              CALL    TSTDSK          ;Test for disk
D641 201B                JR      NZ,DRS1         ;Error => skip
D643 E5                  PUSH    HL              ;Save
D644 D5                  PUSH    DE
D645 DBE1                IN      A,(TRKREG)      ;Get track nbr
D647 57                  LD      D,A             ;Put in D
D648 1E00                LD      E,0             ;Dummy sector
D64A 217DC0              LD      HL,IDHEAD       ;Read ID header
D64D 3E01                LD      A,1
D64F CDC9D6              CALL    RWSR
D652 D1                  POP     DE              ;Restore
D653 E1                  POP     HL
D654 3E27                LD      A,27H           ;(In case error)
D656 2006                JR      NZ,DRS1         ;Error => skip
D658 3A7DC0              LD      A,(IDHEAD)      ;Pick up track
D65B D3E1                OUT     (TRKREG),A      ;Give it to 1797
D65D AF                  XOR     A               ;No error
D65E C1          DRS1:   POP     BC              ;Restore
```

```
D65F B7                  OR      A               ;Status to Z flag
D660 C9                  RET

                ; Seek track D

D661 DBE1       SEEKTR:  IN      A,(TRKREG)      ;There already?
D663 BA                  CP      D
D664 C8                  RET     Z               ;Yes => bye
D665 7A                  LD      A,D             ;Seek track
D666 D3E3                OUT     (DATREG),A
D668 3E1B                LD      A,CSEEK
D66A CDE3D5              CALL    C1797
D66D C5                  PUSH    BC              ;Additional delay
D66E 0614                LD      B,20
D670 CDDBD5              CALL    DELAY
D673 C1                  POP     BC
D674 C9                  RET

                ; Read/Write sector E to/from memory
                ; A=0:  Read sector
                ; A=1:  Read address
                ; A=-1: Write sector

D675 C5         RDWR:    PUSH    BC              ;Save
D676 D5                  PUSH    DE
D677 E5                  PUSH    HL
D678 4F                  LD      C,A             ;Put R/W flag in C
D679 3A02C0              LD      A,(DRVCOD)      ;Get drive code
D67C CB67                BIT     4,A             ;Double density?
D67E 1612                LD      D,18            ;(18 sectors/track)
D680 2802                JR      Z,RW0           ;Yes => skip
D682 160A                LD      D,10            ;10 sectors/track
D684 7B         RW0:     LD      A,E             ;Get sector number
D685 0600                LD      B,0             ;(Side 0 flag)
D687 BA                  CP      D               ;On side 0?
D688 3803                JR      C,RW1           ;Yes => skip
D68A 92                  SUB     D               ;Adjust
D68B 0602                LD      B,2             ;Side 1 flag
D68D D3E2       RW1:     OUT     (SECREG),A      ;Output sector number
D68F CDF0D5              CALL    MOTON           ;Keep motors running
D692 0C                  INC     C               ;Write sector?
D693 2014                JR      NZ,RW4          ;No => skip
D695 0EE4                LD      C,STPORT        ;Point to STPORT
D697 3EA8                LD      A,CWRSEC        ;Get command
D699 B0                  OR      B               ;Include side
D69A D3E0                OUT     (CMDREG),A      ;Output command
D69C 7E         RW2:     LD      A,(HL)          ;Get next byte ready
D69D 23                  INC     HL
D69E ED40       RW3:     IN      B,(C)           ;Read status
D6A0 28FC                JR      Z,RW3           ;No requests => loop
D6A2 F2C2D6              JP      P,RW6           ;Jump on INTRQ
D6A5 D3E3                OUT     (DATREG),A      ;Output byte
D6A7 18F3                JR      RW2             ;Go get next
D6A9 0D         RW4:     DEC     C               ;Read sector?
D6AA 3E88                LD      A,CRDSEC        ;(Read sector command)
D6AC 2802                JR      Z,RW7           ;Yes => skip
D6AE 3EC0                LD      A,CRDADR        ;Read address command
D6B0 0EE4       RW7:     LD      C,STPORT        ;Point to STPORT
D6B2 B0                  OR      B               ;Include side
D6B3 D3E0                OUT     (CMDREG),A      ;Output command
D6B5 ED40       RW5:     IN      B,(C)           ;Read status
D6B7 28FC                JR      Z,RW5           ;No requests => loop
```

```
D6B9 F2C2D6               JP      P,RW6               ;Jump on INTRQ
D6BC DBE3                 IN      A,(DATREG)          ;Read byte
D6BE 77                   LD      (HL),A              ;Save it
D6BF 23                   INC     HL                  ;Point to next
D6C0 18F3                 JR      RW5
D6C2 DBE0         RW6:    IN      A,(STSREG)          ;Read status
D6C4 B7                   OR      A                   ;Status to Z flag
D6C5 E1                   POP     HL                  ;Restore
D6C6 D1                   POP     DE
D6C7 C1                   POP     BC
D6C8 C9                   RET

                  ; Read/Write sector E from track D with up to
                  ; eight retries

D6C9 C5           RWSR:   PUSH    BC                  ;Save
D6CA 47                   LD      B,A                 ;Put R/W flag in B
D6CB 0E08         RWR0:   LD      C,8                 ;Set retry count
D6CD CD61D6       RWR1:   CALL    SEEKTR              ;Seek track D
D6D0 78           RWR2:   LD      A,B                 ;Get R/W flag
D6D1 CD75D6               CALL    RDWR                ;Do read/write
D6D4 2824                 JR      Z,RWR7              ;No error => done
D6D6 0D                   DEC     C                   ;Done 8 retries?
D6D7 2817                 JR      Z,RWR4              ;Yes => skip
D6D9 CB47                 BIT     0,A                 ;Drive not ready?
D6DB 3E20                 LD      A,20H               ;(Error 20 if so)
D6DD 201B                 JR      NZ,RWR7             ;Yes => skip
D6DF CB41                 BIT     0,C                 ;Odd retry?
D6E1 20ED                 JR      NZ,RWR2             ;Yes => skip
D6E3 3E3B                 LD      A,CSTEP             ;Load step command
D6E5 CB49                 BIT     1,C                 ;2nd or 6th retry?
D6E7 2002                 JR      NZ,RWR3             ;No => skip
D6E9 3E0B                 LD      A,CRSTOR            ;Load restore command
D6EB CDE3D5       RWR3:   CALL    C1797               ;Do command
D6EE 18DD                 JR      RWR1                ;Go retry
D6F0 061F         RWR4:   LD      B,1FH               ;Compute error code
D6F2 04           RWR5:   INC     B
D6F3 17                   RLA
D6F4 30FC                 JR      NC,RWR5
D6F6 CDE1D5               CALL    CLEAR               ;Clear 1797
D6F9 78                   LD      A,B                 ;Put code in A
D6FA C1           RWR7:   POP     BC                  ;Restore
D6FB B7                   OR      A                   ;Status to Z flag
D6FC C9                   RET

D6FD              $END:   END
```

```
$END    D6FD    ABORT   D2C8    ARG1    0C0C
ARG10   0C1E    ARG2    0C0E    ARG3    0C10
ARG4    0C12    ARG5    0C14    ARG6    0C16
ARG7    0C18    ARG8    0C1A    ARG9    0C1C
ARGC    0C0A    ARGN    0C0B    ARGX    0C2B
BIN     D43C    BIN1    D440    BL1     D41F
BL3     D447    BL4     D475    BLINK   D419
BLINKF  C016    BNSC    C00A    BRAM    C008
BREAK   C006    BRKADR  0C23    BRKPT   0020
BRKVAL  0C25    BS      0008    C1797   D5E3
C1A     D5E7    C1B     D5EA    CC1     D5A9
CCLEAR  00D0    CCR     0018    CFDRV   C00C
CFFLG   C00B    CFMA    D2FD    CFNSC   C00F
CFS     D0EF    CFS1    D0F8    CFS10   D16A
CFS11   D135    CFS12   D132    CFS2    D124
CFS3    D128    CFS4    D129    CFS5    D145
CFS6    D15C    CFS7    D164    CFS8    D171
CFS9    D173    CFSBP   C010    CFSEC   C00D
CH      0017    CKB1    D2F9    CKBRK   D2DF
CKER    D2A5    CLEAR   D5E1    CLIN    C01B
CLINP   C019    CMDREG  00E0    CMPS    D1DF
CNVCOD  D59A    CNVSAD  D5B2    CONFLG  0C26
COV     D229    COVR    D234    CPS1    D1E3
CPS2    D1E8    CR      000D    CRDADR  00C0
CRDSEC  0088    CRSTOR  000B    CRT     D3C7
CRT1    D3DF    CRT2    D3EB    CRTC    D3F6
CRTC1   D405    CSA1    D5C3    CSA2    D5C5
CSA3    D5D5    CSA4    D5D7    CSEEK   001B
CSL     0015    CSR     0016    CSTEP   003B
CUD     0014    CUL     0011    CUR     0012
CURBLR  C201    CURCHR  C200    CURSOR  0C29
CUU     0013    CWRSEC  00A8    DATREG  00E3
DBREAK  D2C4    DDRV    C001    DELAY   D5DB
DIRBUF  C400    DNAME   C400    DNNIM   D410
DRD     D0A9    DRPORT  00E4    DRS1    D65E
DRSEL   D628    DRVCOD  C002    DRW     D0AE
DSIZE   D552    DSKWSP  C07D    DWR     D0AC
ENT1    D1FC    ENT2    D225    ENTER   D1F0
ERRCOD  C005    ERRFLG  C004    ESC     001B
FCBS    C418    FEXA    0012    FEXT    0008
FF      000C    FFLP    0045    FIRST   C003
FLDA    0010    FNAM    0000    FNSC    000E
FSEC    000C    FSFL    000A    FUFL    000B
GETCH   D177    GETOV   D253    GOV1    D266
GOV2    D291    IDHEAD  C07D    INFOFA  C200
INIT    D541    INSLEN  C214    INSTR   C215
INTA    0C75    INTBL   D416    JUMP    D328
KMAP    0C01    KOPT    0C27    KTAB    0C6F
KTABL   0C6D    LF      000A    LK1     D18E
LK2     D18F    LK3     D1A1    LK4     D1AD
LK5     D1B7    LK6     D1BA    LK7     D1C3
LK8     D1CB    LK9     D1DD    LOOK    D185
MAXDRV  0007    MDRV    C000    MO1     D5F5
MONSTK  0C2C    MOTON   D5F0    NMIA    0C7E
NMIJ    0C7D    NUMN    0C20    NUMV    0C21
NXTFCB  C416    NXTSEC  C414    OUTTA   0C73
OVAREA  C800    OVFCB   C04B    OVNAM   C800
OVRLY   C804    PBMG    C211    PCHR    C240
PCPL    C212    PDC1    D01D    PDC2    D029
PDC3    D03A    PDCROM  D000    PDOSW   D09D
PDSCTB  D51B    PLCT    C017    PLMG    C213
PLPP    C210    PO1     D352    PO10    D3A4
```

```
PO11     D3A9    PO2      D353    PO3      D35E
PO4      D361    PO5      D36F    PO6      D37E
PO7      D387    PO8      D391    PO9      D3A2
PORTO    0C00    POUT     D338    PPOS     C018
PRCH     D3B7    PRCHT    D3AB    PRS      0028
RAF      0C67    RAM      1000    RBC      0C61
RCALH    0010    RDE      0C63    RDEL     0038
RDIR     D0B8    RDWR     D675    RESET    0000
RHL      0C65    RIN      0008    RK1      D48D
RK10     D4E7    RK11     D4F1    RK12     D4FC
RK13     D500    RK2      D49C    RK3      D4A2
RK4      D4AB    RK5      D4B9    RK6      D4BB
RK7      D4C0    RK8      D4C5    RK9      D4E4
RKBD     D481    RKBIT    C012    RKCNT    C014
RKLON    C202    RKROW    C011    RKSHO    C204
RKVAL    C013    ROUT     0030    RPC      0C69
RSP      0C6B    RW0      D684    RW1      D68D
RW2      D69C    RW3      D69E    RW4      D6A9
RW5      D6B5    RW6      D6C2    RW7      D6B0
RWR0     D6CB    RWR1     D6CD    RWR2     D6D0
RWR3     D6EB    RWR4     D6F0    RWR5     D6F2
RWR7     D6FA    RWS1     D572    RWS2     D589
RWS3     D597    RWSCTS   D564    RWSR     D6C9
S1FCB    C055    S2FCB    C069    SCALH    0018
SCALI    D509    SCTB     C100    SCTBS    C07E
SECBUF   C300    SECREG   00E2    SEEKTR   D661
SP2      D504    SSCV     D314    STAB     0C71
STACK    1000    STMON    000D    STPORT   00E4
STSREG   00E0    SYSWSP   C083    TAB      0009
TCH1     D180    TD1      D60A    TD2      D618
TD3      D625    TOP      C000    TRKREG   00E1
TROVN    D296    TSTCH    D179    TSTDSK   D5FB
UINA     0C7B    UINJ     0C7A    UOUTA    0C78
UOUTJ    0C77    USRWSP   C0C0    VRAM     080A
WDIR     D0DA    XOPT     0C28    ZARGS    0060
ZATE     0073    ZB1HEX   007A    ZB2HEX   0068
ZBLINK   007B    ZCFMA    008C    ZCFS     0085
ZCKBRK   008B    ZCKER    008A    ZCOV     0088
ZCOVR    0089    ZCPOS    007C    ZCRLF    006A
ZCRT     0065    ZDRD     0081    ZDSIZE   0080
ZDWR     0082    ZENTER   0087    ZERRM    006B
ZFFLP    005E    ZIN      0062    ZINLIN   0063
ZJUMP    008E    ZKBD     0061    ZLOOK    0086
ZMFLP    005F    ZMRET    005B    ZNIM     0072
ZNNIM    0078    ZNNOM    0077    ZNOM     0071
ZNUM     0064    ZPOUT    008F    ZRDIR    0083
ZRKBD    007D    ZRLIN    0079    ZSCALI   007F
ZSCALJ   005C    ZSOUT    006D    ZSP2     007E
ZSPACE   0069    ZSRLIN   0070    ZSRLX    006F
ZSSCV    008D    ZTBCD2   0067    ZTBCD3   0066
ZTDEL    005D    ZTX1     006C    ZUIN     0076
ZUOUT    0075    ZWDIR    0084    ZXKBD    0074
ZXOUT    006E
```

```
                  ;-----------------------------------------------
                  ;
                  ;       PolyDos 2.0
                  ;
                  ;       Emsg overlay
                  ;       The error message writer
                  ;
                  ;       By Anders Hejlsberg
                  ;       Copyright (C) 1981
                  ;       PolyData microcenter ApS
                  ;
                  ;-----------------------------------------------


                          REFS    SYSEQU
                          REF

C800                      ORG     OVAREA
C800                      IDNT    $,0

                  ; Overlay name

C800 456D7367             DB      'Emsg'

                  ; Overlay entry point

C804 B7                   OR      A
C805 C8                   RET     Z
C806 47                   LD      B,A
C807 213FC8               LD      HL,EMSGTB
C80A 7E           SEARCH: LD      A,(HL)
C80B 23                   INC     HL
C80C B7                   OR      A
C80D 2814                 JR      Z,NOMSG
C80F B8                   CP      B
C810 2807                 JR      Z,PRINT
C812 CB7E         SKIP:   BIT     7,(HL)
C814 23                   INC     HL
C815 28FB                 JR      Z,SKIP
C817 18F1                 JR      SEARCH
C819 7E           PRINT:  LD      A,(HL)
C81A E67F                 AND     7FH
C81C F7                   RST     ROUT
C81D CB7E                 BIT     7,(HL)
C81F 23                   INC     HL
C820 28F7                 JR      Z,PRINT
C822 C9                   RET
C823 EF           NOMSG:  RST     PRS
C824 3F4E6F20             DB      '?No message for error ',0
C83B 78                   LD      A,B
C83C DF68                 SCAL    ZB2HEX
C83E C9                   RET

C83F              EMSGTB: EQU     $

                  ; PolyDos error messages

C83F 53796E74             DC  'Syntax error'
C84B 02                   DB  02H
C84C 546F6F20             DC  'Too many/few parameters'
C863 03                   DB  03H
C864 42616420             DC  'Bad parameters'
```

```
C872 10                 DB  10H
C873 496C6C65           DC  'Illegal character in filename'
C890 11                 DB  11H
C891 46696C65           DC  'Filename too long'
C8A2 12                 DB  12H
C8A3 42616420           DC  'Bad drive identifier'
C8B7 13                 DB  13H
C8B8 46696C65           DC  'Filename missing'
C8C8 14                 DB  14H
C8C9 45787465           DC  'Extension missing'
C8DA 15                 DB  15H
C8DB 44726976           DC  'Drive number missing'
C8EF 20                 DB  20H
C8F0 44726976           DC  'Drive not ready'
C8FF 21                 DB  21H
C900 4469736B           DC  'Disk write protected'
C914 22                 DB  22H
C915 57726974           DC  'Write fault'
C920 23                 DB  23H
C921 5265636F           DC  'Record not found'
C931 24                 DB  24H
C932 43686563           DC  'Checksum error'
C940 25                 DB  25H
C941 4C6F7374           DC  'Lost data error'
C950 26                 DB  26H
C951 42616420           DC  'Bad disk address'
C961 27                 DB  27H
C962 4E6F2064           DC  'No disk or wrong format'
C979 28                 DB  28H
C97A 496C6C65           DC  'Illegal drive number'
C98E 29                 DB  29H
C98F 4469736B           DC  'Disk is full'
C99B 30                 DB  30H
C99C 49206361           DC  'I can''t find that file'
C9B2 31                 DB  31H
C9B3 54686174           DC  'That file already exists'
C9CB 32                 DB  32H
C9CC 44697265           DC  'Directory is full'
C9DD 33                 DB  33H
C9DE 49206361           DC  'I can''t do that to a locked file'
C9FE 40                 DB  40H
C9FF 49206361           DC  'I can''t rename across drives'

                ; DISK BASIC error messages

CA1B 80                 DB  80H
CA1C 4E455854           DC  'NEXT without FOR'
CA2C 81                 DB   81H
CA2D 53796E74           DC  'Syntax error'
CA39 82                 DB  82H
CA3A 52455455           DC  'RETURN without GOSUB'
CA4E 83                 DB  83H
CA4F 4F757420           DC  'Out of data'
CA5A 84                 DB  84H
CA5B 46756E63           DC  'Function call error'
CA6E 85                 DB  85H
CA6F 4F766572           DC  'Overflow'
CA77 86                 DB  86H
CA78 4F757420           DC  'Out of memory'
CA85 87                 DB  87H
CA86 556E6465           DC  'Undefined line'
CA94 88                 DB  88H
```

```
CA95 42616420          DC  'Bad subscript'
CAA2 89                DB  89H
CAA3 446F7562          DC  'Double defined dimension'
CABB 8A                DB  8AH
CABC 44697669          DC  'Division by zero'
CACC 8B                DB  8BH
CACD 496C6C65          DC  'Illegal in direct mode'
CAE3 8C                DB  8CH
CAE4 54797065          DC  'Type mismatch'
CAF1 8D                DB  8DH
CAF2 4F757420          DC  'Out of stringspace'
CB04 8E                DB  8EH
CB05 53747269          DC  'String too long'
CB14 8F                DB  8FH
CB15 53747269          DC  'String expression too complex'
CB32 90                DB  90H
CB33 49206361          DC  'I can''t continue'
CB43 91                DB  91H
CB44 556E6465          DC  'Undefined function'
CB56 92                DB  92H
CB57 4D697373          DC  'Missing operand'
CB66 93                DB  93H
CB67 496C6C65          DC  'Illegal unit number'
CB7A 94                DB  94H
CB7B 556E6974          DC  'Unit already open'
CB8C 95                DB  95H
CB8D 556E6974          DC  'Unit not open'
CB9A 96                DB  96H
CB9B 496E7661          DC  'Invalid format descriptor'
CBB4 97                DB  97H
CBB5 456E6420          DC  'End of file'
CBC0 98                DB  98H
CBC1 496E7661          DC  'Invalid record number'
CBD6 9A                DB  9AH
CBD7 4E756C6C          DC  'Null string'
CBE2 9B                DB  9BH
CBE3 49206361          DC  'I can''t open that unit'
CBF9 9C                DB  9CH
CBFA 556E6974          DC  'Unit not open for input'
CC11 9D                DB  9DH
CC12 556E6974          DC  'Unit not open for output'
CC2A 9E                DB  9EH
CC2B 49206361          DC  'I can''t position that unit'

                ; End of error message table

CC45 00                DB  0

CC46            $END:  END
```

PolyDos

UTILITIES GUIDE

PolyData
microcenter

TABLE OF CONTENTS

Section 1

Introduction

This manual describes the FORMAT, BACKUP, and SZAP utility programs included on your system disk. The above programs are not system commands, but seperate machine code files, i.e. files with the GO extension.

FORMAT is used to format disks, i.e. prepare blank disks for use. Furthermore, it can be used to verify disks. FORMAT is described in section 2.

BACKUP is used to make backup copies of disks. It is described in section 3.

SZAP (SuperZap) is a program that enables you to examine and modify sectors on a disk. It may be used by the experienced system programmer to recover files from a crashed disk. SuperZap is described in section 4.

Section 2

The FORMAT program

The disk format program is used to prepare blank disks for use and to verify the sectors on a disk. As the task of formatting a disk is dependant on the hardware configuration, FORMAT is available in two versions, one for the G809/G815 system, and one for the G805 system. When invoked, FORMAT clears the screen and outputs:

```
PolyDos 2.0 [xxxxx]
Disk Format Program
```

where xxxxx is the implementation name. Next, you must select a drive number for formatting/verification:

```
Which drive (0-X)?
```

where X is the largest drive number supported by your particular implementation (7 in the G809/G815 version, and 3 in the G805 version). Note that the FORMAT program has no special command to terminate itself. Each time a task is completed you are returned to the prompt above. To exit, you may at any time press CTRL/SHIFT/@ which returns you to the command level. Once the drive number has been established, you must select the function to be performed:

```
Format or verify (F/V)?
```

Type 'F' to select disk formatting, or 'V' to select disk verification. When verifying, FORMAT reads each sector on the disk, starting from sector 0000H. If an error occurs, the error number is displayed. If formatting was selected, yet another question appears:

```
Skew factor (0-4)?
```

to which you must answer a digit between 0 and 4. The skew factor determines the order in which the sectors of a track on the disk are numbered. If a disk is formatted with the sectors of each track being in sequential order, i.e. in the order 0, 1, 2, 3, etc., the disk will seem extremely slow to the system. The reason for this is that often after one sector has been read the controlling program spends a short time processing before reading the next sector. By the time the next transfer is requested the R/W head of the drive will be well past the header record of the next sector, if not several more sectors as well. Thus, the disk will have to do almost a complete revolution, before the next sector can be read. To get around this problem, the disk may be formatted with the sectors in a jumbled order, rather than being in sequential order, so that when the sectors are read/written in numerical order, a delay of up to four physical sectors can occur before the next wanted sector is found. The skew factor determines this delay. A skew of 0 indicates that no delay should occur between each transfer (assuming that the sectors are accessed in numerical order), i.e. that the sectors should be ordered sequentially. A skew of

1 indicates that a delay of one sector should occur between each transfer. Thus, in double density the sectors should appear in the order 0, 9, 1, 10, 2, 11,...., 8, 17, and in single density in the order 0, 5, 1, 6, 2, 7,...., 4, 9. As explained above, skew factors of 0 are hardly ever used. Normally, a skew of 1 is selected, but higher skews may be used to suit particular applications.

When the disk has been formatted, it is verified automatically. When the verification completes, FORMAT prompts you:

```
Disk name?
```

to which you must answer by entering a name of up to twenty characters. Once the name has been written to the disk, the disk is ready for use, and you may remove it from the drive.

Section 3

The BACKUP program

The disk backup program is used to make bakcup copies of disks. Obviously, the COPY command can be used for this purpose (e.g. COPY :0 :1;Y to backup all files on drive 0 to drive 1), and has the advantage that is packs the disk on the way, but BACKUP is usually faster, and it requires no system files to be present on the source disk. Also, BACKUP is ideal for converting disks between the two densities supported by the G809/G815 version. On running BACKUP, it will clear the screen and output:

```
PolyDos 2.0
Disk Backup Program
```

Following this a prompt for the source and destination drives will appear:

```
Source drive?
Destination drive?
```

It is permissable to enter the same drive number for both drives (or the same physical drive under G809/G815, e.g. 0 and 4), in which case BACKUP will ask you to insert another disk before each transfer (make sure that you don't get them mixed up!). If an error occur during a disk access the error number is output, for example:

```
Reading: Sector 02E7 >>>> Error 23
         Retry or ignore (R/I)?
```

Two options are offered. 'R' will do a retry (not that 8 retries may already have been attempted), and 'I' will ignore the error and continue from the next sector. Note that if you ignore an erroneously read sector, the data written to the destination disk will of course be incorrect. However, the ignore option may be used to recover as much data as possible from a crashed disk (errors may then be corrected later for example using the SuperZap program). If BACKUP is unable to read a correct copy of sector 0000H, it will backup all sectors on the source disk. Otherwise it will pick up the next-free-sector pointer from the directory information in sector 0000H, and only copy sectors below that pointer. If you are copying from a double density disk to a single density disk you may come upon the message:

```
No room on destination drive.
```

As the message suggests, there is not enough room on the single density disk, to hold all of the sectors in use on the source disk.

Once the backup is completed the program restarts allowing you to make other copies at the same time. To terminate the BACKUP program press CTRL/SHIFT/@.

Section 4

The SuperZap program

The SuperZap program is used to examine and modify disk contents on a one sector basis. SuperZap displays on the screen the contents of a sector in hex as well as in ASCII and allows you to move a cursor about in the image. Single bytes may be set to new values which can be entered in hex as well as in ASCII.

On running SuperZap (by entering SZAP), the screen is cleared and you must select a volume (drive) number. Following this, the first sector of that drive is read into the sector buffer, and an image is displayed. Due to the limited screen size, an entire sector cannot be displayed at one time. Instead, each sector is divied into a lower (bytes 00H-7FH) and an upper (80H-FFH) part. Each part is displayed as 16 lines of 8 bytes. Before each line the address of the first byte in the line is displayed. Then comes the actual data displayed as 8 two-digit hex number, and to the extreme left, 8 ASCII characters are displayed, representing the data as it would be written. SuperZap automatically swaps between the lower and upper parts of a sector when it is found necessary. Whenever you move beyond the current sector, SuperZap automatically updates the disk by writing the sector before the new sector is read. However, updates only occur when changes has been made to the sector. On the right hand of the display, the current volume/sector numbers are displayed. If an error occurs the error number is written to the screen below the sector number. In case of read errors the sector buffer is set to all zeroes.

To set a byte to a new value you must enter two hexdigits ('0'-'9' or 'A'-'F'). When the first digit is entered, the cursor moves to the second digit of the particular byte. When you enter the second digit the sector buffer is updated and you are moved left one byte.

The following commands are recognized by SuperZap (<LA> denotes the left arrow key, <RA> means right arrow, <UA> up arrow, and <DA> down arrow):

<LA> Move left one byte. If the cursor is at the first byte of a display frame, SuperZap displays the previous frame, and moves the cursor to the last byte. <BS> may be used instead of <LA>.

<RA> Move rigth one byte. If the cursor is at the first byte of a display frame, SuperZap displays the next frame, and moves the cursor to the first byte. <SPACE> may be used instead of <RA>.

<UA> Move the cursor up one line, i.e. to the left eight times.

<DA> Move the cursor down one line, i.e. to the right 8 times.

N Move to the next display frame.

P Move to the previous display frame.

S Set sector number. When 'S' is typed the cursor moves to the 'Sct' field. Following the command you must enter three hexdigits giving the address of the sector you want to move to.

V Set volume (drive) number. When 'V' is typed the cursor moves to the 'Vol' field. Following the command you must enter the number of the drive you want to select.

R Read sector. When 'R' is typed, the current sector is read from the disk, thus overwriting the changes you have made.

W Write sector. When 'W' is typed, the contents of the sector buffer is written to the disk. Note however, that a write is automatically done whenever you leave a sector.

<CH> Enter ASCII mode. When <CH> is typed, the cursor moves to the ASCII field of the display. When you are in the ASCII mode, all entries, except <CH>, will be entered directly into the sector buffer. If you press <CH> normal operation is restored.

Q Quit. The quit command terminates SuperZap, and returns you to the command level. If modifications have been made to the sector currently within the buffer, the sector is written to the disk before quitting.

PolyEdit

USERS GUIDE

PolyData
microcenter

TABLE OF CONTENTS

Section 1

Introduction

An editor is a system program that has a special use. To put it briefly, the editor allows you to create and change text. Specifically, the editor displays on the screen the contents of a text file and lets you move back and forth in the text, adding and deleting as you please. When you write text (including text form programs, such as BASIC programs or assembly language source files) you use the editor.

Section 2 of this manual describes to you how to invoke the editor, how to specify which file(s) you want to create and/or edit, and how to exit the editor. Section 3 describes how to operate the editor, i.e. the commands available and the functions they perform. Section 4 describes how to invoke the editor from you own machine code programs.

Section 2

Invoking the editor

The editor consists of two overlay files on the system disk. The first one, called Ecmd.OV, handles the EDIT command and file input/output. The second one, called Edit.OV, is the editor itself, i.e. the program that allows you to edit the files you have selected. To use the editor, both overlay files must be present on the master drive.

2.1 The command line

The command line used to invoke the editor must start with the command word EDIT followed by a list of file specifiers separated by blanks:

```
$EDIT <fspec>[ <fspec>]
```

At least one file specifier must follow the EDIT command word. When the editor has been loaded into memory it prompts:

```
PolyEdit x.x
```

where x.x is the version number. Next, the file specifiers are processed. PolyEdit starts with the first file specifier, which it looks up in the directory. If it exists, it is read into memory, and the next file specifier is processed. The reading of input files continues, each file being merged to the end of the text already read, until there is no more file specifiers on the command line, or an unexisting file is specified. If an unexisting file was specified, it becomes the output file. Otherwise, the first file specifier will function as output file specifier. If the extension is omitted from an input file, the first file with a matching name will be loaded. If the extension is omitted from an output file specifier, .TX is supplied. The default drive is always the master drive.

The examples that follow assume that you have a file called Letter.TX and a file MAINPROG.TX on the disk in drive 0, and a file called Intro.TX and a file called ROUTINES.TX on drive 1. Furthermore they assume that drive 0 is the master drive.

Command line: EDIT Letter
Response: Reading Letter.TX:0.
Output file is Letter.TX:0.
Comments: Since to drive number is included in the file specifier, the file must reside on drive 0. As no output file is specified, the input file specifier is used as output file specifier.

Command line: EDIT REPORT.BS
Response: Output file is REPORT.BS:0.
Comments: Since REPORT.BS does not exist on the master drive, no files are read, and REPORT.BS becomes the output file specifier.

Command line:	EDIT Letter Intro:1
Response:	Reading Letter.TX:0. Reading Intro.TX:1. Output file is Letter.TX:0.
Comments:	As Letter.TX as well as Intro.TX are existing files, both of them are read into memory, Letter.TX being the first file, and Intro.TX being merged to the end of Letter.TX. Since no output file is specified, the first file specifier, i.e. Letter.TX, is selected.

Command line:	EDIT MAINPROG ROUTINES:1 PROGRAM
Response:	Reading MAINPROG.TX:0. Reading ROUTINES.TX:1. Output file is PROGRAM.TX:0.
Comments:	Since PROGRAM.TX does not exist on the master drive it becomes the output file. Note that .TX is supplied by default.

If an error occurs while the command line is being processed, an error message is displayed, and control is returned to Exec.

When the command line has been processed without errors, PolyEdit prompts:

```
Press <SPACE> to continue
```

Press <SPACE> to enter the editor, or, to return to the command level, press CTRL/SHIFT/@. Once inside the editor you can exit in one of two ways. If you press CTRL/SHIFT/@ you are returned directly to the command level in PolyDos, and the text in memory is not saved. If you press CTRL/^ PolyEdit prompts:

```
Writing text to nnnnnn.ee:d.
```

where nnnnnn.ee:d is the output file specifier. If a file exists of the same name and extension as the output file, it is deleted, and the message:

```
(Old file deleted)
```

appears. If an error occurs, e.g. a disk is full error, while the file is being written, PolyEdit outputs:

```
WARNING: Disk is full.
New output file name?
```

Type a new file specifier, and PolyEdit will try write the file again. If you wish, you may insert another disk before entering the new file specifier.

2.2 Reentering PolyEdit

The following command line may be used to reenter PolyEdit:

```
$EDIT;W
```

Assuming that PolyEdit has been coldstarted prior to this command, and that no vital memory areas has been overwritten, the editor prompts:

```
PolyEdit x.x

Output file is nnnnnn.ee:d.

Press <SPACE> to continue
```

where nnnnnn.ee:d is the output file specifier. Press <SPACE>, and you will be returned to the editor.

Section 3

Editor operation

PolyEdit is a character oriented on-screen editor, which means that the display may be likened to a window, which can be moved about over the text. The cursor always resides within the window and by its position it determines where characters are to be deleted or inserted.

There is no limitations on line lengths. If a line is longer than 48 characters it swaps over the edge of the screen and continues on the next line. The piece of text you see on the screen always appears as it would when listed. This means that whenever a carriage return appears in the text, the following characters are displayed starting on a new line, and whenever a TAB character appears the following characters are displayed starting in the next multiple-of-8 column. Note that the top line i.e. the unscrolled line on top of the text screen is also used by PolyEdit, thus axpanding the number of lines displayed to 16.

As you will learn from section 3.2 the cursor can be represented either as a blinking cursor overlaying a character, or a solid non-blinking cursor inserted between two characters. The solid cursor never overlays a character, it just marks the current editing position. When you enter characters they are always inserted before the cursor, and when you delete characters it is always the characters before the cursor you remove.

3.1 Editor commands

The commands recognized by PolyEdit are entered as control characters, i.e. characters with ASCII values less than 20H. A control character is produced from the keyboard by pressing the CTRL key and another key simultaneously, or by pressing a control key, e.g. BACK, ENTER, CS, etc. Whenever you enter a character which is not a command, it is inserted in the text at the current cursor position, and the rest of the text on that line is pushed one character to the right.

Since control characters (characters with ASCII values less than 20H) are interpreted by PolyEdit as editor commands, you cannot enter these characters in the text using the CTRL key or one of the control keys. However, if you use the @ key instead of CTRL, the control character will be inserted instead of executed as a command. Thus, to insert a CTRL/L character in the text you would enter @/L, i.e. press @ and L simultaneously.

In many text editing applications one misses an ALPHA-LOCK key on the NASCOM keyboard, i.e. a key that will revert the SHIFT key function on alphabetic characters. PolyEdit supports this missing feature. Through the CTRL/G command (see section 3.1.5) you can select for the GRAPH key to function either as normally or as an ALPHA-LOCK key which, when depressed, reverts the SHIFT mode of alphabitics.

Editor commands are divided into 5 groups:

Cursor movement commands
Editing commands
Search/replace commands
Block commands
Various commands

The commands in each of these groups are described in the following sections. RA denotes the right arrow key, LA denotes the left arrow key, UA denotes the up arrow key, and DA denotes the down arrow key.

3.1.1 Cursor movement commands

RA Move cursor right one character.

LA Move cursor left one character.

SHIFT/RA Move cursor right to the next multiple-of-8 column.

SHIFT/LA Move cursor left to the next multiple-of-8 column.

UA Move cursor up one line.

DA Move cursor down one line.

SHIFT/UA Move cursor to the first character in the current line, or if the cursor is already at the beginning of a line, to the first character in the line above.

SHIFT/DA Move cursor to the last character in the current line (i.e. to the carriage return ending the line), or if the cursor is already at the end of a line, to the end of the next line.

CTRL/O Move cursor up one page (15 lines).

CTRL/N Move cursor down one page (15 lines).

CTRL/B Move cursor to the beginning of the text, i.e. to the first character in the text.

CTRL/E Move cursor to the end of the text, i.e. to the last character in the text.

3.1.2 Editing commands

BACK Delete character before the cursor. CTRL/H may be used instead of BACK.

CS Delete word before cursor. First, one character is deleted regardless of its value, and then characters before the cursor are repeatedly deleted until a space or a TAB or a carriage return is met. CTRL/L may be used instead of CS.

ESC Delete line before cursor. First, one character is deleted regardless of its value, and then characters before the cursor are repeatedly deleted until a carriage return is met. CTRL/[may be used instead of ESC.

LF Undelete one character. LF is used to recover characters deleted accidentally. CTRL/J may be used instead of LF.

CH Tabulate. Inserts a TAB character before the cursor. CTRL/W (or @/W) or CTRL/I (or @/I) may be used instead of CH.

ENTER New line. Inserts a carriage return before the cursor. CTRL/M (or @/M) may be used instead of ENTER.

CTRL/A Flip alphabetics to end-of-line. All alphabetic characters (A-Z, a-z, [, \,], {, |, and }) from the cursor to the next carriage return are flipped, i.e. upper case characters are turned into lower case, and lower case characters are turned into upper case.

3.1.3 Search/replace commands

CTRL/F Input search string and optionally a replace string, and find the first occurrance. When CTRL/F is pressed, PolyEdit prompts by printing a NULL character. You may now enter a search string of up to 255 characters. The only editing key available is BACK (CTRL/H) which deletes the last entered character. End the entry by pressing CTRL/F or CTRL/X. If CTRL/F is used, no replace string is input. If CTRL/X is used, yet another NULL is printed, and you may now enter a replace string. The maximum length of the replace string is 255 less the length of the search string. Again, end by pressing CTRL/F or CTRL/X. Once the string(s) are input, PolyEdit scans the text for an occurrance of the search string. If found, the cursor is moved to the character position just after the occurrance. If not, the cursor does not move. The search only includes the text after the cursor. To scan all of the text, use CTRL/B before CTRL/F.

CTRL/C Continue search. Continues searching for the search string entered using CTRL/F.

CTRL/X Replace. Replaces the search string by the replace string. CTRL/X only works if used immediately after a CTRL/F, CTRL/C, or CTRL/K command. Furthermore it is required that a replace string was input the last time you used CTRL/F.

CTRL/K Replace and find next. CTRL/K is equivalent to CTRL/X followed by CTRL/C.

3.1.4 Block commands

CTRL/P Set block marker. Block markers are used to delimit blocks in the text to be copied or deleted using CTRL/I or CTRL/D. When CTRL/P is entered, a start block marker is inserted in the text before the cursor. However, if the character before the cursor is already a block marker, no marker is inserted. Instead the existing marker is changed to the opposite type, i.e. a start block marker is changed to an end block marker, and vice versa. Hence, to insert an end block marker enter CTRL/P, which inserts a start block marker, followed by another CTRL/P, which changes the start block marker to an end block marker. Within the text a start block marker is stored as an ACK character (ASCII 06H), and an end block marker is stored as a NAK character (ASCII 15H). Thus, block markers can be inserted by entering @/F and @/U instead of CTRL/P.

CTRL/I Insert block. Inserts the first marked block before the cursor. The block marks are not included. If no blocks are marked, or if the cursor is within the first marked block, CTRL/I is ignored.

CTRL/D Delete block. Deletes the first marked block from the text. The block markers of the block are deleted as well. If no blocks are marked, CTRL/D is ignored.

CTRL/_ Delete all block markers. Scans the text for start and end block markers, i.e. ACK (ASCII 06H) and NAK (ASCII 15H) characters, and deletes them whenever they occur.

3.5 Various commands

CTRL/G Set GRAPH key function. CTRL/G must be followed by a character which determines the GRAPH key mode. 'G' (or 'g') makes the GRAPH key function as usual. 'A' (or 'a') makes the CRAPH key function a an ALPHA-LOCK key which, when depressed, reverts the SHIFT mode.

CTRL/] Flip display flag. The display flag determines if TAB (tabulate) and CR (carriage return) characters are to be shown on the display. If the display flag is set, TAB characters are shown as right arrows, and CR characters are shown as left arrows. If the display flag is clear, no characters are displayed on the screen to represent TABs and CRs. Whenever CTRL/] is entered, the display flag is complemented. The display defaults to reset, i.e. it is reset when the editor is invoked.

CTRL/^ Terminate editor. When CTRL/^ is entered the editor returns to the calling program. If the editor was invoked from an EDIT command (see section 2.1), the text is written to the output file, and control is returned to PolyDos.

3.2 Changing the cursor

Within PolyEdit the cursor can be represented either as a blinking cursor overlaying a character or as a solid non-blinking cursor inserted between two characters.

The EDIT command handler has a special mode which allows you to redefine the cursor character. To invoke this mode, enter the following command line:

```
$EDIT;C
```

which will prompt you:

```
New cursor ASCII value (in hex)?
```

If you enter 0, a blinking cursor is selected. Other values indicate a solid cursor, the value being the ASCII value of the cursor character. What happens now is that the Edit.OV overlay file is read into memory and modified to reflect the new cursor. If no errors occur, the new version is written to the disk, and you are returned to PolyDos.

If your NASCOM is equipped with a graphics character generator (the NAS-GRA ROM), a suitable value for a solid cursor would be 0DBH. It is a semigraphic character with the upper four pixels set.

Section 4

Using the Edit overlay

This section describes to you how to call the Edit overlay from your own machine code language programs. If you are not familiar with machine code programming, you may wish to skip this section (it's in no way required of you to understand it before operating the editor).

The Edit overlay can be invoked from your own programs using the COV and COVR SCAL routines (please refer to the system programmers guide for further details on these system services). The editor uses the directory buffer as workspace. Therefore, it sets DDRV to 0FFH before returning, to indicate that the buffer does not contain a valid directory. Interface to the overlay is done through five variables, all of which must be initialized before the editor is called.

Name	Addr	Size	Describtion
SOFP	CØC0	2	Start-of-file pointer. This location contains a pointer to the start address of the text buffer. The pointer should reserve one byte of free RAM below the start address. Thus, if RAM is available for the text buffer starting from address 2000H, SOFP should read 2001H.
EOFP	CØC2	2	End-of-file pointer. This location contains a pointer to the first free location after the text in the buffer. If no text is present when you call PolyEdit, EOFP should equal SOFP.
BTOP	CØC4	2	Buffer end address. This variable defines the end address of the text buffer. The location pointed to by BTOP is used to store data. Thus, the address in BTOP is the address of the last byte reserved for the buffer and not the address of the first unused location after the buffer.
CURADR	CØC6	2	Cursor address. When PolyEdit is invoked the cursor is moved to the address stored in CURADR, or, in other words, the cursor is moved forwards CURADR-SOFP times. Normally the value stored in SOFP is also stored in CURADR so that when PolyEdit is invoked the cursor is placed at the first character in the text. Before returning PolyEdit stores the cursor address in this location.
DKOPT	CØC8	1	Default keyboard options. PolyEdit will load the value in DKOPT into KOPT (address 0C27H) when it is invoked. If bit 0 is set, unshifted letters entered from the keyboard will be in lower case. If bit 1 is set, the GRAPH key is in the ALPHA-LOCK mode. Before returning PolyEdit stores the value in KOPT into DKOPT whereafter it stores a zero in KOPT.

Below is shown an example of a program using the Edit overlay. It allows you to enter some text in the editor, and when you exit the editor the text is printed on the printer.

```
          REFS    SYSEQU          ;Get symbols from SYSEQU
          REF                     ;Get all of them

;Define buffer parameters

BSTART:   EQU     02000H          ;Buffer start
BEND:     EQU     0C000H          ;Buffer end

;Define interface variables

          ORG     USRWSP

SOFP:     DS      2
EOFP:     DS      2
BTOP:     DS      2
CURADR:   DS      2
DKOPT:    DS      1

;Define program origin

          ORG     1000H
          IDNT    $,$

;Entry point

          LD      HL,BSTART+1     ;Get start address
          LD      (SOFP),HL       ;Put in SOFP
          LD      (EOFP),HL       ;and in EOFP
          LD      (CURADR),HL     ;and in CURADR
          LD      HL,BEND-1       ;Get end address
          LD      (BTOP),HL       ;Put in BTOP
          LD      A,2             ;Set default keyboard
          LD      (DKOPT),A       ;options
          SCAL    ZCOV            ;Invoke PolyEdit
          DB      'Edit'
          LD      HL,(SOFP)       ;Pick up start address
PRINT:    LD      DE,(EOFP)       ;Compare pointer to EOFP
          OR      A
          SBC     HL,DE           ;Finished?
          JR      NC,DONE         ;Yes => skip
          LD      A,(HL)          ;Get character
          PUSH    HL              ;Save pointer
          SCAL    ZPOUT           ;Print character
          POP     HL              ;Restore pointer
          INC     HL              ;Move to next
          JR      PRINT           ;Loop
DONE:     SCAL    ZMRET           ;Back to PolyDos

          END
```

PolyZap

USERS GUIDE

16E2	7E
1ABC	7E
1AFC	5E
1904	7E

⇒ PZAP printer [illegible]

PolyData
microcenter

TABLE OF CONTENTS

Section 1

Introduction to PolyZap

PolyZap is a program that processes source program statements written in ZILOG/MOSTEK Z-80 assembly language. The assembler translates these source statements into object code files compatible with PolyDos file format, and produces a listing of the source program. The symbolic language used to code source programs to be processed by the assembler is called assembly language. The language is a collection of operation code symbols (opcodes), representing machine code instructions, pseudo operations (pseudo-ops), representing directives to the assembler, symbolic names (symbols), and special symbols. The assembly language provides opcodes for all machine code instructions in the Z-80 instruction set. It also provides some pseudo-ops, which specify auxiliary actions to be performed by the assembler.

PolyZap is a two-pass assembler. During the first pass the source program is scanned to develop a symbol table. During the second pass the object file is created with reference to the table developed in pass one. It is during the second pass that the source listing is produced.

1.1 Notations used in this manual

To describe the syntax, or format, of command lines, source statements, and assembly options, the following notations are used throughout the manual:

[...]   Contains an optional element. If selected, the element may only be used once.

{...}   Contains an optional element. If selected, the element may be used any number of times.

<...>   Contains an element name. The forms the element can take on is described in the text.

As an example of these notations, consider the following line, which describes the syntax of an assembler source statement in general:

```
[<label>[:]][ <opcode>[ <operand>{,<operand>}]][;<comment>]
```

The first element in the line is an optional label. If selected, the label can optionally be followed by a colon. After this comes an opcode, also optional, which, if selected, must be preceded by at least one space to separate it from the label field. If the opcode was selected, it may be followed by a list of operands. If the operand list is specified, the first operand must be preceded by at least one space to separate it from the opcode field. If more than one operand is given, each operand must be separated from the previous one by a comma. The last field of a line is the comment field, which, if selected, must be preceded by a semicolon.

Section 2

Coding assembly language programs

Programs written in assembly language consist of a sequence of source statements. Each source statement consists of ASCII characters ending with a carriage return.

2.1 Source statement format

Each source statement may include up to four fields: A label field, an opcode field, an operand field, and a comment field. Each of these fields, and their appearance, are described in the following sections.

PolyZap does not differ between upper and lower case letters within the following assembly language elements: Symbols, operation codes, pseudo-ops, register names, condition codes, operators, and constants. Thus, the source statement:

```
START:  LD   HL,0A59EH
```

is exactly the same as:

```
start:  ld   hl,0a59eh
```

2.1.1 The label field

The label field occurs as the first field of a source statement. It may either be empty, or contain a symbol name, optionally followed by a colon. A symbol may have any length, but normally symbols are not longer than seven characters, as this would disturb the assembly listing format.

When a symbol appears in the label field, normally it will be assigned the value of the program counter, i.e. the address of the first byte of object code generated by the source statement. However, the EQU, SET, and DEFL pseudo-ops treat labels differently (see section 2.2).

2.1.2 The opcode field

The opcode field occurs after the label field and, if present, must be preceded by at least one space. Entries in the opcode field may be one of two types: A Z-80 operation code, e.g. ADD, LD, INC, which will be translated into its corresponding Z-80 machine code instruction, or a pseudo operation, representing a directive to the assembler. The pseudo-ops are described in section 2.2.

2.1.3 The operand field

The contents of the operand field depends on the operation code in the opcode field. Some opcodes don't require an operand (e.g.

NOP, CCF, END), in which case the operand field is empty. Other opcodes require a fixed number of operands, and some allow for a varying number of operands. If the operand field is not empty, it must be separated from the opcode field by at least one space, and if it is to contain more than one operand, the operands must be separated by commas. If the opcode field is empty, the operand field must also be empty.

2.1.4 The comment field

The comment field is always the last field of a source statement. If it is present, it must begin with a semicolon (;). The comment field is not in any way processed by the assembler, but it is included in the assembly listing for documentation purposes.

2.2 Symbols

Symbols are one of the most important features of assembly language. A symbol is a name with an associated value (16-bit), the name being used rather than explicitly stating the value.

A symbol is declared to the assembler by placing it in the label field of a source statement line. The value assigned to the symbol depends on the opcode of that line. If an EQU, SET, or DEFL pseudo-op appears in the opcode field, the value assigned to the symbol is given by the expression in the operand field. Otherwise the symbol will be assigned the value of the location counter (program counter). The latter use of symbols are called labels. A symbol may occur only once in a label field, unless it is used with a SET or a DEFL pseudo-op.

The following characters may be used to form symbols:

Alphabetic characters:	A-Z and a-z
Numeric characters:	0-9
Special characters:	$. ? @ _

A symbol must begin with an alphabetic or a special character, and may contain any number of alphabetic, numeric, or special characters after that. Note that the assembler does not differ between upper and lower case letters.

There are certain reserved keywords that may not be used as symbols. These are register names (e.g. A, D, HL, IX), condition codes (e.g. Z, NC, PE), operators (e.g. NOT, AND, SHR), and a single $ character.

2.3 The location counter

The location counter is an internal register of the assembler giving the address of the object code currently being generated. The initial value of the location counter is 0, but it can be changed by an ORG pseudo-op. The current value of the location counter may be referenced within an expression by using a single $ character as a symbol. Normally, the value returned is the address of the first byte of object code generated by the source statement. However, when using the pseudo-ops DB, DW, DEFB, and DEFW with multiple operands, the loaction counter will be incremented after coding each operand. Hence, the $ symbol will yield the address of the first byte of object code generated by the operand within which it is used.

2.4 Constants

Constants represent quantities of data that do not vary in value during execution of a program. Constants are either numeric constants, litteral constants (character constants), or strings.

Numeric constants can be any 16-bit integer represented in one of four bases: Hexadecimal, decimal, octal, or binary, with decimal being the default base. A numeric constant must always start with a digit (0-9). Hexadecimal constants are 'H' postfixed (e.g. 15H, 23C0H, 0FFFFH), decimal constants are optionally 'D' postfixed (e.g. 21D and 34801D, which is the same as 21 and 34801), octal constants are either 'O' or 'Q' postfixed (e.g. 13Q, 177777O), and binary constants are 'B' postfixed (e.g. 11B, 01101010B).

Character constants consists of a single ASCII character enclosed in single or double quotes (e.g. 'A', "$", '"'). The delimiter quotes may be used as characters if they appear twice (i.e. '''' and """").

Strings consist of two or more ASCII characters enclosed in single or double quotes. Similar to character constants, the delimiter qoutes can be used as characters if they appear twice for each occurrance desired (e.g. 'That''s all folks'). Everywhere a string is allowed, a character constant is also allowed.

2.5 Expressions

An expression is a combination of symbols, numeric or character constants, algebraic operators, and parentheses. The expression is used to specify a value which is to be used as an operand. Note that enclosing an expression entirely in parentheses indicates a memory address. Thus, the source statement line LD HL,(DATA+3) is NOT equivalent to LD HL,(DATA+3)*1. Expressions will evaluate into 16-bit values. Overflow resulting from arithmetic operations are not reported; instead the result will be truncated to the low order 16 bits.

The operators recognized by PolyZap, and their priority are listed below:

Operator	Function	Priority
HIGH	Isolate high order byte	1
LOW	Isolate low order byte	1
-	Unary minus (2's complement)	2
+	Unary plus (ignored)	2
*	Multiplication	3
/	Division	3
MOD or \	Modulo	3

SHL	Logical shift left	3
SHR	Logical shift right	3
+	Addition	4
-	Subtraction	4
EQ or =	Equal	5
NE or <>	Not equal	5
GE or >=	Greater than or equal	5
LE or <=	Less than or equal	5
GT or >	Greater than	5
LT or <	Less than	5
NOT	Logical NOT (1's complement)	6
AND or &	Logical AND	7
OR or ^	Logical OR	8
XOR or %	Logical XOR	8

All operators involving alphabetic characters must be separated from their operands by at least one space. The byte isolation operators (HIGH and LOW) isolate the high or low order 8 bits of their operand. HIGH X is equivalent to X/100H, and LOW X is equivalent to X AND 0FFH. The relational operators (EQ, NE, GE, LE, GT, and LT) interpret their arguments as unsigned integers, and returns 0 if the relation is false or -1 (0FFFFH) if the relation is true.

2.6 Symbol table files

Symbol table files (or symbolfiles) provide a meothod of referencing symbols defined in another assembly. A symbolfile contains the complete symbol table of its associated source file(s), i.e. the table created by PolyZap during assembly which gives the names and values of all symbols defined by the source file(s). Through the REFS and the REF pseudo-ops you can extract key sombols, i.e. addresses of subroutines and workspace locations from a symbolfile defined by another program.

The usefullness of this system is demonstrated by the situation in which you have a main program that sharing a number of utility subroutines and workspace locations with a collection of overlays (the concept of overlays is discussed in the PolyDos System Programmers Guide). Basically the overlays all run at address C800H and can only run one at a time. Therefore, they cannot communicate between each other, but only act as large swappable subroutines to the main program. The symbolfile system is used in this example to pass address information from the main program to the overlays. The main program creates a symbolfile, and the overlays reference key symbols using REFS and REF. The classic example of this use is in the PolyDos equate file (SYSEQU.SY), which defines the PolyDos workspace, the NAS-SYS workspace, the subroutine numbers, the ASCII character codes, etc. SYSEQU is included on your system disk, and it is suggested that you use it to referencing system locations. A listing of SYSEQU is given as an appendix to the PolyDos System Programmers Guide.

Section 3

Pseudo operations

The assembler pseudo operations (pseudo-ops) are instructions to the assembler rather than instructions to be directly translated into machine code. In the description of the various pseudo-ops, the syntax, or format, of the pseudo-op source statement line is given first. The following syntactical elements are used:

<label>	Statement label.
<expr>	Assembler expression.
<expression>	Assembler expression.
<string>	Character string.
<comment>	Any string of ASCII characters.

3.1 ORG - Define program counter origin

```
ORG <expression> [;<comment>]
```

The ORG pseudo-op changes the loction counter (program counter) to the value specified by the expression in the operand field. Subsequent statements are assembled to load into memory locations starting with the new location counter value. The expression may not contain forward references.

If object file creation was requested when PolyZap was invoked (see section 3), enough zeroes are put into the object file to fill the space from the last instruction assembled (before the ORG pseudo-op) to the address given by the expression in the ORG. Note that there is no way of ORGing backwards, i.e. to a lower address than the current location counter value, as the assembler can only output more object code and cannot retract what it has previously output. If you try doing a backwards ORG, you will get an error. However, if the backwards ORG occurs before any code is generated, PolyZap will allow it, and will only change the location counter. This makes it possible to put a group of DS statements in an area of memory beyond the program area, but to have these DS statements occur at the front of the program if desired.

3.2 IDNT - Define object file identity

```
IDNT <expression>,<expression> [;<comment>]
```

The IDNT pseudo-op is used to define the load and execute addresses of the object file. The first expression gives the load address, and the second expression gives the execute address. IDNT may only be used once within a program. If there is no IDNT statement in a program, and object file creation is requested, and error occurs.

3.3 EQU - Equate symbol to a value

```
<label>[:] EQU <expression> [;<comment>]
```

The EQU pseudo-op assigns the symbol in the label field the value given by the expression in the operand field. The expression may not involve forward references, a the symbol in the label field may not be a symbol already defined.

3.4 SET - Set symbol value

```
<label>[:] SET <expression> [;<comment]
```

The SET pseudo-op is identical to the EQU pseudo-op, except that no error is generated if the symbol in the label field is already defined.

3.5 DEFL - Define label

```
<label>[:] DEFL <expression> [;<comment]
```

The function performed by the DEFL pseudo-op is the same as that of a SET pseudo-op (see section 3.4).

3.6 REFS - Reference file specify

```
REFS <file specifier> [;<comment>]
```

The REFS pseudo-op gives the file specifier of the symbolfile to be used by subsequent REF pseudo-ops. The default extension of the file specifier is .SY, and the default drive is the master drive. The symbolfile remains open until a new REFS is given, or until a REF with no label is encountered.

3.7 REF - Reference a symbol

```
[<label>[:]] REF [;<comment>]
```

REF will search in the current symbolfile for the symbol given in the label field. If it exists, a symbol is created with the value of the correcponding symbol in the symbolfile. If no label is given, all symbols in the current symbolfile are copied to the symbol table.

3.8 DB - Define byte(s)

```
[<label>[:]] DB <value>{,<value>} [;<comment>]
```

The DB pseudo-op will generate bytes of data in the object code. The <value> can either be an expression or a string. In the case of an expression, one byte of data is generated, which means the the value of the expression must be within the range -128 to 127 or 0 to 255. The number of bytes generated by a string is given by its length. Each byte of data will be the seven bit ASCII value (high order bit is zero) of its associated character.

3.9 DC - Define character string

```
[<label>[:]] DC <string> [;<comment>]
```

DC stores the characters in <string> in successive memory locations beginning with the current location counter. As with DB, characters are stored with the high order bit set to zero. However, DC stores the last character of the string with the high order bit set to one.

3.10 DW - Define word(s)

```
[<label>[:]] DW <expr>{,<expr>} [;<comment>]
```

The DW pseudo-op will place words (adresses) in the object code. Each expression generates a word, i.e. two bytes, in the standard Z-80/8080 byte reversed form.

3.11 DS - Define storage

```
[<label>[:]] DS <expression> [;<comment]
```

The DS pseudo-op is used to reserve data areas that need no initial value. The effect of the DS pseudo-op is that the value of the expression in the operand field is added to the location counter, thus reserving the specified number of bytes.

If an object file is being created, zeroes will be filled into areas reserved using DS. However, filling only occurs where it is necessary, i.e. only if the DS statement is followed by some statements that generate object code.

3.12 DEFB - Define byte(s)

```
[<label>[:]] DEFB <expr>{,<expr>} [;<comment>]
```

The DEFB pesudo-op is identical to the DB pseudo-op, except that strings are not allowed as operands.

3.13 DEFM - Define message

```
[<label>[:]] DEFM <string> [;<comment>]
```

DEFM will place into the object code the seven-bit ASCII values (high order bit is zero) of the characters in the string given in the operand field.

3.14 DEFW - Define word(s)

```
[<label>[:]] DEFW <expr>{,<expr>} [;<comment>]
```

The functions performed by the DEFW pseudo-op is the same as those of a DW pseudo-op (see section 3.10).

3.15 DEFS - Define storage

```
[<label>[:]] DEFS <expression> [;<comment>]
```

The DEFS pseudo-op is identical to the DS pseudo-op (see section 3.11).

3.16 END - End of source program

```
END [;<comment>]
```

The END pseudo-op marks the end of the source program. Source statement lines following the END pseudo-op will not be processed, and will not be printed in the source listing.

3.17 Conditional assembly

The conditional pseudo-ops allow selective skipping of source statement lines. A skipped line is completely ignored except for a quick check to if it contains another conditional pseudo-op. There are five conditional pseudo-ops: IF, ELSE, ENDIF, COND, and ENDC, COND and ENDC being identical to IF and ENDIF. Sections of source statement lines are delimited by an IF/ENDIF pair, with possible ELSEs in between. The sections can be nested within one another to a depth of 15 levels.

3.17.1 IF

```
IF <expression> [;<comment>]
```

If the value of the expression in the operand is false (zero), the statement lines following the IF pseudo-op are skipped. If the value of the expression is true (non-zero) the statements following the IF pseudo-op is assembled normally.

3.17.2 ELSE

```
ELSE [;<comment>]
```

The ELSE pseudo-op acts to switch from skipping to non-skipping and non-skipping to skipping mode between an IF and an ENDIF. An arbitrary number of ELSEs may occur within an IF/ENDIF pair, each time the result being reversion of the skipping mode.

3.17.3 ENDIF

```
ENDIF [;<comment>]
```

The ENDIF pseudo-op gives the end of a section of conditional source statements, started by its matching IF pseudo-op.

3.17.4 COND

```
COND <expression> [;<comment>]
```

The COND pseudo-op is identical to the IF pseudo-op (see section 3.17.1).

3.17.5 ENDC

```
ENDC [;<comment>]
```

The ENDC pseudo-op is identical to the ENDIF pseudo-op (see section 3.17.3).

Section 4

Operating PolyZap

PolyZap is supplied on your system disk as a machine code program file called PZAP.GO. The format of the command line used to invoke PolyZap is:

```
        PZAP <source>[,<source>][ <object>][;<options>]
```

where <source> denotes a source file specifier, <object> denotes an optional object file specifier, and <options> denote an optional option list. The default drive is always the master drive. If no extension is given to a source file specifier, the first file of a matching name is used. The default extension for the object file is .GO, i.e. a machine code program file. PolyZap will accept up to 8 source files for assembly. During pass one and pass two the source files are read one by one, in the order they appear on the command line, and assembled, until an END statement is met, or until the last source file has been processed. When a new file is read, the symbol table extracted from prevoius source files is preserved. Thus, the input files appear to PolyZap as one contignuous file. Each time a source file is read into memory a message is displayed:

```
Pass xxx: nnnnnn.ee:d.
```

where xxx is either 'one' or 'two', and nnnnnn.ee:d is the file specifier.

If the assembler runs out of memory when reading a source file, or if the symbol table overflows the available memory, the message:

```
Memory overflow
```

is output, and control is transferred back to PolyDos. To avoid memory overflow, split your source file into two or more smaller source files.

If PolyZap detects an error, an error message will be output followed by the erroneous line, even if no listing was requested. If errors occur during pass one, pass two will be aborted unless you have specifically forced it using the F assembly option.

At any time you may press CTRL/SHIFT/@ to abort PolyZap and return to the command level in PolyDos.

4.1 Assembly options

The assembly options may be zero or more of the options described in this section. The assembly options appear as the last entry on the command line, and if any are specified they must be prefixed by a semicolon, to separate them from the rest of the command line. Some examples of command lines involving assembly options:

```
                    $PZAP BACKUPS;S
                    $PZAP TYPES Type.OV;F
                    $PZAP GMS1,GMS2 GAME;LPGD
                    $PZAP SYMBOLS;C
```

4.1.1 The L option

The L option instructs PolyZap to output an assembly listing during pass two. If the options list includes a P option, the listing will be directed to the printer. Otherwise the screen will be used for output. When listing to the screen PolyZap will pause and blink the cursor each time 15 lines has been output. To continue press any key.

4.1.2 The P option

The P option specifies that assembler output should be sent to the printer. Furthermore it instructs PolyZap to print a heading and a page number on top of each page. If the P option appears in the option list you will be prompted:

```
                              Heading?
```

The maximum length of the heading is 36 characters.

4.1.3 The S option

The S option instructs PolyZap to output an alphabetically sorted symbol table listing at the end of pass two.

4.1.4 The C option

The C option requests that PolyZap save the symbol table created during assembly in a symbol table file. The symbols in a symbol table file can be accessed from another assembly using the REFS and REF pseudo-ops (see section 3.6 and 3.7). If the C option appears in the option list you will be prompted:

```
                          Symbolfile name?
```

Answer by entering a file specifier. The default extension is SY, and the default drive is the master drive. Note that the assembler will not create a symbol reference file if errors occur during assembly.

4.1.5 The G option

The G option instructs the assembler to print all object codes generated by DB, DC, DW, DEFB, DEFM, and DEFW pseudo-ops, and not just the first four bytes.

4.1.6 The D option

The D option instructs PolyZap to omit the printing of lines containing conditional pseudo-ops, and lines skipped as an effect of these, in the assembly listing.

4.1.7 The F option

The F option instructs PolyZap to force the second pass through the source program, even if errors occur during the first pass.

4.2 Assembler error handling

Assembly errors detected by PolyZap are displayed before the actual line containing the error. Errors are accumulated, and the total number of errors is printed an the end of each pass. If no listing was requested, assembly error messages are still displayed to indicate the the assembly process did not proceed normally. The format of an error report is:

```
>>>> ERROR      Error message
```

The code generated to a source statement in error is not predictable. Some error conditions will produce code, others will not, depending on the type of error. Below all error messages are described:

Parentheses error

The parantheses in an operand do not balance.

Error in string

A string or a character constant is empty or an ending quote is missing. Note that the beginning and the ending qoute must be of the same type.

Error in constant

An illegal digit was detected in a constant.

Undefined symbol

A symbol appearing in an expression in the operand has not been declared as a label. A symbol value of zero is assumed.

Syntax error

The operand contains an illegal character, or a semicolon is missing in front of a comment.

Truncation error

The value of the operand exceeds the range of a single byte (8 bits). It must be within the range -128 to 127 or within the range 0 to 255. A value of zero is assumed.

Offset error

The offset at an instruction using indexed addressing is not within the range -128 to 127, or the address at an instruction using program counter relative addressing

(JR and DJNZ) is not within the range $-126 to $+129, or the offset at a JP (IX) or JP (IY) is not zero. An offset of zero is assumed.

Invalid operand

The operand constellation is not valid.

Unknown instruction

The symbol appearing in the opcode field is neither a valid Z-80 operation code nor a valid pseudo operation.

Invalid label

The symbol in the label field contains an illegal character.

Label missing

No symbol appears in the label field at one of the pseudo-ops EQU, SET, and DEFL.

Reserved word

The symbol appearing in the label field is a reserved word (A, B, C, D, E, L, H, M, N, P, R, Z, AF, BC, DE, HL, IX, IY, NC, NZ, PE, PO, SP, HIGH, LOW, MOD, SHL, SHR, EQ, NE, GE, LE, GT, LT, NOT, AND, OR, XOR and $).

Double defined label

The symbol appearing in the label field has already been used as a label.

Illegal backwards ORG

At this point of assembly it is not possible to ORG backwards, as one or more bytes of object code has already been generated.

Too many IF/COND's

The maximum nesting level of IF/COND conditional pseudo-ops is 15.

No prior IF/COND

The conditional pseudo-op in the source statement has no matching IF/COND pseudo-op.

ENDIF/ENDC missing

This error can only occur at the end of a source program. It indicates that one or more IF/COND conditional pseudo-ops are still in effect, although no more source statement lines are present.

Bad symbolfile name

The symbolfile specifier given in the operand is syntactically incorrect. Refer to the PolyDos Users Guide for a description of file specifiers.

No such symbolfile

The symbolfile specified in the operand does not exist.

Symbolfile unreadable

An error occurred when reading a symbolfile. Try load the symbolfile into memory using the LOAD command to determine the type of error.

Symbolfile too big

There is not enough memory to read the specified symbolfile. Split your source file into two or more smaller source files, to free more memory.

Symbol not in symbolfile

There is no symbol of the name you specify in the current symbolfile.

IDNT can only be used once

You are only allowed to have one IDNT statement within your source file.

IDNT missing

An IDNT statement line is missing from your source file. To create an object file it must be present.

PolyDos

DISK BASIC GUIDE

PolyData
microcenter

TABLE OF CONTENTS

Section 1

Introduction to DISK BASIC

PolyDos DISK BASIC is a collection of new commands to the NASCOM 8K ROM BASIC. Features of DISK BASIC include loading and saving program files on disk, sequential and random access data files, printer interfacing, program error trapping, automatic line numbering and renumbering. Programs written for the NASCOM 8K ROM BASIC will run under PolyDos DISK BASIC with no modifications at all.

Section 2

Invoking DISK BASIC

DISK BASIC consists of two disk files, one called BSfh.OV, which is the file handler overlay that handles the BASIC command and DISK BASIC program file execution, and one called BSdr.BR, which contains the DISK BASIC commands. Both these files must be present on the master drive to run DISK BASIC.

2.1 The BASIC command

The BASIC command is used to cold and warmstart the DISK BASIC interpreter. On running DISK BASIC from a BASIC command the following prompt message is output:

```
PolyDos DISK BASIC Version v.v
(C) 1981 Poly-Data microcenter
xxxxx Bytes free
```

where v.v is the version number, and xxxxx is the number of bytes available to the BASIC program and its variables. To warmstart DISK BASIC, add a W option to the BASIC command, thus BASIC;W. If you attempt to warmstart DISK BASIC prior to a coldstart, or if you in the meantime has been executing programs that use the same memory areas as DISK BASIC, you are greeted:

```
I can't warmstart DISK BASIC
```

and returned to the command level. Note that if you used the BASIC command to coldstart DISK BASIC, don't use the Z in NAS-SYS to warmstart it, as Z does not activate the DISK BASIC routines properly.

2.2 Executing BASIC program files

When you execute a BASIC program file, by typing its file specifier as a command, it will be loaded into memory an RUNed automatically. What actually happens is that PolyDos invokes the BASIC file handler BSfh.OV, which loads the DISK BASIC routines file, called BSdr.BR. When control is transferred to BSdr, it loads your program file and RUNs it.

PolyDos DISK BASIC supports two kinds of program files: Memory image files and text format files. A memory image file is an exact copy of the program workspace saved on disk. Memory image files are fast loading and consumes little space, but they cannot be listed or edited by PolyEdit. A text format file on the other hand can be listed and edited just as any other text file, but it is slower to load and consumes more disk space than a memory image file. Normally one uses text format files during the development phase, and memory image files to contain the finished program. You and DISK BASIC can tell memory image files from text format files, by looking at the files load address (the address displayed in the column labelled 'Load' of a disk directory displayed by a DIR;E command). A load address of zero indicates a text format file, other values a memory images file.

You don't have to use the BASIC editor (i.e. NAS-SYS editing facilities) to create text format files. Instead it is suggested that you use PolyEdit. Not only is it a better editor, but it enables you to enter lines of more than 48 characters (up to 72 characters are allowed). On loading a text format file, DISK BASIC ignores all lines that doesn't begin with a line number. Hence, by omitting the line number, you can insert comment lines that will appear in the text file, but are ignored when the program is loaded into memory. Consider the following example of a text format BASIC program file possibly created using PolyEdit:

```
This program will input your name
and spell it backwards
100 input "Hi there, what's your name";n$
110 print "Backwards your name is ";
120 for a=len(n$) to 1 step -1
130   print mid$(n$,a,1);
140 next
150 print: monitor
```

When loaded into memory (e.g. by executing the program or by LOADing it), it will appear like this:

```
100 INPUT "Hi there, what's your name";N$
110 PRINT "Backwards your name is";
120 FOR A=LEN(N$) TO 1 STEP -1
130 PRINT MID$(N$,A,1);
140 NEXT
150 PRINT: MONITOR
```

Note that the comments has been removed, and all lower case letters, outside of string qoutes, has been converted into upper case.

If an error occurs on loading a program (e.g. a line number greater than 65529 or a disk read error), an error message will be displayed, and DISK BASIC enters direct mode. At this point some program lines may be present.

Section 3

DISK BASIC commands

The commands of DISK BASIC are divided into two groups: Direct mode commands, which works only in the direct mode, and global commands, which may be used in program statements as well as in the direct mode. The descriptions use the following notations:

[...] Contains an optional element. If present the element may only be used once.

{...} Contains an optional element, which, if present, may be used any number of times.

ABC The names of the commands are printed in upper case letters. All elements outside of the angle brackets (<>) must be specified as-is. For example the element {,<var>} requires the comma to be specified each time the optional element is selected.

<...> The angle brackets contains a syntactical element which is described in the text.

3.1 Direct mode commands

Direct mode commands are only valid when used in the direct mode, i.e. as a response to the 'Ok' prompt output by BASIC. The commands in this group are:

LOAD	Load a BASIC program file
SAVE	Save program using memory image format
SAVET	Save program using text format
EXEC	Load and RUN a BASIC program file
AUTO	Automatic line numbering
REN	Renumbering
FIND	Locate search string

3.1.1 The LOAD command

```
LOAD <filename>
```

The LOAD command is used to load BASIC program files from disk into memory. <filename> must be a string expression giving the file specifier of the file to be loaded, e.g. "Pingpong:1". If no drive number is specified, the file is loaded from the master drive. The program in memory is erased before the new program is loaded.

3.1.2 The SAVE command

```
SAVE <filename>
```

SAVE will save the program currently in memory as a disk file using memory image format. <filename> must be a string expression giving the file specifier of the program file to be created. If no extension is specified DISK BASIC defaults to .BS, and if no drive number is specified the file is created on the master drive. If any files exist of the name you specify they will be deleted.

3.1.3 The SAVET command

```
SAVET <filename>
```

SAVET works the same as SAVE, except that the file created is a text format file.

3.1.4 The EXEC command

```
EXEC <filename>
```

EXEC works the same as LOAD, except that the program being loaded is executed automatically.

3.1.5 The AUTO command

```
AUTO [<start>[,<inc>]]
```

AUTO provides automatically output line numbers. <start> is an expression giving the first line number, and <inc> is an expression giving the line number increment. If <inc> is omitted, 10 is assumed. If <start> is omitted, 100 is assumed. When a line is entered, AUTO looks at its line number and adds the increment to form the number to be output at the next line. This means that you can backspace over the line number output by AUTO, and enter a new one, from which AUTO will count at the next line. Consider the following example:

```
AUTO 200,5

200 FOR A=1 TO 10
205 GOSUB 500
210 NEXT: END
500 PRINT D(A)*C(A),D(A)/C(A)
505 RETURN
```

In line 500 AUTO actually output a line number of 215, which was changed to 500 by the typist. As you see AUTO continued counting from 500 instead of 215, the next line number being 505.

To deactivate AUTO enter a line that doesn't start with a line number, e.g. a blank line produced by pressing <ESC> followed by <ENTER>.

3.1.6 The REN command

```
REN [<start>[,<inc>]]
```

REN renumbers the program currently in memory to start with the line number given by the expression <start> with a line number increment given by the expression <inc>. If <inc> is omitted 10 is assumed. If <start> is omitted 100 is assumed. Line number references at the following statements will be renumbered:

GOTO	GOSUB	IF..THEN
ON..GOTO	ON..GOSUB	RESTORE
LIST	RUN	SETERR

If a statement of one of the above mentioned types contains a reference to an undefined line number the reference will be renumbered to 65529 which is the highest line number possible. Such illegal references can be loacted later using a FIND "65529" command.

3.1.7 The FIND command

```
FIND <string>
```

FIND will list all lines containing the search string given by the string expression <string>. Each time the number of lines given by the most recent LINES command has been output, FIND stops and blinks the cursor. Press <ESC> to terminate FIND or any other key to continue. Note that FIND will find all occurrances of the search string including parts of line numbers and reserved words. Hence, FIND "100" will list all lines containing the string "100" (excluding the qoutes), as well as line 100. If your string is to include a double quote you must use a CHR$(34) function call.

3.2 Global commands

Global commands can be used as direct mode commands as well as in program statements. As you will note, all global commands start with the keyword SET. This may seem a little odd but it is the only way of implementing extra commands that can be used as program statements. The global commands are divided into four groups:

Data file I/O commands
Program file commands
Printer control commands
Various commands

3.2.1 Data file I/O commands

The data file I/O commands are probably the most important addition offered by DISK BASIC, since they allow you to maintain data files on disks. PolyDos DISK BASIC supports two types of data files:

Sequential files

Sequential files use the same internal format as text files. Each "record" is a string of ASCII characters ended by a carriage return. There is no fixed length on lines in a sequential file as opposed to random access files described below. This leads to the following restrictions on sequential file access: Reading a sequential file can only be done from the beginning of the file moving towards the end one line at a time and writing to a sequential file can only be done by adding lines to the end of the file. Only strings can be read from and written to sequential files. In case of numeric variables you will have to use the STR$(N) function to write them and the VAL(S$) function to read them.

Random access files

A random access file consists of a fixed number of records, each record containing a fixed number of fields. A field can be one of three types: An integer, i.e. a whole number between -32768 and 32767, a real, i.e. a floating point number, and a string of a fixed maximum length (however not more than 255 characters). As the length of each record is known, DISK BASIC can calculate the position of specific records in the disk file. Hence, you can read and write anywhere in the file as you please.

Data file input/output is done through file channels, also known as units. When you open a file for processing you assign to it a unit. Every time you want to access the file you reference the number of the unit assigned to it instead of the file name. DISK BASIC supports 4 units numbered from 0 to 3. Thus, you can access four files from your program at the same time. When you close a file the unit you assigned to it is released and ready to be assigned to another file.

3.2.1.1 The SETNEW command

```
SETNEW(<unit>),<filename>[,<type>[,<format>,<nrec>]]
```

The SETNEW command is used to open a data file and assign it to a unit. The unit must be in its closed state or otherwise an error occurs. The elements in the format descriptor are explained below.

<unit> An expression representing the unit number to be assigned to the data file (0 to 3).

<filename> A string expression giving the file specifier of the file to be assigned to the unit. If the extension is omitted, .DT is assumed. If the drive number is omitted, the master drive is assumed.

<type> This parameter is optional. Its presence specifies that a new file is to be created and its value specifies the access type of the file. It may be S for sequential output or R for random access.

<format> This parameter is only used in the case of a new random access file to be opened. It is a string expression representing the internal format of each record in the random access file.

<nrec> This parameter is only used in the case of a new

random access file to be opened. It is an expression representing the total number of records in the file.

As you see from the above description, to open an existing file you need only specify the unit number and the file name thus leaving the <type>, <format>, and <nrec> fields unspecified. DISK BASIC will itself figure out the type of the file. This is done by looking at the file load and execute addresses. If the load address is zero the file is considered a sequential file which will be opened for input. If the coldstart address is non-zero the file is considered a random access file which will be opened for both input and output. The load address specifies the number of records in the file and the execute address specifies the record length in bytes.

If the <type> field is given DISK BASIC assumes that you want to open a new file. If the file type is S a new sequential file will be created and opened for output. The <format> field and the <nrec> field should not be specified when creating sequential files. If the file type is R a new random access file will be created. In this case the <format> field and the <nrec> field must be specified. The <nrec> field is an expression giving the number of records in the file. The maximum number of records is 32767. The <format> field is a string expression giving the format of each record, i.e. the number of fields within the record, and the type of each field. The character I is used to indicate an integer, R is used to indicate a real, and S is used to indicate a string. In the case of a string, the ASCII value of the following character defines the maximum length of the string. Below is shown some examples of format descriptor strings:

"I"
Indicates that each record contains one field which will store integer values.

"IIR"
Indicates that each record contains three fields, the first and the second one being integers, and the third one being a real.

"IS"+CHR$(36)+"S"+CHR$(48)
Indicates that each record contains three fields, the first one being an integer, the second one being a string of maximum length 32, and the third one being a string of maximum length 48. The format descriptor could also be written "IS$S0", however this is not very informative.

The format descriptor string may not exceed 45 characters in length. On creating a new random access file, all fields within each record will be cleared, i.e. integers and reals assume the value 0, and strings become empty (length 0). Records in a random access file are numbered from 0 to NR(<unit>)-1, where NR(<unit>) is the number of records in the file. It is, however, possible to position the record pointer at record number NR(<unit>), but any attemps to read or write at this position will produce an error.

If you open an existing sequential file it is only possible to read form it, and if you create a new sequential file it is only possible to write to it. To add lines to an already existing sequential file, you will have to open a unit to the old file and a unit to a new file, and copy all elements from the old file to the new file, before adding extra lines. It is not possible to have more than one sequential output file opened on each drive.

Some examples of SETNEW commands:

```
SETNEW(0),"TEXT.TX",S
```

Create a new file called TEXT.TX on the master drive, and assign to it unit number 0. As TEXT.TX is a new file, it is only possible to write to it.

```
SETNEW(2),"REPORT"
```

Open the file called REPORT.DT on drive 1, and assign to it unit number 2. As REPORT.DT is an existing file, the file itself defines the type of I/O it will permit (sequential or random access) by the value of its load address. Thus, your program must "know" what type of I/O it is allow to do.

```
SETNEW(1),"DATA",R,"IS"+CHR$(20)+"R",1000
```

Create a new random access file called DATA.DT on the master drive, and assign to it unit number 1. DATA.DT will contain 1000 records (numbered from 0 to 999), each record containing an integer, a string of maximum length 20, and a real, in that order.

3.2.1.2 The FM$ file variable

The FM$ dimension is a reserved variable. Each time a random access file is assigned to a unit, FM$(<unit>) is assigned the format descriptor of that file. This is especially useful when accessing already existing random access files of an unknown internal format. FM$(<unit>) is treated by BASIC as any other string variable. Thus, it is possible for you to assign values to it, however this is strongly discouraged. When a sequential file is assigned to a unit, FM$(<unit>) is undefined.

3.2.1.3 The NR file variable

The NR dimension is a reserved variable. Each time a file is assigned to a unit, the number of records in that file, or zero if the file is a sequential file, is assigned to NR(<unit>). This is especially useful for determining the type of an existing file. NR(<unit>) is treated by BASIC as any other variable. Thus it is possible for you to assign values to it, however this is strongly discouraged.

3.2.1.4 The EOF file variable

The EOF dimension (actually only EO need be specified) is a reserved variable. Each time a line is read from a sequential file, or when a sequential file is opened for input, EOF(<unit>) is assigned a boolean value, reflecting the status of the unit. If EOF(<unit>) is false, the file contains more lines. If EOF(<unit>) is true, the file pointer is at the end of the file. Trying to read from a sequential file, when EOF(<unit>) is true, will result in an error. For sequential output files and for random access files, EOF(<unit>) is undefined.

3.2.1.5 The SETINP command

```
SETINP(<unit>),<var>{,<var>}
```

The SETINP command is used to input data from a unit. The unit must be in its opened state, or otherwise an error occurs. The <var> field(s) denote variable identifiers.

On reading from a sequential file, the variable(s) specified must be of type string (i.e. $ variables). When a variable is read, all characters up to, but not including, the next carriage return in the file will be returned (assuming that this is not more than 255 characters), and the carriage return character will be skipped. If the line contains more than 255 characters, only the first 255 characters will be returned. If the line read was the last line in the file, the end-of-file variable of the unit involved will be set to true (-1). Note that it is not possible to read from a sequential file which was opened as a new file.

On reading from a random access file, the type of the variable to be read must match that of the field pointed to by the internal file pointer: Integer fields and real fields must be read into numeric variables, and string fields into string variables. When a variable has been read, the pointer advances to the next field in the record. If you specify more variables in the statement line than there are fields in the record, an error message will be produced. If you specify less variables in the statement line, than there are fields in the record, the internal file pointer will be left to point at the next field. This allows you to split up reading of records into more SETINP statements, but it can also be a source of confusion, if administrated inproperly, as it is possible to leave the file pointer in the middle of a record.

3.2.1.6 The SETOUT command

```
SETOUT(<unit>),<expr>{,<expr>}
```

The SETOUT command is used to write to a unit. The unit specified must be in its opened state, or otherwise an error occurs. The <expr> field(s) denote expressions.

On writing to a sequential file, the expression(s) specified must be of type string. Each time a string value is written to the file, a carriage return will be output automatically. Note that it is only possible to write to a new sequential file, i.e. a file opened with the S specification.

On writing to a random access file, the type of the expression to be written must match that of the field pointed to by the internal file pointer: Numeric expressions must be written to integer or real fields, and string expressions into string fields. If the length of a string expression is greater than the maximum length of the string field it is to be written to, only the leftmost characters will be transferred. When a value has been written, the pointer advances to the next field in the record. If you specify more variables in the statement line than there are fields in the record, an error message will be produced. If you specify less variables in the statement line than there are fields in the record, the internal file pointer will be left to point at the next field. This allows you to split up writing of records into more SETOUT statements, but it can also be a source of confusion, if administrated inproperly, as it is possible to leave the file pointer in the middle of a record.

3.2.1.7 The SETPOS command

```
SETPOS(<unit>),<recnbr>
```

The SETPOS command is used to move the internal file pointer of the unit specified to the record given by the expression <recnbr>. The file pointer will be positioned at the first field of the record. <recnbr> should be within the range 0 to NR(<unit>)-1, where NR(<unit>) is the number of records in the file. If <recnbr> is greater than NR(<unit>), an error will be produced. If <recnbr> equals NR(<unit>), the file pointer will be positioned at the end of the file. Any attempts to read or write in this position will result in an error. SETPOS on sequential files will produce an error.

3.2.1.8 The SETCLS command

```
SETCLS(<unit>)
```

The SETCLS command is used to close (release) the unit specified. If data has been written to the sector currently contained in the internal file buffer, the buffer will be written to the disk. If the file assigned to the unit is a sequential output file (i.e. a new file), it will be entered into the disk directory, and all files with the same name and extension will be deleted. Thus, the creation of a new sequential file is not completed before a SETCLS(<unit>) statement is executed, as opposed to the creation of a new random access file, which is entered into the directory when the SETNEW command is executed. However this DOES NOT mean that the SETCLS(<unit>) statement can be omitted when working on random access files.

3.2.2 Program file commands

The program file commands are SETLOAD, which is used to load machine code subroutine files into memory, and SETCHAIN, which will load and execute any BASIC program file without clearing the variable workspace.

3.2.2.1 The SETLOAD command

SETLOAD <filename>

The SETLOAD command is used to load a file into memory under program control. <filename> is a string expression giving the file specifier of the file to be loaded. The default drive is the master drive.

The files loaded using SETLOAD are typically machine code files, but any file type of files can be handled. The file will be loaded into memory starting at its load address. Note that no checking is done to assure that system memory areas are not overwritten. The SETLOAD should not be used to load BASIC program files. Instead use the LOAD or the EXEC command from direct mode, or the SETCHAIN command from programs.

3.2.2.2 The SETCHAIN command

SETCHAIN <filename>

The SETCHAIN command will load an execute a BASIC program file, without clearing the variable workspace. <filename> is a string expression giving the file specifier of the file to be CHAINed. The default drive is the master drive.

3.2.3 Printer control commands

The printer control commands are used to control output to the printer. Two commands are available:

SETPRON	Printer on
SETPROFF	Printer off

SETPRON will turn on printer output, which means that subsequent output will be directed to the printer. SETPROFF turns off printer output. Note that the printer output function does not output to the printer the characters typed from the keyboard in the direct mode and as response to INPUT statements. These inputs will be echoed to the VDU in the usual way.

3.2.4 Various commands

In addition to the commands described in the preceding section, PolyDos DISK BASIC supports three commands. These are:

SETERR	Trap program errors
SETREAD	Input variables with editing
SETCLEAR	Define string space and memory size

3.2.4.1 The SETERR command

SETERR[<lineno>]

The SETERR command causes program control to be transferred to the line specified in case of errors. <lineno> must be the line number of an existing program line. The error service routine can obtain information about the error condition in the system variables EL (error line) and EN (error number). EL contains the line number of the error, and EN contains the error number (see appendix A). When the error service routine is invoked, the SETERR function is turned off to avoid a system "hang-up", should the error service routine itself contain an error. If used without a line number argument, the SETERR command will turn off the error trapping function. If the line given by <lineno> does not exist, and an error has occurred, DISK BASIC will output the error message "?Undefined line in xxxxx", where xxxxx is the line number of the SETERR command. In this case the initial error line and error number can be accessed through EL and EN.

3.2.4.2 The SETREAD command

SETREAD <strvar>

The SETREAD command will display the contents of the string variable <strvar> and allow you to edit it. The following editing keys are available (<LE> denotes the left arrow, and <RI> denotes the right arrow):

<LE>	Move the cursor left
<RI>	Move the cursor right
SHIFT/<LE>	Delete character
SHIFT/<RI>	Insert character
<CS>	Clear input field
<BS>	Backspace one character
<ENTER>	Entry complete

The length of the input field is determined by the length of the string when the SETREAD command was invoked. Only characters within the input field will be affected by the editing commands. For instance, if you SETREAD a string of length 10, inserting a character, when the cursor is in the first position of the input field, will only move left the nine characters following the cursor. The length of the string returned by SETREAD is always the same as the length upon entry. The carriage return (<ENTER>) ending the entry will not be echoed.

3.2.4.3 The SETCLEAR command

SETCLEAR <strsp>[,<memtop>]

The SETCLEAR command will erase all variables and set the size of the string space to the number of bytes given by the expression <strsp>. If the <memtop> field is specified, it should be an expression giving the address of the highest memory address DISK BASIC is allowed to access. Values greater than 32767 must be specified as negative numbers, computed from zzzzz=xxxxx-65536, where xxxxx is the desired value and zzzzz is the value to be used. The only difference between the SETCLEAR command and the CLEAR command is that SETCLEAR allows for values greater than 32767 to be specified as described above.

Appendix A

DISK BASIC memory map

```
FFFF +------------------------------------+
     |         NASCOM 8K ROM BASIC        |
E000 +------------------------------------+
     |               Unused               |
D800 +------------------------------------+
     |             PolyDos ROM            |
D000 +------------------------------------+
     |          PolyDos workspace         |
C000 +------------------------------------+
     |         DISK BASIC routines        |
B000 +------------------------------------+
     |       DISK BASIC file buffers      |
A900 +------------------------------------+
     |              User RAM              |
MTOP +------------------------------------+
     |                                    |
     |          Program/variables         |
     |                                    |
10FA +------------------------------------+
     |         ROM BASIC workspace        |
1000 +------------------------------------+
```

When DISK BASIC is coldstarted, MTOP is set to A900H, thus reserving all unused RAM for BASIC programs and variables. The value of MTOP can be lowered using the SETCLEAR command (see section 3.2.5.3). The overlay area is used by the error message writer overlay Emsg.OV. Emsg is loaded by DISK BASIC when it is invoked. Thus, you may remove the system disk once DISK BASIC is up and running.

Appendix B

Useful hints

If you during program execution want to insert a new disk in one of the drives, you must first make absolutely sure that all units relating to that drive are closed, and next POKE -16383,255, to inform PolyDos that the directory has to be read into memory before any disk transactions can take place.

If you want to extract parameters from the command line invoking your program, use this short routine to copy the contents of the command line into a string variable (in this case CL$):

```
500 CL$="": FOR P=-16357 TO -16311
510 CL$=CL$+CHR$(PEEK(P)): NEXT: RETURN
```

Note that this method assumes that your program was executed from the command level in PolyDos, and not from an EXEC command or a SETCHAIN statement.

The routine shown below will test to see if the command file mode is active, and, if so, abort it:

```
600 IF PEEK(-16373)=0 THEN RETURN
610 PRINT "*** Command file abort ***"
620 POKE -16373,0: RETURN
```

Very often you will need a routine to input one character from the keyboard without echoing it to the screen. Start your program with POKE 4158,223, and each time you want to read an input character execute CH=INP(123), which will blink the cursor until a key is pressed, and return its ASCII value. To restore normal INP operation, execute POKE 4158,219. Don't use other "port" values than 123. It will cause strange things to happen, and may very well crash the system.

Appendix C

Error messages

This appendix lists all error messages and their associated error numbers. When you are using the SETERR function to trap program errors, the error number can be accessed through the EN variable.

Errors reported by PolyDos:

For a full explanation of these errors please refer to the PolyDos Users Guide.

16	Illegal character in filename
17	Filename too long
18	Bad drive identifier
19	Filename missing
32	Drive not ready
33	Disk write protected
34	Write fault
35	Record not found
36	Checksum error
37	Lost data error
38	Bad disk address
39	No disk or wrong format
40	Illegal drive number
41	Disk is full
48	I can't find that file
49	That file already exists
50	Directory is full
51	I can't do that to a locked file

Errors reported by ROM BASIC:

For an explanation of these errors please refer to the NASCOM 8K ROM BASIC manual. The two-letter error codes normally returned by BASIC are listed enclosed in parentheses:

128	NEXT without FOR (NF)
129	Syntax error (SN)
130	RETURN without GOSUB (RG)
131	Out of data (OD)
132	Function call error (FC)
133	Overflow (OV)
134	Out of memory (OM)
135	Undefined line (UL)
136	Bad subscript (BS)
137	Double defined dimension (DD)
138	Division by zero (/0)
139	Illegal direct (ID)
140	Type mismatch (TM)
141	Out of stringspace (OS)
142	String too long (LS)
143	String expression too complex (ST)
144	I can't continue (CN)
145	Undefined function (UF)
146	Missing operand (MO)

Errors reported by DISK BASIC:

147 Illegal unit number. The unit number is not within the range 0 to 3.

148 Unit already open. An attempt was made to open a unit which has not yet been closed.

149 Unit not open. An attempt was made to access a unit which has not yet been opened.

150 Invalid format descriptor. The format descriptor is empty or more than 45 characters long or it contains invalid field descriptors (i.e. not "I", "R", or "S"+CHR$(x)).

151 End of file. An attempt was made to read from a sequential file which has its end-of-file flag set, or from an unexisting record in a random access file.

152 Invalid record number. The record number specified is out of range.

154 Null string. An empty string is not allowed here.

155 I can't open that unit. You are trying to open a new sequential file on a drive which already has a sequential output file on it.

156 Unit not open for input. You are not allowed to read from a sequential output file.

157 Unit not open for output. You are not allowed to write to a sequential input file.

158 I can't position that unit. You are not allowed to use the SETPOS command on sequential files.