GEOS

Tricks & Tips

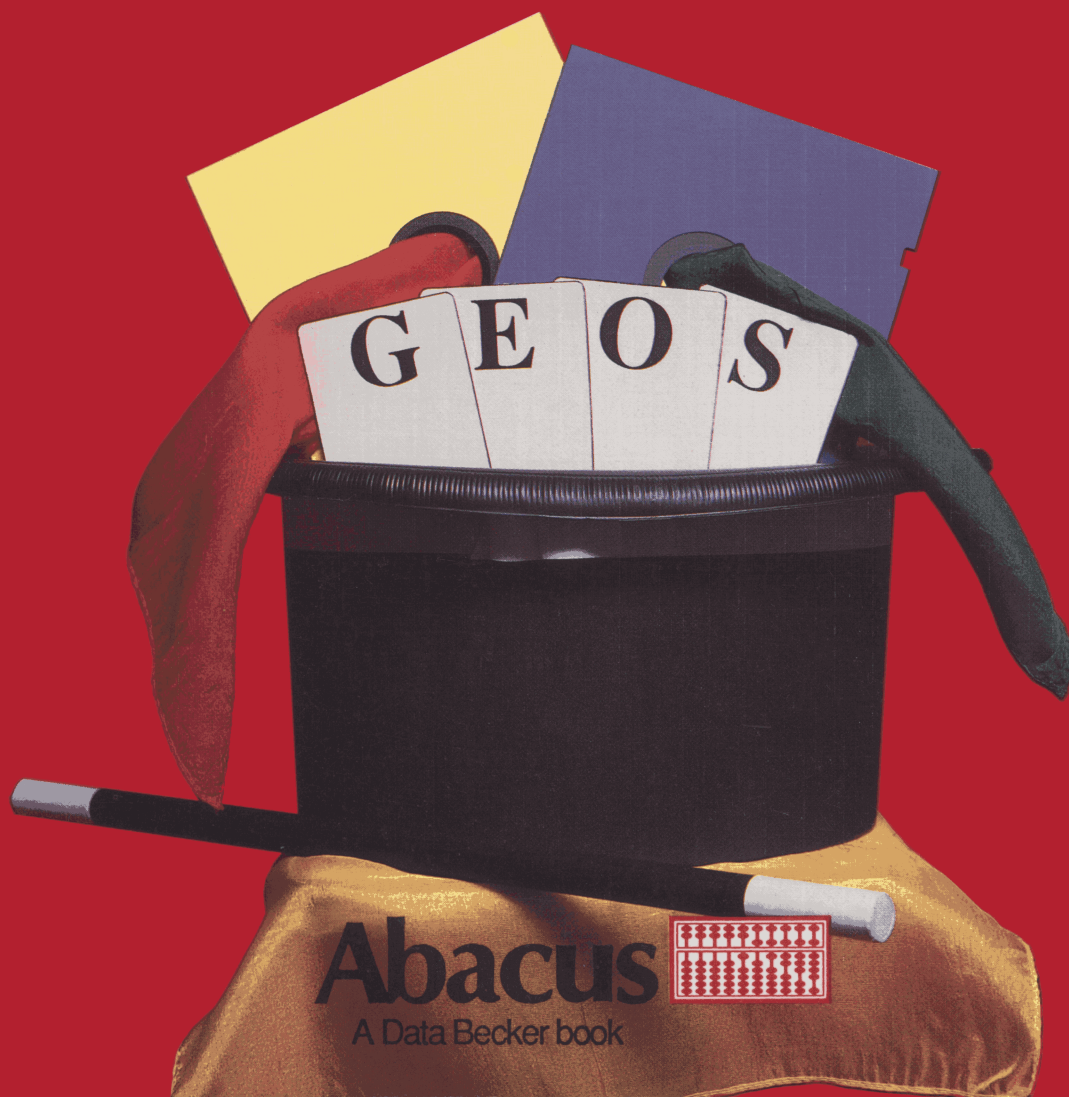**For all COMMODORE 64 owners who use GEOS**
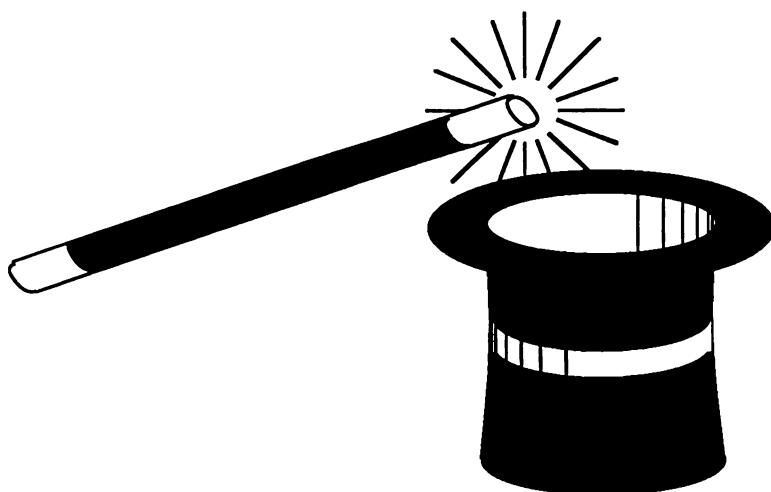**Includes Version 1.3**

Abacus
A Data Becker book

GEOS

Tricks & Tips

By M. Kerkloh and R. Tornsdorf

A Data Becker book published by

Abacus

Commodore 64, Commodore 64C, Commodore 128, Commodore 1541 and
Commodore 1571 are trademarks or registered trademarks of Commodore
Electronics, Ltd.

GEOS, deskTop, geoPaint and geoWrite are Copyright © 1985, Berkeley
Softworks.

Preface

GEOS™, the new Commodore user interface, has been available on the US market for several months now. Since we work with GEOS daily and are very familiar with the internal workings of the system, we naturally read all we could find about GEOS in the various magazines with great interest. We often sat at our computer and asked ourselves whether we should laugh or cry.

The information printed in the magazines as "absolute know-how" was often nothing more than superficial repetitions of passages from the manual, or even half-truths or less. After a while we had our fill of this sort of reporting, and we decided to write a book of our own that on one hand cleared up the half-truths, and on the other hand presented the many tips and tricks that we had picked up in our work with GEOS.

When we had compiled a list of the possible tips and tricks, we ourselves were surprised how extensive and different the various hints were. They range from predefined tabs and margins in geoWrite to a complete machine language monitor which can be used like any other accessory under GEOS and can also store programs as accessories.

The high point is a program which we humbly call the FONT Editor. Not only does it let you customize characters in all of the existing fonts, it also allows you to create your own type styles from 0 to 63 points in size. The FONT Editor uses the GEOS environment with pull-down menus and is a real accessory.

In addition there is also general information that can simplify your work with GEOS but also will teach you a good deal about how it works. For machine language programmers we have included the GEOS jump table, including descriptions of the routines and what you have to pay attention to. Over 150 routines are available to you from GEOS.

Since the monitor and FONT Editor alone are large programs and therefore difficult to type in, there is also an optional program diskette available for this book.

But now we finally come to the actual start of our tips and tricks and wish you as much fun as you read and try them out as we had in discovering them.

The authors.

Table of Contents

Chapter 1

Tricks and Tips for the User Interface

1.1 Load and go

What good are the greatest tips when you can't even load GEOS properly? There are several things which can stop you right at the beginning from working with GEOS:

GEOS has an excellent copy-protection scheme. It is checked at the moment the message "BOOTING GEOS..." appears and the disk makes two short, somewhat threatening noises. The shorter the pause between these two noises, the faster GEOS has discovered the copy protection. If it is not found within a certain time, GEOS will jump to RESET after some self-destroying actions (the copy-protection code in memory).

After you have worked with GEOS for a while you will be able to tell from these two noises and the gap between them whether the BOOT procedure (load and go) has worked or not (about a half a second is the maximum). What can you do if GEOS will not accept the copy protection on the original diskette?

Tip 1: The BOOT pause

If you experience an unsuccessful attempt, don't boot GEOS again right away. Wait a moment until the disk drive stops running. Often this is enough to start a successful BOOT procedure.

Tip 2: Disk drive alignment with the "bump"

If GEOS still won't boot, the disk drive head may be out of alignment. A "bump" may help the problem. This is the head hitting against a stop in the disk drive. This "bump" is performed before every load in order to position the head precisely. You can also create this bump with a small program:

(CAUTION: You should not start this "bump" with a very warm drive. If the drive has been on for several hours and you have done a lot of loading and saving recently, you should turn the 1541 off for about 10 minutes or else the head may go even farther out of alignment. If the drive was just turned on, you don't have anything to worry about.)

```
10 REM "Bump"
20 REM "This program moves the head to the stop"
30 REM "and aligns it for correct loading."
40 OPEN 15,8,15,"I":REM"Open channel 15 as command
channel"
50 PRINT#15, "M-W"CHR$(0)CHR$(0)CHR$(0)CHR$(192)
60 CLOSE 15
```

Save this program and then start it with RUN. You will hear the head bump against the stop. Now GEOS should be able to find the copy protection properly when booting.

Tip 3: Turning the printer off

One last tip which may decide whether or not you can begin to work with GEOS at all. If GEOS should stop shortly before the end of the loading process, when the menu bar was already displayed at the top of the screen, simply turn off your printer. GEOS in general has some problems with printers. For example, our Epson FX-80 would not run with the printer driver intended for it, but only as a Commodore-compatible printer.

If you have a printer connected, the best thing to do is to turn in on just before you are about to print and then turn it off again afterward. If your printer has a large buffer, print from the application that created the file rather than from the deskTop. When you print from the deskTop and GEOS is actually done (because the rest of the document is in the printer buffer), GEOS will generate a RESET. It will not be completed, however, because the printer is still on. Then despite the buffer in your printer, you have to wait until the printer is done printing and you can turn it off.

By the way, don't be surprised if an extra disk icon with a question mark appears at the right side of the screen after the printer is turned off. GEOS appears to recognize the printer on the serial bus as another disk drive and therefore displays the extra icon. Since this "pseudo disk drive" didn't answer properly, the question mark is displayed.

If this question mark bothers you, click the option RESET under special (with the printer off) and it will disappear.

1.2 deskTop

GEOS gives you a very comfortable and convenient way to use your C64. The deskTop is provided so that all of the everyday procedures like copying, deleting, and renaming files or copying, renaming, and formatting disks, and so on can be easily accomplished. The following tips can be used to get even more power out of the deskTop:

Tip 4: Selecting different names for the diskettes

Give all of your diskettes different names. GEOS recognizes disks by the names and can get confused if you try to copy something between two disks with the same name. Suddenly icons disappear from the border or GEOS will terminate the copy process.

Tip 5: Use the original disk only for booting

Take the original disk out of the disk drive immediately after booting and put it in a safe place. Not only is pouring coffee over it or deleting an important file disastrous. It will no longer be possible to boot from the disk if you place GEOS, GEOS BOOT, or GEOS KERNAL on the border and then exit GEOS!

Don't copy *single files* from the original disk. Imagine you just put something on the border and for some reason the power fails. You will then have to reboot, but some necessary file may not be found (because it is on the border and therefore outside the normal directory).

Tip 6: Always click OPEN after changing disks

Don't forget to select open after changing disks. In the least harmful situation the deskTop will request the disk which you just removed. More unpleasant things can happen, however.

Tip 7: Everything you need for working with an application must be on one disk

Generally you can insert another diskette only when using the deskTop (and initializing the disk with open). If you have a lot of time and a diskette which doesn't contain anything you want to keep (it must be formatted), do the following:

Create a dummy document with geoPaint. Call it simply "Dummy" and draw a couple of lines. Move the geoPaint window to several different places on the page so that geoPaint has to save more than once. At the end set the geoPaint window back to the top.

Take the disk out of the drive and insert the disk which doesn't contain anything you want to keep and click the SCROLL arrow.

CAUTION: The contents of this disk may be quite unusable after this.

Move the geoPaint window with the SCROLL pointer to the right and then down.

The results are rather interesting: either you will see surrealistic drawings, or the pointer arrow will be garbage.The reason for this is that geoPaint is not all resident in memory at one time and must load certain sections for certain functions (such as scrolling). If you remove the disk first, something other than this program module will be loaded instead, and GEOS may crash.

Therefore, copy everything you will need, applications and accessories, onto one disk so that you won't have to change disks.

Tip 8: VALIDATE and SCRATCH from GEOS only

GEOS and the Commodore DOS (the disk operating system which you normally use from BASIC) are *somewhat* compatible. This allows you to use GEOS with all of your old files created without GEOS and store BASIC programs on disks with GEOS files.

The important thing is the little word "*somewhat.*" We advise you not to save onto GEOS disks from BASIC. Sometime you will delete a file accidentally or validate the disk. This has devastating consequences on a GEOS diskette. The reason for this is quite simple: GEOS stores more information on a disk than Commodore DOS (Disk Operating System). For example, an extra sector is allocated for the border and an INFO sector is allocated for every GEOS file. Such sectors will not be recognized by the normal Commodore DOS however, and will therefore be released with the normal validate command (OPEN 15,8,15,"V0;":CLOSE 15). When you then save something from BASIC, these sectors will also be used and this will make the GEOS disk unusable.

Never use the commands scratch and validate from BASIC on GEOS diskettes. If this happens accidentally, do not store anything more on this disk before you have executed a validate on it from GEOS. The GEOS command

naturally recognizes the additional used sectors and allocates them in the BAM (Block Allocation Map), protecting them from being overwritten.

Tip 9: Don't use system names for your own files

Never give a system name to a document created in GEOS. By this we mean the names of all the files stored on the original disk and also the names of the data files created by GEOS programs: Notes and Preferences. You risk either a system crash (GEOS hangs up) or total loss of your data with such a name.

For instance, if you name a document "Notes", the accessory note pad will stop reading on the first control character (geoWrite marks the various types and tabs with these control characters). In addition, your geoWrite "Notes" file will probably be destroyed in the process. If you call a graphic image "Preferences", saving from the preference mgr will destroy your graphic image.

Tip 10: VALIDATE makes room

Use the GEOS command validate when you get the chance (but only from GEOS). Sometimes you can obtain a few extra kilobytes on your disk this way. It seems that GEOS doesn't always release all of the sectors when deleting a file. If you have worked with a disk for a long time and you only have a little bit of space, and you want to take a small break, click validate. Depending on the number and size of the present files this may take several minutes, but GEOS may show some extra kilobytes on the disk after it's done.

Tip 11: Never turn the disk drive off

Never turn the disk drive off while working with GEOS. (There is only one exception: When you want to connect a second drive, you will be explicitly told to turn it off). GEOS puts part of the Speeder (disk drive accelerator) in the memory of the disk drive. When you turn the drive off, this Speeder will be lost. This will not just cause all subsequent disk operations to run slowly, it will simply cause GEOS to display an error message at the next disk access and you will no longer be able to access the disk! If you try to add a drive the GEOS system will crash.

Tip 12: Preferences can be set up for each disk

You can make your own settings for colors, mouse speed, etc. by saving from the `preference mgr`. The important thing is that these values can be set differently for every disk. These values will be read and set when a disk is opened with the GEOS `open` command. This way you can set the maximum mouse speed on a `deskTop` disk, while selecting a slower speed for a geoPaint disk for precise work.

You don't have to worry that the date you set will be changed by opening a disk with other `Preferences`. All settings will be taken from `Preferences` except the date and time, which will be unchanged.

By the way, if you want to experiment with a "modified pointer," that is, you don't like the shape of the arrow, you should note when making your new shape that the upper left corner is the "click point." Even if your pointer consists only of one point in the lower right corner, the upper left corner still counts when clicking.

Tip 13: Automatically selecting the right printer

Since GEOS has several printer drivers stored on the original disk, you have to choose the appropriate one before printing. If this is too much trouble for you or you forget it often, there are two ways around it:

- Remove all printer drivers from a disk except for the one for your printer. This will then become the default driver, so you won't have to select it every time.

- If, for some reason, you want to have several printer drivers on your working disk, just make the first one the one you want to use most often. GEOS uses the first driver it finds on the disk as the default.

NOTE: These two solutions work only on the disk from which GEOS is booted.

Tip 14: Just the right input device

The same thing applies to the input drivers too, by the way. The default pointing device is the joystick. Berkeley Softworks has other input devices available now

(mouse, Koala Pad, light pen). If you have multiple input drivers on the disk, just make the desired one first. (applies only to the disk from which GEOS is booted).

Tip 15: Proper order saves paging

The order of the files (icons) on the disk is important for more than just the selection of the printer and input drivers. You can also save a lot of effort and paging if you organize your disk well. For example, if you have a disk with a number of geoPaint pictures, put the pictures you're currently working on, on the first page. Then you don't have to do any paging to find the file you want.
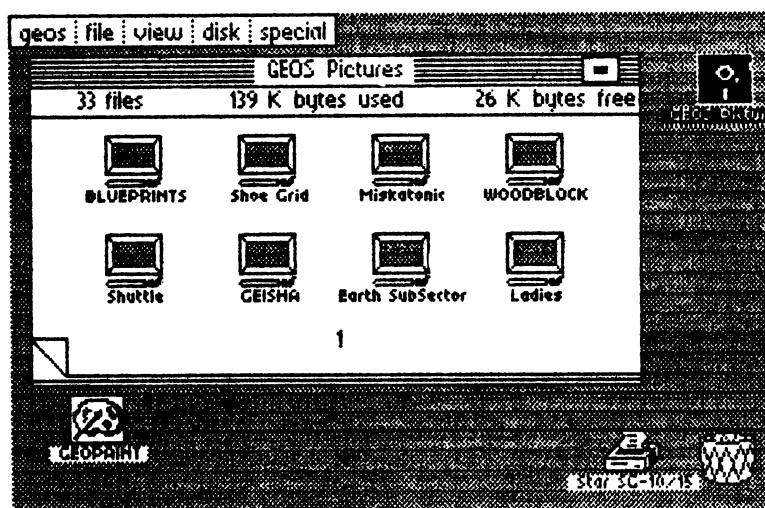
Figure 1: Page one screen

Tip 16: Putting icons on another page

To move an icon to a given page (such as the first), it must be placed on the border. Then select the desired page and move the icon somewhere on this page. If there is space there, GEOS will put it there. It is not possible to double-click the icon, then select the desired page and move it. When you start paging the icon will automatically be put back in its old place, and you won't have accomplished anything.

Tip 17: Put system files on the last pages

The best thing to do is to put the FONTS, INPUT DRIVERS, PRINTERS, NOTES, and PREFERENCES on the last pages so that these files cannot be accessed directly. This way you can't double-click notes to load the note pad, for instance. The programs which require these system files are also found on the last pages.
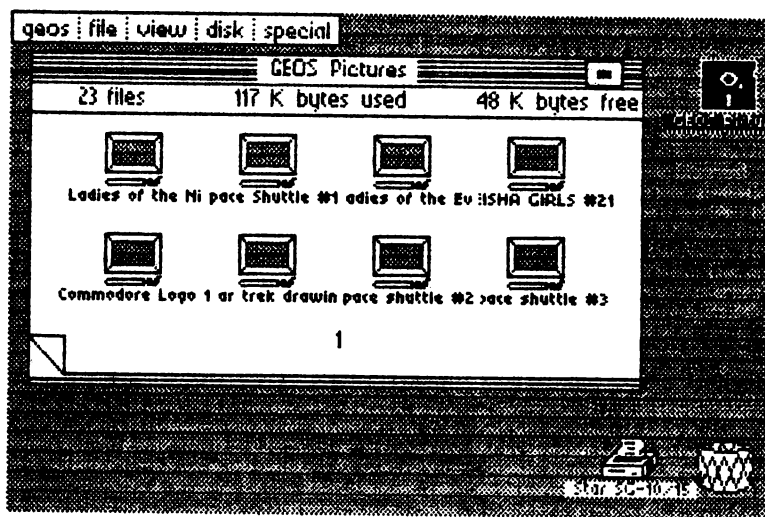
Figure 2: Filenames too long!

Tip 18: Don't use too long a name

Don't give your files names which are too long. These will be centered under the icon by the deskTop. If you use more than about thirteen characters (it depends on the width of the characters since the font uses proportional spacing), they will not all be readable. Things like the date, time, and comments should be kept in the INFO sector.

Tip 19: Date and time help with your work

Use the ability to store the date, time, and comments with each file. The first two pieces of information are stored with each file by GEOS itself. Set the time and date using the preference mgr when you start working with GEOS.

You can then display the latest version of your graphic picture by selecting by date under view. The last-created file will always be on top.

Tip 20: Comments in the INFO window

You can save yourself a lot of confusion if you comment documents. Instead of calling your fourth attempt at wishing Peter a merry Christmas on 12/18 "PETER12/18Xmas4", just call the letter "Peter4" and the date will be set in the INFO sector, to which you can add the comment "Christmas greetings."

Tip 21: Creating new pages in the directory

If for some reason you need another page for icons, you can't just turn to it. For example, if you have five pages and you need a sixth, the deskTop will go back to the first page if you try to go beyond the fifth. You can create another page by creating files until deskTop needs a new page. The simplest way is to follow our example:

Copy as short a file as possible (1K) to the last page. Make sure that in each case the write-protect is not active on this file. If you don't know how this goes, see Tip 16 again. Duplicate this file until the new file no longer fits on the last page. deskTop will then automatically create a new page, even if there was still room on the previous pages.

When duplicating (file, duplicate), you must select new names. Just add a number to the end of the filename. After the new page has been created, you can remove the superfluous files from the previous page (and hopefully you used a file which wasn't locked, or you'll have to remove the write-protect from each file individually).

While you're having lots of fun creating new empty pages, remember that it's not so easy to get rid of empty pages again.

Tip 22: Saving old versions with DUPLICATE

With the command duplicate you can save a lot of work with texts and pictures. Let's say you have a graphic "Landscape" which is so good that you want to develop it in two different directions. Exit geoPaint and duplicate the graphic (new name: "Landscape2"). Change the name of the first one to

"Landscape 1" (rename). You can now experiment with "Landscape 1" in one direction and with "Landscape 2" in another.

In this manner you can also easily create a copy if you want to try some major changes to a document and don't want to lose the original in case things don't work out.

Tip 23: Form letters with DUPLICATE

You can also use the duplicate command to make something like a form letter. This is something where only the name and address are different from letter to letter. Create a master letter, such as an "Invitation", and leave space for the name and address. Then duplicate the letter as often as needed. All you have to do is add the names and addresses to the files with geoWrite. If you give the files appropriate names when duplicating them (Smith Inv., Jones Inv., ...) you not only save a lot of work, you also know whose invitation is being printed.

Tips 24: Remove extra icons with RESET

If you copy files from diskette A to diskette B, the icons on disk A will still remain on the border. We are often too lazy to select "A" again with open and move the icons back. Also, if you are looking for a document on several disks, and there are several icons on the border of each, your border will soon be full. You can't just replace these icons in the deskTop window because the deskTop will misunderstand that as a request to copy the files. There is a simple solution:

In order to remove all of the icons from previous disks from the border, just select RESET under special. Only the icons which were on the border of the inserted (current) disk will remain. All others will disappear.

Tip 25: GEOS convert V1.0 has no risks

If you want to use a disk with GEOS which was not created (formatted) by GEOS, GEOS will ask if you want to convert the disk to GEOS FORMAT. You may have been annoyed at this point because you were afraid you might lose some data. (FORMAT sounds rather dangerous), while on the other hand you can't copy files from a non-GEOS FORMAT disk. GEOS identifies its disk

by attempting to place an icon on the border of a disk, since this is possible only on a GEOS disk.

You can let your disks be converted to GEOS FORMAT without worrying. GEOS requires an extra sector in which to place information for the border. Normally track 19, sector 8 is used for this, but if this is used, GEOS will look for a free sector. This way no data will be lost.

You should not use this procedure on program disks which you have purchased. These diskettes often have copy-protection, which can be destroyed by using GEOS.

Tip 26: SWAP file - When GEOS doesn't fill the empty space

You may know that when you create a new file (document, graphic, or notes), GEOS searches from the beginning for a free space for the icon which belongs to the file. Curiously, this doesn't always seem to work. For example, if you have a space free on the first page and then you create a notes file with the note pad, it will not appear on the first page but somewhere else. How is this possible?

Before GEOS loads an accessory, the part of the computer's memory that will be changed will be saved on disk. GEOS calls this a "SWAP file". Naturally, GEOS searches for a place for this file starting on page one. When an accessory is then loaded (such as note pad), which then creates a file itself, the space on the first page is occupied, so the next free space is taken.

When you exit the accessory, the original memory of the C64 will be restored. The SWAP file will be loaded at the appropriate location and then deleted. A blank space now appears on the first page in the deskTop. Usually you never see a SWAP file.

If you want to take a look at such a secret file once, load an accessory and then just turn the computer off (not if you are processing important information or during loading or saving, of course). You will be able to see this secret SWAP file by loading a directory in BASIC (load "$", 8). The name will be listed as "SWAP file". When GEOS is rebooted and this disk is inserted GEOS will automatically delete the file.

```
 0 "Disk name           " 01 2a
75    "desktop"                    usr<
90    "geowrite"                   usr<
22    "PREFERENCE MGR"             usr
13    "ALARM CLOCK"                usr
15    "CALCULATOR"                 usr
17    "NOTE PAD"                   usr
 3    "nOTES"                      usr
26    "cALIFORNIA"                 usr
23    "cORY"                       usr
13    "dWINELLE"                   usr
34    "rOMA"                       usr
46    "uNIVERSITY"                 usr
11    "font editor"                usr
13    "[sWAP fILE"                 usr
```
or for GEOS V1.2
```
13    "sWAP fILE"                  usr
```

Figure 3: the SWAP file

Tip 27: When accessories can't be loaded anymore

This SWAP file is the reason that there must be a certain amount of room on the disk in order to use an accessory. If this room is not available, GEOS cannot save anything on the disk and (usually) outputs an appropriate error message. If GEOS returns to the deskTop without an error message after a few seconds of loading, you probably don't have enough free space on the disk. Sometimes no error messages will be given, especially with programs which store something themselves (such as the preference mgr: Preferences).

Tip 28: How much space does an accessory need?

It isn't all that easy to figure out how much memory space a given accessory needs. You can't just select INFO. This will certainly display the size of the program, but not the memory requirements. How does GEOS know how large a program is, how much space is free on the disk, and how much must be stored to the disk as a SWAP file when the accessory is loaded? The size of the file, which is displayed in the INFO sector in kilobytes, is taken from the directory entry. There bytes 28 and 29 contain the number of blocks. GEOS divides this number by four and can then output the number in Kbytes.

The number of free blocks on the disk is obtained from the BAM (Block Availability Map). The size of the SWAP file when loading an accessory is determined from the INFO sector. There bytes 71, 72 specify the load address and bytes 73, 74 specify the end address. GEOS calculates the memory requirement from the difference. With a simple trick, however, you can easily learn how much space an accessory needs when loading.

Create a SWAP file for each accessory by turning the computer off after the accessory is loaded. Load the directory from BASIC and note the number of blocks the SWAP file occupies. Divide by 4 and you know how large the file is, and since the memory requirement of the accessory is the same size, you know how much memory it needs.

Tip 29: Paging with the keyboard

You can save some work and time if you don't use the joystick to page through the various directory pages. You can just press the corresponding number instead. Using the keyboard often simplifies certain processes, even if a user interface with a "mouse" is a good idea.

Tip 30: CLOSE can be omitted

If you change disks as described in the manual, you first have to click `close` and then `open`. `Close` can be omitted. `deskTop` will accept the new disk if you just click `open`. This will save you some time and effort, because after `open` the pull-down menu disappears and you have to click `disk` and then `open` again. You can also just click the dark disk icon in the upper right corner.

Tip 31: All accessories under GEOS

You don't have to page through the directory to find the accessory you want. Just click `geos` and the `deskTop` will show you all of the available accessories. You can then click the one you want.
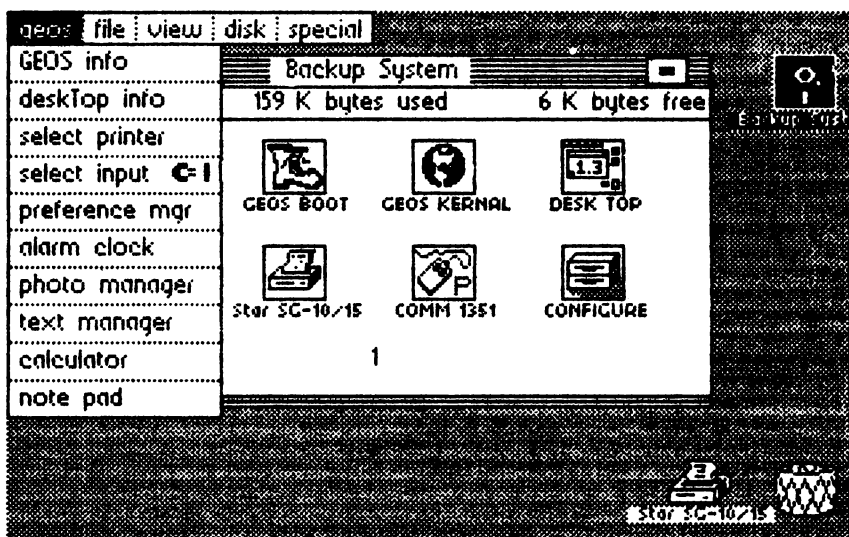
Figure 4: GEOS accessories

Tip 32: Working copies

In order to work with GEOS, you need work disks.
There are three ways to make them:

• If you have two disk drives you can use the command copy under disk.

• You can make a copy of the original disk with DISK COPY and then delete all of the unneeded files from the copy. This takes a while, though, since important files are write-protected.

• You can format a disk and then copy all of the files you need to it, one by one.

In any event you can save yourself some time if you create a copy of the original application disk once and then remove DISK COPY and unneeded printer drivers. Don't use this disk as a work disk—set it aside. Now when you need a work disk, make a copy of the "master work disk" with DISK COPY. For a geoWrite work disk, for example, you just have to delete geoPaint in order to make room for your documents.
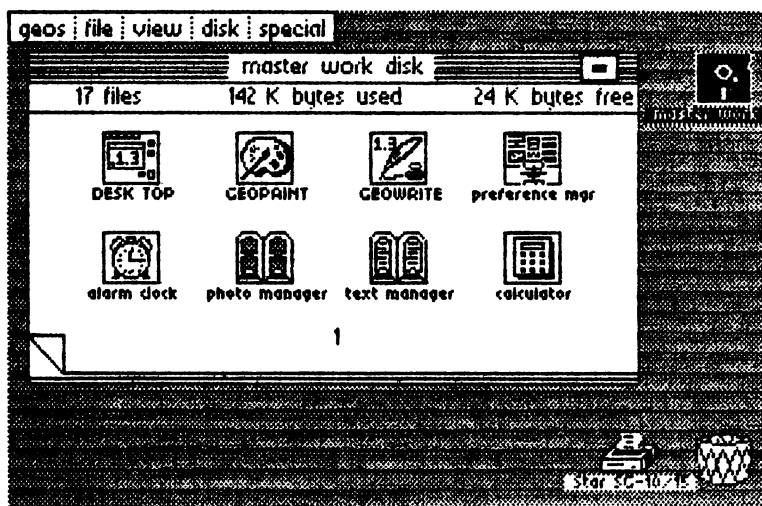
Figure 5: Master work disk

1.3 geoPaint

Tip 33: Double-clicking pictures and documents

You can save a lot of time and effort if instead of first loading the application (such as geoPaint) and then selecting the desired file (picture), you load the picture you want along with geoPaint. This is quite simple.

Pictures and documents are called "Application Data" (GEOS file type = 7). GEOS can immediately recognize from this file that a picture belongs to geoPaint, for example. When you double-click it, GEOS will find the application it belongs to in the INFO sector. If you look at the INFO sector of a picture once, you will see how GEOS knows what application it belongs to.

The application will be loaded and it will pass the name of the file which you double-clicked. This avoids the whole procedure with CREATE, OPEN, QUIT, and subsequent selection.

Unfortunately you can't double-click notes or preferences to load the corresponding accessory. These files have file type 4 (= system file) and GEOS allows almost no actions for this file type. (No double-click, no duplicate, no rename).

By the way, there is an interesting effect if you click a TEXT or PHOTO ALBUM. Don't be confused by the error message:

"This file can't be opened from the deskTop"

Of course the photo manager is on the disk. But the PHOTO ALBUM has an Application Data file type of 7. GEOS therefore searches for the application photo manager, and since it is an accessory, GEOS doesn't find it.

Tip 34: Precise copying in geoPaint

If you want to copy or move a section in geoPaint, mark it out with the section marker and then select copy or move in the status line in the lower right. Then you must click in the marked area in order to be able to move or copy it. The location where you click is very important. The point of the picture which you click will be right at the position of the cross at the destination.

This may sound rather complicated. Let's take an example: You have drawn a rectangle and now want to copy it exactly below another rectangle so that the corners match up. Mark the first rectangle and click right at the desired corner in the marked region. Now move the cross to the desired location on the second rectangle and click when the cross points to the corner. Rectangle 1 will be copied in such a manner that the previously clicked location will begin right at the tip of the arrow.
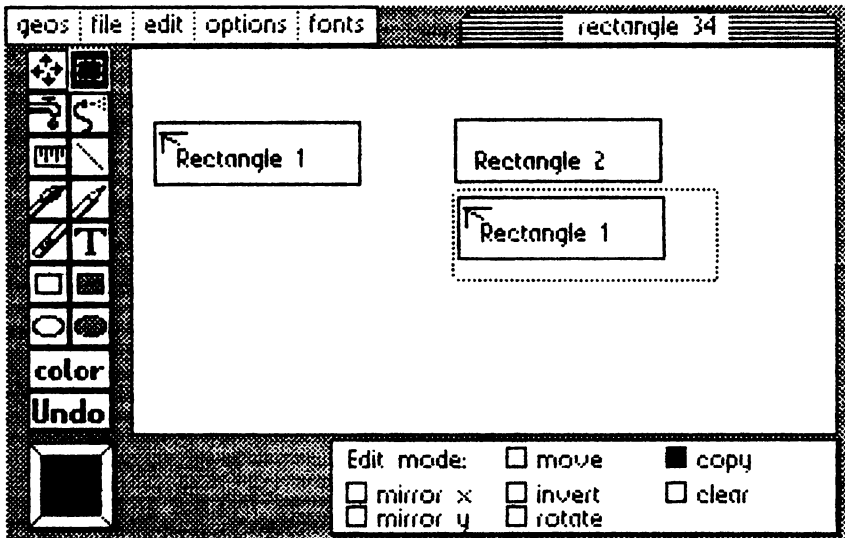
Figure 6: Copying in geoPaint

Tip 35: Fast movements with the page pointer

If you want to move quickly from the upper corner or a page to the lower corner, scrolling with the scroll arrows is tiring and time-consuming. At the bottom of the screen you see the page pointer, a symbolic sheet which displays your entire page. There is a small rectangle on it which displays the position of the geoPaint window on the page. Simply click on the small rectangle and place it down at the bottom. The bottom of the page will appear immediately in the window.

Tip 36: The empty pattern in the pattern menu can also be used

In the pattern menu, the first pattern on the left is completely empty. If you use this pattern to fill a surface (water faucet) in which there is something already

19

drawn, nothing happens. If you use the spray can with this pattern, the sprayed area will become "blurred," meaning that more and more points will be removed until the surface is completely empty (what we're actually doing is spraying with the background color). Spraying text like this can be used to produce interesting effects. The letters appear old and weathered.
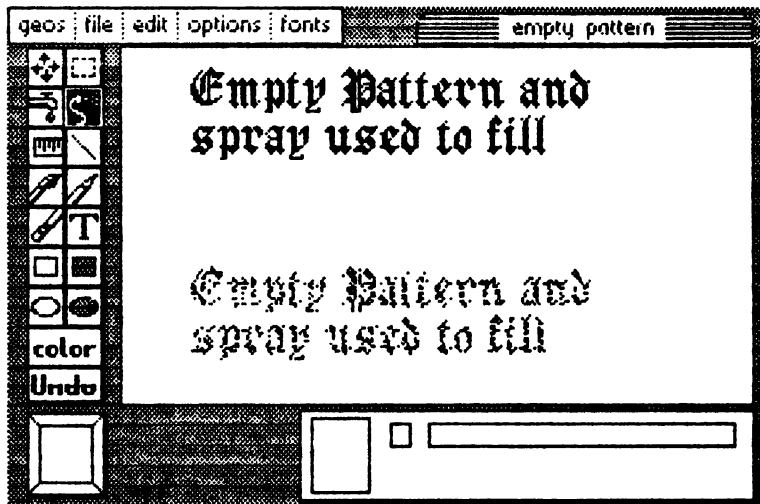
Figure 7: Empty pattern filling

Tip 37: Mixing multiple patterns

You can fill a surface with one or more patterns. But before you do this you should save your drawing with update. Unwanted effects can sometimes happen (bright spots). Moreover, the proper order of the patterns is important. The first pattern you use should be the one with the fewest points set and which does not create any closed regions. The brick pattern is not a good one for this since only one brick will be filled with the next pattern.

Tip 38: Problems when missing type and patterns

Sometimes words and patterns don't mix. First, certain letters which contain closed regions will not be filled. Second, parts of the characters can sometimes be "overwritten" and become unreadable. Before you fill a surface which contains text, you should save the current state of the drawing with update.
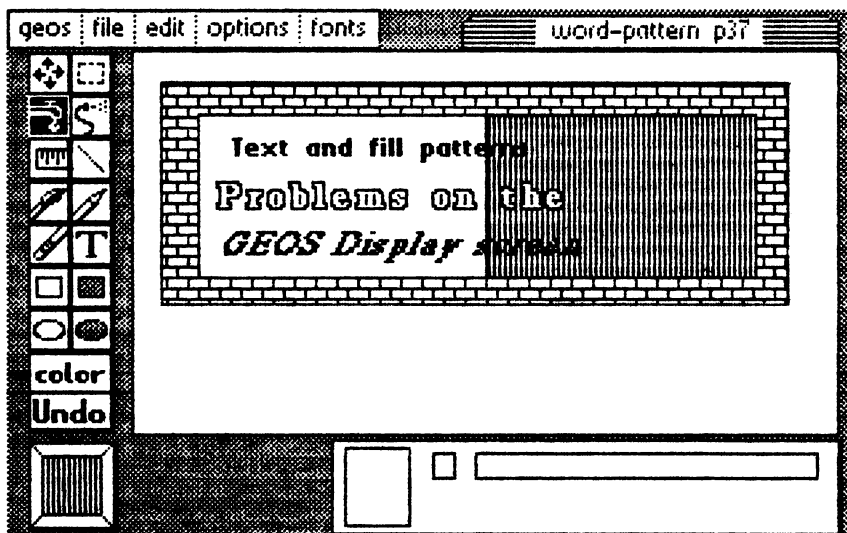
Figure 8: Mixing type and fill patterns

Tip 39: Wild effects with brushes and patterns

The fact that the various brushes can all paint with the patterns can lead to some effects which are difficult to create with something like the pencil. Try these:

- The small point brush or a small line brush with a dotted pattern. This creates different dotted lines.

- The long dotted brush. This produces a good effect with almost any pattern.
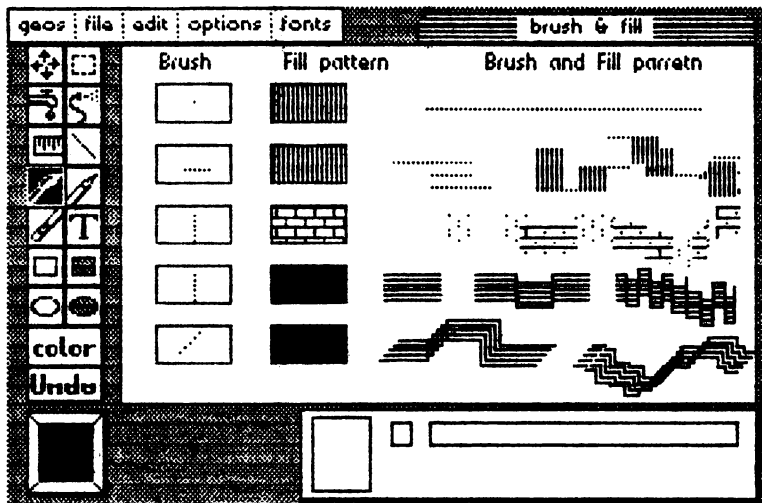
Figure 9: Pixel and pattern combinations

Tip 40: Double-clicking in the tool menu

Many of the functions on the tool menu have double functions. If this tool is already active (clicked once and dark) and you double-click on it, the following tools have the following additional functions:

-   Brush: `change brush`—select one of 32 brushes

-   Pencil: Switch between `pixel edit` and `normal edit`.

-   Eraser: Erases the window (not the entire page).

Tip 41: More tools in Pixel Edit

In the `pixel edit` mode more tools can be used than are mentioned in the manual. Some have different functions, however.

-   Scroll arrows: switch to "Set magnifier." Clicking the scroll arrow is the fastest way to set the magnifier.

22

- Water faucet: fills surfaces as in `normal edit`. Note: Filling ends at the borders of the `pixel edit` window.

- Spray can: spray function; the area effect of the spray can will not correspond to the displayed size of the can but to a much larger area. The magnifier shows the enlarged effect.

- Ruler: normal function.

- Line: normal function.
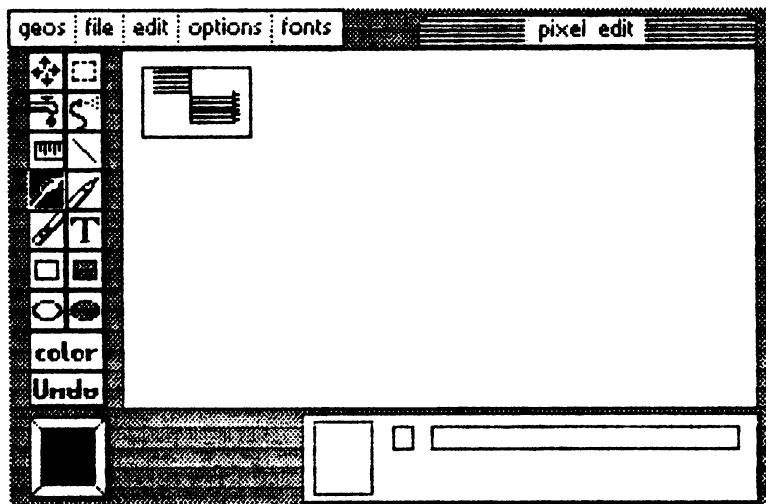
Figure 10: Pixel edit

Paint brush: paints with the selected brush and pattern. As with the spray can, the result is much larger than the brush.

- Eraser: normal function.

- Pencil: sets and erases individual points. In `pixel edit` it has three functions (no effect [just mouse movement], setting, erasing). If you click on an empty location, "Set points" will be enabled, while if you click on a set point, "Erase points" will be enabled.

- Rectangle: normal function.

- Filled rectangle: normal function.

- Undo: normal function.

Tip 42: Activate UNDO immediately

The Undo function allows you to undo the effects of the last operation. You must not do anything else between the action you want to undo and selecting Undo, however. Do not click on a pull-down menu before Undo. Undo is possible only once. You cannot undo two successive operations.

Tip 43: How can I exit the pattern or brush menu?

If you click change brush or the pattern selection and you decide not to change anything, you can't just exit. In the pattern selection you can simply click the lower left pattern square, while in change brush you must select the current brush in order to exit.

Tip 44: Moving objects across the entire page

Unfortunately you cannot simply frame parts of your drawing with the section marker and move out of the geoPaint window in order to place it in some other part of the page. When you have marked out the area and clicked the scroll arrow, the marking will disappear immediately. In spite of this it is possible to move parts of graphics across the page. To do this, just put them in the PHOTO SCRAP (cut, copy). Then move the geoPaint window to the desired location and paste the section back in.

Tip 45: Drawing library with PHOTO SCRAP and PHOTO ALBUM

You can save a lot of work with the PHOTO SCRAP by putting often-used objects there. Let's say you are working on your first electrical schematic. You can copy each newly-drawn component into the PHOTO SCRAP. You can't just copy the component in with copy or the current contents of the PHOTO SCRAP will be erased. Put the contents of the PHOTO SCRAP on a blank part of your page with paste. Then mark the component and copy it here with copy. Now you can copy your new library back into the PHOTO SCRAP.

Tip 46: Caution with UNDO after CUT, COPY, and PASTE

The Undo function does not work on these three commands; in fact, it disturbs things more than it helps. A cut Undo will restore the removed section of the graphic, but the PHOTO SCRAP will be changed, of course. If you select the Undo function after PASTE, the material you pasted will not disappear and the dotted border will remain as well. If a pull-down menu falls over this border it will disappear when the menu does.

1.4 geoWrite

Tip 47: Right margin at 5 saves time and makes text easier to read

The left and right margins in geoWrite are not placed very well, in our opinion. You can write quickly and easily if you set the right margin to "5" at the beginning of your work. To do this, move the arrow to the right as far as possible. At the right edge next to the "7" you will find a strange symbol—the marker for the right margin. If you click on it you will get a moving "M" which you can simply move to the "5" and let go of the mouse button. Now GEOS doesn't have to switch between the left and right halves of the page and the screen does not have to be redrawn every time.
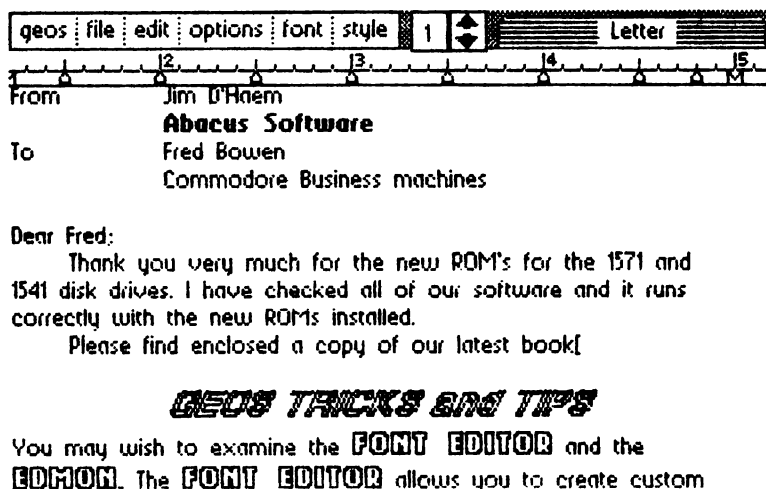
Figure 11: Right margin

Incidentally, the marker for the right margin is also an "M". The only reason it looks so strange is that the other seven tabs are also sitting at this position. Once you have placed all eight tab stops between the left and right margins the margin marker will be a plain "M" (we did this in Figure 11).

26

Tip 48: Removing tabs easily

Tabs are set by pointing to the desired location and clicking. To move a tab, just point at it and click and move it to the new location. There you can let go of the button. To remove a tab, just click it (holding down the button again) and move it up. When you let go the position will be cleared.

But if you want to remove all eight tabs this way, you have a lot of work to do. It's easier just to move the right margin as far left as possible. All tabs which are now to the right of the right margin will be removed, and you can then move the right margin back to the desired position. Actually, geoWrite doesn't delete the tabs, it just moves them all back to the right margin. When you then move the right margin, all of the "deleted" tabs move with it. In any event, the tabs are reset and disappear.

Tip 49: BSW 9 point saves time and effort

Write your documents in BSW 9 point first. For one, it is a relatively small type, so you can see more text in the geoWrite window. geoWrite is a WYSIWYG program (What You See Is What You Get). This has many advantages, but it also has its disadvantages. If you can't see something you also can't mark it, and so you can't copy it or cut it out. If you can only see five lines of a larger type on the screen, but you want to change 20, you have to do a lot of scrolling.

Also, geoWrite must access the disk for each changed type in order to display the font. If you use a lot of fonts right away when writing, geoWrite will have to access the disk each time the screen is redrawn. The extreme case of this is when you use a different font for every character. Then you can go have a cup of coffee every time GEOS redraws a page.

When you have written and corrected your document, mark all of the desired locations and change the type.

geoWrite does not have to access the disk to change the type styles (**bold**, underline,.outline...). These changes are calculated instead of being loaded. You can therefore use various type styles on the first draft without having to sacrifice any speed.

Tip 50: Changing type: Better to go from University 6 point to 24 point than the other way around

It is quite easy to change long passages of text from a small type size to a larger one. Usually you can see the entire section in the geoWrite window, you can mark it, and select the new type size without any problems. The reverse process is much more difficult, however, and you should avoid it if possible.

Let's take an example: You have an entire page in University 24 point and you want to convert it to University 6 point. Mark as much text as you can see and click the small type. geoWrite will convert the marked text properly, but the result is only 1-2 lines of University 6 point. The remaining text will naturally remain in the other type size because it wasn't marked, and you have to repeat the process. This can take quite a while. Going from small to large saves time and your nerves.

Tip 51: Text formatting and pasting pictures

You can paste drawings created in geoPaint into geoWrite documents. Unfortunately there are some restrictions:
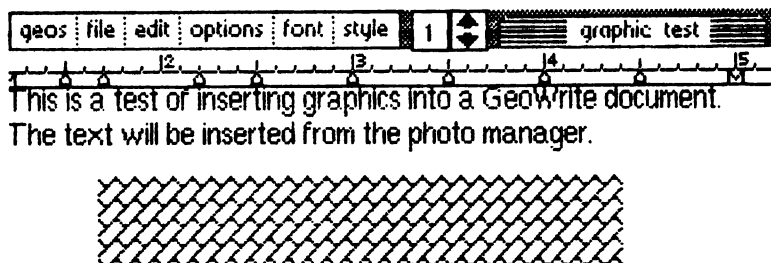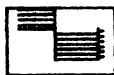
Figure 12: Graphic insert

- A pasted graphic occupies the entire width. No text can be placed to the right or left of the graphic.

- Graphics are automatically centered and cannot be moved.

28

Chapter 2

Programming Tricks

2.1 Printing notes

GEOS has developed certain standards for a number of programs. For example, the various fonts are stored in such a manner that they can be accessed and recreated by geoPaint as well geoWrite (and our FONT Editor). Unfortunately, files created by the different programs are not compatible with each other. This means that it is impossible to print the notes created with the note pad, for example. Notes is stored as a system file (file type = 04) and the user really can't do much with such a system file. It cannot be duplicated or renamed, and the program which is created cannot be loaded by clicking on the system file.

Note pad also has some other characteristics which make cooperation with geoWrite very difficult. The settings for the margins and tabs are stored at the beginning of a geoWrite file and they must be set to reasonable values or geoWrite will crash when it tries to read the text. Note pad stores its text with no settings, however. This means that even if we change the file type we still can't print the file with geoWrite.

Conversely, it is also impossible to use geoWrite files as notes. If you look at a disassembly of note pad you will clearly see that the developer was under time pressures. In particular there are no control characters provided for different fonts or type styles. If note pad reads a file created by geoWrite (after some tricks), for example, it will crash on the first control character.

We weren't satisfied with this situation, so we wrote a program which will print notes. This program must solve two problems:

- It must read a GEOS file in VLIR format

- It must convert the GEOS ASCII character set to CBM ASCII

The following program "PRINT NOTES" does both of these for you and is also very easy to use. After typing it in and saving it, just insert the disk containing the notes you want to print. After starting and pressing a key it will search the directory for notes. All pointers in the VLIR sector will then be read and stored in an array.

The individual pages will first be displayed on the screen. After each page you can press RETURN to see the next page or select the printer output directly with "P". You may also have to adapt the printer output to your printer. We sent a carriage return (CHR$(13)) and a linefeed (CHR$(10)) to the printer after each line. You may have to omit the linefeed if your printer double spaces.

On each page you will automatically get a title with a page number. You can also output the date here or anything else you want.

We have a little suggestion for those who make heavy use of the note pad and have filled up all the pages with text. You can easily alter the program so that only certain pages will be printed, such as:

a)   from page 1 to page x

b)   from page x to the end

c)   from page x to page y

d)   only the pages marked on the screen

For cases a), b) and c), you just have to read the appropriate values with an INPUT statement and adapt the output loop at line 740 to work with these values. You should first check to make sure that x and y line in the range from 1 to AN.

In case you could dimension an extra column with NO(128,3) instead of NO(128,2). When displaying the pages you could allow "Y", "N", or "P" as the input. For "Y" you set NO(I,3) to "1" and for "N" you set it to "0". In the printing loop you then print only the pages which contain a "1" in the third column.

2.1.1 The program "PRINT NOTES"

```
10 DIM NO(128,2)
20 PRINT CHR$(147);CHR$(14)
30 PRINT"           note pad PRINTER!"
40 PRINT
50 PRINT"      INSERT note pad DISKETTE !"
60 PRINT""
70 PRINT"        AND PRESS ANY KEY"
80 GET A$:IF A$="" THEN 80
90 OPEN5,8,15,"I"
100 OPEN3,8,3,"#"
110 T=18:S=1
120 SC$="/OTES" :SG$="": REM SHIFTED N THEN otes
130 FOR I=1 TO LEN(SC$)
140 : A=ASC(MID$(SC$,I,1))
150 : GOSUB 940:REM "C64 TO GEOS"
160 : SG$=SG$+CHR$(A)
170 NEXTI
180 : REM
190 PRINT#5,"U1:"3;0;T;S
200 PRINT#5,"B-P:"3;0
210 GOSUB 910:NT=A
220 GOSUB 910:NS=A
230 BP=2:GE=0 :
240 FOR I=1 TO 8:
250 : PRINT#5,"B-P:"3;BP
260 : GOSUB 910:FT=A:
270 : GOSUB 910:TT=A:
280 : GOSUB 910:SS=A:
290 : FI$=""
300 : IF FT=0 THEN 370
310 : FOR J=1 TO 16:GOSUB 910
320 :    IF A=160 THEN J=16:GOTO 340
330 :    FI$=FI$+A$ :
340 : NEXTJ
350 : PRINTFI$
360 : IF FI$=SG$ THEN PRINT"FOUND",TT,SS:I=10:GE=1: GOTO380
370 : BP=BP+32
380 NEXTI
390 IF GE=0 THEN IF NT>0 THEN T=NT:S=NS:GOTO 190
400 IF GE=0 THEN PRINT"SORRY NO NOTES":CLOSE5:END
410 REM "READ OUT VLIR"
430 PRINT#5,"U1:"3;0;TT;SS
440 ANZ=0:BP=2
450 : PRINT#5,"B-P:"3;BP
460 : GOSUB 910:NT=A:PRINTNT " NOTE TRACK"
470 : GOSUB 910:NS=A:PRINTNS " NOTE SECTOR"
480 : IF NT <1 OR NT>35 OR NS>21THEN520
490 : NO(AN,1)=NT:NO(AN,2)=NS
500 : AN=AN+1:BP=BP+2:
510 : IF AN <127 THEN 450
520 IF AN=0 THEN PRINT"NO PAGES":CLOSE5:END
530 REM "READ NOTES"
540 FOR I=0 TO AN-1 :PRINT"-----------------------------"
```

33

```
550 : PRINT"PAGE: ";I+1
560 X = 1
570 : T=NO(I,1):S=NO(I,2)
580 : IF T<1 OR T>35 THEN PRINT"ERROR":CLOSE1:CLOSE5: END
590 : PRINT#5,"U1:"3;0;T;S
600 : PRINT#5,"B-P:"3;2
610 : GOSUB 910
620 : IF A=0 THEN 680
630 : GOSUB 980:REM "GEOS TO C64"
640 : IF A>31 THEN PRINTCHR$(A);
650 : IF A=13 THEN X=1: PRINTCHR$(A);
660 : IF X>30 THEN IFA=32THEN:X=1: PRINTCHR$(13);
670 : X=X+1:  GOTO 610
680 : PRINT:INPUT "N=NEXT PAGE    Q=QUIT";N$:IFN$="Q"THEN I=AN
690 NEXTI
700 INPUT "PRINTER Y/N";N$
710 IF N$<>"Y"  THEN 890
720 REM "PRINT NOTES"
730 OPEN 1,4,7
740 FOR I = 0 TO AN-1
750 :PRINT#1,"--------------------PAGE  ";I+1;"  --------------------"
760 :X=1
770 : T=NO(I,1):S=NO(I,2)
780 : IF T<1 OR T>35 THEN PRINT"ERROR":CLOSE1:CLOSE5:STOP
790 : PRINT#5,"U1:"3;0;T;S
800 :PRINT#5,"B-P:"3,2
810 GOSUB 910
820 : IF A = 0 THEN 875
830 : GOSUB 980: REM"GEOS TO 64"
840 : IF A>31 THEN PRINT#1,CHR$(A);
850 : IF A=13 THEN X=X+1:PRINT#1,CHR$(A)
860 : IF X>50 THEN IF A=32 THEN X=1:PRINT#1,CHR$(A)
870 : X=X+1:  GOTO 810
875 PRINT#1,CHR$(13);CHR$(13)
880 NEXTI :PRINT#1:CLOSE1
890 CLOSE3:CLOSE5
900 END
910 GET#3,A$:A=ASC(A$+CHR$(0))
920 RETURN
930 REM COMO TO GEOS
940 IF A>64 AND A<91 THEN A=A+32:RETURN
950 IFA>192ANDA<219THEN A=A-128:RETURN
960 RETURN
970 REM GEOS TO COMO
980 IF A>64 AND A<91 THEN A=A+128:RETURN
990 IFA>96ANDA<123THEN A=A-32:RETURN
1000 RETURN
```

2.1.2 Documentation for the program "PRINT NOTES"

In this book you will find a whole set of programs which help you and ease or improve your work. But maybe you might think a routine of ours to not be so useful as we did, or you may miss one that we thought was unnecessary. Our programs are really suggestions (that work, of course), and we encourage you to adapt them to your own needs. Therefore, most of the programs in this book are documented, and it should be fairly easy for you to understand the operation of each program. Maybe you'll even learn a few tricks from them.

We have not documented some programs for a variety of reasons, and even one because it was too difficult to understand ("Backup copy").

But here is the documentation for the program "PRINT NOTES":

10	The array NO(128,2) contains the pointers to the possible pages in the note pad. NO(I,1) contains the track and NO(I,2) contains the sector.
20-80	After the screen message the program waits until the disk with notes has been inserted and a key is pressed.
90	The command channel is opened and the inserted disk is initialized.
100	Another channel to the disk drive is opened and a buffer is assigned to it.
110	T is used as the variable for the current track and S for the current sector. At the beginning both are intialized to the first sector of the disk directory.
120	SC$ contains the desired filename (notes) which is to be searched for. The C means that the characters are in Commodore format. The variable SG$ ("G" for GEOS) is initialized as an empty string.
130-170	The program converts in a loop. At the end SG$ contains the desired file entry in the GEOS encoding. The subroutine at line 940 is used for this. It converts a CBM ASCII value in A to GEOS ASCII. The converted value is returned in A. The program is purposely kept general in lines 120 to 170 in order to be able to find other file entries as well.
190	Here is the loop which reads the directory. The current sector is loaded into the buffer.

200   The buffer pointer is set to the first byte (byte 0). This contains the track of the following sector (if present).

210   The subroutine gets a character from the drive buffer and passes the corresponding ASCII value in A. This will be assigned to the variable NT (new track).

220   The next byte contains the sector. It will be assigned to the variable NS (new sector).

230   BP (buffer pointer) will be set to the file type of the first file entry. The variable for found (GE) will be set to "not found."

240   Start of the check look for the eight file entries in a sector.

250-280   File type, track, and sector of the file entry are fetched and stored in variables.

290   The filename is placed in FI$. FI$ must first be initialized to a null string.

300   If the file type is 0, the file is deleted and is not considered.

310-340   Up to 16 characters are read in a loop and combined into the filename. Since shorter names are padded with shifted spaces (= 160), the loop will be terminated if this character is encountered.

350   Screen output of the filename.

360   If the desired and found names match, an appropriate message will be printed. In addition, the start track and start sector of the first data block will be printed. GE will be set to found and the loop will be exited.

370-380   The buffer pointer will be set to the file type of the next file and a new pass through the loop will have begun.

390   If the file entry (notes) has not been found yet and there is another directory block, track and sector will be sent to this block and the program will jump to read it.

400   If there are no more blocks and the file entry was not found, a appropriate message will be printed and the program is terminated.

420	TT and SS contain the track and sector of the VLIR pointer block. If for some reason an invalid value was read, the program will end with an error message.
430	Read the pointer block for notes.
440	The number of pages found is set to 0, and the variable for the buffer pointer is set to the third byte because the first two always contain $00 and $FF.
450	Set the buffer pointer to the value which the variable contains. This line is used more than once.
460-470	Each pair of bytes contain the track and sector of a page of notes. These are read from the block and displayed.
480	If the values are not in the permitted range, no more pointers will be read. If NT=0, we know that no more pages will follow.
490	Track and sector are stored in the array .
500	The number of pages is incremented by one and the buffer pointer is incremented by two.
510	If the maximum number of pages has been reached, the next pointer will be read.
520	Here we check to see if any pages are present at all. This is important because if you exit the note pad immediately after entering it you will get an empty note book.
540	Output loop for all existing pages.
550	Each page is given a page number. The date can also be printed on each page here.
560	X will later contain the number of previously printed characters in the line. This can be used to achieve word wrap, without which the screen would become too full to read easily. X is initialized here.
570	Track and sector of the current page are fetched from the array.
580	The value of the track is again checked for validity.
590	Read a page into the buffer.

600	The buffer pointer is set to the first text byte. The two bytes before it always contain $00 and $FF to indicate that there is no following block and that the block contains 255 valid data bytes.
610	A character is read from the buffer. The character is passed in A$ and its ASCII value in A.
620	A zero means the end of the page in notes.
630	GEOS ASCII is converted to Commodore ASCII.
640	If the character is not a control character, it will be printed.
650	If the character is an end-of-line, a RETURN will also be printed on the screen. In addition, the counter for the number of characters in the current line will be reset.
660	At the 30th character in the line, a new line will be started if a space is encountered (A=32). This keeps the text readable on the screen.
670	The line counter is incremented by 1 and we jump to read again.
680	At the end of the page the next page can be printed on the screen by pressing <RETURN>. The printer output can be started directly with "P".
690	End of output loop.
700	Confirm printing with "Y".
730	An output channel to the printer is opened.
740	Output loop for all existing pages.
750	The pages will be separated by a dashed line on the printer. We chose not to start each page on a new sheet of paper to speed up the printing and to save paper.
760-880	This is essentially the same output loop as for the screen output. If your printer prints one line on top of another, add a linefeed (CHR$(10)) to the end of line 850. A new line is printed at the first space (CHR$(32)) encountered after the 50th character. If you want to change this value, replace the 50 in line 860 with another value.

910-920   Subroutine: Here a character is taken from the disk buffer, and the valid ASCII value is determined. A CHR$ (0) is appended to each character before the ASCII value is determined. Otherwise the command ASC (A$) would produce an error with an empty string (A$=" ").

930-960   Subroutine: Here the CBM ASCII value is converted into a GEOS ASCII value. You may be able to use this routine in your own programs.

930-960   Subroutine: A GEOS ASCII value is converted to a CBM ASCII value.

2.2 geoWrite

geoWrite offers you possibilities for the appearance of your text that were previously impossible with a C64. These include proportional type, different type styles, and different fonts. But these possibilities have their price. We don't mean the purchase price of GEOS, we mean the printing speed. Text must be printed as graphics, and this causes the printing speed to drop dramatically.

If you look at the result on paper, you are usually willing to pay this price. But it often happens that after printing something out you notice some errors that you overlooked before and you have to print the document again. This can cause you to lose valuable time, and we can only hope that you don't find more errors in the second printing.

If you look at the way geoWrite files are stored on the disk, you will see that the text is not stored as graphics but as a stream of ASCII characters. In this stream there are also control characters for the formatting and the various types and styles. These characters are not converted into graphics until they are outputted on the screen or printer. This gave us the idea of writing a program which prints geoWrite files simply as text instead of as graphics. This allows you to quickly print a copy of your document that you can use for proof-reading, correct the errors, and then print out a final version with geoWrite.

2.2.1 GEOPRINT: Printing geoWrite files as text

The program GEOPRINT is loosely based on the program "PRINT NOTES".
Several important changes and extensions must be taken into account:

- The text is stored by geoWrite somewhat differently than it is by `note pad`.

- geoWrite stores the settings for the margins and tabs at the start of the file. These settings must be skipped.

- Various control characters for type changes occur in geoWrite files and these must be ignored.

- A geoWrite file can consist of a maximum of 63 pages, while `notes` can have up to 127.

Before we get to the program GEOPRINT, we want to give you a few suggestions for using it. After you have started it, insert the disk containing the file you want to print. After pressing a key you will be asked to enter the name of your file. The program will search the directory until it finds this name. All valid names in the directory will be displayed, so you can read them on the screen in case you forgot the exact name.

If the program finds the file entry, the pointers to the VLIR structure will be read into the array. You can then decide if you want the output to go to the screen or to the printer. After each page press RETURN, or press Q if you want to terminate the output.

GEOPRINT is not particularly fast. We use it only in the compiled form, in which the speed is quite satisfactory. If you have the BASIC 64 compiler from Abacus, go ahead and compile it. Both versions are included on the optional program diskette. If you don't, there are a number of ways the program can be made faster. We have not done this in the listed version to prevent it from being more complicated than necessary.

41

2.2.2 The program GEOPRINT

```
10 DIM NO(64,2)
40 PRINT CHR$(147);CHR$(14)
60 PRINT"         geoWrite-TEXT PRINTER!"
80 PRINT"         ----------------------"
100 PRINT
120 PRINT
140 PRINT"         PLEASE INSERT DISKETTE !"
160 PRINT
180 PRINT"         AND PRESS ANY KEY       "
200 GET A$:IF A$="" THEN 200
220 OPEN5,8,15,"I"
240 OPEN3,8,3,"#"
260 T=18:S=1
280 PRINT:PRINT
300 PRINT:PRINT
320 INPUT "         TEXT NAME:";SC$
340 SG$=""
360 FOR I=1 TO LEN(SC$)
380 : A=ASC(MID$(SC$,I,1))
400 : GOSUB 2480:REM "C64  TO  GEOS"
420 : SG$=SG$+CHR$(A)
440 NEXTI
460 :REM
480 PRINT#5,"U1:"3;0;T;S
500 PRINT#5,"B-P:"3;0
520 GOSUB 2420:NT=A
540 GOSUB 2420:NS=A
560 BP=2:GE=0 :
580 FOR I=1 TO 8:
600 : PRINT#5,"B-P:"3;BP
620 : GOSUB 2420:FT=A:
640 : GOSUB 2420:TT=A:
660 : GOSUB 2420:SS=A:
680 : FI$=""
700 : IF FT=0 THEN 840
720 : FOR J=1 TO 16:GOSUB 2420
740 :    IF A=160 THEN J=16:GOTO 780
760 :    FI$=FI$+A$ :
780 : NEXTJ
800 : PRINTFI$
820 : IF FI$=SG$ THEN PRINT"FOUND",TT,SS:I=10:GE=1: GOTO860
840 : BP=BP+32
860 NEXTI
880 IF GE=0 THEN IF NT>0 THEN T=NT:S=NS:GOTO 480
900 IF GE=0 THEN PRINT"SORRY NO  ";SC$;" FOUND":CLOSE5:END
920 REM "VLIR READ"
940 IF TT<0 OR TT>35 THEN PRINT"ERROR ":END
960 PRINT#5,"U1:"3;0;TT;SS
980 ANZ=0:BP=2
1000 : PRINT#5,"B-P:"3;BP
1020 : GOSUB 2420:NT=A:PRINTNT:REM "TEXT-TRACK"
1040 : GOSUB 2420:NS=A:PRINTNS:REM "TEXT-SECTOR"
1060 : IF NT <1 OR NT>35 OR NS>21THEN1140
```

```
1080 :  NO(AN,1)=NT:NO(AN,2)=NS:PRINT NT,NS
1100 :  AN=AN+1:BP=BP+2:
1120 :  IF AN <64   THEN 1000
1140 IF AN=0 THEN PRINT"NO PAGES":CLOSE5:END
1160 REM ****************************
1180 REM "TEXT READ"
1185 :  INPUT"SCREEN/PRINTER      S/P";N$
1186 IF N$<>"P" AND N$<>"S" THEN 1185
1187 :  IF N$<>"P" THEN 1200
1190 :  OPEN1,4,8:REM NO LINEFEED
1193 :  PRINT#1,CHR$(27)CHR$(64):REM INITIALIZE PRINTER
1195 :
1200 FOR I=0 TO AN-1
1205 :  IF N$="S" THEN PRINT"------------------------------"
1210 :  IF N$="P" THEN PRINT#1,"------------------------------";CHR$(10)
1220 :  IF N$="S"THEN PRINT"PAGE : ";I+1
1225 :  IF N$="P"THEN PRINT#1,"PAGE : ";I+1;CHR$(10)
1240 :  X=1 :PO=22
1260 :  T=NO(I,1):S=NO(I,2)
1280 :  IF T<1 OR T>35 THEN PRINT"ERROR ":CLOSE1:CLOSE5: END
1300 :  PRINT#5,"U1:"3;0;T;S
1320 :  PRINT#5,"B-P:"3;0
1340 :  GOSUB 2420:TT=A
1360 :  GOSUB 2420:SS=A
1380 :  MA=255: REM "MAX POINTER"
1400 :  PRINT#5,"B-P:"3;PO
1420 :  IF TT=0 THEN MA=SS-1
1440 :  GOSUB 2420:PO=PO+1
1460 :  IF A<>23 THEN 1780
1480 :  FOR K =1TO 3
1500 :     PO=PO+1:IF PO>MA THEN 1580
1520 :     GOSUB 2420
1540 :     SA(K)=A :REM"TEXT STYLE"
1560 :  NEXT K :GOTO 1440
1580 :  T=TT:S=SS:IF TT=0 THEN PRINT"ERROR ":CLOSE5:END
1600 :  PRINT#5,"U1:"3;0;T;S
1620 :  PRINT#5,"B-P:"3;0
1640 :  GOSUB 2420:TT=A
1660 :  GOSUB 2420:SS=A
1680 :  MA=255: REM "MAX POINTER"
1700 :  PO=2
1720 :  PRINT#5,"B-P:"3;PO
1740 :  IF TT=0 THEN MA=SS
1760 :  GOTO 1520
1780 :  IF A<>12 THEN 1840
1800 :  IFN$="P"THEN PRINT#1,CHR$(13)CHR$(10)
1805 :  IFN$= "S"THEN PRINTCHR$(13)CHR$(10)
1820  :IFN$="P"THENPRINT#1,"-----------------------------------------
";CHR$(10)
1825 :  IFN$= "S"THEN PRINT"-----------------------------------------"
1840 :  IF A=0 THEN PRINT"ERROR ":CLOSE5:END
1860 :  GOSUB 2560:REM "GEOS  TO  C64"
1880 :  IF N$="S"THEN IF A>31 THEN PRINTCHR$(A);
1885 :  IF N$="P"THEN IF A>31 THEN PRINT#1,CHR$(A);
1900 :  IF N$="S"THEN IF A=13 THEN X=1: PRINTCHR$(A);
1905 :  IF N$="P"THEN IF A=13 THEN X=1: PRINT#1,CHR$(A);CHR$(10)
```

```
1920 : IFN$="S" THEN IF X>30 THEN IFA=32THEN:X=1: PRINTCHR$(13);
1925 :IFN$="P"THENIFX>60THENIFA=32THEN:X=1:PRINT#1,CHR$(13);CHR$(10)
1940 : X=X+1:IF PO<=MA THEN GOTO 1440
1945 IF TT>0 THEN T=TT:S=SS:PO=2:GOTO 1300
1947 IF N$="P" THEN PRINT#1,CHR$(13);CHR$(10)
1950 : IF N$="P" THEN 1980
1960 : PRINT:INPUT "NEXT PAGE      Q=QUIT";M$
1965 : IFM$="Q"THEN I=AN
1980 NEXTI
2380 CLOSE1: CLOSE3:CLOSE5
2400 END
2420 GET#3,A$:A=ASC(A$+CHR$(0))
2440 RETURN
2460 REM COMO  TO  GEOS
2480 IF A>64 AND A<91 THEN A=A+32:RETURN
2500 IFA>192ANDA<219THEN A=A-128:RETURN
2520 RETURN
2540 REM GEOS  TO  COMO
2560 IF A>64 AND A<91 THEN A=A+128:RETURN
2580 IFA>96ANDA<123THEN A=A-32:RETURN
2600 RETURN
```

2.2.3 Documentation for the program GEOPRINT

20	The array NO(128) will contain the pointers to the pages of the text.
40-200	After the screen message the program waits until you insert the disk containing the file and press a key.
220	The command channel is opened and the inserted disk is initialized.
240	Another channel is opened to the disk drive and a buffer is assigned to it.
260	T is used as the variable for the current track and S for the current sector. At the beginning these are both set to the first sector in the directory of the disk.
280-340	SC$ contains the filename to be found. The "C" means that the characters are in Commodore format. The variable SG$ ("G" for GEOS) is initialized to a null string.
360-440	The program converts characters in a loop. At the end SG$ contains the file entry in GEOS encoding.
480	Here begins the loop for reading the directory. The current sector is read into the buffer in this line.
500	The buffer pointer is set to the first byte (byte 0). This contains the track of the following sector (if present).
520	The subroutine fetches a character from the disk buffer and returns the ASCII value in A. This is assigned to the variable NT (new track).
540	The following byte contains the sector of the next directory block. It is assigned to the variable NS (new sector).
560	BP (buffer pointer) is set to the file type in the first file entry. The variable for found (GE) is set to not found.
580	Here starts the loop for checking the eight file entries in a directory block.
600-660	File type, track, and sector of the file entry are fetched and stored in the variables FT, TT, and SS.

45

680	The filename is constructed in F I $. F I $ must therefore be initialized to a null string.
700	If the file type is 0, the file is deleted and we don't have to bother with it.
720-780	Up to 16 characters are read in a loop and combined into the filename. Since shorter names are padded with shifted spaces (=160), the loop will be terminated on this character.
800	Output of the filename to the screen. This allows you to check if the desired file was skipped and you entered the name wrong.
820	If the name found and the name desired match, an appropriate message is printed. The start track and start sector of the first data sector are displayed. The variable GE is set to found and the loop is exited.
840-860	The buffer pointer is set to the next file entry and a new pass through the loop is started.
880	If a file entry with the desired name was not found and there is another directory block, the track and sector will be set to this block and the program will jump to read again.
900	If there are no more blocks and the name was not found, an appropriate message will be displayed and the program will be ended.
940	TT and SS contain the track and sector of the VLIR pointer block. If for some reason an illegal value is read, the program will end with an error message.
960	Read the pointer block.
980	The number of pages found will be set to 0, and the variable for the buffer pointer will be set to the third byte because the first two always contain $00 and $FF.
1000	Set the buffer pointer to the value in the variable. This line is used more than once.
1020-1040	Each pair of bytes contain the track and sector of the first data block of one page of text. These will be read from the VLIR sector.

1060   If the values do not lie in the allowed range, no more pointers will
             be read. If NT=0, then we know that no more pages follow.

1080   Track and sector of the first data block are stored in the array and
             displayed.

1100   The number of pages is incremented by 1 and the buffer pointer is
             incremented by two.

1120   If the maximum possible number of pages has not been reached,
             the next pointer is read.

1140   Here we check to see if there are any pages present at all.

1185   Output to the printer or screen? The answer is stored in N$. The
             same program is used for both outputs.

1187   If the output is to go to the screen, the printer initialization is
             skipped.

1190-1195   Printer initialization. A channel is opened to the printer and it is
             initialized. If your printer double spaces you may have to change
             the open command to open 1,4,7 (check your interface or
             printer manual). The initialization sequence applies only to an
             Epson printer. If you have another printer, insert the appropriate
             values or omit line 1193 entirely.

1200   Output loop for all pages.

1205   Each page starts with a dashed line on the screen.

1210   All output goes to the printer if printer output was selected. It
             would have been simpler to use the CMD command, but this will
             lead to errors when reading the disk buffer, so we had to choose
             this more complicated route. In the following you will find each
             output once for the screen and once for the printer.

1220   Each page has a page number.

1240   X contains the number of previously outputted characters on the
             line. This allows a sort of word wrap so that the screen is easier to
             read. X is initialized here. In the following PO will be used as a
             counter for the number of bytes in a sector. The program must
             know when the data in a sector is done and a new sector must be
             read. Since geoWrite stores the margins and tabs before the actual
             text, these must be skipped with PO=22.

1260	Track and sector of the current page are fetched from the array.
1280	The value for track is checked for validity.
1300	The buffer pointer is set to the start of the buffer. There we find either the track and sector of the next data block or a 0 and the number of valid bytes.
1340-1360	The first two bytes are fetched from the buffer and stored.
1380	The largest possible value for the buffer pointer is set. It must be variable because the last sector of a page can contain fewer valid bytes.
1400	Set the buffer pointer to the value of the variable PO. This is still 22 in order to read past the settings.
1420	If TT=0, there is no following sector and SS contains the number of valid data bytes. Then the maximum value MA for the pointer is set in the buffer PO.
1440	A character is fetched from the buffer. The character is passed in A$ and its ASCII value in A.
1460	The following program block will be executed if A=23. This is the control character for type change and the following bytes must be skipped.
1480	This loop skips over the next three bytes in the file. These contain control characters for the type and type style.
1500	Increment pointer. If the pointer is greater than the number of valid bytes, a new sector must be loaded.
1520	Get control characters.
1540	Save control characters. In our program they are not used. You could write your own routine to process the control characters and make some changes in the printer output (such as enabling underlining, *italics*, or **boldface**).
1560	End of the control character loop. Back to the normal input.
1580	End of the data in this sector. A new sector must be read. The variables TT and SS contain its address. If no more sectors are

present, an error has occurred and the program ends with an appropriate message.

1600   The next sector is read into the drive buffer.

1620   The buffer pointer is set to the start of the buffer in order to read the first two bytes. These contain the following sector or the number of valid bytes.

1640-1660   Track and sector of the next sector are read and stored.

1680   Set the upper limit for the pointer in the drive buffer to the largest possible value.

1700   Set the variable for the buffer pointer to the first data byte.

1720   Set the buffer pointer to the corresponding variable.

1740   Correct maximum value for pointer if needed (see line 1420).

1760   Back to the loop to read control characters.

1780   Here the program continues if there are no control characters for type change in line 1460. In geoWrite CHR$ (12) stands for the end of a page. If the character just read is not this control character, the following passage is skipped.

1800   If the character was a control character for the end of a page and the output is going to the printer, a carriage return and linefeed will be sent to the printer.

1805   For screen output these control characters will be printed on the screen.

1820-1825   The pages will be separated by a dashed line.

1860-65   GEOS ASCII is converted to Commodore ASCII.

1880-85   If the character is not a control character, it will be printed.

1900-05   If the characters are an end-of-line, a RETURN will be printed on the screen. Also, the counter for the number of characters in the current line is reset.

1920-25   A space occurring after the 30th character in the line will cause a new line to be started on the screen. This prevents words from

being broken at the end of the line. Output to the printer will start a new line after the first space after the 60th character.

1940 The line counter is incremented by one and if the number of valid bytes has not been exceeded, we jump to read more.

1945 If there is another sector it will become the current sector and the program jumps to read the current sector.

1980 End of the loop: all pages printed.

2380-2400 Close the open channels and end of program.

2420-2440 Subroutine: A character is fetched from the disk buffer and the valid ASCII value is determined. A CHR$ (0) is appended to each character before the ASCII value is determined. Otherwise the command ASC (A$) would produce an error with an empty string (A$="").

2460-2520 Subroutine: Here the CBM ASCII value is converted into a GEOS ASCII value. You may be able to use this routine in your own programs.

2540-2600 Subroutine: A GEOS ASCII value is converted to a CBM ASCII value.

2.3 The Text "CONVERTER"

You may have noticed that there are two different methods of word processing:

One is distinguished by its graphic capabilities, and is relatively slow because of the time-consuming screen operations.

The other type of text processing is quite fast, but doesn't offer the many possibilities of high-resolution graphics.

geoWrite belongs to the first category. The possibilities which it offers are truly enormous. Any segment of text can be printed in a completely different type style, for example. But anyone who has worked with programs of the caliber of TEXTOMAT from Abacus Software will be disappointed by the speed of the GEOS application. A writer used to such speed will not want to use the slow geoWrite for "normal" documents. Unfortunately, this program only accepts the files which it created itself. You can't, for example, write something with TEXTOMAT and then load it into geoWrite.

We want to present to you a tool which will allow you to write a document on any word processing program you like, then read this file (complete with a "real" icon) into geoWrite. Here you can use all of the formatting capabilities of geoWrite. geoWrite is fast enough to allow small changes to the text.

The tool, of course, is nothing more than a program. It is printed in the next section.

2.3.1 The listing of the converter

The program "Converter" printed below is written completely in BASIC. The following points should be noted when entering it:

-   The semicolon is the most important command in the program! You will not get a SYNTAX ERROR if one is missing.

-   The remarks (REMs) can be omitted.

-   The DATA lines with the same contents (32) can be duplicated by editing the line number. This makes the program easier to enter.

```
5 REM CONVERTER BY RUEDIGER KERKLOH
10 DIM Z$(255),K(23)
15 GOSUB 655:REM STANDARD-TABLE SETUP
20 GOSUB 610:REM HEADER LINE OUTPUT
25 PRINT" 1 = GENERATE GEOS-TEXT":PRINT
30 PRINT" 2 = CHANGE TABLE    ":PRINT
35 PRINT" 3 = LOAD    TABLE   ":PRINT
40 PRINT" 4 = SAVE    TABLE":PRINT
45 PRINT:PRINT" ENTER CHOICE!"
50 GETW$:W=VAL(W$):IF W<1 OR W>4 THEN 50
55 GOSUB 610:REM HEADER LINE OUTPUT
60 OPEN 1,8,15,"I:0":GOSUB 635
65 ON W GOSUB 85,465,520,565
70 CLOSE 1
75 GOTO 20:REM PROGRAM LOOP
80 :
85 PRINT" PLEASE INPUT NAME OF THE TEXT TO "
90 PRINT " BE CONVERTED !"
95 PRINT:INPUT AT$
100 GOSUB 610
105 PRINT " HOW MANY CHARACTERS PER PAGE 1-4580) ":PRINT
110 INPUT AZ
115 IF AZ<1 OR AZ>4580 THEN 100
120 :
125 REM PUT DATA INTO HELP FILE
130 OPEN 2,8,2,AT$+",R":GOSUB 635
135 M$="TEXT-DUMMY"
140 PRINT#1,"S:";M$+"*"
145 OPEN 3,8,3,M$+STR$(SZ+1)+",S,W":GOSUB 635
150 :
155 REM HEAD TEXT SETUP
160 FOR I=0 TO 23
165 PRINT#3,CHR$(K(I));
170 NEXT I
175 :
180 REM TEXT-DUMMYS SETUP
185 P=0:REM CHARACTER COUNT PER PAGE
190 GET#2,T$:IF T$="" THEN T$=CHR$(0)
195 S=ST AND 64:T=ASC(T$)
200 IF T<161 OR T>170 THEN 255
```

```
205 T1=T:T=69:REM LOWERCASE    "E"
210 IF T1=161 THEN T1=79:GOTO 250
215 IF T1=165 THEN T1=65:GOTO 250
220 IF T1=166 THEN T1=85:GOTO 250
230 IF T1=168 THEN T1=207:GOTO 250
235 IF T1=169 THEN T1=193:GOTO 250
240 IF T1=170 THEN T1=213:GOTO 250
245 IF T1=167 THEN T1=83:T=T1
250 PRINT#3,Z$(T1);:REM CODE
255 PRINT#3,Z$(T);:REM CODE
260 IF S THEN 270:REM END OF FILE
265 P=P+1:IF P<AZ THEN 190
270 PRINT#3,CHR$(0);:REM END OF TEXT
275 CLOSE 3:REM TEXT-DUMMY CLOSE
280 X$=M$+STR$(SZ+1):GOSUB 365:REM GET T & S
285 E$=C$:F$=D$:REM TEXT-DUMMY MARK
290 X$="DUMMY":GOSUB 365:REM T & S
295 OPEN 5,8,5,"#":GOSUB 635
300 PRINT#1,"U1";5;0;ASC(C$);ASC(D$)
305 PRINT#1,"B-P";5;SZ*2+2
310 PRINT#5,E$;F$;:REM ENTRIES
315 PRINT#1,"U2";5;0;ASC(C$);ASC(D$)
320 CLOSE5
325 SZ=SZ+1
330 GOSUB 610:REM TITLE LINE
335 PRINT" PAGE NUMBER: ";SZ
340 IF SZ=63 AND S=0 THEN PRINT:PRINT" TEXT TOO LONG !":GOTO 350
345 IF S=0 THEN 145
350 PRINT#1,"S:";M$;"*"
355 CLOSE 2:RETURN
360 :
365 REM GET T & S DATA
370 OPEN 4,8,4,"#":GOSUB 635
375 A$=CHR$(18):B$=CHR$(1):REM DIR
380 A=ASC(A$):B=ASC(B$)
385 PRINT#1,"U1";4;0;A;B:REM EMPTY
390 GET#4,A$,B$:REM NEXT SECTOR
395 FOR I=0 TO 7:REM NUMBER OF ENTRIES
400 : K=0
405 : PRINT#1,"B-P";4;I*32+3
410 : GET#4,C$,D$:REM T & S OF DATA
415 : IF D$="" THEN D$=CHR$(0)
420 : FOR J=1 TO LEN(X$)
425 :    GET#4,W$
430 :    IF W$=MID$(X$,J,1) THEN K=K+1
435 : NEXT J
440 : IF K=LEN(X$) THEN 450:REM FOUND
445 NEXT I:GOSUB 635:GOTO 380
450 CLOSE 4
455 RETURN
460 :
465 REM CHANGE TABLE
470 INPUT " CHARACTER CODE NUMBER:    ";Q
475 IF Q<0 OR Q>255 THEN 470
480 PRINT:PRINT" NORMAL GEOS-CODE:";ASC(Z$(Q))
485 NC=ASC(Z$(Q))
```

```
490 PRINT:PRINT " NEW GEOS-CODE:":PRINT
495 INPUT" (RETURN=UPDATE    , 0=QUIT)";NC
500 IF NC<1 OR NC>255 THEN  510
505 Z$(Q)=CHR$(NC):GOSUB 610:GOTO 465
510 RETURN
515 :
520 REM LOAD TABLE
525 INPUT" NAME OF TABLE";NT$
530 OPEN 2,8,2,NT$+",S,R":GOSUB 635
535 FOR I=0 TO 255
540 GET#2,Z$(I)
545 NEXT I
550 CLOSE 2
555 RETURN
560 :
565 REM SAVE TABLE
570 INPUT " NAME FOR THE NEW TABLE";NT$
575 OPEN 2,8,2,NT$+",S,W":GOSUB 635
580 FOR I=0 TO 255
585 PRINT#2,Z$(I);
590 NEXT I
595 CLOSE 2
600 RETURN
605 :
610 PRINT CHR$(147):REM CLR HOME
615 PRINT TAB(9);"*** TEXT-CONVERTER ***"
620 PRINT:PRINT
625 RETURN
630 :
635 INPUT#1,F,FT$,FT,FS
640 IF F<>0 THEN PRINT:PRINT" ";FT$:STOP
645 RETURN
650 :
655 RESTORE
660 FOR I=0 TO 255
665 READ Z:Z$(I)=CHR$(Z)
670 NEXT I
675 READ Z:IF Z=-1 THEN 685
680 PRINT:PRINT" ERROR IN DATA!":STOP
685 FOR J=0 TO 23
690 READ K(J)
695 NEXT J
700 RETURN
705 :
710 REM DECODER-TABLE
715 DATA 32,32,32,32,32,32,32,32,32,32
720 DATA 32,32,32,13,32,32,32,32,32,32
725 DATA 32,32,32,32,32,32,32,32,32,32
730 DATA 32,32,32,33,34,35,36,37,38,39
735 DATA 40,41,42,43,44,45,46,47,48,49
740 DATA 50,51,52,53,54,55,56,57,58,59
745 DATA 60,61,62,63,64,97,98,99,100,101
750 DATA 102,103,104,105,106,107,108,109,110,111
755 DATA 112,113,114,115,116,117,118,119,120,121
760 DATA 122,91,92,93,94,13,96,65,66,67
765 DATA 68,69,70,71,72,73,74,75,76,77
```

```
770 DATA 78,79,80,81,82,83,84,85,86,87
775 DATA 88,89,90,123,124,125,126,127,32,32
780 DATA 32,32,32,32,32,32,32,32,32,32
785 DATA 32,32,32,32,32,32,32,32,32,32
790 DATA 32,32,32,32,32,32,32,32,32,32
795 DATA 32,32,32,32,32,32,32,32,32,32
800 DATA 32,32,32,32,32,32,32,32,32,32
805 DATA 32,32,32,32,32,32,32,32,32,32
810 DATA 32,32,32,65,66,67,68,69,70,71
815 DATA 72,73,74,75,76,77,78,79,80,81
820 DATA 82,83,84,85,86,87,88,89,90,123
825 DATA 124,125,126,127,32,32,32,32,32,32
830 DATA 32,32,32,32,32,32,32,32,32,32
835 DATA 32,32,32,32,32,32,32,32,32,32
840 DATA 32,32,32,32,32,32,-1
845 :
850 REM TEXT HEADER DATA     (TABS ETC.)
855 DATA 24,0,48,1,64,0,144,0,224,0
860 DATA 48,1,48,1,48,1,48,1,48,1
865 DATA 23,9,0,0
```

2.3.2 Using the converter

Save the program after you type it in so you can use it later.

SAVE "CONVERTER", 8

Before we try to convert an existing document, we have to make some preparations. First create a disk which contains only geoWrite. geoWrite must then be loaded from this disk. Select the option Create New Document from the menu that appears. The new document must be given the name "DUMMY" because our program will look for this later. Make sure the name is entered in uppercase.

As soon as the white writing surface appears the program geoWrite can be exited again with Quit.

The reason we do this is to create a file with an appropriate icon on the disk. If we had done this in BASIC, our program would have been much longer.

Next, the document which is to be converted to geoWrite format must be copied to the disk. This can be done either with GEOS or with any other copy process.

After this is done we can finally load our CONVERTER. It is set up for documents created with TEXTOMAT. Naturally many other text formats can be converted as well. Below we describe the necessary changes.

The program is started with RUN. After a few seconds the following menu appears:

```
1 = CREATE GEOS DOCUMENT
2 = CHANGE TABLE
3 = LOAD TABLE
4 = SAVE TABLE
```

At the moment we are just interested in the first option. If we select it, we will be asked to enter the name of the file to be converted. Enter this exactly as it would appear in directory listing. The asterisk can be used in the filename.

You will then be asked how many characters are to appear on a page. Normally 1500 is a good value. For longer documents you may have to change this so that the maximum number of pages which GEOS allows (63) is not exceeded.

The necessary number of pages can be calculated as follows:

P = (254 / number of characters per page) * number of blocks in the document as shown in the directory.

The result is rounded to whole pages. Here is an example:

A document occupies a total of 42 blocks according to the directory. There are to be 165 characters on a page. This results in the following number of pages needed:

$$P = (254/165)*42$$

which we round to 65 pages. We don't have this many available, however, so after a half hour of hard work our program would simply announce "Text too long."

To fix this we could just raise the number of characters per page to 170, for example. It's a good idea to calculate the number of pages beforehand so you don't run into this problem.

When you have made the last input the process starts running: The file is converted and the number of completely written pages is displayed on the screen. If errors occur in connection with the disk drive while the program is running, it will stop and print an appropriate error message.

Unfortunately, the conversion process requires a fair amount of time since both the C64 and the serial bus are rather slow. The end of the procedure is indicated by the appearance of the main menu again.

There is one little matter yet to be taken care of under GEOS, but it must be done: A validate!

If this little option in the disk menu is not selected, the whole conversion process may be for naught. The reason for this behavior should be clear from the program documentation.

Now we can finally load the document as a geoWrite file. Naturally, the name can also be changed under GEOS from DUMMY to a more appropriate name.

We should explain how to adapt our program for use with other word processing programs.

It seems as if every computer manufacturer believes it has to develop its own "standardized" code for encoding characters. Commodore really screwed things up by switching the upper and lower case letters. The use of codes greater than 127 is also confusing. Normally the seventh bit is used only to ensure correct data transfer (parity bit).

To its credit, GEOS adheres to the real ASCII code. The fact that this creates problems with "old" documents, however, is obvious.

In many cases the default conversion which our program performs will be successful. The only condition is that the file must be stored on the disk sequentially (as a SEQ or PRG file).

In cases where incorrect characters appear in geoWrite, it will be necessary to make changes to the conversion table. We have installed help functions in our program for this purpose. They allow the encoding to be changed, provided you know the scheme used by the word processor.

For example, if a program uses the value 97 to represent the letter "A", we simply assign the real ASCII value 65 to this 97.

The C64 manual contains a list of the real ASCII values. For GEOS only the values from 32 to 127 are of interest in this table. In addition, the lowercase alphabet must be placed behind the inverse characters at 97.

These changes are made with the second program option `Change Table`. You will first be asked for the "source code number." This is the decimal value of the incorrect character in your file. If your file uses the value 98 for a "B", for example, the number 98 must be entered here.

The program will tell you what real ASCII code is assigned this value at the moment. If a 66 is printed here, everything's in order, because this corresponds to a "B" (press RETURN). When you have corrected all of the incorrect characters, a zero ends the recoding.

The entire table can be saved so you don't have to go through this process every time you want to covert a file. The fourth menu option is provided for this purpose (`Save Table`). The table must be given a name. The table can be loaded again later with the third menu option `Load Table`.

2.3.3 Documentation for the converter

For those who are interested, we will explain the operation of the converter here. To understand it, you must know how the disk drive channels work and the structure of VLIR files.

In order to understand the individual actions of the converter more easily, we will first go into the principle involved:

A text file under geoWrite has the VLIR structure. There is a link block on the disk which points to the various tracks and sectors of the document. Each page in geoWrite has one entry in this link block.

But since the normal DOS does not recognize VLIR files, we have to use a bit of a detour to create such a structure. The DUMMY file is used for this, which must be recreated before each conversion.

The tracks and sector of each converted page must be placed in the now-present link block. This is done as follows:

For each page we create a normal sequential file which we call "TEXT-DUMMY". The filename also contains the corresponding page number. We can easily place the text page in this file as we convert it. The actual conversion is done with the array Z$. The program then gets the track and sector of this file from the directory and enters the values in the VLIR link block.

This process is repeated until there is no more text. At the end, all TEXT-DUMMY files are deleted. But since this releases the blocks occupied by these files in the BAM, and since they aren't really free, we must perform a validate in GEOS. A write operation preceding the validate would certainly delete part of the new document.

Here is a description of the individual program steps:

10-15	Initialize arrays for the conversion table and the header of the GEOS document. The header contains specifications about the margins and tabs (see Section 3.5.7).
25-65	Create menu, evaluate input, and jump; first open command channel and initialize disk.
70-75	Close command channel and program loop.
80-120	Enter data for menu 1.
130	The old text file is opened.

140   Any existing "TEXT-DUMMY" is deleted from the disk.

145   A "TEXT-DUMMY SZ" is opened. "SZ" stands for the current page
             number in the program. At the beginning, SZ=0.

160-170   Specifications for the margins and tabs must be entered on each
             page.

185-275   The actual conversion is performed in these lines. The data from
             channel 2 is converted and sent to channel 3 TEXT-DUMMY.

             P is the character counter. It is set to zero at each new page. S
             contains the disk status and is 1 when the end of the file is
             reached.

280-285   Here we branch to the subroutine which passes the track and
             sector of the first data block of the file X$ in the variable C$ and
             D$. In this case T and S of the current TEXT-DUMMY are fetched
             and placed in E$ and F$.

290   We jump to the same subroutine with the "VLIR-Dummy". The T
             and S of the link block are fetched in C$ and D$.

295-320   The link block is read into the disk buffer and the T and S of the
             TEXT-DUMMY are entered in the appropriate locations. The block
             is then written back to the disk.

325-345   Evaluate the page number SZ. If the source text is done (S=1), the
             conversion is over.

350-360   The TEXT-DUMMY files are deleted again. Return to the main
             menu.

366-455   Subroutine:

             The track and sector of the first data block of the file in X$ are read
             from the directory and stored in C$ and D$, respectively. One
             sector of the directory is read into the drive buffer and the desired
             filename is compared with all eight entries in this sector. If all of
             the characters match, the file is found.

             The desired data is in front of the filename, and is therefore read
             before the name comparison (line 410).

465-510   This is the modification program for the conversion table.

520-550   The conversion table data is read here.

565-600   Save routine for the conversion table.

610-625   The program message is printed.

635-645   If a disk error occurs, it will be printed here and the program will
             stop.

655-670   The conversion table is set up.

685-700   The data for the text header is fetched.

710-840   Decoder table "TEXTOMAT"

850-865   Data for the text header (two values fetched at a time):

             Left and right margin, eight tabs, specifications for the type style
             used.

Chapter 3

Changing GEOS

3.1 Problems with modifying GEOS

Nobody's perfect!

This applies especially to the developers of GEOS. There are no really serious errors in the program package, but we found a number of things while working with the user interface that we felt could have been done better. One of these is sensible margin settings when starting geoWrite.

Some of our suggestions fall under the term "cosmetic." For example, it bothers us that the notes created by the note pad cannot be distinguished from the main program except by the filename, because they have the same icon.

These are only some of the points in our long list to which we have dedicated this chapter. Naturally, we don't just suggest possible changes, we also give you the opportunity and the program code to actually implement each improvement.

In many cases it is enough to change one or two memory locations in GEOS to achieve very useful effects. Here lies the problem: Unlike the old C64 operating system, we can't just POKE into memory and then save a modified program where possible (such as geoWrite).

You may get the idea to just load the program to be changed with the C64 operating system with LOAD "name", 8, enter the appropriate POKEs, and then save the whole thing again with SAVE "name", 8.

This would be possible if GEOS didn't use a completely different (and better) file format. The error message that results from such an attempt (FILE TYPE MISMATCH) is a result of this incompatibility. Even if you try to change the file type, there are additional difficulties because of the different character coding that GEOS uses.

We have written a program to let us make purposeful changes to GEOS files. It consists of a BASIC segment and a machine language routine stored in DATA statements.

The function is based on the following idea: Since it is difficult to load a GEOS file into the computer to modify it, we will go the other way around: A small machine language program will be copied into the disk drive by the BASIC portion and will poke the bytes to be changed directly on the disk. The new values will be sent to the program as mail. This method is also faster because the transfer of data over the normally very slow serial bus is kept to a minimum.

You should enter the following listing of our modifier very carefully because errors in input will not only cause the program not to work, they may also wreak havoc on your disks.

Naturally, the checksum provided offers a fairly good protection against typing errors. In spite of this, this program is a very powerful tool, so you should be careful. **All modifications should be made only to copies and never on the original disk.**

3.2 The Modifier

```
10 REM *    GEOS-MODIFIER          *
15 REM * AUTHOR: RUEDIGER KERKLOH *
20 REM
25 POKE 53280,0:POKE 53281,0
30 PRINT CHR$(147);CHR$(14);CHR$(8);CHR$(5)
35 PRINT TAB(8);"*** GEOS MODIFIER ***":PRINT
40 PRINT"PLEASE INSERTA BACKUP COPY THAT HAS"
45 PRINT"THE FILES TO BE CHANGED !"
50 GOSUB 440
55 :
60 REM * GET TRACK AND SECTOR *
65 PRINT CHR$(147):PRINT:PRINT
70 INPUT" FILE NAME";X$
75 GOSUB 460:F$=Y$
80 FOR I=1 TO 16-LEN(Y$)
85 :    F$=F$+CHR$(160)
90 NEXT I
95 OPEN 1,8,15,"I:0"
100 OPEN 2,8,2,"#0"
105 PRINT:PRINT" SEARCHING ...":PRINT
110 T=18:S=1
115 PRINT#1,"U1";2;0;T;S
120 GET#2,T$,S$
125 T=ASC(T$+CHR$(0)):S=ASC(S$)
130 FOR I=0 TO 7
135 :    PRINT#1,"B-P";2;I*32+2
140 :    GET#2,Z$,P$,Q$
145 :    IF Z$="" THEN 195
150 :    PRINT" ";
155 :    FOR J=1 TO 16
160 :        GET#2,X$:GOSUB 460
165 :        M$=MID$(F$,J,1)
170 :        IF X$=M$ THEN K=K+1
175 :        PRINT Y$;
180 :    NEXT J
185 :    IF K=16 THEN 220
190 :    K=0:PRINT
195 NEXT I
200 IF T<>0 THEN 115
205 CLOSE 1
210 PRINT:PRINT" FILE DOES NOT EXIST !"
215 END
220 PRINT:PRINT CHR$(145);">"
225 GET#2,IT$,IS$,SC$
230 PRINT:PRINT" FILE STRUCTURE: ";
235 IF SC$="" THEN SC$=CHR$(127):PRINT"SEQUENTIAL":GOTO 245
240 SC$=CHR$(0):PRINT"VLIR"
245 PRINT:PRINT
250 PRINT"  1  = REPLACE TEXT ":PRINT
255 PRINT"  2  = REPLACE BYTES "
260 PRINT:PRINT:PRINT"  ENTER CHOICE "
265 GET W$:IF W$<>"1" AND W$<>"2" THEN 265
270 ON VAL(W$) GOSUB 280,320
```

```
275 CLOSE1:RUN
280 PRINT CHR$(147):PRINT
285 PRINT"  INPUT SEARCH TEXT":PRINT
290 INPUT X$:GOSUB 460:AT$=Y$
295 PRINT:PRINT
300 PRINT"  INPUT NEW TEXT":X$="":PRINT
305 INPUT X$:X$=X$+CHR$(0):GOSUB460:NT$=Y$
310 GOTO 345
315 :
320 PRINT CHR$(147):PRINT
325 PRINT" INPUT SEARCH BYTES:":PRINT
330 GOSUB 520:AT$=Y$:PRINT:PRINT
335 PRINT" INPUTNEW BYTES:":PRINT
340 GOSUB 520:NT$=Y$
345 IF LEN(AT$)+LEN(NT$)<57 THEN 355
350 PRINT:PRINT" INPUT TOO LONG !":GOSUB 440:GOTO 245
355 REM * TRANSFER DATA TO DISKETTE *
360 PRINT#1,"B-P";2;0
365 FOR I=1 TO 196
370 :    READ D
375 :    PRINT#2,CHR$(D);:W=W+D
380 NEXT I
385 IF W<>22349 THEN PRINT:PRINT"ERROR IN DATA !":END
390 PRINT#2,SC$;   :REM FILE STRUCTURE
395 PRINT#2,P$;Q$; :REM START SECTOR
400 PRINT#2,CHR$(LEN(AT$));
405 PRINT#2,CHR$(LEN(NT$));
410 PRINT#2,AT$;NT$;
415 PRINT#1,"M-E";CHR$(0);CHR$(3)
420 INPUT#1,D,A$,A,A:CLOSE1:PRINT
425 IF D=0 THEN PRINT"DATA FOUND AND CHANGED !":GOTO 440
430 PRINT"DATA NOT FOUND !"
435 :
440 PRINT:PRINT" <RETURN>"
445 GETW$:IFW$=""THEN445
450 RETURN
455 :
460 Y$=""
465 FOR A=1 TO LEN(X$)
470 :    B=ASC(MID$(X$,A,1))
475 :    IF B<192 THEN 485
480 :    B=B-96
485 :    IF B<65 THEN 500
490 :    IFBAND32THENB=BAND223:GOTO500
495 :    B=B OR 32
500 :    Y$=Y$+CHR$(B)
505 NEXT A
510 RETURN
515 :
520 Y$="":I=1
525 :  PRINT" BYTE NR.";I;":";:INPUT D$
530 :  IF VAL(D$)>255 THEN 525
535 :  IF D$="" THEN 550
540 :  Y$=Y$+CHR$(VAL(D$)):D$=""
545 :  I=I+1:GOTO525
550 RETURN
```

```
555 REM
560 DATA172,197,3,173,198,3,174,196,3,208,34,132,8,133,9,169,128,133
565 DATA1,165 1,48,252,238,196,3,173,196,3,16,3,76,94,225,10,170,189
570 DATA0,4,240,238 168,189,1,4,133,11,132,10,169,128,133,2,165,2
575 DATA48,252,164,16,162,2 189,0,5,217,201,3,208,8,200,204,199,3,144
580 DATA6,176,62,160,0,132,16,232 208,233,152,240,37,132,16,189
585 DATA0,5,157,0,6,232,208,247,165,10,133 12,165,11,133,13,136,152
590 DATA73,255,168,174,199,3,189,201,3,153,0,6,232 200,208,246,173,0,5
595 DATA240,150,133,10,173,1,5,133,11,24,144,166,232 138,56
600 DATA237,199,3,24,101,16,170,173,199,3,24,101,16,168,185,201,3
602 DATA 157,0,5,200,232,206,200,3,173,200,3,56,229,16
605 DATA208,237,162,144,134 2,165,2,48,252,165,16,240,6,134,3,165,3
610 DATA48,252,76,188,230
```

3.3 Using the Modifier

After you have entered our program you should first save it to disk with SAVE "MODIFIER", 8. For the following work, prepare a copy of the GEOS applications disk with the DISK COPY program so that you can still access the original program if there are any errors. **All modifications should be made only to copies and never on the original disk.**

When we talk about the GEOS disk, we mean only this copy and never the original.

Now you can load the modifier again (LOAD "MODIFIER", 8). After you start the program it will ask you to insert a copy of GEOS and press RETURN. The program will then ask for the name of the file to be changed. For our first attempt we will enter "Hello". The program will search for this file on the disk by reading in every existing name and comparing it with "Hello". At the same time, the names read in will be displayed on the screen. In this case the list will consist of all the files, since there is no "Hello" program on the GEOS disk.

Note how "ROMA" is printed in this list, because we will enter this name in our second attempt, and the program is sensitive to upper- and lower-case lettering. On the second pass, enter the name of this font file. When it is found by the program, the following menu appears:

        1 = Replace text
        2 = Replace bytes

You can now decide whether you want to replace certain text strings or certain sequences of bytes in the selected file (called "ROMA" in this case). In both cases the program will ask for the data to be replaced and then the data to be inserted instead.

We don't tell the program the address of a given memory location in which to write a new value, we give a kind of search key which occurs only once in the program. If this search key is found, the new values will be entered at this location. You must make sure that a given search key does not appear more than once in a program (at minimum not before the location you want to replace) or the new data will be put in the wrong place. In this chapter we have made sure that the strings or byte sequences we present occur only once.

For the first attempt, enter a 1 for "Replace text". You will then be asked to enter the search key, the string to be replaced. For example, have the program search for "Casablanca". You can replace the search text with itself since the word "Casablanca" is not in the font file, so nothing will be replaced. The "MODIFIER" program will respond "Data not found" in such a case.

If you made an error in entering the DATA lines, the search process will not be started and a corresponding error message will be printed. Our program calculates a checksum of the DATA before the search routine is activated in the disk drive and compares this with the value it is supposed to be. If there is a discrepancy, you have to check the DATA again and compare them with the printed listing.

When you get the message "Data not found!", try the menu option "Replace bytes" the second time around.

You will again be asked for the search key, but in this case it consists of individual bytes which must be entered one after the other. Enter a five for the first byte. If you just press RETURN for the second byte, the search key will be terminated and you will be asked to enter the bytes to replace the old values.

Just enter five again for the first byte. If you simply press RETURN in answer to the prompt for the second byte, the search process will start, but this time it will be successful. The message "Data found and changed" will confirm this. In a program which is several kilobytes long a five should occur at least once, and based on our input it will be replaced by another five. We haven't changed the file at all.

We have now demonstrated the use of the program. We hope that you can appreciate how powerful this tool is. It goes without saying that it can also destroy programs if given incorrect input. As long as the program in the disk drive has not been started, the program can be stopped with RUN/STOP + RESTORE and started again.

71

3.3.1 How it works

This section is directed to those readers who want to know more about the operation of our modifier. If this doesn't interest you, you can skip this section.

First we will explain the BASIC portion, and to understand the program you should have some programming experience in BASIC. The second section follows the documentation for the search routine that runs in the disk drive. Since it is written completely in machine language, knowledge of this language is necessary in order to be able to follow the explanations.

3.3.2 Documentation of the BASIC portion

25	Set screen color to zero (black).
30	Clear screen, enable upper case, white characters.
50	Wait for keyboard input.
65	Clear the screen.
75	The subroutine at line 460 converts characters of a string to the format used by GEOS. The result is passed in Y$.
80-90	The name entered will be padded with "shift spaces" (CHR$ (160)) to the maximum length of 16 characters.
95	Open command channel to the drive and initialize the disk.
100	Open transfer channel for data. All data which is sent over this channel refers to buffer zero in the disk drive (at $0300 in RAM).
110	Set track and sector to the first directory entry.
115	Execute "block read" command. The data will end up in buffer zero (T=track, S=sector).
120	Save track and sector of the next block. The data for this is always in the first two locations of the block.
125	Since converting a track 0 to integer format would lead to an ILLEGAL QUANTITY ERROR, a CHR$ (0) is appended here.

130-195	The eight filenames of the block read are compared with the filename entered. If the name is not present and there is another block (T not zero), it will be read and the process is started over.
135	The read pointer for buffer zero is set to the file type of the next filename.
140	The file type, track, and sector of the data are read.
145	If the file type is equal to zero (=""), the next entry is fetched.
155-180	The sixteen characters of the filename are read and compared with the filename.
185	K is the counter for the matches. If it contains a 16, the file is found.
190	Otherwise the counter is reset and we keep searching.
200	If another directory sector is present, it will be read.
205-215	File not found.
225	In the GEOS format there is additional information behind the filename—the track and sector of the INFO block (IT$ and IS$) and the file structure (SC$). The INFO block contains information about the icon and the entries which appear in the GEOS "File Info". IT$ and IS$ are not used.
235	A zero in the file structure field stands for "SEQUENTIAL."
240	File structure=1: VLIR format. The variable SC$ will be used later.
245-275	Menu selection.
280-290	Input the old text. It is converted to GEOS format and placed in AT$.
300-310	The same is done with the new text, after it is terminated with a zero. The result is placed in NT$.
320-330	Input of the old bytes to be searched for.
335-340	Input of new bytes.

345-350	Since there is only room for 56 bytes behind the program in the disk buffer, we have a check here.
360-380	The machine language program is copied to buffer zero ($0300). A checksum is calculated at the same time.
385	Error handling, if the checksum does not match.
390-410	The data which the program needs for the search is transmitted. These include the file structure and the start sector of the data, as well as the lengths of the old and new strings, and the old and new strings themselves.
415	The search program is started.
420-450	The message is read from the error channel and evaluated. The search program prepares this message. It is either OK or RECORD NOT PRESENT. Only the corresponding error number is evaluated.
460-510	Subroutine which converts a C64 string to GEOS format.
520-550	This subroutine accepts the input of bytes via the keyboard in a text string.
555-605	Here is the actual search program in the form of DATA statements.

3.3.3 Documentation for the search program

The program was created with a screen-oriented editor and therefore does not have line numbers. There are, however, labels in the program (such as START) to which we will make reference.

```
              * = $0300     ;Start address in disk drive RAM

              JOB1 = $01    ;Job for buffer 1
              JOB2 = $02    ;Job for buffer 2
              JOB3 = $03    ;Job for buffer 3
              JOB4 = $04    ;Job for buffer 4

              T1   = $08    ;Track  for buffer 1
              S1   = $09    ;Sector for buffer 1
              T2   = $0A    ;Track  for buffer 2
              S2   = $0B    ;Sector for buffer 2
              T3   = $0C    ;Track  for buffer 3
              S4   = $0D    ;Sector for buffer 3
              T4   = $0E    ;Track  for buffer 4
              S4   = $0F    ;Sector for buffer 4

              FLAG = $10    ;(s. Text)

              ;* READ BLOCK IN BUFFER 1 *

      START   LDY TRACK     ;SEND POINTER TO BLOCK
              LDA Sector    ;FROM BASIC

              LDX COUNT     ;STRUCTURE?
              BNE GET4      ;SEQUENTIAL

              STY T1        ;ELSE SET POINTER TO
              STA S1        ;VLIR LINKER BLOCK

              LDA #$80      ;JOB: READ SECTOR
              STA JOB1      ;     IN BUFFER 1

      WAIT1   LDA JOB1      ;WAIT UNTIL JOB
              BMI WAIT1     ;IS DONE

              ;* GET NEXT ENTRY
      GET1    INC COUNT
              LDA COUNT     ;1-127 ENTRIES
              BPL GET2      ;NO END

              JMP $E15E     ;"RECORD NOT PRESENT"
                           ;FROM ERROR CHANNEL

      GET2    ASL A         ;TIMES TWO (T & S)
              TAX
              LDA $0400,X   ;START: NEW FILE
              BEQ GET1      ;NO ENTRY (T=0!)
              TAY
              LDA $0401,X
```

```
GET4    STA S2        ;T & S FOR BUFFER 2
        STY T2

GET3    LDA #$80      ;READ BLOCK
        STA JOB2

WAIT2   LDA JOB2      ;WAIT UNTIL JOB
        BMI WAIT2     ;IS DONE

        ;* LOOK FOR DATA IN BUFFER 2  *

        LDY FLAG      ;POINTER: OLD TEXT
        LDX #2        ;POINTER IN BUFFER

SRCH2   LDA $0500,X ; BLOCK READ
        CMP TEXT,Y    ;OLD TEXT
        BNE SRCH1     ;UNEQUAL

        INY           ;INCREMENT
        CPY LENA      ;ALL COMPARED?
        BCC SRCH3     ;NO
        BCS SRCH4     ;YES

SRCH1   LDY #0        ;CLEAR FLAG
        STY FLAG      ;OVERLAP FLAG
SRCH3   INX           ;BUFFER POINTER
        BNE SRCH2     ;NOT DONE YET

        ;* READ BUFFER  *
        TYA           ;OVERFLOW?
        BEQ SRCH5     ;NO--CONTINUE

        ;* EVENTUAL OVERLAPPING *

        STY FLAG      ;SET

GRENZ1  LDA $0500,X ;BUFFER 2 -> 3
        STA $0600,X
        INX
        BNE GRENZ1

        LDA T2        ;BUFFER 3 OF T & S
        STA T3
;SET UP
        LDA S2
        STA S3

        DEY
        TYA           ;CONVERT AMOUNT
        EOR #$FF      ;IN POINTER
        TAY           ;(E.G. $01-> $FF)
                      ;(OR $02-> $FE)

        LDX LENA      ;ENTER NEW TEXT
SRCH6   LDA TEXT,X  ;IN BUFFER 3
        STA $0600,Y
```

```
                INX
                INY
                BNE SRCH6

        SRCH5   LDA $0500   ;NEXT TRACK
                BEQ GET1    ;NOT AVAILABLE

                STA T2
                LDA $0501   ;NEXT SECTOR
                STA S2
                CLC
        ;UNCONDITIONAL JUMP
                BCC GET3

        SRCH4   ;* VALUE FOUND *
                ;
                ;X-REG: TO LAST OLD BYTE

                INX         ;POINTER TO STARTING
                TXA         ;POSITION
                SEC
                SBC LENA
                CLC         ;PLUS THE BYTES
                ADC FLAG    ;ALREADY ENTERED
                TAX

                LDA LENA    ;POINT TO CURRENT
                CLC         ;NEW BYTE
                ADC FLAG
                TAY

        SET1    LDA TEXT,Y  ;ENTER
                STA $0500,X ;NEW TEXT
                INY
                INX
                DEC LENB
                LDA LENB
                SEC
                SBC FLAG
                BNE SET1

                LDX #$90    ;WRITE SECTOR
                STX JOB2

        WAIT3   LDA JOB2    ;WAIT UNTIL JOB
                BMI WAIT3   ;IS DONE

                LDA FLAG    ;OVERLAP?
                BEQ SET2    ;NO

                STX JOB3    ;ELSE WRITE
                            ;BUFFER 3
        WAIT4   LDA JOB3
                BMI WAIT4

        SET2    JMP $E6BC   ;PUT "O.K." IN ERROR CHANNEL
```

```
                ;DATA RANGE TO BE SET ASIDE FROM BASIC


                COUNT  * = * + 1    ;LINKER POINTER
                                    ;OR VLIR FLAG

                TRACK  * = * + 1
                SECTOR* = * + 1

                LENA   * = * + 1    ;LENGTH OLD TEXT
                LENB   * = * + 1    ;LENGTH NEW TEXT

                TEXT   * = * + 56   ;OLD & NEW TEXT

                    .END
```

Memory usage:

The buffers are used as follows in the disk drive RAM:

Buffer 0 (at $0300)

Contains the search program from the DATA with the variables passed by the BASIC program.

Buffer 1 (at $0400)

Contains the first block for VLIR files (link block). This is not chained to other blocks, as in sequential files. It contains a maximum of 127 pointers to track and sector of data, which are then chained like sequential files. More about the VLIR format is found in Section 2.1. This buffer is not used if the file is in sequential format.

Buffer 2 (at $0500)

The blocks of the program to be searched are read into this buffer and then compared byte by byte with the values passed by the BASIC program (TEXTA).

Buffer 3 (at $0600)

Temporary storage for a program block. If the data to be searched for is right at the end of the program section in buffer 2, but not all of the bytes have been compared, then we don't know for sure if the block read is the right one or not. The desired changes are made in buffer 2, but the block is not saved back to the disk yet. Instead, buffer 2 is copied to buffer 3 and the next sector of the program is read into buffer 2. Only if this verifies that the location was correct will the new bytes be entered here and the two buffers (2 and 3) written back.

The memory location FLAG contains the number of bytes which have already been replaced in buffer 3. A zero signals that there was no overlap.

Program flow:

We have to distinguish between two operating modes of the search program. In the first operating mode the file to be searched is sequential, and in the second it has VLIR structure. The information about which file structure is being used is passed by the BASIC program in the variable SC$. Here 0 means VLIR and 127 means sequential. The value will be received and stored in the variable COUNT.

If the file structure is sequential, a section of the program will be skipped. This section, which extends to GET4, serves to load the link block in buffer 1 ($0400) for VLIR files. The track and sector of the next entry are fetched from this link block.

At GET4 the Y register and the accumulator holds the track and sector of the first data block. For sequential files these data come from the BASIC program, while for VLIR files they are the entries in the link block. This data block will be read into buffer 2($0500) at GET3. The sector data is then compared with the old data (TEXTA). The function of FLAG was mentioned above.

When the program block has been searched, the program tests to see if there was an overlap. If not, the next program block is read and tested (SRCH5). If there are no more blocks, it will try to get another record from the link block at GET1. Since this exists only for VLIR files, the test is designed such that a "RECORD NOT PRESENT" will be outputted immediately for sequential files. This is because the BASIC program passes a 127 in the memory location COUNT instead of the zero for VLIR files. Incrementing this to 128 causes the test to fail and the search process ends.

If there is an overlap, buffer 2 is copied to buffer 3 at GRENZ1. In addition, the track and sector of this block are also stored so that we can write buffer back to the disk later if necessary.

Since the Y register contains the number of matches, we can easily turn it into a pointer which points to the bytes in question in buffer 3. These bytes are then replaced by the new ones. The necessary offset to the new TEXT is created from LENA.

If all of the old bytes are found by the comparison process (SRCH2), we branch to SRCH4. Here the rest of the new bytes are entered. If there was no overlap, this will be all of the new bytes. Buffer 2 will then be written back with the modified data. If there was an overlap, buffer 3 must also be written back. The program recognizes this from the contents of memory location COUNT. The OK message is then placed on the error channel.

3.4 Uses for the modifier

Now there's nothing more keeping us from using our modifier. But before we get started, we should first explain the "rules of the game."

Each of the modification suggestions we make refers to a very specific file on the GEOS diskette (**the copy!**). Naturally, we will tell you the name of this file each time and write it so that our program can also find it in (upper/lowercase).

You will be able to tell from the context whether text or individual bytes will be replaced. You must select the correct menu option in modifier based on this. Of course, you can use "Replace bytes" for all of the work, but you will then have to convert any text strings to the corresponding ASCII values. If you replace a string in this manner remember to end it with a zero. A zero in GEOS is the universal termination character. In "Replace text" it is automatically added to the new text. In addition, you should note that a new text can be shorter than the original, but it may never by longer or important data in the program may be overwritten.

We want to mention one property of the decimal search keys we use: The search bytes we specify are always the values which stand directly before the actual values to be replaced. Therefore they must also be placed in the list of new bytes each time.

The advantage of this method is that the modifier can find the location every time, even if it is been changed before.

3.4.1 Modifying the error messages

The modifier program can be used to customize the deskTop error messages. Start the modifier program and insert the copy of GEOS. **All modifications should be made only to copies and never on the original disk.** Enter the name deskTop for the file. Select 1. Alter string from the menu and enter the following as the search string:

```
Would you like it converted?
```

Enter the following as the replacement text:

```
Convert this C-64 diskette?
```

The program will run and add your customized error message. Notice that the replacement string must be shorter than the original screen, if it is longer it may overwrite some important information on the GEOS diskette. Have fun making your own custom version of GEOS.

The original deskTop was loaded off your master diskette and the error messages in memory have not been changed. The changed deskTop can be loaded by starting an application program (geoWrite or geoPaint) and then quitting the application program with the disk containing the modified deskTop in the disk drive. The modified deskTop will be loaded and you will have a customized version of GEOS.

Here is a list of error messages that may be changed.

```
1          Operation cancelled due to
2          disk error
3          This is a NON-GEOS disk.
4          K bytes used
5          K bytes free
6          Please insert disk
7          A maximum of eight files may
8          be placed on the border
9          This file can't be printed
10         from the deskTop
11         This file is write protected
12         and can't be deleted
13         Please insert a disk which
14         already exists
15         Please insert destination
16         Put disk to format in drive: A
17         and enter a name for it
18         Replace the contents of
```

```
19          with the contents of
20          Plug in & turn ON new drive.
21          This operation may only be
22          performed on files from the
23          current disk
24          This operation may not be
25          performed on System files
26          performed on Non-Geos disks
27          Please enter new filename:
28          Please enter new disk name:
29          deskTop not on disk
30          File from other disk
31          Write Protect
32          Disk too full
33          Directory full
34          File not found
35          Device not found
36          Missing or unformatted Disk
37          Write protect tab on disk
38          Printer
39          printer
40          This file can't be opened
41          In drive.
42          Canceled
43          DRIVE A
44          DRIVE B
```

Here are a few examples of custom GEOS error messages:

Figure 13-a: DRIVE A/ DRIVE B changed to UNIT 8/ UNIT 9
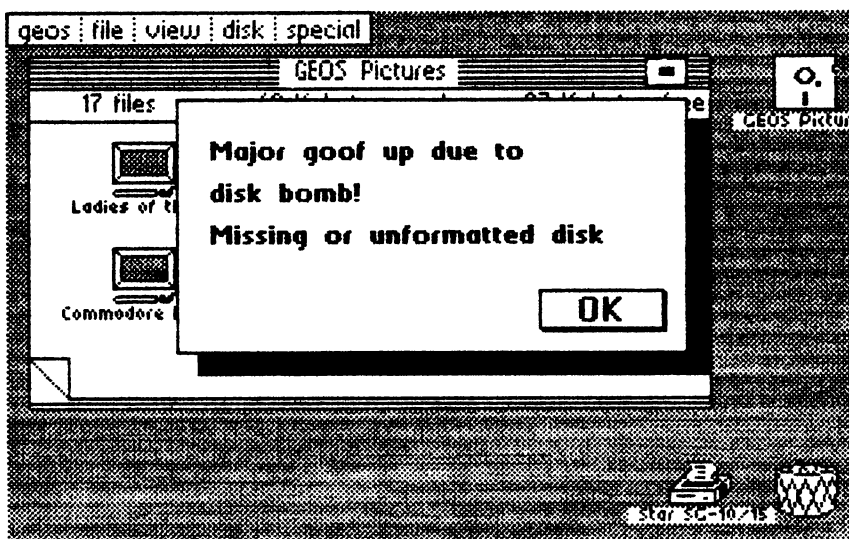
Figure 13-b: Modified disk error message example
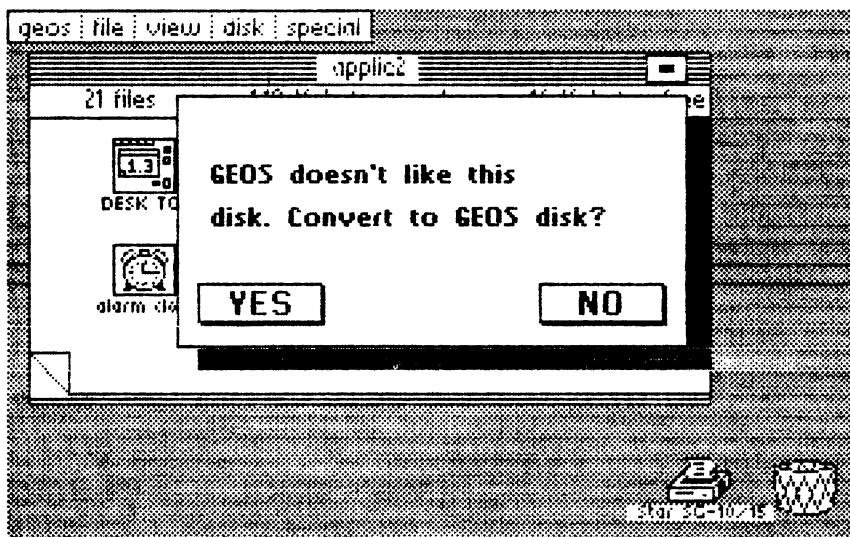
Figure 13-c: Modified error message example

3.4.2 Changing the position of the calculator
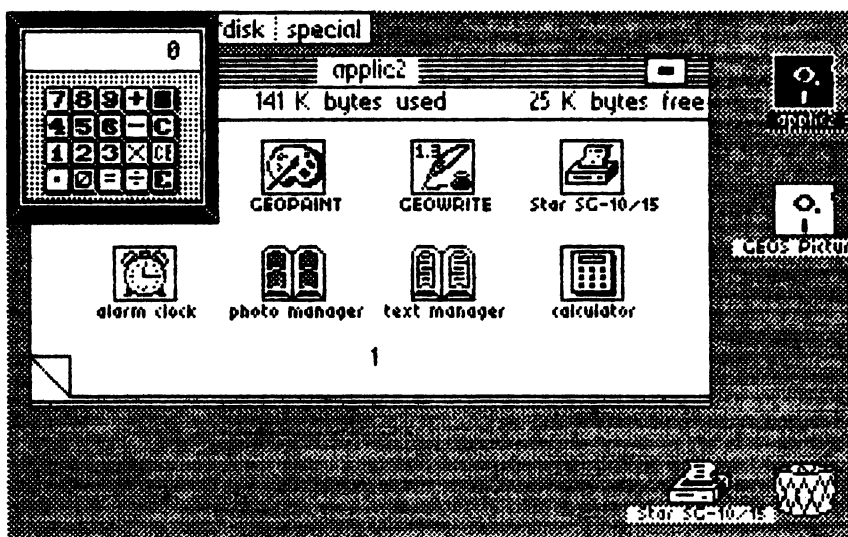
Figure 14: New calculator position

The calculator appears in the center of the screen, which can be quite aggravating. Using the modifier you can reposition the calculator to appear in the upper left hand corner of the screen. **All modifications should be made only to copies and never on the original disk.** Run the modifier and input `calculator` as the filename, enter the following bytes:

```
1st byte:   96
2nd byte: 169
3rd byte:  (RETURN)
```

To repostion the calculator enter these values:

```
1st byte:   96
2nd byte: 169
3rd byte:   0
4th byte: 133
5th byte:   4
6th byte: 169
7th byte:   0
8th byte:  (RETURN)
```

When a 17 is in the third byte instead of a 0 and a 56 in the seventh byte instead of a 0 the calculator appears in the normal position.

3.4.3 Changing the `notes` icon

The `notes` icon is the same as the `note pad` icon, which can be confusing for the beginner. By changing the data in the `note pad` accessory that writes out the `notes` file we can alter the `notes` icon so it is different from the `note pad` icon. **All modifications should be made only to copies and never on the original disk.** Run the modifier and input `note pad` as the filename, enter the following bytes:

```
1st byte: 249
2nd byte: 144
3rd byte:   0
4th byte:  25
5th byte: (RETURN)
```

To change the icon. Enter these values:

```
 1st byte: 249
 2nd byte: 144
 3rd byte:   0
 4th byte:  25
 5th byte: 148
 6th byte: 130
 7th byte:  25
 8th byte: 150
 9th byte: 105
10th byte:  25
11th byte: 144
12th byte:   0
13th byte:  25
14th byte: 148
15th byte:   0
16th byte:  25
17th byte: 150
18th byte: 166
19th byte: 153
20th byte: 144
21st byte:  83
22nd byte:  25
23rd byte: 144
24th byte:   0
25th byte:  24
26th byte: 156
27th byte:  34
28th byte:  25
29th byte: 148
30th byte: 122
31st byte: 153
32nd byte: (RETURN)
```

The `note pad` will now create an icon with `notes` on it.

3.4.4 Tabs and margins in geoWrite

This time we want to make a very useful suggestion for improvement. It refers to the word processing program geoWrite. **All modifications should be made only to copies and never on the original disk.**

When you create a new document with this program, the two margin settings and the tabs have default settings which are not very useful. In most cases you will set the margins so that the width of the document will fit in the displayed screen area. This prevents the horizontal scrolling when you reach the right edge. When you are done, then the margins can be set to the desired positions for printing. In the default setting, however, the right margin it outside the visible range so that whenever you create a new document you have to find the "M" on the ruler and move it to the left.

You can avoid this step in the future if you make a small change to geoWrite. We have found the locations in this program which contain the default values for the settings.

Of the twenty memory locations reserved for this purpose, four apply to the two margin positions and the remaining sixteen are for the eight tabs. Two memory locations are provided per marker (margin or tab). The reason for this is that a byte can only store values from 0 to 255, but geoWrite stores the positions in pixel format, in which the minimum distance from tab to tab is eight points.

With 60 possible tab positions we can get values up to 479, which could not be stored in a single byte. For example, if we want a marker at the 53rd position (counting from zero), we calculate the value for the memory locations as follows:

53 * 8 = 424 (pixel value of the position)

Since 424 can be divided once by 256, the high-order byte has the value one, and the low-order byte gets the remainder of the division, which is 168. With this procedure we can assign any of the two margins and eight tabs new values. The location in geoWrite which contains the default values can be found by searching for the following bytes with the modifier:

```
1st byte: 251
2nd byte:  96
3rd byte: (RETURN)
```

The following table contains the values we modified for the margin positions. In addition, the first three tabs are set. The meaning of the bytes is explained so you can use your own values. LO stands for the low-order byte and HI stands for the high-order byte.

```
1st byte: 251
2nd byte:  96
3rd byte:  24 (left margin, LO)
4th byte:   0 (left margin, HI)
5th byte:  48 (right margin, LO)
6th byte:   1 (right margin, HI)
7th byte:  64 (1st tab, LO)
8th byte:   0 (1st tab, HI)
9th byte: 144 (2nd tab, LO)
10th byte:   0 (2nd tab, HI)
11th byte: 224 (3rd tab, LO)
12th byte:   0 (3rd tab, HI)
13th byte:  48 (4th tab, LO)
14th byte:   1 (4th tab, HI)
15th byte:  48 (5th tab, LO)
16th byte:   1 (5th tab, HI)
17th byte:  48 (6th tab, LO)
18th byte:   1 (6th tab, HI)
19th byte:  48 (7th tab, LO)
20th byte:   1 (7th tab, HI)
21st byte:  48 (8th tab, LO)
22nd byte:   1 (8th tab, HI)
23rd byte: (RETURN)
```

If you want to use your own values here, you must make sure that all tabs are to the left of the right margin, that they have smaller values than the right margin. Unset tabs (tabs 4-8 are unset above), are set to the right margin. Even if just one tab is greater then the right margin, geoWrite will crash.
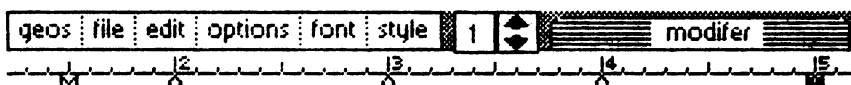
Figure 15: Preset tabs and margins in geoWrite

3.4.5 Modifying text in accessories

The modifier program can be used to customize the accessory text messages. Start the modifier program and insert the copy of GEOS that contains the `text manager` or `photo manager`. **All modifications should be made only to copies and never on the original disk.** Enter the name `text manager` or `photo manager` for the file. Select `1. Alter string` from the menu and enter the following as the search string:

`Please Select Option`

Enter the following as the replacement text:

`Please enter choice`

The program will run and add your customized accessory text message. Notice that the replacement string must be shorter than the original screen, if it is longer it may overwrite some important information on the GEOS diskette. Have fun making your own custom version of GEOS.
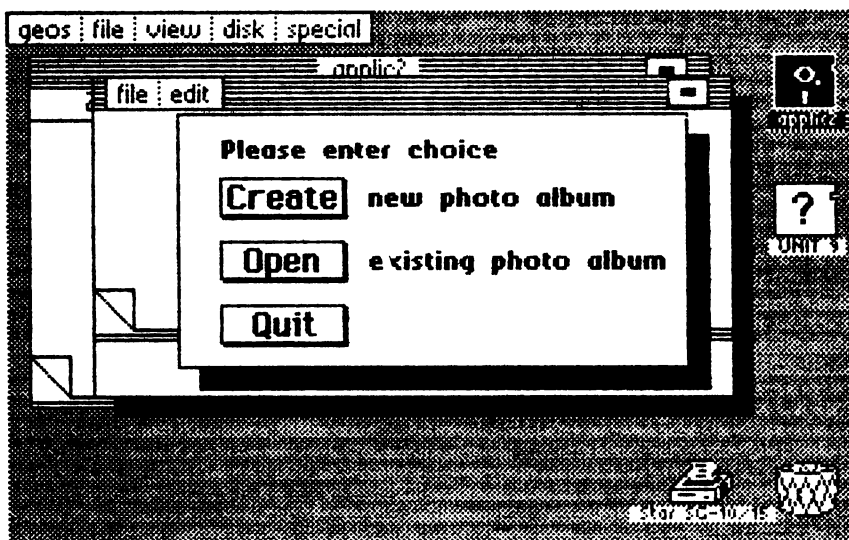
Figure 16: Modified `photo manager`

Chapter 4

New Accessories for GEOS

4. New Accessories for GEOS

When you think of GEOS, terms from the user interface (windows, pull-down menus, icons) usually come to mind, and this is certainly one of the most important new developments for the C64. But one essential new development which makes GEOS possible on the C64 is quickly forgotten: accessories. You can use these tools practically any time without losing what you're working on at the moment.

We were so impressed by the accessories during our work with GEOS that we had a lot of fun trying to write our own. In this chapter you will find two new programs which can really help you while working with GEOS:

-   a program which you can use to change existing character sets or create new ones in GEOS with the help of pull-down menus.

-   a complete machine language monitor with the usual capabilities. You can look at memory locations, disassemble parts of programs (even the entire GEOS KERNAL), and create and run your own machine language routines at any time.

One special feature is that the monitor allows you to save your own programs directly as accessories. You don't have to worry about the file type, start and end addresses, and the INFO sector. Just save it, exit the monitor, and double-click your program's icon.

If you have your ideas for a program under GEOS, you will not only find the listing of the FONT Editor in this chapter, you will also find complete documentation which shows you exactly how an accessory is bound into GEOS and how you can make good use of the existing routines.

We think that the more programmers who use GEOS's powerful capabilities and write corresponding programs, the more popular GEOS will become.

4.1 The FONT Editor

One of the biggest advantages of word processing under GEOS is certainly the ability to change the appearance of the characters in a document. All you have to do in geoWrite, for example, is select a character set under FONT and start typing and the letters will appear in this new style. You can select from the fonts (character sets) on the disk currently in the drive.

The first disappointment we experienced was not finding a variety of foreign characters in the font files on the original disk. There is University font text ranging from 6 to 24 point, but nowhere will you find a "ü". This isn't too surprising since the program was written in America and designed for the English-speaking American market.

But for using GEOS in other countries and for writing correspondence or other documents in a foreign language, we really need some of these foreign characters. In principle it is possible to modify individual characters of a font with the help of a machine-language monitor. We used this technique to modify our first font file by adding characters with German umlauts.

Then we toyed with the idea of at least printing the listing of such a modified character set in the book so that you could do the same thing without a monitor. In the end we came up with a better idea.

We have developed a program which you can use to modify (edit) entire fonts to heart's content. Not only can you change an "o" to "ö" by adding a couple of dots, you can define entire characters in proportional type (which is always used under GEOS).

How would you like a Greek character set for scientific or math work? Or subscripts and superscripts for chemists? Naturally, you can also define your own graphic symbols, such as the standard C64 symbols on the keys which cannot be accessed directly under GEOS.

The program is written flexibly so that existing character sets can be changed and entirely new ones can be created in heights ranging from 1 to 62 points. You still have to type in the program just like you would a font that we had modified, but the difference is that you get an "active" program out of the FONT Editor instead of just dry data.

Before you start the rather tiresome task of entering the DATA lines of our program into your computer, we want to make your mouth water a little to encourage you to make it all the way through .

4.1.1 Using the FONT Editor

Starting the program

First an important note: When working with our Editor, the original GEOS disk should be out of your reach. If something ever goes wrong, you can always go back to original fonts. **Copy the fonts you want to modify to your work disk.**

Also, we recommended a small change to the font name in order to avoid confusion later. Since the time and date are automatically placed in the INFO sector every time a file is modified, you should initialize these values with the `preference mgr` before loading the editor.

Since the editor is an accessory it can be loaded any time, even while you are using geoPaint or geoWrite. All you have to do is click on its name, which will appear as an option under the GEOS menu in the upper left corner of the screen. Double clicking the corresponding icon on the `deskTop` accomplishes the same thing.

To work with the FONT Editor you need a fair amount of free space on your disk. The SWAP file created when loading occupies 64 blocks, or about 16K. In addition you also need space if you want to create a new font or size. You should therefore have about 25K of free disk space when starting the FONT Editor.

If everything is working properly, a rectangle will appear in the middle of the screen with two pull-down menus.
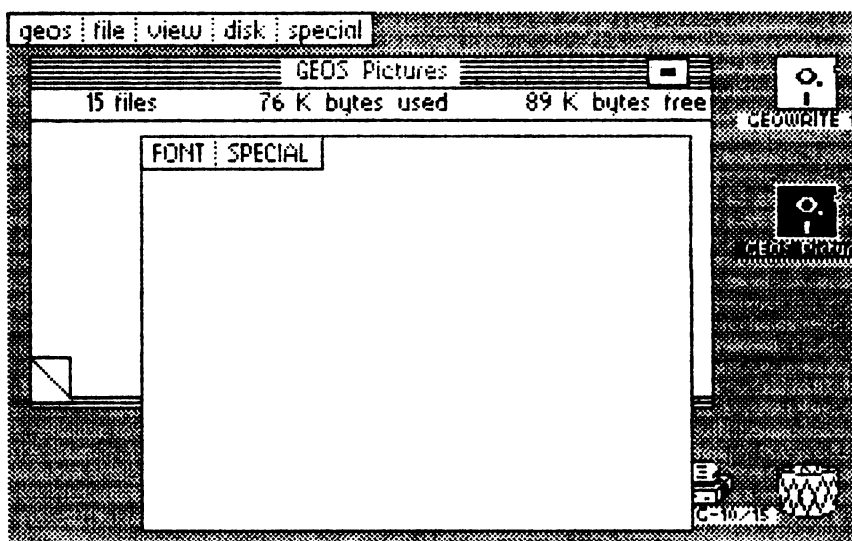
Figure 17: GEOS FONT Editor opening screen

When you click the menu FONT or SPECIAL, a menu will come down: FONT
has the options LOAD, SAVE, UPDATE, and CLR MEM. SPECIAL contains
QUIT and NEW DISK. For the sake of simplicity we will just list the option
below, and not the menu in which it is found.

You don't have to be concerned about starting the program from inside geoPaint
or geoWrite—you won't lose any data.

Modifying a font

In this section we will explain how to adapt an existing font file for your own
uses. Insert your work disk containing the font to be edited and select NEW
DISK with the mouse. The names of the first five fonts will be read into a list
and this list will appear after you click the option LOAD.
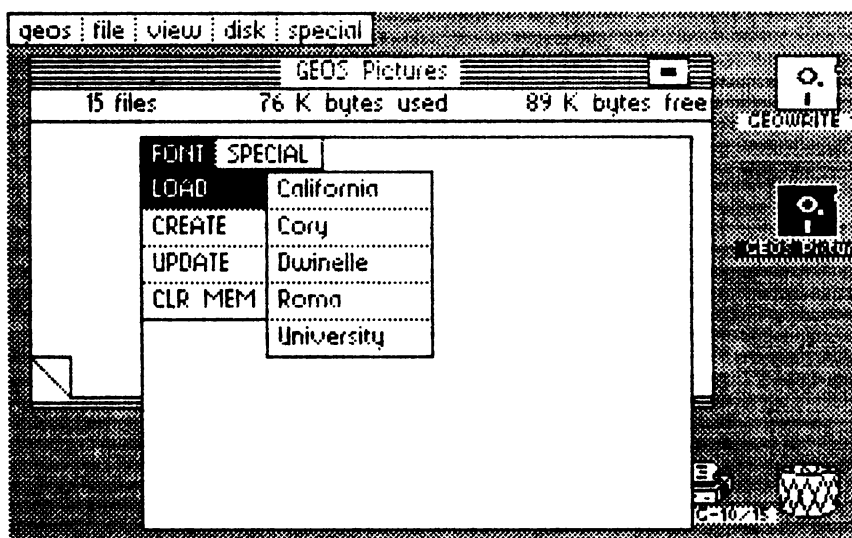
Figure 18: FONT Editor LOAD menu

It was necessary to limit the list to five entries because we had a limited amount of memory available. If you don't find your font in this list, you can rearrange the order of the fonts on the deskTop. Click QUIT to do this. When the disk drive stops running and the deskTop appears, put the font you wanted and another that you don't absolutely need on the border. Then put them both back in reverse order in the deskTop window. The order of the two fonts is not reversed. Naturally, the FONT Editor must be loaded again.

If your font is now in the load list, you must select the point size after selecting the font. The data will then be loaded. This will take a few seconds depending on the size of the file. We will use "University 12 point" as an example.

Now it gets exciting: Press the "a" key on the keyboard. The "a" will promptly appear in the right of the program window, and after a short delay it will also appear to the left, but this time much larger and with a black border.
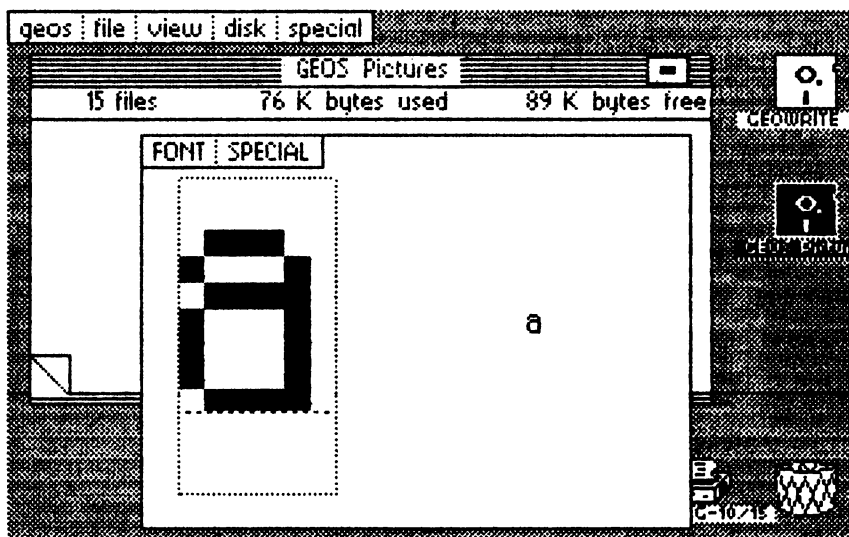
Figure 19: The letter a

You can now edit the character in this bounded field (edit field) with the joystick. Pressing the button will change the state of a given dot. A white dot will become black, and vice versa. The change will immediately be reflected in the "life-size" version to the right.

A line in editing field shows where the baseline of the character will be. All points below this line will be displayed as descenders. For a "g", for example, the line goes through the center of the letter because the lower curve of the character extends somewhat below the rest of the characters.

Try adding an umlaut to the letter "o" once. Press the "o" key so that this character is displayed in the edit field. Then set the appropriate points by clicking them.
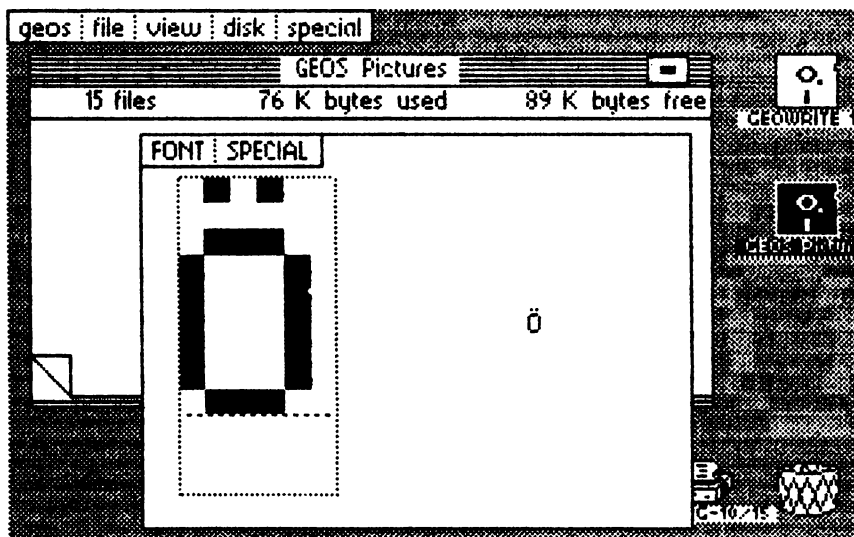
Figure 20: O-umlaut (ö)

When the "ö" is done, don't just exit the FONT Editor right away to try out your new character. What good is the nicest "ö" if you don't have a regular "o" any more? We should put the umlauts on keys that aren't needed for word processing. So we'll put the "ö" on the ":" key on the normal C64 keyboard.

First press the ":" key. The colon will appear and you can start replacing it with the "ö". You might start setting and clearing points on the screen, but after a short time you will probably be looking at the screen and scratching your head. Something isn't quite right. The character ":" is only two columns wide. We can hardly fit an "ö" in that space.
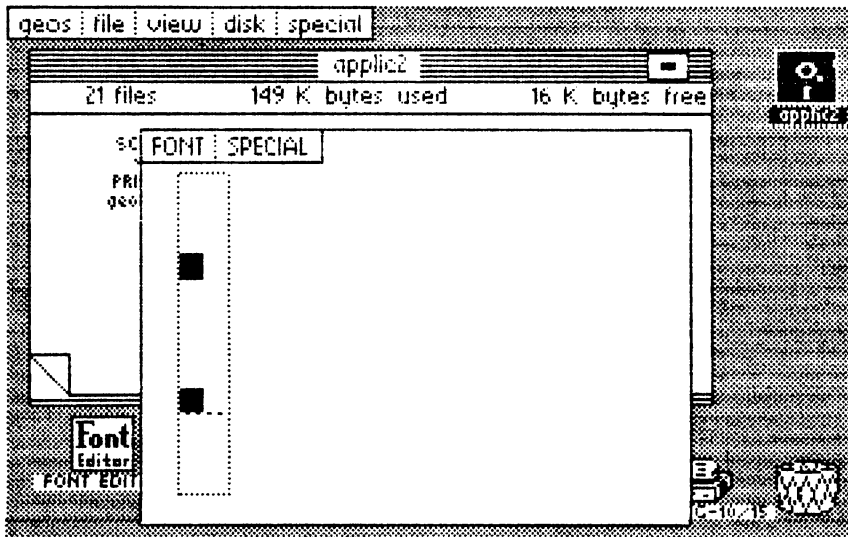
Figure 21: Colon (:)

This problem arises because the fonts are made of proportional-width characters. There is a different width for each character. If this couldn't be changed, our FONT Editor wouldn't be worth the paper it's printed on.

It was certainly quite a programming problem to make the width of each character changeable, because the fonts are stored in a very complicated manner, but the program had to have this feature. Press a key that you want to assign. If the field that appears is too small, position the mouse directly to the right of the edit field border and click until the field has the desired size. You can then begin with the definition of the character.
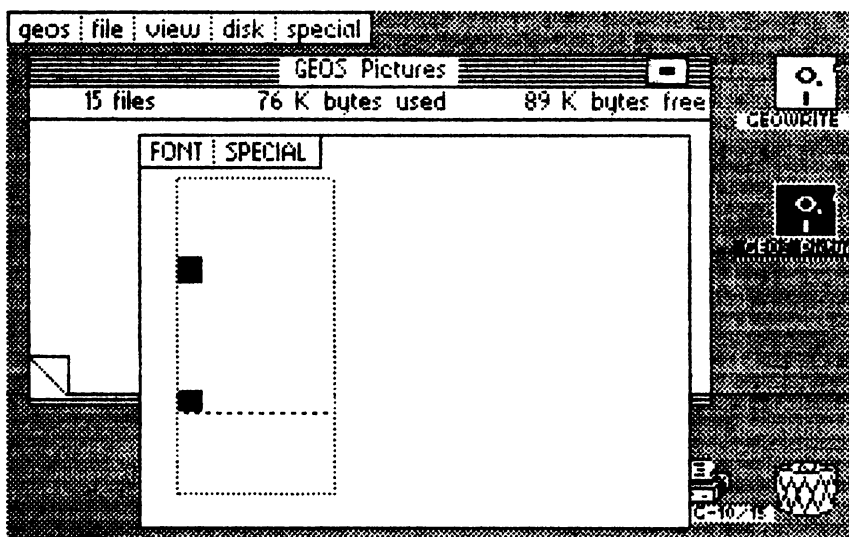
Figure 22: New width for :

Here we would like to refer you to Section 4.1.4, "Examples with the FONT Editor." It really isn't so easy to create the umlauts. In order to find room for the dots, the actual characters must be made a little smaller. Therefore we show you some examples after the printed program which show the easiest way to change the characters. If you run into difficulties and something doesn't work right away, take a look at Section 4.1.4.

Naturally you can change character sizes at any time. Making a character smaller causes some of the information to be lost, however, because the removed data is erased. If you reduce the size of an "a", for example, by one column and then enlarge it again, the "new old" column will be blank.

While you are working it is a good idea to select the option UPDATE now and then because this copies the font file to the disk, and if the computer should crash you will only lose the work you did since the last time you UPDATEd the file. You must UPDATE before exiting the program with QUIT or all your work will be gone.

You are now ready to define new fonts in a given point size (namely that of the loaded file) and then to save it under the same name. Nothing will have changed in the disk directory (except the entries in the INFO sector of the font). The result is not a new file, but a modification of an existing one. If you want to use the unmodified font, you can make a copy of the old font from the deskTop with DUPLICATE before you modify it.

Creating a new font

At the beginning we mentioned that it is possible to define completely new fonts without loading an existing one. The command for this is CREATE, as you probably discovered in the menu. Now you may click it. But be careful: This command will erase the font file memory. If you made some changes previously that you want to keep, you must select UPDATE first.

One condition must be met before we can begin: Since a new entry will be created in the directory, the program needs at least one font file on the current disk. The editor itself is not very creative and has to steal the icon from another font. Naturally we could have stored this data in the program, but we figured there were enough DATA lines already, right?

You can use the LOAD option to determine if there is a suitable file on the disk. If this results in "* No File *" you know what to do: copy a font to the disk in the deskTop. If at least one entry is present, we can get going: After you have selected CREATE, the cursor will appear next to the menu bar. It will wait until you enter a name for the font you are about to create. There must be a two-digit number followed by a colon in front of the actual name. This number is very important for the correct operation of the font.

Since a computer can work with numbers better than it can with names, GEOS distinguishes the fonts internally by a number. BSW has the number 0, California the number 1, and so on, in the order Cory, Dwinelle, Roma, and University. The numbers 0 through 5 are thus already present. Theoretically you could continue with 06 for your own character sets.

But since other fonts will be offered by Berkeley Softworks, we recommend that the numbering for your own fonts start with 30. This should leave enough room for future expansion fonts. For the first time enter "30:xxx" where "xxx" stands for the name you want to give to the font.
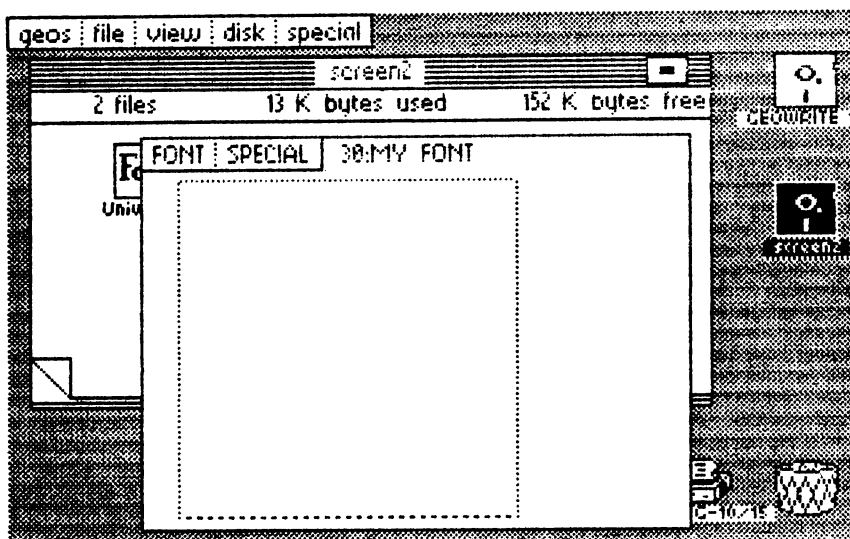
Figure 23: 30:MY FONT

If your imagination has deserted you, you can just use the three x's, since you can change the name later from the deskTop when you think of something better. As always, RETURN ends the input.

Now the input can begin: After pressing a key, you see nothing except a box. This again borders the editable area for the printed character. Pressing the fire button inside the field inverts a region. The only problem is that the entire field is just one such region, and therefore the whole thing turns black when you press the fire button. You have to be quite talented to define readable characters in this manner.
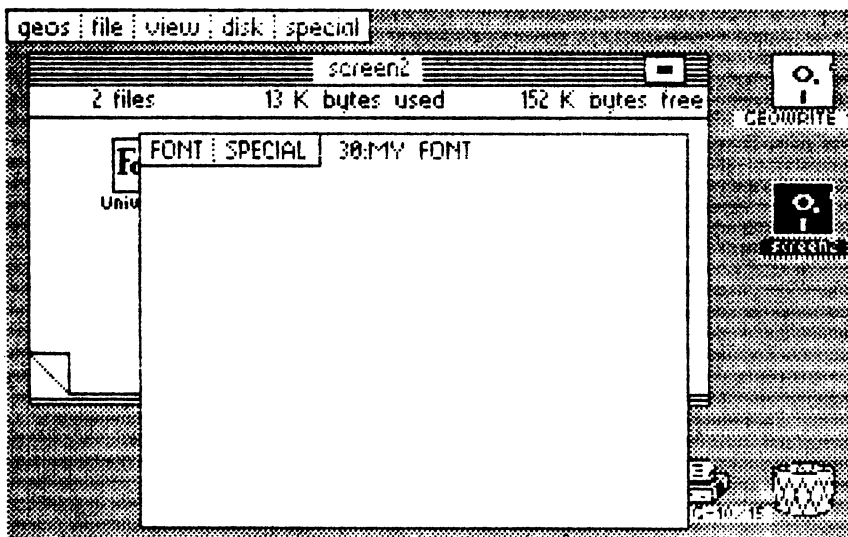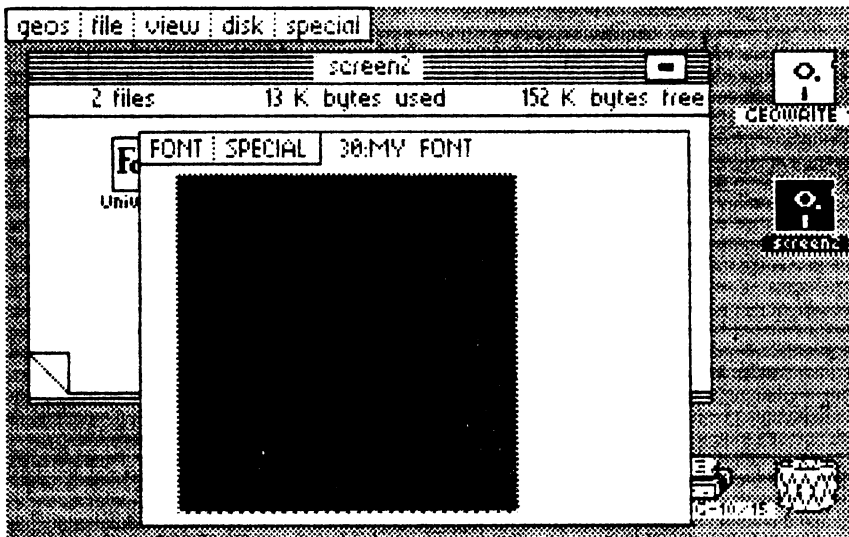
Figure 24: The first character, a large point

Since we don't have this gift, we added a function which allows you to set the height of a font. If you have read the last section carefully (especially the part about changing the width of a character), you can probably guess how to change the height of the character in the FONT Editor. Right: If the mouse is clicked at the upper edge of the window, the font will increase in size by one line, which will be added at the bottom. The last line can be removed in a similar manner, only this time the mouse is clicked past the lower edge of the window.

This selection must be made before each new font is created. It decides directly the point size under which the file will be stored later. For example, if you immediately click UPDATE, a font "xxx 1 Point" will be created on the disk. It is therefore an important decision that must be made at the beginning.

At least as important is the determination of the baseline for the font. This specifies how many points descenders will occupy. This horizontal line will be visible in 2 point size or greater and can be moved down by clicking the mouse at the far right, outside the white field. When it reaches the bottom it will appear again in the first line.

The descenders must be accounted for when calculating the point size. For example, a 9-point font really allows only about seven lines for "normal" letters. The unused two lines on the bottom are available for the descenders of certain characters (like "Q", "q" and "y").

The following figure shows the same point from the previous figure. We have clicked five times on the right, eight down, and six to the far right. This resulted

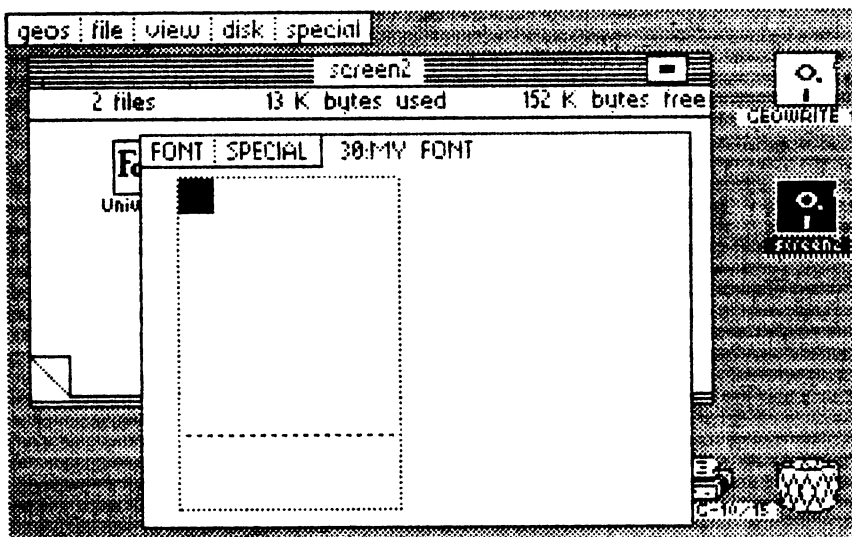in a 9-point font with two points for descenders and six columns for this character.

Figure 25: A nine-point font with two-point descenders

One last point: Selecting the option CLR MEM (clear memory) will wipe out the font currently in the computer immediately and without warning.

The information about font height and individual character width will not be deleted by CREATE. Therefore, it is possible to get earlier information (e.g., using LOAD ROMA 10 Point) and build a new character set based upon this pre-loaded information using CREATE. This means that it is not possible to "stretch" individual characters. It will only allow you to reset pixels in a font through CREATE. If, however, CLR MEM is selected, all the information is deleted, and a 1-point character set is created, which gives all the characters a width of 1.

Ending the program

Before selecting NEW DISK or QUIT, the diskette already in the drive must be selected for UPDATE, or else the changes made later to the font being edited will not be saved.

Before QUITting, the loader diskette with the FONT Editor must be inserted, or else the system will crash.

To exit simply select QUIT. But be careful, the disk drive runs for a reason after you click QUIT: When the program is loaded, important data is saved on the disk. GEOS pros know what this is—a SWAP file, which is loaded again when you quit. Naturally, this is found only on the disk from which the FONT Editor was loaded. If the correct disk is not inserted, the computer commits hara-kiri, but only in software. Everything will be all right when you restart GEOS.

You should be made aware of one thing yet when using the FONT Editor with geoWrite or geoPaint. It can occur that the new font will not be immediately available. This is due to the conception of the accessories and not an error in the editor. Accessories can change things on the disk without having to inform the applications what they are doing. If this happens to you, exit the application and start it again.
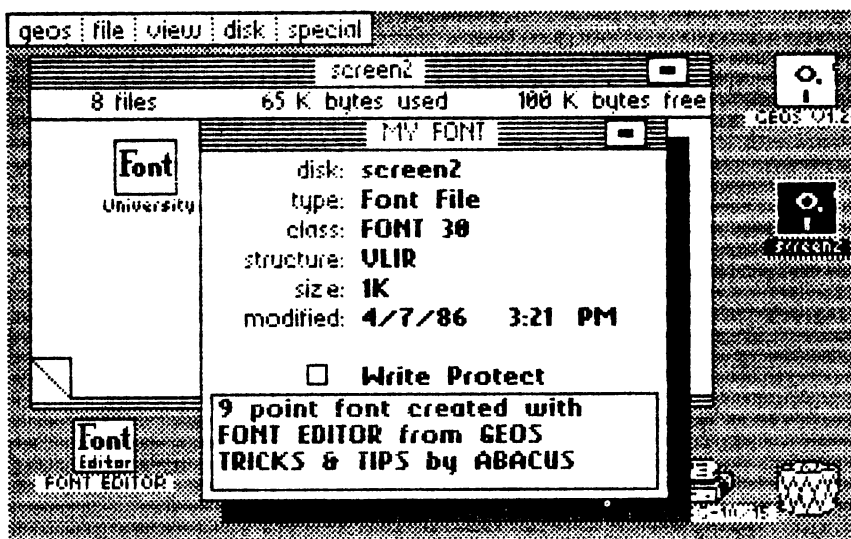
Figure 26: Class: FONT 30

In this figure you can see what the INFO sector of a new font looks like. The entry under "class" is very important so that you later know what numbers you have already assigned to your own fonts.

4.1.2 Reference guide

We put together this little section so you don't have to page through the whole tutorial section to find an answer to your question. This section contains a brief description of the menu options and their function in alphabetical order.

CLR MEM: Clears the working memory of the individual characters and a 1-point character set is created, which gives all the characters a width of 1.

CREATE: Used to create completely new fonts. After selecting the option, the cursor will appear next to the menu so that you can enter a two-digit number followed by a colon and the name of the new font.

IMPORTANT: You cannot use a number more than once because then GEOS will not be able to distinguish between the fonts. For fonts you create yourself you will find this identifier in the INFO screen for the font under class (such as class: Font 31).

The numbers 00 to 05 are already used. Since additional fonts may come out on the market, it's a good idea to start your own numbering at 30.

There must be some font file already on the disk. In addition, there must be enough room on the disk (amount needed depends on the point size, but at least 5K).

Fonts opened by LOAD and CREATE are automatically updated during editing.

FONT: Displays the first menu.

LOAD: Displays the first five fonts on the current disk.

NOTE: After clicking LOAD it is no longer possible to work with the current font. Fonts opened by LOAD and CREATE are automatically updated during editing. You should also save your work with UPDATE.

When the name of a font is clicked, the available sizes will appear. After clicking a size the corresponding font will be loaded.

105

NEW DISK: This must be clicked every time the disk is changed. Corresponds to the command open under disk in deskTop. In addition, the current work will be ended and the names of the new fonts will be read into the LOAD list.

NOTE: If you have made changes to fonts on the old disk which you want to keep, you must first select UPDATE. Also, NEW DISK must be selected before the disks are changed in this case because this will close old files.

UPDATE: Stores the current file to disk if changes have been made. Any old data will be overwritten. The program must be ended with QUIT since this is the only way the current file will be closed properly (don't just turn off the computer!).

SPECIAL: Displays the second menu.

QUIT: Correct way to exit the program. The disk from which FONT Editor is loaded must be inserted first. Important data must first be saved with UPDATE.

4.1.3 The listing

Enter the program very carefully because a single incorrect digit can have fatal consequences on the operation. We have included a checksum which will catch some errors, but it certainly isn't fool proof. One error can compensate for another and both will go undetected. If you have trouble finding the error, have someone help you since it's easy to read right over your own errors.

If everything works properly, the cursor will appear after you start the program. With it you can enter the name of the program under which it will be saved on the disk. The file will be placed in GEOS format through the BASIC header. This includes specifications which are necessary for it to work correctly (load and start address, end address, etc.). If you then boot GEOS you will see the new file with a rather ugly striped icon. If you find this disturbing, there are three things you can do about it:

1.   Use the Berkeley Softworks icon editor to change the icon.

2.   Modify the section of the loader program that defines the icon. This is in line 115 in which the 63 bytes which generate the striped pattern are passed. The construction corresponds to that of a sprite.

3.   The FILE MASTER from the Abacus book *GEOS Inside and Out* offers an easy way to change program icons.

Here finally, is the listing for the BASIC loader:

```
5 open 1,8,15,"i:0":rem error channel
10 restore:print chr$(147):rem clr home
15 rem * calculate check sum *
20 for i=1 to 2225
25 read a:s=s+a
30 print chr$(19);2347-i;chr$(157);" ";
35 next:restore
40 if s<>252030 then print "Error in DATA !":stop
45 rem * file output *
50 input "Filename";f$
55 open 2,8,2,f$+",u,w":gosub 255
60 for i=1 to 2347:read a
65 print#2,chr$(a);:next:gosub 255
70 close2
75 rem * put info-sector *
80 a=1:b=0
85 t=a:s=b:print#1,"b-a:0";t;s
90 input#1,n,t$,a,b:if a=18 then a=19:b=0
95 if n>0 then 85
100 open 3,8,3,"#0":print#3
105 print#1,"u1";3;0;t;s:rem sector read
110 print#3,chr$(0)chr$(255)chr$(3)chr$(21)chr$(191);:rem info copy
115 fori=1 to 63:print#3,chr$(85);:next:    rem striped-icon definition (!)
```

```
120 print#3,chr$(130)chr$(5)chr$(0);
125 print#3,chr$(0)chr$(65); :rem start
130 print#3,chr$(64)chr$(127);:rem end
135 print#3,chr$(0)chr$(65); :rem init
140 fori=1 to 179
145 print#3,chr$(0);:rem rest
150 next
155 print#1,"u2";3;0;t;s:rem write
160 rem * in directory entry *
165
170 print#3
175 a$=chr$(18):b$=chr$(1):rem directory
180 a=asc(a$):b=asc(b$):print#1,"u1";3;0;a;b
185 get#3,a$,b$:rem next sector
190 for i=0 to 7:rem number records
195 print#1,"b-p";3;i*32+5
200 for j=1 to len(f$)
205 get#3,w$:if w$<>mid$(f$,j,1) then k=0:goto 215
210 k=k+1
215 next j
220 if k=len(f$) then 230:rem found
225 next i:gosub 255:goto 180
230 print#1,"b-p";3;i*32+21
235 print#3,chr$(t)chr$(s)chr$(0)chr$(5);
240 print#1,"u2";3;0;a;b:rem dir back
245 close1:print"o.k.":end
250 rem *****************************
255 input#1,n,t$:if n then print$:stop
260 return
265 rem
300 rem
301 data162,7,181,112,157,73,74,202,16,248,162,0,32,45,68,162,0,189,37,73
302 data149,6,232,224,6,208,246,169,0,32,57,193,32,36,193,169,255,32,39,193
303 data32,196,67,32,75,193,162,49,160,73,134,2,132,3,169,1,32,81,193,96
304 data173,183,132,208,24,173,4,133,201,32,144,17,201,127,176,13,162,90
305 data160,65,142,169,132,140,170,132,32,162,70,96,173,183,132,208,250,169
306 data255,141,62,74,173,23,74,197,60,144,7,32,138,70,32,162,70,96,160,0
307 data166,41,109,63,74,197,60,176,11,200,202,208,245,32,103,70,32,162,70
308 data96,140,65,74,165,59,240,7,32,129,68,32,162,70,96,162,0,172,64,74
309 data173,22,74,10,10,10,197,58,144,7,32,220,69,32,162,70,96,109,63,74
310 data197,58,176,11,232,136,208,245,32,252,68,32,162,70,96,172,65,74,173
311 data4,133,56,32,147,68,32,162,70,96,72,32,36,72,104,174,61,74,240,10
312 data32,144,193,169,0,133,2,133,3,96,133,2,10,10,10,10,101,2,105,82,133
313 data2,169,0,105,82,133,3,160,0,177,2,153,45,74,240,3,200,208,246,32,116
314 data194,138,208,210,168,162,0,189,9,74,153,167,82,200,232,224,11,208
315 data244,192,110,208,238,169,0,141,202,73,141,69,74,32,122,194,224,8,240
316 data64,152,240,56,173,202,73,10,10,10,109,202,73,109,202,73,109,202,73
317 data105,2,168,174,69,74,232,169,0,248,24,105,1,202,208,251,216,170,74
318 data74,74,74,240,5,9,48,153,167,82,200,138,41,15,9,48,153,167,82,238
319 data202,73,238,69,74,208,185,32,119,194,174,202,73,173,196,73,202,48
320 data5,24,105,14,208,248,141,197,73,173,202,73,9,128,141,202,73,162,196
321 data160,73,134,2,132,3,96,72,32,144,193,162,45,160,74,134,2,132,3,32
322 data116,194,169,255,141,150,132,104,72,32,122,194,152,240,250,104,56
323 data233,1,16,243,162,21,160,83,134,16,132,17,162,0,160,20,134,6,132,7
324 data32,140,194,138,208,3,32,0,73,32,119,194,96,32,144,193,32,36,72,173
325 data61,74,208,244,32,141,193,32,205,71,32,232,67,32,75,193,169,27,32
```

108

```
326 data69,193,32,159,193,47,57,120,0,250,0,169,0,141,42,74,141,44,74,133
327 data4,162,125,160,0,169,49,134,24,132,25,133,5,169,15,133,6,162,42,160
328 data74,134,2,132,3,162,31,160,67,142,163,132,140,164,132,32,186,193,96
329 data173,44,74,201,58,208,192,32,138,193,162,82,160,82,32,14,73,208,120
330 data162,77,157,0,129,232,208,250,189,110,73,157,77,129,240,3,232,208
331 data245,169,32,157,77,129,173,42,74,157,78,129,173,43,74,157,79,129,169
332 data0,157,80,129,173,43,74,41,15,72,173,42,74,41,15,170,104,202,48,5
333 data24,105,10,208,248,141,128,129,162,45,160,74,142,0,129,140,1,129,162
334 data0,160,129,134,20,132,21,169,0,133,22,32,237,193,138,208,143,162,45
335 data160,74,134,2,132,3,32,116,194,138,208,13,32,137,194,138,240,250,224
336 data9,208,3,32,0,73,32,119,194,96,32,144,193,32,57,72,96,32,144,193,173
337 data163,132,13,164,132,240,59,32,196,67,96,162,0,189,26,74,157,21,83
338 data232,224,8,208,245,169,0,170,168,152,157,29,83,232,169,0,157,29,83
339 data232,200,192,98,208,240,32,205,71,166,44,164,45,134,4,132,5,169,0
340 data133,2,169,20,133,3,32,120,193,96,32,144,193,169,0,141,62,74,32,36
341 data72,32,225,193,162,7,189,73,74,149,112,202,16,248,162,194,160,62,142
342 data156,132,140,155,132,96,32,144,193,169,0,141,62,74,32,36,72,32,225
343 data193,169,5,133,17,138,208,49,162,82,160,82,134,14,132,15,169,8,133
344 data16,169,0,133,22,133,23,169,0,141,61,74,32,59,194,138,240,20,169,255
345 data141,61,74,189,253,73,157,82,82,232,224,12,208,245,169,1,208,7,169
346 data5,56,229,17,240,229,170,9,128,141,170,73,173,164,73,202,48,5,24,105
347 data14,208,248,141,165,73,96,174,21,83,232,236,24,83,208,2,162,0,142
348 data21,83,32,205,71,96,8,142,65,74,140,66,74,56,233,32,10,168,177,42
349 data24,109,65,74,133,112,200,177,42,105,0,133,113,165,112,41,7,73,7,133
350 data114,70,113,102,112,70,113,102,112,70,113,102,112,162,0,236,66,74
351 data240,14,165,112,24,101,39,133,112,144,2,230,113,232,208,237,165,112
352 data24,101,44,133,112,165,113,101,45,133,113,160,0,166,114,189,34,74
353 data133,114,40,144,4,81,112,145,112,49,112,240,2,56,96,24,96,160,192
354 data177,42,24,105,1,141,70,74,200,177,42,105,0,141,71,74,162,3,78,71
355 data74,110,70,74,202,208,247,238,70,74,165,39,205,70,74,176,60,164,41
356 data136,140,70,74,169,127,32,201,193,170,202,172,70,74,169,127,24,32
357 data147,68,166,39,160,0,177,112,172,70,74,145,112,165,112,56,233,1,133
358 data112,165,113,233,0,133,113,202,208,231,206,70,74,208,208,238,22,83
359 data32,205,71,165,44,24,101,39,141,70,74,174,64,74,160,0,173,4,133,24
360 data32,147,68,173,70,74,56,229,112,141,71,74,164,41,136,140,70,74,174
361 data64,74,202,172,70,74,173,4,133,24,32,147,68,160,0,166,114,202,138
362 data49,112,74,8,72,138,73,255,49,112,145,112,104,17,112,145,112,165,112
363 data141,187,69,165,113,141,188,69,162,1,172,71,74,40,136,240,6,126,255
364 data255,232,208,247,206,70,74,16,189,173,4,133,56,233,31,10,170,254,29
365 data83,208,3,254,30,83,232,232,224,194,144,242,96,165,44,24,101,39,141
366 data70,74,174,64,74,202,240,241,160,0,173,4,133,24,32,147,68,173,70,74
367 data56,229,112,141,71,74,164,41,136,140,70,74,174,64,74,202,172,70,74
368 data173,4,133,24,32,147,68,165,112,141,34,70,165,113,141,35,70,174,71
369 data74,202,240,7,24,62,255,255,202,208,250,8,160,0,166,114,202,138,49
370 data112,40,42,72,165,114,10,56,233,1,73,255,49,112,145,112,104,17,112
371 data145,112,206,70,74,16,185,173,4,133,56,233,31,10,170,189,29,83,233
372 data0,157,29,83,201,255,208,3,222,30,83,232,232,224,194,144,235,96,173
373 data24,83,201,62,176,27,238,24,83,32,205,71,162,0,164,41,136,169,32,24
374 data32,147,68,169,0,168,145,112,200,196,39,208,249,96,174,24,83,224,1
375 data240,16,202,142,24,83,202,236,21,83,176,3,142,21,83,32,205,71,96,162
376 data0,189,43,73,149,6,232,224,6,208,246,32,36,193,162,195,160,0,169,120
377 data134,24,132,25,133,5,173,4,133,32,69,193,162,81,160,74,134,4,132,5
378 data162,1,160,8,134,2,132,3,32,120,193,173,4,133,32,201,193,240,195,141
379 data64,74,197,41,176,2,165,41,133,112,162,255,169,128,232,56,229,112
380 data176,250,142,63,74,169,0,141,66,74,174,64,74,202,142,65,74,172,63
381 data74,169,0,141,75,71,24,109,66,74,136,208,250,141,74,71,160,4,14,74
```

```
382 data71,46,75,71,136,208,247,169,82,24,109,74,71,141,74,71,133,118,169
383 data74,109,75,71,141,75,71,133,119,174,65,74,172,66,74,173,4,133,24,32
384 data147,68,172,63,74,8,8,162,0,40,126,255,255,8,232,224,16,208,246,40
385 data40,136,208,237,206,65,74,16,216,174,63,74,202,240,32,165,118,133
386 data116,165,119,133,117,165,116,24,105,16,133,118,165,117,105,0,133,119
387 data160,15,177,116,145,118,136,16,249,48,221,238,66,74,173,66,74,197
388 data41,240,3,76,251,70,32,156,71,32,217,71,169,223,37,57,133,57,96,160
389 data0,162,0,189,20,74,149,2,232,224,6,208,246,152,72,240,16,230,5,165
390 data2,24,105,16,133,2,144,2,230,3,136,208,240,169,144,145,2,32,66,193
391 data104,168,200,192,128,208,210,96,169,21,133,2,169,83,133,3,32,204,193
392 data96,173,23,74,133,6,174,63,74,24,202,48,4,101,41,208,249,233,0,133
393 data7,169,0,133,9,133,11,173,22,74,10,10,10,133,8,174,63,74,202,48,5
394 data109,64,74,208,248,233,0,133,10,169,85,32,39,193,165,6,166,38,24,109
395 data63,74,202,16,250,233,0,133,24,169,153,32,24,193,96,169,0,141,169
396 data132,141,170,132,141,163,132,141,164,132,32,57,72,32,75,193,96,173
397 data62,74,240,250,162,45,160,74,134,2,132,3,32,116,194,169,127,32,201
398 data193,170,164,41,136,169,127,24,32,147,68,165,112,56,233,20,133,6,133
399 data112,165,113,233,83,133,7,133,113,162,21,160,83,134,16,132,17,165
400 data41,141,150,132,32,143,194,32,119,194,162,45,160,74,32,14,73,162,254
401 data232,232,189,130,129,41,31,240,42,197,41,240,38,144,241,160,28,185
402 data129,129,153,131,129,185,128,129,153,130,129,185,96,129,153,98,129
403 data185,95,129,153,97,129,136,136,232,232,224,28,144,224,152,170,169
404 data0,133,6,173,128,129,10,38,6,10,38,6,10,38,6,10,38,6,10,38,6,10,38
405 data6,5,41,157,130,129,165,6,157,131,129,165,112,157,97,129,165,113,157
406 data98,129,174,19,132,172,20,132,134,4,132,5,162,0,160,129,134,10,132
407 data11,32,231,193,169,0,141,62,74,96,162,60,160,65,142,163,132,140,164
408 data132,32,205,71,96,134,14,132,15,32,11,194,138,208,12,162,0,160,132
409 data134,20,132,21,32,41,194,138,96,46,194,50,0,0,1,55,183,195,0,255,0
410 data46,58,50,0,118,0,2,110,73,128,66,73,115,73,128,93,73,59,115,50,0
411 data97,0,132,123,73,128,164,73,128,73,0,202,66,135,73,0,174,67,142,73
412 data0,181,67,59,87,79,0,129,0,130,150,73,0,252,67,155,73,0,31,68,70,79
413 data78,84,0,83,80,69,67,73,65,76,0,76,79,65,68,0,67,82,69,65,84,69,0
414 data85,80,68,65,84,69,0,67,76,82,32,77,69,77,0,81,85,73,84,0,78,69,87
415 data32,68,73,83,75,0,59,73,97,0,159,0,129,82,82,64,204,65,99,82,64,204
416 data65,116,82,64,204,65,133,82,64,204,65,150,82,64,204,65,59,243,159
417 data0,212,0,58,167,82,0,139,66,178,82,0,139,66,189,82,0,139,66,200,82
418 data0,139,66,211,82,0,139,66,222,82,0,139,66,233,82,0,139,66,244,82,0
419 data139,66,255,82,0,139,66,10,83,0,139,66,42,32,78,79,32,70,73,76,69
420 data32,42,0,32,32,32,88,32,80,79,73,78,84,0,81,74,8,62,16,1,0,12,0,1
421 data8,0,202,0,1,2,4,8,16,32,64,128,32
```

4.1.3.1 The FONT Editor program

Here is the source code for FONT Editor.

```
;*******************************
;*  Prgr. : Font-Editor V1.0  *
;*  Update: 21.06.87          *
;*  Author: Ruediger Kerkloh  *
;*******************************

        * = $4100
    .offs  $c000-*

    bsout = $c145
  fontoff = $c14b
  openrec = $c274
 closerec = $c277
   append = $c289
     next = $c27a
     read = $c28c
    write = $c28f
   muster = $c142
   clrmem = $c178
    input = $c1ba
 mouseoff = $c18d
  mouseon = $c18a
     word = $8504
    ttjob = $84a3
  mausjob = $84a9
  mausxlo = $3a
  mausxhi = $3b
    mausy = $3c
     line = $26
    bytes = $27    ;Total bit width
     hoch = $29    ;character height
     tab1 = $2a    ;linker table
     tab2 = $2c    ;data table
       p1 = $70
       p2 = $72
       p3 = $74
       p4 = $76

        ldx #$07   ;store register
  r1    lda p1,x
        sta rett,x
        dex
        bpl r1

        ldx #0
        jsr init   ;initialize directory

        ldx #0     ;window data
  cl1   lda wndw1,x
        sta $06,x
        inx
```

```
            cpx #6
            bne cl1

            lda #$00
            jsr $c139    ;Fill in

            jsr $c124    ;fill pattern

            lda #$ff
            jsr $c127    ;border

            jsr nulfont  ;default value

            jsr fontoff

            ldx #<pull   ;Pull-Down Menu
            ldy #>pull
            stx $02
            sty $03
            lda #1
            jsr $c151    ;create

            rts          ;Ready


            ;* Keyboard job *

    tt      lda $84b7    ;Pulldown active?
            bne rtn3     ;yes!

            lda $8504     ;Character
            cmp #32
            bcc rtn3      ;Is the character illegal?
            cmp #127
            bcs rtn3

            ldx #<button ;Button  init.
            ldy #>button
            stx mausjob
            sty mausjob+1

    tt1     jsr show     ;Show character

    rtn3 rts


    button ;* Fire button job *

            lda $84b7    ;Pulldown active?
            bne rtn3     ;yes!

            lda #$ff      ;Update necessary
            sta flag2

            lda pos+3    ;Y starting coordinate of picture
            cmp mausy
```

112

```
        bcc ok50

        jsr kompry   ;Mouse too far up
        jsr show
        rts

ok50 ldy #$00      ;row counter
     ldx hoch      ;(number of Y fields)

ok32 adc faktor
     cmp mausy
     bcs ok31      ;Goal
     iny
     dex
     bne ok32
                   ;Mouse too low
     jsr expandy
     jsr show
     rts

ok31 sty spalte

     lda mausxhi
     beq ok49

     ;Mouse all the way to the right:

     jsr gline
     jsr show
     rts

ok49 ldx #$00      ;Column counter
     ldy breit     ;(number of X fields)

     lda pos+2     ;x start of screen in
     asl a         ;mouse coordinates
     asl a
     asl a

     cmp mausxlo
     bcc ok34

     ;Mouse left near border:

     jsr komprx
     jsr show
     rts

ok34 adc faktor
     cmp mausxlo
     bcs ok33      ;Goal
     inx
     dey
     bne ok34

     ;Mouse right, near border:
```

```
        jsr expandx ;expand
        jsr show    ;and display
        rts

ok33 ldy spalte
     lda word
     sec
     jsr pixel    ;inverse.

     jsr show     ;display

ok30 rts


     ;********
     ;* load *
     ;********

ap0  pha

     jsr close

     pla

     ldx flag1    ;Files available?
     beq ok35     ;yes!

ok36 jsr $c190    ;Else exit;

     lda #$00     ;and no submenu
     sta $02
     sta $03
     rts


ok35 sta $02      ;click-field times 17
     asl a
     asl a
     asl a
     asl a
     adc $02
     adc #<d
     sta $02
     lda #0
     adc #>d
     sta $03

     ldy #0          ;take names
lo2  lda ($02),y
     sta fname,y
     beq lo1
     iny
     bne lo2

lo1  jsr openrec
```

```
        txa             ;($02/$03 - Name)
        bne ok36        ;Error

        tay             ;enter "point"
ok43 ldx #0             ;(10 times)
ok42 lda t2,x
        sta p,y
        iny
        inx
        cpx #11
        bne ok42
        cpy #110
        bne ok43

        lda #0
        sta anzp        ;Number of entries
        sta pnt         ;Point size

ok38 jsr next           ;Read entry
        cpx #8
        beq ok45        ;All read

        tya             ;Track =0 ?
        beq ok39        ;then it isn't available

        lda anzp        ;times 11 + 2
        asl a
        asl a
        asl a
        adc anzp
        adc anzp
        adc anzp
        adc #2
        tay             ;Convert data points

        ldx pnt         ;to number
        inx
        lda #0
        sed
        clc
ok44 adc #1
        dex
        bne ok44
        cld

        tax             ;Left decimal value
        lsr a
        lsr a
        lsr a
        lsr a
        beq ok46
        ora #$30
        sta p,y

ok46 iny               ;right decimal value
        txa
```

```
          and #$0f
          ora #$30
          sta p,y

          inc anzp
ok39 inc pnt          ;continue
          bne ok38

ok45 jsr closerec

          ldx anzp
          lda pp1       ;y-up
ok40 dex
          bmi ok41
          clc
          adc #14       ;Establish menu
          bne ok40
ok41 sta yuntp      ;length

          lda anzp      ;Vertical menu
          ora #$80
          sta anzp

          ldx #<pp1     ;Design sub-sub-menu
          ldy #>pp1
          stx $02
          sty $03

          rts

;***********************
;* Select point size *
;***********************

ap1  pha              ;store

          jsr $c190     ;Exit

          ldx #<fname
          ldy #>fname
          stx $02
          sty $03
          jsr openrec

          lda #$ff      ;Reset
          sta $8496     ;REC-Pointer

          pla
ok48 pha

ok47 jsr next
          tya
          beq ok47     ;No entry

          pla
          sec
```

```
            sbc #1
            bpl ok48

            ldx #<memo     ;Buffer
            ldy #>memo
            stx $10
            sty $11

            ldx #$00       ;max. 5 K
            ldy #$14
            stx $06
            sty $07

            jsr read
            txa
            bne rtn2       ;Error

            jsr ttinit

rtn2 jsr closerec

rtna rts


            ;**********
as2   ;* Create *
            ;**********

            jsr $c190
            jsr close

            lda flag1      ;no fonts
            bne rtna       ;for icon!

            jsr mouseoff

            jsr fonton

            jsr clr1       ;Delete set


            ;Namen eingeben

            jsr fontoff

            lda #27
            jsr bsout      ;plain text

cr5   jsr $c19f      ;i-rectangle

            .byte 47,57,120,0,250,0

            lda #0
            sta fnmbr      ;Clear buffer
            sta fnmbr+2    ;No colon
            sta $04        ;Job flag for keyboard
```

```
        ldx #125      ;x-lo
        ldy #0        ;x-hi
        lda #49       ;y

        stx $18
        sty $19
        sta $05

        lda #15       ;Maximum number
        sta $06

        ldx #<fnmbr   ;address for name
        ldy #>fnmbr
        stx $02
        sty $03

        ldx #<cr4     ;Keyboard job
        ldy #>cr4
        stx ttjob
        sty ttjob+1

        jsr input     ;Input names

rtnb rts




cr4  lda fnmbr+2     ;":" used?
     cmp #$3a
     bne cr5         ;Syntax error

     jsr mouseon

     ldx #<d         ;get icon (Info)
     ldy #>d         ;from
     jsr getinfo
     bne cr6         ;error

     ldx #$4d        ;clear info-sector
clin sta $8100,x     ;a=0 !
     inx
     bne clin

cr8  lda m1,x        ;Enter class
     sta $814d,x     ; ("FONT XX")
     beq cr7
     inx
     bne cr8

cr7  lda #$20        ;Blank
     sta $814d,x
     lda fnmbr       ;Enter number
     sta $814e,x
```

```
          lda fnmbr+1
          sta $814f,x
          lda #0           ;End with a null
          sta $8150,x

          lda fnmbr+1      ;File number
          and #$0f         ;(plus 1)
          pha
          lda fnmbr
          and #$0f
          tax
          pla
     cr11 dex
          bmi cr10
          clc
          adc #10          ;convert to hex
          bne cr11
     cr10 sta $8180        ;enter in info

          ldx #<fname      ;Pointer to name
          ldy #>fname
          stx $8100
          sty $8101

          ldx #$00         ;Info-data: $8100
          ldy #$81
          stx $14
          sty $15

          lda #0           ;page
          sta $16

          jsr $c1ed        ;Create file
          txa
          bne rtnb         ;Error

          ldx #<fname      ;Open file
          ldy #>fname
          stx $02
          sty $03

          jsr openrec
          txa
          bne cr6          ;Error

     app  jsr append       ;fill in: $00/$FF
          txa
          beq app
          cpx #9
          bne cr6

          jsr ttinit   ;Keyboard on

     cr6  jsr closerec

     rtnc rts
```

```
                ;**********
        as3     ;* Update *
                ;**********

                jsr $c190
                jsr update      ;Write current REC
                rts

                ;************
        as4     ;* clr mem *
                ;************

                jsr $c190

                lda ttjob       ;Not allowed
                ora ttjob+1
                beq clr3

                jsr nulfont     ;font reset

                rts


        nulfont

                ldx #0          ;Font head
        cr1     lda default,x
                sta memo,x
                inx
                cpx #8
                bne cr1

                lda #0          ;Tab 1 set
                tax
                tay
        cr2     tya
                sta memo+8,x
                inx
                lda #0          ;hi
                sta memo+8,x
                inx
                iny
                cpy #98
                bne cr2

                ;* Delete point data *

                jsr fonton

        clr1    ldx tab2        ;Start of data
                ldy tab2+1
                stx $04
                sty $05
```

```
        lda #0          ;5 kByte
        sta $02
        lda #20
        sta $03

        jsr $c178       ;clr

clr3 rts


        ;********
        ;* Quit *
        ;********

bp1  jsr $c190

        lda #0
        sta flag2

        jsr close

        jsr $c1e1       ;new disk

        ldx #$07        ;Return register
r2      lda rett,x
        sta p1,x
        dex
        bpl r2

qu1  ldx #$c2           ;Return swap
        ldy #$3e
        stx $849c
        sty $849b

        rts

        ;************
        ;* New Disk *
        ;************

bp2  jsr $c190

        lda #0
        sta flag2

        jsr close

        jsr $c1e1       ;new disk

init lda #5             ;Maximum number of files
        sta $11

        txa
        bne lop1        ;Error!

        ldx #<d         ;Address for names
```

```
        ldy #>d
        stx $0e
        sty $0f

        lda #8          ;File type being searched for
        sta $10         ;is "font file"

        lda #0          ;No selection after names
        sta $16         ;in info sector
        sta $17

        lda #0
        sta flag1       ;Files found

        jsr $c23b       ;Search for files
        txa
        beq lop1        ;O.K.

nd1     lda #$ff        ;"no files"
        sta flag1
lop3    lda t1,x
        sta d,x         ;enter
        inx
        cpx #12
        bne lop3
        lda #1
        bne ok3         ;1st menu point

lop1    lda #5          ;compute number
        sec
        sbc $11
        beq nd1         ;No file

ok3     tax
        ora #$80        ;Vertical menu
        sta anzf

        lda as1
lop2    dex             ;Get menu length from it
        bmi ok2
        clc
        adc #14         ;Set length
        bne lop2
ok2     sta yuntf       ;of a menu


        rts



gline ;Move to line below

        ldx memo        ;Current position
        inx
        cpx memo+3      ;down already?
        bne gl1         ;no
```

122

```
        ldx #0          ;1 point over 0
gll     stx memo
        jsr fonton      ;init.
        rts



;*******************
;*** Subroutines ***
;*******************

pixel   ;Parameter:
        ;Akku :  ASCII-character
        ;Carry:  = 0, Then read
        ;        = 1, Then insert
        ;X-Reg:  Column
        ;y-Reg:  Row
        ;(No illegal values!)

        ;Result )only after 1
        ;C = 1 , if bit is set reading):
        ;C = 0 , otherwise
        ;pl:     Address containing the bit
        ;p2:     Bit sought in address


        php             ;Store carry

        stx spalte
        sty zeile

        sec
        sbc #32
        asl a           ;2-Byte table
        tay             ;Pointer to tabl

        lda (tabl),y ;Offset-Bits
        clc
        adc spalte      ;Search for value;
        sta pl          ;lo
        iny
        lda (tabl),y ;hi
        adc #0          ;Transfer
        sta pl+1

        lda pl          ;bit being searched for
        and #$07
        eor #$07
        sta p2

        lsr pl+1        ;divide by 8;
        ror pl          ;Gives start byte
        lsr pl+1
        ror pl
        lsr pl+1
        ror pl
```

123

```
        ldx #0        ;Plus line times bytes
ok8  cpx zeile
        beq ok7

        lda p1
        clc
        adc bytes
        sta p1        ;lo
        bcc ok6
        inc p1+1      ;hi
ok6  inx
        bne ok8

ok7  lda p1
        clc            ;Plus starting address
        adc tab2       ;of data
        sta p1
        lda p1+1
        adc tab2+1
        sta p1+1

        ldy #0        ;p1 Points to byte
        ldx p2        ;p2 comtains hex value
        lda dual,x    ;Convert to bits
        sta p2

        plp            ;Read or invert?
        bcc get

        eor (p1),y    ;Turn bit
        sta (p1),y

get  and (p1),y    ;Test bit
        beq ok9        ;Not set

        sec            ;set
        rts

ok9  clc
rtn4 rts




expandx ;Parameter:
        ;$8504:    ASCII-character
        ;Result:widened by 1 bit

        ldy #192      ;New bit width +1
        lda (tab1),y ;lo
        clc
        adc #1
        sta hreg1
        iny
        lda (tab1),y ;hi
        adc #0
```

```
                sta hreg2

                ldx #3        ;Division by 8 gives
        ex1     lsr hreg2     ;necessary
                ror hreg1     ;bytes
                dex
                bne ex1
                inc hreg1     ;Round it off

                lda bytes     ;Is - width
                cmp hreg1     ;New should - width
                bcs ex2       ;is > = should: OK

                ;Insert byte

                ldy hoch      ;Line counter
                dey
                sty hreg1


        ex4     lda #127      ;Last character
                jsr $c1c9     ;Get width
                tax
                dex           ;columns
                ldy hreg1     ;rows
                lda #127
                clc
                jsr pixel     ;Get byte after p1

                ldx bytes     ;Look around
        ex3     ldy #0
                lda (p1),y
                ldy hreg1     ;in order to shift
                sta (p1),y    ;linebytes
                lda p1
                sec
                sbc #1        ;Bug
                sta p1
                lda p1+1
                sbc #0
                sta p1+1
                dex
                bne ex3

                dec hreg1     ;Line
                bne ex4

                inc memo+1    ;New value of
                jsr fonton    ;shifted bits

        ex2     ;Bit einfuegen

                lda tab2      ;Starting address: Data for
                clc                        first line
                adc bytes
                sta hreg1
```

```
          ldx breit    ;Spalte des ersten zu
          ldy #0       ;schiebenden Bits
          lda word
          clc
          jsr pixel    ;Starting address of entire
                       ;shift procedure

          lda hreg1    ;Use that to compute
          sec          ;the number of bits
          sbc p1       ;to be shifted
          sta hreg2    ;(1 too many!)

          ldy hoch
          dey
          sty hreg1    ;Line counter

ex9       ldx breit    ;Get starting address
          dex
          ldy hreg1
          lda word
          clc
          jsr pixel

          ldy #0

          ldx p2       ;Last bit of the character
          dex          ;Get mask from it
          txa
          and (p1),y   ;New character bits
          lsr a        ;to the right
          php          ;Store carry
          pha          ;Store value
          txa          ;Mask
          eor #$ff     ;Delete new character's
          and (p1),y   ;bits
          sta (p1),y
          pla
          ora (p1),y   ;Shifted entries
          sta (p1),y

          lda p1
          sta ex6+1    ;lo
          lda p1+1
          sta ex6+2    ;hi

          ldx #1
          ldy hreg2    ;Number of bytes +1
          plp

ex8       dey
          beq ex7      ;Ready

ex6       ror $ffff,x
          inx
          bne ex8
```

```
ex7   dec hreg1
      bpl ex9


      ;Adjust table 1

      lda word
      sec
      sbc #31
      asl a
      tax

ex11  inc memo+8,x
      bne ex10
      inc memo+9,x

ex10  inx
      inx

      cpx #194
      bcc ex11

ex5   rts



komprx


      ;Drop bit

      lda tab2      ;Starting address: Data
      clc                       of first line
      adc bytes
      sta hreg1

      ldx breit     ;Column which should be
      dex           ;dropped
      beq ex5       ;left!

      ldy #0        ;(line)
      lda word
      clc
      jsr pixel     ;Ending address of
                    ;entire shift routine

      lda hreg1     ;Compute the number
      sec           ;of bytes that should be
      sbc p1        ;shifted
      sta hreg2     ;(1 too many!)

      ldy hoch
      dey
      sty hreg1     ;Line counter
```

```
kox5 ldx breit     ;Get starting address
     dex
     ldy hreg1
     lda word
     clc
     jsr pixel

     lda p1
     sta kox2+1   ;lo
     lda p1+1
     sta kox2+2   ;hi

     ldx hreg2     ;Number of bytes +1
     dex
     beq kox3
     clc
kox2 rol $ffff,x
     dex
     bne kox2
kox3 php           ;Store carry

     ldy #0
     ldx p2        ;Clear-bit of character
     dex           ;from which mask is taken
     txa
     and (p1),y    ;Bits of next Character
     plp
     rol a         ;store
     pha           ;to the left
     lda p2        ;bit
     asl a
     sec
     sbc #1
     eor #$ff
     and (p1),y
     sta (p1),y
     pla
     ora (p1),y
     sta (p1),y

kox4 dec hreg1
     bpl kox5


     ;Compare table 1

     lda word
     sec
     sbc #31
     asl a
     tax

kox6 lda memo+8,x
     sbc #0
     sta memo+8,x
     cmp #$ff
```

128

```
                 bne kox7
                 dec memo+9,x

        kox7 inx
                 inx

                 cpx #194
                 bcc kox6

                 rts




        expandy

                 lda memo+3
                 cmp #62      ;max-Point
                 bcs exy1

                 inc memo+3   ;Add 1

                 jsr fonton   ;Re-initialize set

                 ldx #0       ;column
                 ldy hoch     ;new line
                 dey
                 lda #32      ;Address of first
                 clc          ;data
                 jsr pixel

                 lda #0       ;Clear
                 tay
        exy2 sta (p1),y
                 iny
                 cpy bytes
                 bne exy2

        exy1 rts




        kompry

                 ldx memo+3   ;min-Point
                 cpx #1
                 beq ky1
                 dex
                 stx memo+3   ;Take 1

                 dex
                 cpx memo      ;G-Line limit?
                 bcs ky2       ;O.K.
                 stx memo

        ky2  jsr fonton
        ky1  rts
```

129

```
show ;* Display current character *

     ldx #0       ;Window data
sh2  lda wndw2,x
     sta $06,x
     inx
     cpx #6
     bne sh2

     jsr $c124    ;Fill

     ldx #195     ;Original
     ldy #0
     lda #120     ;line position
     stx $18
     sty $19
     sta $05

     lda $8504
     jsr bsout    ;output

     ldx #<buff   ;Clear buffer
     ldy #>buff
     stx $04
     sty $05

     ldx #1       ;1 Byte
     ldy #8       ;8 Pages
     stx $02
     sty $03

     jsr clrmem

     lda word     ;Get character
     jsr $c1c9    ;width
     beq ky1      ;Mark control
     sta breit    ;code
     cmp hoch
     bcs sh1      ;Too wide
     lda hoch
sh1  sta p1

     ldx #$ff     ;Determine
     lda #128     ;distance factor
ok5  inx
     sec
     sbc p1       ;(at least 1!)
     bcs ok5
     stx faktor

     lda #0
     sta zeile    ;Counter

ok12 ldx breit    ;"4" width, i.e.,:
```

```
        dex            ;columns 0-3
        stx spalte     ;counter

        ldy faktor
        lda #0         ;Starting address of
        sta dat+2      ;hi
        clc            ;Computer buffer line:
ok10    adc zeile      ;Buff + line * factor*16
        dey
        bne ok10
        sta dat+1      ;LO

        ldy #4         ;times 16
ok11    asl dat+1      ;LO
        rol dat+2      ;HI
        dey
        bne ok11

        lda #<[buff+1]
        clc            ;plus Buffer-start
        adc dat+1
        sta dat+1
        sta p4
        lda #>[buff+1]
        adc dat+2
        sta dat+2
        sta p4+1

ok19    ldx spalte     ;Column
        ldy zeile
        lda word
        clc
        jsr pixel      ;Get bit

        ;Enter in buffer

        ldy factor

dat2    php
        php
        ldx #0         ;Shift buffer line
dat1    plp
dat     ror $ffff,x
        php
        inx
        cpx #16
        bne dat1

        plp            ;Garbage
        plp            ;valid Carry

        dey
        bne dat2

        dec spalte
        bpl ok19
```

```
                ;Duplicate line

                ldx faktor
        ok22 dex
                beq ok20

                lda p4
                sta p3
                lda p4+1
                sta p3+1

                lda p3      ;Duplicate line
                clc
                adc #16
                sta p4
                lda p3+1
                adc #0
                sta p4+1

                ldy #15     ;Copy
        ok21 lda (p3),y
                sta (p4),y
                dey
                bpl ok21
                bmi ok22

        ok20 inc zeile    ;Ready?
                lda zeile
                cmp hoch
                beq ok23

                jmp ok12    ;Continue

        ok23 jsr out      ;Display buffer

                jsr border  ;Set border

                lda #$df    ;Clear button jobs
                and $39
                sta $39

                rts


                ;* Display buffer *

        out  ldy #$00     ;128 lines

        ok27 ldx #0
        ok24 lda pos,x    ;Data over position
                sta $02,x
                inx
                cpx #6
                bne ok24
```

```
        tya            ;Store
        pha
        beq ok26

ok28 inc $05           ;Y-position:Y-times +1
        lda $02        ;Adress    :Y-times +16
        clc
        adc #16
        sta $02
        bcc ok29
        inc $03
ok29 dey
        bne ok28

ok26 lda #$90
        sta ($02),y    ;Bit-Map Byte

        jsr muster     ;Output line

        pla
        tay
        iny
        cpy #128       ;All lines out?
        bne ok27       ;No!

        rts


fonton

        lda #<memo     ;Character set on
        sta $02
        lda #>memo
        sta $03
        jsr $c1cc
        rts


border

        lda pos+3      ;y-up
        sta $06

        ldx faktor     ;y-down
        clc
bo4  dex
        bmi bo3
        adc hoch
        bne bo4
bo3  sbc #0         ;-1 (in field)
        sta $07

        lda #0         ;X-Position
        sta $09        ;left hi
        sta $0b        ;right hi
```

```
          lda pos+2    ;left lo
          asl a
          asl a
          asl a
          sta $08

          ldx faktor   ;right border
     bo2  dex
          bmi bo1
          adc breit
          bne bo2

     bo1  sbc #0       ;right lo  (in field)
          sta $0a

          lda #$55
          jsr $c127    ;Draw

          lda $06      ;up
          ldx line     ;toplines -1
          clc
     bo5  adc faktor
          dex
          bpl bo5
          sbc #0       ; (in field)
          sta $18

          lda #$99     ;draw
          jsr $c118

          rts


     close

          lda #0
          sta mausjob    ;Save
          sta mausjob+1
          sta ttjob
          sta ttjob+1

          jsr update

          jsr fontoff

     up1  rts



     update

          lda flag2     ;Not necessary
          beq up1       ;or not allowed
                        ; (Button pressed)
```

```
              ldx #<fname
              ldy #>fname
              stx $02
              sty $03
              jsr openrec

              lda #127        ;Last character
              jsr $c1c9       ;Get width
              tax             ;Column
              ldy hoch        ;Last line
              dey
              lda #127
              clc
              jsr pixel       ;Get last address

              lda p1          ;length=end address
              sec             ;       minus beginning
              sbc #<[memo-1]  ;       plus one
              sta $06
              sta p1
              lda p1+1
              sbc #>[memo-1]
              sta $07
              sta p1+1

       up2    ldx #<memo      ;Starting address
              ldy #>memo
              stx $10
              sty $11

              lda hoch        ;Point height = REC #
              sta $8496

              jsr write       ;enter REC

              jsr closerec

              ldx #<fname     ;Info-sector
              ldy #>fname
              jsr getinfo     ;Get after $8100

              ldx #$fe
       less   inx
              inx             ;(x=0,2,4,...)
              lda $8182,x     ;Font-ID (lo)
              and #$1f        ;only #
              beq exist       ;Free entry
              cmp hoch        ;with current comparison
              beq exist       ;already exists
              bcc less        ;continue search

              ;*** Note: Only maximum size of 15
              ;*** points possible ( bug in info
              ;*** sector!!)

              ldy #28         ;advance
```

135

```
       shft lda $8181,y    ;Font-ID hi
            sta $8183,y
            lda $8180,y              lo
            sta $8182,y

            lda $8160,y    ;Size in bytes
            sta $8162,y
            lda $815f,y
            sta $8161,y

            dey
            dey
            inx
            inx
            cpx #28
            bcc shft

            tya
            tax



       exist ;Data entered
            ;x + $8182 from LOW ID


            lda #0         ;clear
            sta $06

            lda $8180      ;font number LOW
            asl a
            rol $06
            asl a
            rol $06
            asl a
            rol $06
            asl a
            rol $06
            asl a
            rol $06
            asl a
            rol $06
            ora hoch       ;Point height

            sta $8182,x    ;ID LOW
            lda $06
            sta $8183,x    ;ID HIGH

            lda p1         ;size lo
            sta $8161,x
            lda p1+1
            sta $8162,x    ;size hi

            ldx $8413      ;info sector
            ldy $8414
            stx $04
```

```
            sty $05

            ldx #$00
            ldy #$81
            stx $0a
            sty $0b

            jsr $c1e7       ;rewrite.

            lda #0          ;updated
            sta flag2

            rts


    ttinit

            ldx #<tt        ;Initialize keyboard
            ldy #>tt
            stx ttjob
            sty ttjob+1

            jsr fonton

            rts


    getinfo

            stx $0e
            sty $0f
            jsr $c20b       ;Look for entry
            txa
            bne get1        ;Error
            ldx #$00
            ldy #$84
            stx $14
            sty $15
            jsr $c229       ;Info after $8100
            txa
    get1 rts


            ;* Data range *


            ;window data

    wndw1 .byte 46,194,50 ,0,0  ,1 ;Position
    wndw2 .byte 55,183,195,0,255,0 ;Position

            ;Pull-Down Menu

    pull .byte 46,58,50,0,118,0 ;Position
         .byte $02                ;Number
```

```
        .byte <m1,>m1,$80,<mp1,>mp1
        .byte <m2,>m2,$80,<mp2,>mp2

mp1     .byte 59,115,50,0,97,0   ;Position
        .byte $84                  ;Number

        .byte <a1,>a1,$80,<as1,>as1
        .byte <a2,>a2,$00,<as2,>as2
        .byte <a3,>a3,$00,<as3,>as3
        .byte <a4,>a4,$00,<as4,>as4

mp2     .byte 59,87,79,0,129,0   ;Position
        .byte $82                  ;Number

        .byte <b1,>b1,$00,<bp1,>bp1
        .byte <b2,>b2,$00,<bp2,>bp2

        ;Menu name

m1      .text 'font'
        .byte 0
m2      .text 'special'
        .byte 0

a1      .text 'load'
        .byte 0
a2      .text 'create'
        .byte 0
a3      .text 'update'
        .byte 0
a4      .text 'clr mem'
        .byte 0

b1      .text 'quit'
        .byte 0
b2      .text 'new disk'
        .byte 0

        ;Sub-Eintrag: Font-Files

as1     .byte 59                   ;y-up

yuntf * = * + 1                    ;y-down

        .byte 97,0,159,0           ;x-Position

anzf  * = * + 1                    ;Number

.byte <[0*17+d],>[0*17+d],$40,<ap0,>ap0
.byte <[1*17+d],>[1*17+d],$40,<ap0,>ap0
.byte <[2*17+d],>[2*17+d],$40,<ap0,>ap0
.byte <[3*17+d],>[3*17+d],$40,<ap0,>ap0
.byte <[4*17+d],>[4*17+d],$40,<ap0,>ap0

        ;Sub-sub-entry: Point size
        ; (maximum 10 entries)
```

138

```
ppl   .byte 59              ;y-up

yuntp * = * + 1             ;y-down

      .byte 159,0,212,0      ;x-Position

anzp  * = * + 1             ;Number

.byte <[0*11+p],>[0*11+p],0,<ap1,>ap1
.byte <[1*11+p],>[1*11+p],0,<ap1,>ap1
.byte <[2*11+p],>[2*11+p],0,<ap1,>ap1
.byte <[3*11+p],>[3*11+p],0,<ap1,>ap1
.byte <[4*11+p],>[4*11+p],0,<ap1,>ap1
.byte <[5*11+p],>[5*11+p],0,<ap1,>ap1
.byte <[6*11+p],>[6*11+p],0,<ap1,>ap1
.byte <[7*11+p],>[7*11+p],0,<ap1,>ap1
.byte <[8*11+p],>[8*11+p],0,<ap1,>ap1
.byte <[9*11+p],>[9*11+p],0,<ap1,>ap1


t1    .text '* no file *'
      .byte 0

t2    .text '   x point'
      .byte 0

      ;Data for output

pos   .byte <buff,>buff,8,62,16,1

default ;Data for new font head

  .byte $00,$0c,$00,$01,$08,$00,$ca,$00

dual    ;Data for point calculation

  .byte 1,2,4,8,16,32,64,128

      ;div. help register

fnmbr  * = * + 3    ;File number
fname  * = * + 16   ;Room for filenames
flag1  * = * + 1    ;No File
flag2  * = * + 1    ;Button pressed
faktor * = * + 1    ;expansion factor
breit  * = * + 1
spalte * = * + 1
zeile  * = * + 1
maus   * = * + 2
pnt    * = * + 1
hreg1  * = * + 1    ;help registers
hreg2  * = * + 1
hreg3  * = * + 1
rett   * = * + 8
```

139

```
        ;Buffer for Editor

buff    * = * + [128*16] + 1

        ;Buffer for font names

d       * = * + 85        ;(max 5 pieces)

        ;Point size-text for buffer


p       * = * + [11*10] ;(max 10 Entries144
)


        ;Character set buffer

memo    * = * + [5*1024]

        .end
```

4.1.4 Examples with the FONT Editor

In this section we would like to offer you some help in modifying your fonts. As an example, we will go through adding German umlauts to the font University 12 point. Before you start the FONT Editor you should rename the font to something like "University.g", where the "g" indicates that the font contains the German umlauts.

We do not recommend that you copy the font since each GEOS font has a specific identifier. Two fonts with the same identifier may confuse GEOS.

We will put the umlauts on the keys "colon," "semicolon," "at-sign," and "plus." These means that some important characters will no longer be available, but the layout will correspond to that of a German typewriter. When the umlauts are finished you can put the colon on the SHIFT + comma = left bracket and the semicolon on SHIFT + period = right bracket. If you then exchange the "z" with the "y" you'll have a complete German layout.

Do you have enough free space on the disk and is University.g one of the first five fonts? Good, then you can load the German font University.g in 12 point. Before you start making changes you should just take a look at some different characters in upper and lower case. This will give you a better idea of how these look in the edit field. When you're done, press SHIFT + "o" to get an "O" on the screen.
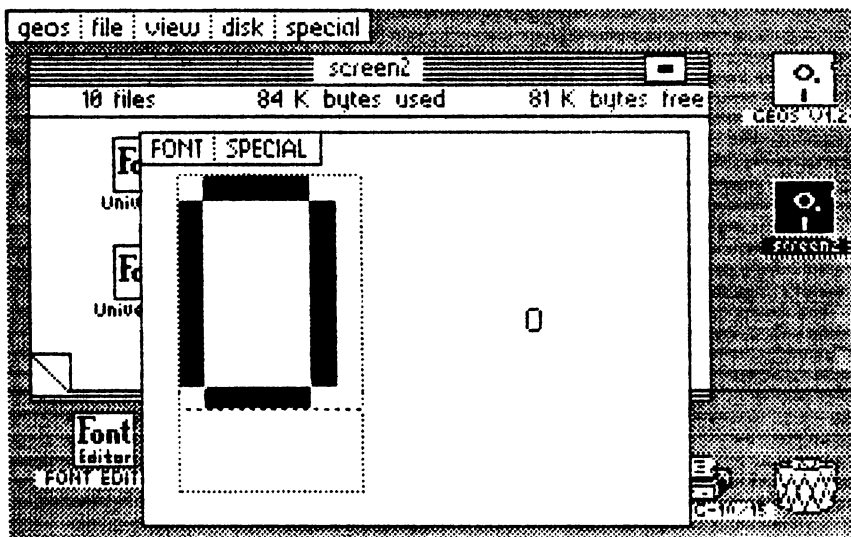
Figure 27: No room for umlaut points

You can see that there is no room above the "O" for the two dots which we have to add for a "Ö". This means that we cannot use the characters o, a, and u directly as patterns for our umlauts. To make the character look good we need at least two lines above the characters. This leaves one line free between the character and the dots.

Let's start with the lower case "ö". We will first look at the plain "o". It occupies seven lines and is five columns wide.
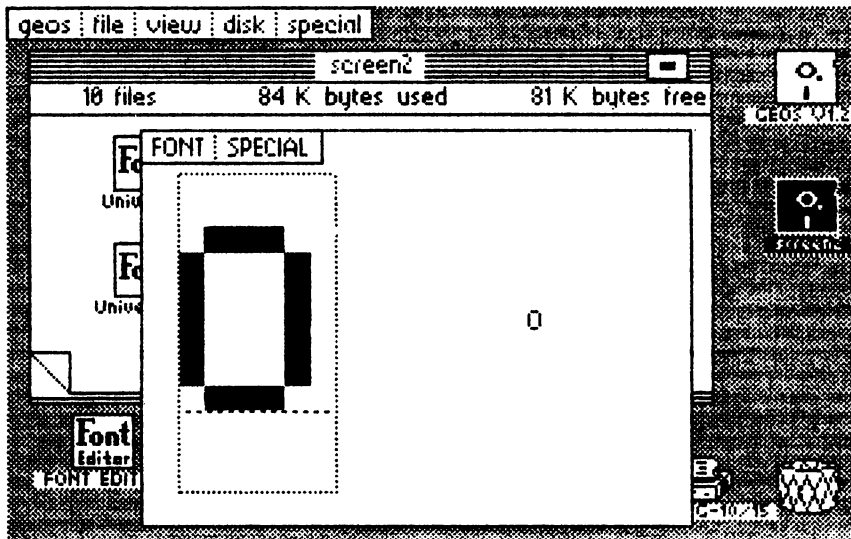
Figure 28: The o serves as a model

The edit field is six columns wide because every character needs a blank column on the right to separate it from the next character in the text. Now we can begin to convert the colon into an "ö". First we will make the editor field the same width as the "o". Click four times to the right of the edit field. You can quickly compare what you're working on with the "o" by pressing the "o" key and then going back to the colon.

Now we can start to create the "ö". It should start one line lower than the "o". The easiest thing to do is to take a look at the "o", move the arrow to the second line (the first blank line below the upper bar, and then get the colon back in the edit field. The character is to begin where the arrow is located. This is precisely the line in which the dot is already. Erase it and set the first dot next to it on the right.

Since the upper bar of the "o" is three points wide, set two more points on the right. A quick look at the "o": The vertical bars must be in columns 1 and 5 in the next line.

Draw these bars to the second line from the bottom and erase the rest of the colon (the set point in the lower left). It should look something like this:
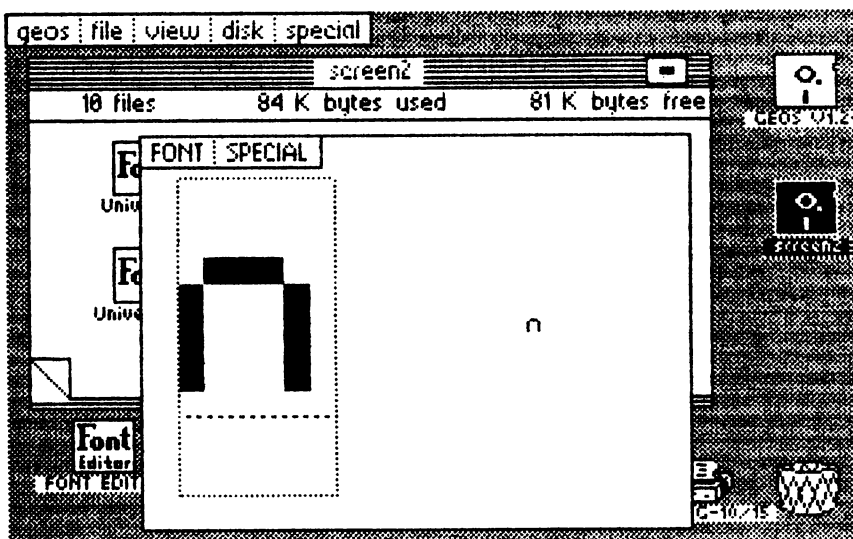
Figure 29: The half done ö

If you draw a bar on the bottom corresponding to the one on the top, the reduced "o" is done. Now you just have to set the two dots for the umlaut and the "ö" is ready. Leave a point free between the two and from the upper bar of the "o". Now your new "ö" should look like this:
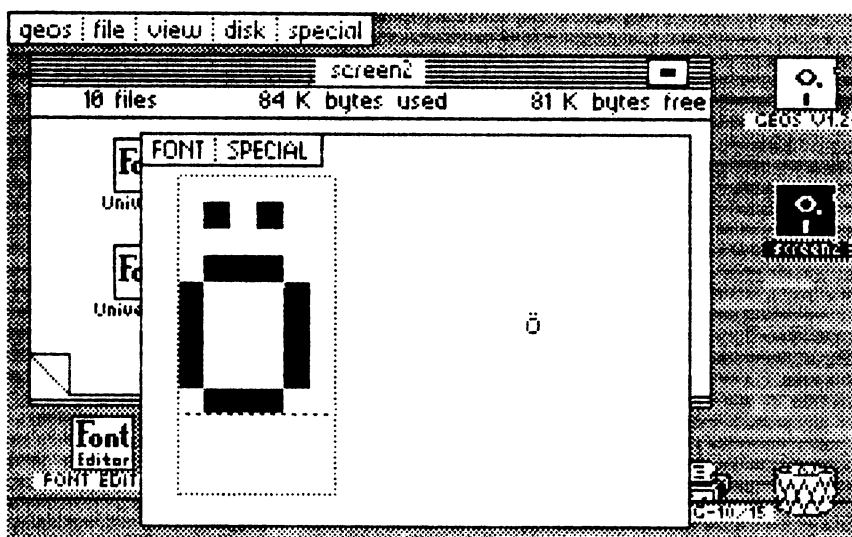
Figure 30: The complete ö

First save your work by clicking UPDATE. It would be a shame if everything you've done so far was lost for some reason.

Now you can create the additional characters. The next one will be the lower case "ä", which we will put on the semicolon. We widen this until it has the same width as the "a". Then we take the "a" as a pattern, but make it one line lower. The result should look something like this:
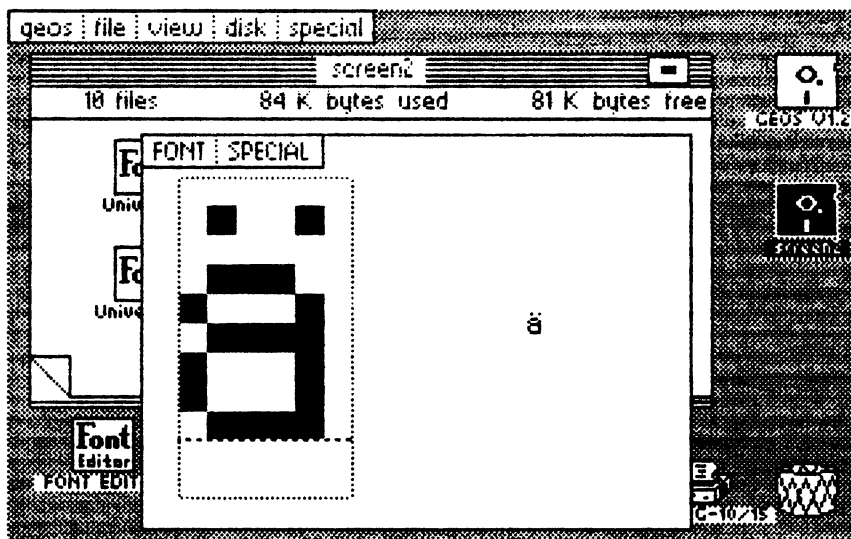
Figure 31: The complete ä

Now that we're on such a roll we also put the lower case "ü" on the at-sign (@) to the right of the p. The "ü" is particularly easy to create.
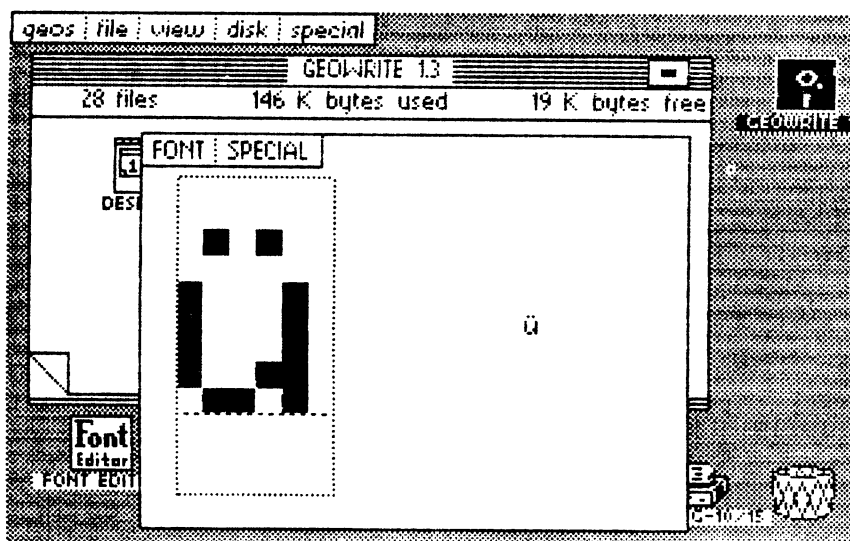
Figure 32: The complete ü

Now we've really earned ourselves a break and we should save our current work with UPDATE. Naturally you can also exit the FONT Editor and try out the new character set in geoWrite.

Next we create the missing "ß". We will assign it to the plus key next to the "0". We will allow two points for the descender. This is how our "ß" turned out:
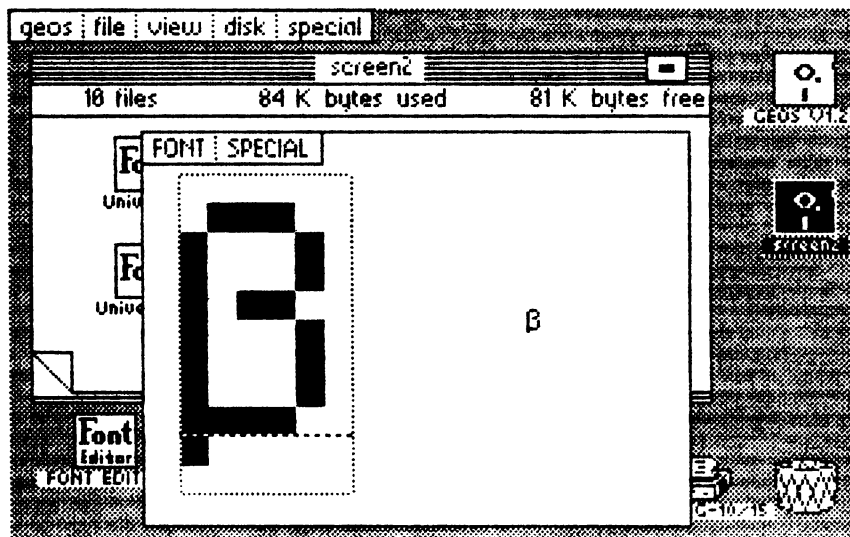
Figure 33: The ß in University 12 point

145

Now we have the three upper-case umlauts left. We will start with the "Ö". We will start it two points lower than the "O" so that there is enough room for the dots. Otherwise it is easy to make the "Ö" if you switch back and forth from the "O".
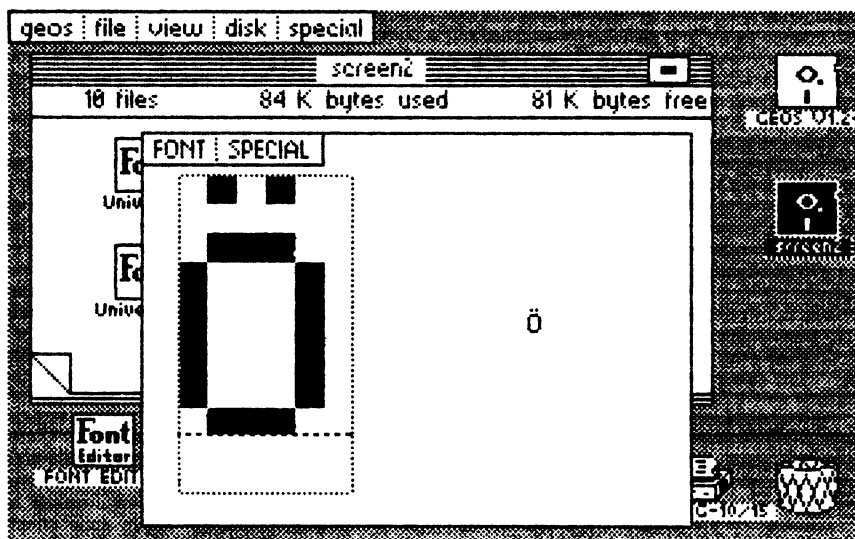
Figure 34: The complete Ö

For the next character, the "Ä" we just have to leave one row open at the top because the "A" only has one single point in the middle. The two dots are easily visible without the one-line gap. The "Ä" looks like this:
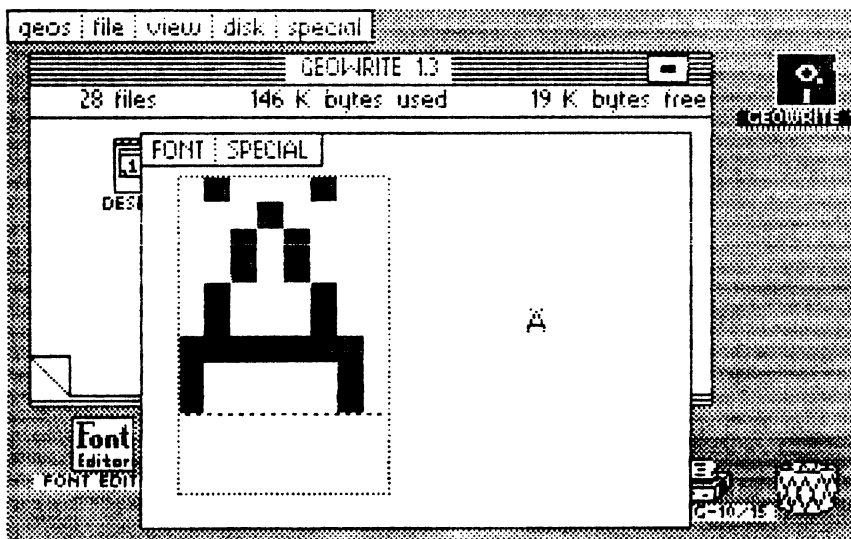
Figure 35: The complete Ä

When you get to the "Ü" you will run into a problem. When you press SHIFT + "at-sign" you won't see some unimportant characters, you'll see a lower-case "ü". Pressing harder on the SHIFT key won't change matters any. In GEOS the "at-sign" has the same code with or without the SHIFT key, so we had to find an alternative.

When you press COMMODORE + "at-sign" you get the same character as when you press SHIFT + 7, so we will put the "Ü" on this key. Later when you write documents you will have to remember to press COMMODORE + "at-sign" for a "Ü", and SHIFT + character for the other characters. There's nothing we can do about this in GEOS. Here's the "Ü":
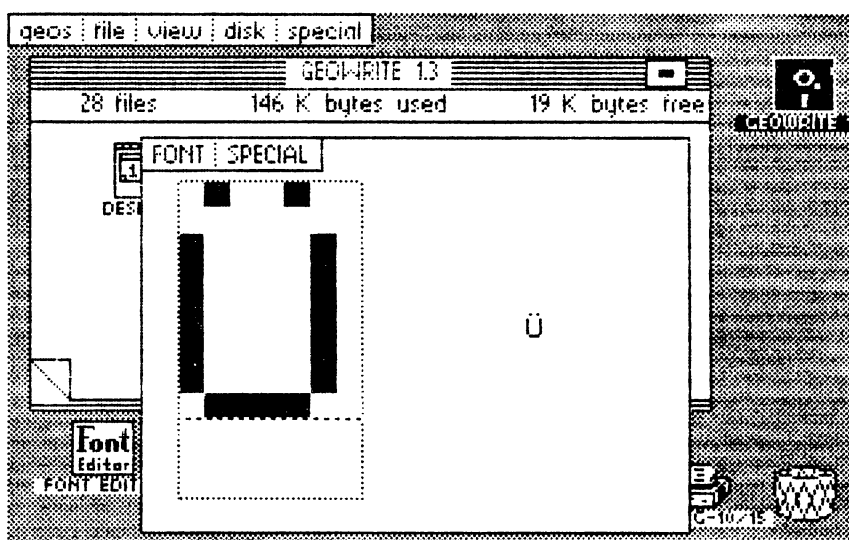
Figure 36: The complete Ü

Before you exit the FONT Editor and rush off to write your first geoWrite document with umlauts, you must not forget to save the result with UPDATE. Then you can have it with geoWrite or read one last tip that will make your work with geoWrite easier.

One thing that disturbed us about geoWrite, in addition to the missing German umlauts, is that right-justification is not possible. This isn't necessary for every document, certainly, but we would like to be able to do it when needed. Previously we had tried to imitate it by putting an extra space between some words. This has two disadvantages, however.

First of all, a space is too wide, so the right margin can still be off by several points. Second, the gap between the words becomes too large and the effect is rather crude as a result. With our FONT Editor we now have a way to get around this.

We made a space that is only one point wide. We added it to the font because we still need the regular space, of course. We would have liked to have assigned the narrow space to SHIFT + space, but this isn't any different than a regular space. Here too the two key combinations have the same code. We put this character on COMMODORE + asterisk.

We can now move text one point to the right by pressing this key combination. You can use this character between several different words until the right margin is flush with the other lines and it will be accurate to the point.

148

We have one request in regard to your work with the FONT Editor. Please do not use the function keys and do not press the CTRL key together with another key.

In conclusion we want to show what is possible with the FONT Editor and hope that it makes the job of typing it in somewhat more bearable.
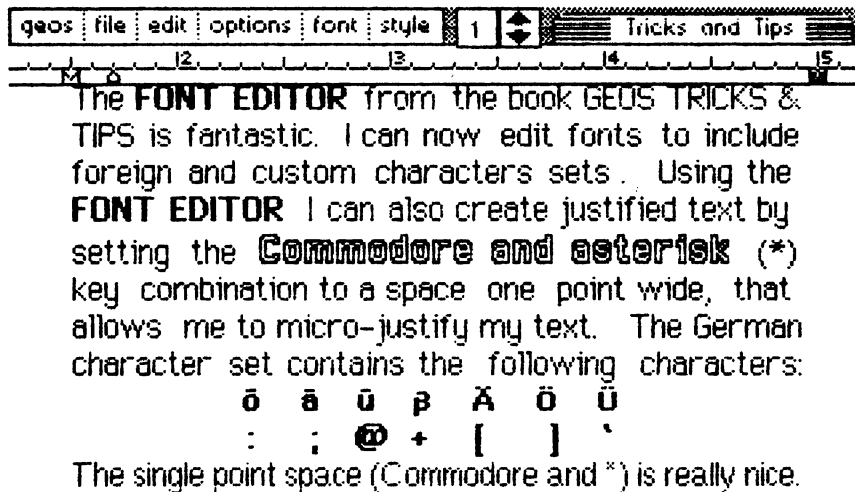
Figure 37: This shows what can be done with the FONT Editor

4.2 The machine language monitor (EDMON)

If you want to get to know the internals of GEOS, you'll need some tools. After we had worked with a conventional monitor for some time we noticed more and more how important a GEOS-based monitor is. It should be loadable at any time, may not change any parts of programs in memory, and should have all of the standard commands.

This section contains precisely these tools that we use constantly for our own work. We have improved it to make it as bomb-proof and easy to use as possible. Despite this, you should not use EDMON when you have irreplaceable data in the computer. Working with a machine language monitor is always potentially hazardous to your data, so you should always use working disks instead of your program disks when using EDMON.

Although EDMON is a very short program for the power it has (it occupies less than 4K), you still have to enter a fairly large number of DATA statements. If you don't feel up to the job of entering the data and then finding all your mistakes, you can order the optional program disk for this book.

The following section contains the BASIC program which will produce the complete EDMON after you correctly enter and run it. EDMON can then be started directly from GEOS. Be sure to save the BASIC program before you run it and click validate before you start EDMON, so that the blocks it occupies will be marked as occupied on the disk.

4.2.1 Listing of EDMON

```
10 open 1,8,15,"i:0":rem error channel
12 restore:print chr$(147):rem clr home
14 rem * calculate check sum*
16 for i=1 to 3950
18 read a:s=s+a
20 print chr$(19);3950-i;chr$(157);" ";
22 next:restore
24 if s<>425984 then print "Error in DATA !":stop
26 rem * file output *
28 input "Filename";f$
30 open 2,8,2,f$+",u,w":gosub 110
32 for i=1 to 3950:read a
34 print#2,chr$(a);:next:gosub 110
36 close2
38 rem * info-sector read *
40 a=1:b=0
42 t=a:s=b:print#1,"b-a:0";t;s
44 input#1,n,t$,a,b:if a=18 then a=19:b=0
46 if n>0 then 42
48 open 3,8,3,"#0":print#3
50 print#1,"u1";3;0;t;s:rem sector write
52 print#3,chr$(0)chr$(255)chr$(3)chr$(21)chr$(191);:rem info kopf
54 fori=1 to 63:print#3,chr$(85);:next:    rem stripped-icon definition (!)
56 print#3,chr$(130)chr$(5)chr$(0);
58 print#3,chr$(0)chr$(96);:rem start
60 print#3,chr$(224)chr$(113);:rem end
62 print#3,chr$(0)chr$(96); :rem init
64 fori=1 to 179
66 print#3,chr$(0);:rem rest
68 next
70 print#1,"u2";3;0;t;s:rem write
72 rem * in directory entry *
74
76 print#3
78 a$=chr$(18):b$=chr$(1):rem directory
80 a=asc(a$):b=asc(b$):print#1,"u1";3;0;a;b
82 get#3,a$,b$:rem next sector
84 for i=0 to 7:rem number records
86 print#1,"b-p";3;i*32+5
88 for j=1 to len(f$)
90 get#3,w$:if w$<>mid$(f$,j,1) then k=0:goto 94
92 k=k+1
94 next j
96 if k=len(f$) then 100:rem found
98 next i:gosub 110:goto 80
100 print#1,"b-p";3;i*32+21
102 print#3,chr$(t)chr$(s)chr$(0)chr$(5);
104 print#1,"u2";3;0;a;b:rem dir back
106 close1:print"o.k.":end
108 rem ****************************
110 input#1,n,t$:if n then print$:stop
112 return
```

```
114 rem
115 data32,156,193,32,202,97,162,29,160,96,142,163,132,140,164,132,162,227
116 data160,96,142,167,132,140,168,132,76,49,98,173,4,133,170,169,128,13
117 data182,132,141,182,132,138,201,32,176,102,201,13,208,3,76,188,98,201
118 data17,208,15,174,116,111,224,14,240,4,238,116,111,96,32,181,96,96,201
119 data30,208,11,174,115,111,224,39,240,3,238,115,111,96,201,16,208,9,174
120 data116,111,240,3,206,116,111,96,201,8,208,9,174,115,111,240,3,206,115
121 data111,96,201,29,208,17,174,115,111,240,11,206,115,111,169,32,32,148
122 data96,206,115,111,96,201,18,208,3,76,40,98,201,19,208,244,76,202,97
123 data72,32,179,97,174,116,111,202,48,6,32,188,97,76,155,96,172,115,111
124 data104,145,144,174,115,111,224,39,240,3,238,115,111,96,32,179,97,162
125 data14,165,144,133,146,165,145,133,147,32,188,97,160,39,177,144,145,146
126 data136,16,249,202,208,233,169,32,160,39,145,144,136,16,251,169,64,13
127 data182,132,141,182,132,96,44,182,132,80,68,120,162,0,32,60,193,165,12
128 data133,144,165,13,133,145,138,72,24,105,10,170,32,60,193,162,40,160
129 data0,177,14,145,144,165,14,24,105,8,133,14,165,15,105,0,133,15,165,144
130 data24,105,8,133,144,165,145,105,0,133,145,202,208,223,104,170,232,224
131 data150,144,192,88,169,27,32,69,193,169,11,32,69,193,169,13,32,69,193
132 data32,179,97,162,0,236,116,111,240,13,169,13,32,163,97,32,188,97,232
133 data224,15,208,238,160,0,204,115,111,208,10,165,24,141,190,132,165,25
134 data141,191,132,177,144,201,127,240,6,32,163,97,76,134,97,140,117,111
135 data152,10,10,24,109,117,111,72,169,32,32,163,97,104,197,24,176,245,133
136 data24,200,192,40,208,201,165,5,56,229,41,141,192,132,169,0,141,182,132
137 data173,180,132,41,128,9,1,141,180,132,96,142,110,111,140,111,111,32
138 data69,193,174,110,111,172,111,111,96,162,136,160,111,134,144,132,145
139 data96,165,144,24,105,40,133,144,165,145,105,0,133,145,96,169,160,133
140 data47,169,0,174,15,192,224,19,176,2,169,30,141,135,111,32,75,193,169
141 data9,133,38,165,41,32,192,193,32,179,97,162,14,160,39,169,32,145,144
142 data136,16,251,32,188,97,202,16,241,162,0,189,103,111,149,6,232,224,6
143 data208,246,169,0,32,57,193,32,36,193,173,104,111,133,24,169,0,133,8
144 data133,9,169,1,133,11,169,63,133,10,169,255,32,24,193,169,0,141,115
145 data111,141,116,111,96,169,79,141,175,132,169,98,141,176,132,104,104
146 data173,135,111,240,12,104,141,97,136,104,141,98,136,186,142,99,136,0
147 data216,141,2,2,140,5,2,142,4,2,173,11,136,141,3,2,104,170,56,233,1,141
148 data1,2,104,72,233,0,141,0,2,138,72,173,254,255,141,8,2,173,255,255,141
149 data7,2,186,142,6,2,142,9,2,88,224,80,176,17,162,64,32,12,102,162,132
150 data154,169,255,72,169,78,72,76,124,98,32,69,100,162,66,169,42,32,13
151 data100,169,114,208,47,169,0,133,139,174,9,2,154,169,46,162,13,32,13
152 data100,32,155,194,76,195,193,32,158,194,169,0,141,114,111,32,179,97
153 data174,116,111,202,48,6,32,188,97,76,202,98,32,255,98,201,0,240,213
154 data201,46,240,245,201,32,240,241,162,14,221,32,110,240,6,202,16,248
155 data76,79,102,134,180,138,10,101,180,170,189,29,99,72,189,28,99,72,96
156 data140,111,111,172,114,111,192,40,240,12,238,114,111,177,144,201,127
157 data208,5,169,32,44,169,0,172,111,111,96,44,214,104,44,44,105,44,20,105
158 data44,80,105,44,106,105,44,155,105,44,207,105,44,233,105,44,239,105
159 data44,30,106,44,246,106,44,109,106,44,204,107,44,197,104,44,43,106,142
160 data112,111,140,113,111,201,13,208,27,169,128,141,182,132,32,227,96,169
161 data0,141,115,111,32,57,96,44,182,132,80,3,32,227,96,76,112,99,32,32
162 data96,174,112,111,172,113,111,96,32,255,98,201,0,240,2,56,96,169,32
163 data24,96,32,125,104,176,5,32,125,104,144,48,32,195,99,10,10,10,10,141
164 data0,1,32,125,104,144,229,32,195,99,13,0,1,56,96,32,119,99,32,132,99
165 data144,13,72,32,137,99,144,6,133,128,104,133,129,96,104,96,32,119,99
166 data32,119,99,24,96,201,58,8,41,15,40,144,2,105,8,96,32,165,99,144,27
167 data165,128,133,130,165,129,133,131,32,165,99,162,2,181,127,72,181,129
168 data149,127,104,149,129,202,208,243,96,169,63,76,79,102,162,1,181,127
169 data72,181,128,32,254,99,104,72,74,74,74,74,32,22,100,170,104,41,15,32
```

152

```
170 data22,100,72,138,32,72,99,104,76,72,99,24,105,246,144,2,105,6,105,58
171 data96,165,128,141,1,2,165,129,141,0,2,96,133,138,160,0,32,66,100,177
172 data128,32,254,99,32,75,100,198,138,208,241,96,32,66,100,169,127,44,169
173 data13,32,72,99,96,230,128,208,6,230,129,208,2,230,139,96,152,72,32,69
174 data100,104,162,46,32,13,100,76,63,100,32,137,99,144,11,162,0,129,128
175 data193,128,240,3,76,238,99,32,75,100,198,138,96,169,3,133,128,169,2
176 data133,129,169,4,96,72,32,66,100,104,162,7,133,138,169,24,6,138,42,32
177 data72,99,202,48,8,152,240,242,32,66,100,208,237,96,133,138,162,0,165
178 data138,61,59,111,93,39,111,240,7,232,224,20,144,241,162,14,189,79,111
179 data72,74,74,74,74,170,104,41,15,201,14,96,160,68,32,86,100,32,243,99
180 data32,66,100,160,0,177,128,32,162,100,134,138,32,66,100,177,128,32,254
181 data99,200,198,138,208,243,32,66,100,32,63,100,200,192,4,144,245,160
182 data0,177,128,162,31,221,47,110,240,26,202,16,248,162,23,177,128,61,103
183 data110,93,79,110,240,6,202,16,243,76,198,101,138,24,105,32,170,138,133
184 data138,10,101,138,170,160,3,189,127,110,32,72,99,232,136,208,246,177
185 data128,32,162,100,176,222,168,169,127,32,72,99,192,8,240,37,192,9,176
186 data38,169,36,192,2,208,5,169,35,32,72,99,169,36,192,0,208,2,169,127
187 data192,1,208,2,169,65,192,5,240,4,192,6,208,7,169,40,32,72,99,169,36
188 data32,72,99,192,12,208,11,32,209,101,138,32,254,99,152,76,254,99,202
189 data240,17,202,240,7,160,2,177,128,32,254,99,160,1,177,128,32,254,99
190 data160,0,177,128,32,162,100,201,3,240,39,201,4,240,32,201,5,240,17,201
191 data6,240,21,201,10,240,23,201,11,240,16,201,8,240,4,96,32,191,101,169
192 data41,76,72,99,32,180,101,169,89,44,169,88,162,44,76,13,100,162,2,169
193 data63,32,72,99,202,16,250,96,166,129,160,1,177,128,16,1,202,24,105,2
194 data144,1,232,24,101,128,144,1,232,168,169,130,56,96,160,0,177,128,32
195 data162,100,176,51,201,12,240,217,201,3,138,72,200,177,128,72,202,202
196 data240,4,200,177,128,170,104,168,104,96,32,69,100,169,63,32,72,99,32
197 data25,102,162,43,189,121,109,8,41,127,32,72,99,232,40,16,243,96,162
198 data73,32,12,102,162,49,32,25,102,32,243,99,76,167,98,162,53,32,12,102
199 data48,246,32,119,99,32,132,99,144,9,133,133,32,137,99,133,132,176,88
200 data169,63,32,72,99,76,167,98,32,206,99,144,243,32,62,102,32,119,99,201
201 data45,240,33,201,43,208,205,165,132,170,165,133,168,165,130,229,128
202 data133,132,165,131,229,129,133,133,138,56,229,132,133,132,152,229,133
203 data133,133,165,132,197,128,165,133,229,129,144,22,165,130,229,128,170
204 data165,131,229,129,168,138,24,101,132,133,132,152,101,133,133,133,56
205 data96,160,0,177,128,32,162,100,32,75,100,202,208,250,96,72,165,130,197
206 data128,165,131,229,129,144,4,165,139,240,3,76,167,98,104,96,160,0,162
207 data0,176,18,32,182,102,161,128,145,132,32,75,100,200,208,243,230,133
208 data24,144,238,32,182,102,177,130,129,132,165,130,208,2,198,131,198,130
209 data165,132,208,2,198,133,198,132,24,144,230,32,119,99,32,119,99,133
210 data180,162,0,134,128,134,129,201,36,240,85,201,37,240,62,201,33,208
211 data80,32,119,99,201,58,176,43,233,47,144,37,168,6,128,38,129,165,128
212 data166,129,10,38,129,10,38,129,24,101,128,144,1,232,133,128,152,24,101
213 data128,133,128,138,101,129,133,129,76,22,103,105,48,96,32,119,99,201
214 data32,208,3,32,119,99,74,73,24,208,7,38,128,38,129,76,73,103,73,24,42
215 data96,32,119,99,170,201,71,176,30,233,47,144,26,201,10,144,6,201,17
216 data144,18,233,7,162,3,38,128,38,129,202,16,249,5,128,133,128,76,99,103
217 data138,96,169,0,162,2,157,10,2,202,16,250,162,15,6,128,38,129,248,160
218 data2,185,10,2,121,10,2,153,10,2,136,16,244,202,16,234,216,162,5,32,200
219 data103,9,48,157,0,1,202,16,245,162,4,189,0,1,201,48,208,3,202,208,246
220 data96,169,16,14,12,2,46,11,2,46,10,2,42,144,244,96,32,167,194,201,22
221 data208,248,76,167,98,169,0,141,115,111,169,68,32,92,100,76,203,100,162
222 data3,180,128,181,132,148,132,149,128,202,16,245,96,196,132,138,229,133
223 data144,11,24,152,229,134,138,229,135,176,2,24,96,56,96,152,72,138,72
224 data164,128,166,129,32,252,103,104,170,104,168,96,152,56,229,128,72,138
225 data229,129,168,24,104,105,126,176,1,136,200,208,6,56,233,128,24,200
```

```
226 data96,162,93,32,12,102,162,49,32,25,102,32,243,99,56,96,24,152,101,136
227 data168,138,101,137,170,96,24,200,96,32,139,103,234,189,0,1,32,72,99
228 data202,16,247,96,32,119,99,162,7,32,119,99,201,32,240,249,74,73,24,208
229 data30,38,138,202,16,239,165,138,56,96,32,119,99,201,48,144,15,201,58
230 data144,243,201,65,144,7,41,223,201,71,144,233,24,96,160,0,32,119,99
231 data144,42,201,32,240,247,201,34,240,15,32,128,104,32,140,99,144,23,145
232 data128,32,75,100,208,228,32,119,99,144,14,201,34,240,219,145,128,32
233 data75,100,208,240,76,79,102,96,32,165,99,32,148,104,160,119,32,86,100
234 data32,243,99,76,167,98,162,0,32,25,102,32,69,100,162,46,169,82,32,13
235 data100,32,66,100,173,0,2,32,254,99,173,1,2,32,254,99,32,66,100,173,7
236 data2,32,254,99,173,8,2,32,254,99,173,2,2,160,255,32,133,100,32,122,100
237 data32,43,100,76,167,98,32,206,99,32,182,102,32,215,103,160,77,32,86
238 data100,32,243,99,169,8,32,43,100,240,235,32,165,99,144,54,32,32,100
239 data32,165,99,144,46,165,128,141,8,2,165,129,141,7,2,32,99,104,144,31
240 data141,2,2,32,122,100,208,10,32,187,99,32,168,99,144,15,169,8,133,138
241 data32,119,99,32,100,100,208,248,76,167,98,76,79,102,32,206,99,176,22
242 data169,255,133,130,133,131,208,14,32,167,194,201,0,240,249,201,22,208
243 data3,76,167,98,169,5,133,180,198,180,48,234,32,215,103,32,182,102,32
244 data198,100,32,168,102,240,238,32,165,99,144,44,32,119,99,32,132,99,144
245 data18,168,32,162,100,152,160,0,145,128,200,202,240,5,32,132,99,176,245
246 data32,225,103,32,69,100,32,168,102,32,225,103,162,10,142,115,111,96
247 data76,79,102,32,206,99,144,248,32,119,99,32,132,99,144,240,162,0,32
248 data182,102,129,128,32,75,100,24,144,245,32,87,102,76,202,102,32,165
249 data99,144,3,32,32,100,174,6,2,154,120,173,7,2,141,21,3,173,8,2,141,20
250 data3,173,0,2,72,173,1,2,72,173,2,2,72,173,3,2,174,4,2,172,5,2,64,162
251 data194,160,62,142,156,132,140,155,132,76,167,98,32,206,99,32,238,103
252 data162,11,169,2,134,128,133,129,32,148,104,165,128,24,233,11,144,39
253 data141,10,2,32,238,103,32,182,102,32,215,103,172,10,2,177,128,217,11
254 data2,208,11,136,16,246,162,85,32,25,102,32,243,99,32,75,100,24,144,223
255 data76,79,102,169,0,170,157,0,129,232,208,250,32,116,107,144,239,32,132
256 data99,144,234,240,232,201,13,176,228,141,69,129,169,130,141,68,129,32
257 data206,99,144,215,166,128,164,129,142,71,129,140,72,129,142,75,129,140
258 data76,129,166,130,164,131,142,73,129,140,74,129,162,118,160,111,142
259 data0,129,140,1,129,162,3,160,21,142,2,129,140,3,129,169,191,141,4,129
260 data162,63,169,85,157,4,129,202,208,250,169,0,133,22,162,129,133,20,134
261 data21,32,237,193,138,240,19,201,3,208,4,162,144,208,8,138,9,48,141,30
262 data110,162,160,32,12,102,76,167,98,169,0,174,135,111,157,94,136,32,116
263 data107,144,109,162,118,160,111,134,14,132,15,32,11,194,138,208,215,32
264 data132,99,144,70,201,0,240,61,201,1,208,82,174,1,132,172,2,132,134,4
265 data132,5,162,255,32,162,107,176,24,162,0,160,128,134,10,132,11,32,228
266 data193,173,2,128,56,233,2,170,173,3,128,233,0,168,134,16,132,17,169
267 data255,133,6,133,7,32,255,193,76,104,107,162,0,32,162,107,162,0,160
268 data132,134,20,132,21,32,17,194,138,240,3,76,233,106,76,167,98,76,79
269 data102,162,0,32,119,99,144,37,201,32,240,245,201,34,208,29,32,119,99
270 data201,34,240,15,157,118,111,232,224,16,208,241,32,119,99,201,34,208
271 data7,169,0,157,118,111,56,96,24,96,32,165,99,144,34,138,208,6,166,128
272 data165,129,176,9,165,128,233,2,170,165,129,233,0,172,135,111,153,96
273 data136,138,153,95,136,169,1,153,94,136,56,96,76,79,102,32,119,99,144
274 data248,201,32,240,247,32,142,99,32,171,99,144,236,32,119,99,176,9,32
275 data66,100,32,242,100,76,55,109,201,32,240,238,41,223,141,12,2,32,119
276 data99,41,223,141,11,2,32,119,99,41,223,141,10,2,169,57,133,138,198,138
277 data48,189,162,2,165,138,10,101,138,168,189,10,2,217,127,110,208,236
278 data200,202,16,244,166,138,142,10,2,162,4,169,32,157,11,2,202,16,250
279 data162,4,32,119,99,144,88,201,32,240,247,201,35,208,49,32,125,104,176
280 data17,201,39,208,6,32,119,99,24,144,23,201,36,208,10,32,125,104,32,140
281 data99,176,11,144,7,201,37,208,0,32,102,104,144,106,133,130,162,2,134
```

154

```
282 data180,208,65,32,119,99,201,36,240,249,157,11,2,32,128,104,144,13,32
283 data140,99,144,14,164,130,132,131,133,130,169,0,157,11,2,202,16,222,169
284 data12,133,180,198,180,48,81,165,180,10,10,101,180,105,4,168,162,4,189
285 data11,2,217,61,109,208,233,136,202,16,244,160,0,185,79,111,41,15,197
286 data180,240,50,200,192,20,144,242,165,180,208,4,230,180,208,232,201,7
287 data208,3,24,144,21,201,9,208,23,173,10,2,201,8,176,16,234,164,130,166
288 data131,32,32,104,133,130,169,12,133,180,144,198,76,79,102,174,10,2,224
289 data32,144,4,189,71,110,44,169,255,73,255,57,39,111,29,47,110,32,162
290 data100,197,180,208,178,202,240,15,202,240,6,160,2,165,131,145,128,160
291 data1,165,130,145,128,160,0,165,138,145,128,32,162,100,138,24,101,128
292 data133,128,144,2,230,129,32,69,100,169,97,32,92,100,32,243,99,169,10
293 data141,115,111,96,32,32,32,32,32,32,32,32,32,65,32,32,32,0,35,32,32
294 data88,44,0,32,32,89,44,0,41,88,44,0,40,89,44,41,0,40,32,32,32,32,0,32
295 data41,0,0,40,32,32,32,0,0,32,88,44,0,0,32,89,44,0,0,13,127,127,127,127
296 data80,67,127,127,73,82,81,127,127,78,127,86,127,35,127,66,127,68,127
297 data73,127,90,127,67,127,65,67,127,88,82,127,89,82,127,83,208,104,133
298 data32,69,82,82,79,210,32,73,78,160,77,73,83,83,73,78,71,32,43,47,173
299 data76,79,87,32,83,84,65,67,203,73,76,76,69,71,65,76,32,67,79,68,197
300 data13,46,70,79,85,78,68,160,66,82,65,78,67,200,90,69,82,79,32,80,65
301 data71,197,79,85,84,32,79,70,32,77,69,77,79,82,217,73,76,76,69,71,65
302 data76,32,76,73,78,69,32,78,85,77,66,69,210,76,79,65,196,68,73,83,75
303 data32,79,82,32,68,73,82,32,70,85,76,76,204,73,47,79,32,35,183,160,114,82
304 data109,77,100,68,102,116,103,120,108,115,97,119,104,144,176,240,48,208
305 data16,80,112,0,24,216,88,184,202,136,232,200,234,72,8,104,40,64,96,170
306 data168,186,138,154,152,56,248,65,129,225,34,1,66,160,162,161,193,2,33
307 data97,76,132,134,98,230,198,224,192,36,32,120,227,227,227,227,227,227
308 data227,227,227,227,227,227,227,223,231,231,227,231,231,243,243,247,255
309 data255,66,67,67,66,67,83,66,69,81,66,77,73,66,78,69,66,80,76,66,86,67
310 data66,86,83,66,82,75,67,76,67,67,76,68,67,76,73,67,76,86,68,69,88,68
311 data69,89,73,78,88,73,78,89,78,79,80,80,72,65,80,72,80,80,76,65,80,76
312 data80,82,84,73,82,84,83,84,65,88,84,65,89,84,83,88,84,88,65,84,88,83
313 data84,89,65,83,69,67,83,69,68,69,79,82,83,84,65,83,66,67,82,79,76,79
314 data82,65,76,83,82,76,68,89,76,68,88,76,68,65,67,77,80,65,83,76,65,78
315 data68,65,68,67,74,77,80,83,84,89,83,84,88,82,79,82,73,78,67,68,69,67
316 data67,80,88,67,80,89,66,73,84,74,83,82,83,69,73,108,158,32,162,150,10
317 data138,128,9,25,16,1,17,8,3,4,0,12,28,20,255,223,255,255,223,159,143
318 data159,31,31,31,31,31,15,3,28,159,28,28,28,56,59,57,34,36,17,16,34,34
319 data59,44,37,38,16,31,39,16,57,58,35,0,82,65,67,0,160,0,0,63,1,48
```

4.2.1.1 The EDMON Program

Here is the source code for the EDMON program.

```
            * = $6000
            .offs $e000 - *


;Attention: Changes for German v1.2
;see 'USA'

       bsout = $c145
       dojob = $84a7
       ttjob = $84a3
       sflag = $84b6 ;Scroll-Flag
        fifo = $c2a7
          p1 = $90
          p2 = $92



init   jsr $c19c    ;Initialize mouse from
       jsr clral    ;screen
                    ;buffer
       ldx #<ein1   ;by TT
       ldy #>ein1
       stx ttjob
       sty ttjob+1

       ldx #<print  ;Output
       ldy #>print
       stx dojob
       sty dojob+1

       jmp moni     ;Coldstart

       ;Character input

ein1   lda $8504    ;get
ok1    tax
       lda #$80     Flag for "Line output"
       ora sflag
       sta sflag
       txa
       cmp #32
       bcs nostr    ;No control character
       cmp #$0d     ;Return
       bne cnt5
       jmp return   ;Line computation

cnt5   cmp #$11     ;Cursor down
       bne cnt1

cnt6   ldx zeile
       cpx #14      ;Maximum line
       beq no1
```

156

```
        inc zeile
        rts

nol     jsr scup       ;Scroll
        rts

cnt1    cmp #$1e       ;Cursor right
        bne cnt2

        ldx spalte
        cpx #39        ;Maximum line
        beq no2
        inc spalte
no2     rts

cnt2    cmp #$10       ;Cursor left
        bne cnt3

        ldx zeile      ;Minimum line
        beq no3
        dec zeile
no3     rts

cnt3    cmp #$08       ;Cursor left
        bne cnt4
        ldx spalte     ;Minimum columns
        beq no4
        dec spalte
no4     rts

cnt4    cmp #$1d       ;Delete
        bne knt5
        ldx spalte     ;Minimum line
        beq no5
        dec spalte
        lda #32        ;Blank
        jsr nostr      ;entry
        dec spalte
no5     rts

knt5    cmp #$12       ;Home
        bne knt6
        jmp home

knt6    cmp #$13       ;CLR Home
        bne no5        ;End (previous)
        jmp clral      ;Re-initialize

        ;Input in buffer

nostr pha              ;Store character

        jsr reset      ;Buffer pointer

        ldx zeile      ;line times 40
nxt1    dex
```

157

```
        bmi rdy1

        jsr incp1      ;Next line
        jmp nxt1

rdy1    ldy spalte
        pla            ;character
        sta (p1),y

        ldx spalte
        cpx #39        ;maximum columns
        beq rdy2
        inc spalte
rdy2    rts


 ;Scroll buffer

scup    jsr reset      ;buffer pointer

        ldx #14        ;Pointer

nxt2    lda p1         ;Old line
        sta p2
        lda p1+1
        sta p2+1
        jsr incp1      ;Copy
        ldy #39        ;next line
nxt3    lda (p1),y
        sta (p2),y
        dey
        bpl nxt3

        dex
        bne nxt2

        lda #32
        ldy #39        ;Fill line
nxt4    sta (p1),y     ;with spaces
        dey
        bpl nxt4
                       ;Scroll flag
        lda #$40
        ora sflag
        sta sflag

        rts            ;Ready

        ;Display buffer contents

        ;Scroll current bitmap

print   bit sflag
        bvc oks        ;Not used

        ;10 Ten-line scroll
```

```
          sei

          ldx #0          ;line

ne        jsr $c13c       ;Target line
          lda $0c
          sta p1          ;mark it;
          lda $0d
          sta p1+1

          txa             ;store it
          pha

          clc
          adc #10
          tax

          jsr $c13c       ;Get line

          ldx #40

          ldy #0
nd        lda ($0e),y
          sta (p1),y

          lda $0e
          clc
          adc #$08
          sta $0e
          lda $0f
          adc #0
          sta $0f

          lda p1
          clc
          adc #$08
          sta p1
          lda p1+1
          adc #0
          sta p1+1

          dex
          bne nd

          pla
          tax
          inx             ;Next get line
          cpx #150        ;Ready?
          bcc ne

          cli

          ;* display current line *

oks       lda #$1b        ;Normal text
```

```
                 jsr bsout

                 lda #$0b      ;home
                 jsr bsout

                 lda #$0d      ;CR
                 jsr bsout

                 jsr reset     ;Buffer pointer

                 ldx #0        ;Line counter

        nextz cpx zeile        ;Gone over?
                 beq cont2     ;yes!

        cont1 lda #$0d         ;Else, output CR
                 jsr out

                 jsr incpl     ;Next line

                 inx
                 cpx #15
                 bne nextz

                 ;Line found

        cont2 ldy #0

        nexts cpy spalte
                 bne nextt

                 lda $18       ;Cursor column
                 sta $84be
                 lda $19
                 sta $84bf

        nextt lda (pl),y       ;Line output
                 cmp #$7f      ;(Tab.)
                 beq tab       ;correction character

                 jsr out       ;Else output
                 jmp cnto      ;coutinue

        tab    sty soll        ;Position
                 tya           ;times 5 (TAB width)
                 asl a
                 asl a
                 clc
                 adc soll

        cntp pha               ;store Should

                 lda #32       ;Display <space>
                 jsr out

                 ;Compare Should with Is position
```

```
          pla
          cmp $18        ;lo
          bcs cntp       ;Should >= Ist

          sta $18        ;New Is value

cnto      iny
          cpy #40
          bne nexts

          lda $05        ;Cursor line
          sec
          sbc $29
          sta $84c0

          lda #0         ;Finish output
          sta sflag

          lda $84b4      ;Sprite on
          and #$80
          ora #$01
          sta $84b4

          rts

     ;Output under register storage

out       stx xreg1
          sty yreg1
          jsr bsout      ;output
          ldx xreg1
          ldy yreg1
          rts

     ;Reset buffer pointer

reset ldx #<buff
          ldy #>buff
          stx p1
          sty p1+1
          rts

     ;Increment buffer pointer by 1 line

incp1 lda p1
          clc
          adc #40
          sta p1
          lda p1+1
          adc #$00
          sta p1+1
          rts

     ;Initialize screen buffer
```

```
      clral lda #$a0        ;only $A0 hires
            sta $2f

            lda #$00

            ldx $c00f        ;geos-version.
            cpx #$13         ;V1.3: No offset
            bcs v13

           ;lda #13          ;Offset V1.2 Germany
            lda #30          ;Offset V1.2 USA

      v13   sta vers

            jsr $c14b
            lda #$09
            sta $26
            lda $29
            jsr $c1c0        ;Cursor init

            ;Clear buffer

            jsr reset        ;Buffer pointer
            ldx #14          ;Number of lines -1
      nxtz  ldy #39          ;Number of columns -1
            lda #32          ;<space>
      nxts  sta (p1),y
            dey
            bpl nxts

            jsr incp1        ;Next line

            dex
            bpl nxtz

            ;Clear background

            ldx #$00
      nextd lda backd,x
            sta $0006,x
            inx
            cpx #$06
            bne nextd

            lda #$00         ;Display
            jsr $c139        ;pattern
            jsr $c124        ;calculated

            lda backd+1      ;Draw lines to scale
            sta $18
            lda #0
            sta $08
            sta $09

            lda #$01
            sta $0b
```

```
                    lda #$3f
                    sta $0a

                    lda #$ff
                    jsr $c118      ;Draw

              ;Reset pointer

        home   lda #0
                    sta spalte
                    sta zeile
                    rts

              ;Monitor

        moni

        ab lda #<ac
              sta $84af    ;execute GEOS break
              lda #>ac     ;(System Error ...)
              sta $84b0

              pla          ;Over-read
              pla

              lda vers     ;geos v1.3 ?
              beq entry

              pla
              sta $8861    ;Return address
              pla          ;(GEOS V1.2 USA)
              sta $8862
              tsx
              stx $8863

              ;pla
              ;sta $8850    ;Return address
              ;pla          ;(GEOS V1.2 Germany)
              ;sta $8851
              ;tsx
              ;stx $8852


        entry

              brk          ;Entry point

              ;Register from $0200 on.

        ac cld
              sta $0202    ;Status (Break)
              sty $0205
              stx $0204
              lda $880b    ;Accumulator
              sta $0203
```

163

```
        pla
        tax
        sec
        sbc #$01
        sta $0201    ;PC-Lo
        pla
        pha
        sbc #$00
        sta $0200    ;PC-Hi
        txa
        pha

        lda $fffe    ;Get IRQ vector
        sta $0208
        lda $ffff
        sta $0207

ae tsx               ;Mark
        stx $0206    ;stack pointer
        stx $0209
        cli
        cpx #$50     ;>$50 ?
        bcs af       ;No, O.K.
        ldx #$40;
        jsr dq       ;Else, Stack Overflow
        ldx #$84
        txs
        lda #><ac-1>  ;Entry as return
        pha
        lda #<<ac-1>
        pha
        jmp ae       ;New search

af jsr bv            ;send CR
        ldx #'b      ;"Break"
        lda #$2a     ;*
        jsr bn       ;output
        lda #114     ;R-command
        bne aj

        ;Read commands

ag lda #$00          ; (Overflow address)
        sta $8b      ;command read initialization

        ldx $0209    ;Stackpointer
        txs
ah lda #$2e          ;Display point
        ldx #$0d
        jsr bn
        jsr $c29b    ;Cursor on
        jmp $c1c3    ;To GEOS job loop


        ;** Return expression **
```

164

```
        return

             ;Pointer positioning

             jsr $c29e   ;cursor off

             lda #0
             sta lpoint

             jsr reset   ;Buffer pointer

             ldx zeile   ;line times 40
        gt2  dex
             bmi gt1

             jsr incpl   ;Next line

             jmp gt2

        gt1  jsr get  ; Get character

        aj cmp #$00
           beq ah
           cmp #$2e   ; Skip point
           beq gt1
           cmp #$20   ; Skip space
           beq gt1

           ;Search for command

           ldx #14      ;Pointer table--compare
        ak cmp code,x  ;with table
           beq al       ;O.K.
           dex
           bpl ak       ;No text
           jmp dx       ;"Display "?"

        al stx $b4      ;Compute pointer index
           txa          ;in table
           asl a
           adc $b4
           tax
           lda am+2,x  ;Get address (-1)
           pha
           lda am+1,x
           pha
           rts          ;and entry


           ;Get characters from screen buffer

        get

           sty yreg1
           ldy lpoint
```

```
        cpy #40
        beq empty

        inc lpoint
        lda (p1),y
        cmp #$7f     ;Tab
        bne ntab
        lda #$20     ;Space

        .byte $2c

empty

        lda #$00     ;Empty

ntab

        ldy yreg1
        rts


        ;Table of command addresses

        ;'rRmMdDftgxlsawh'

am  bit r-1       ;r
        bit ig+1     ;R
        bit id+2     ;m
        bit ih+1     ;M
        bit ik+2     ;d
        bit ip+1     ;D
        bit is+2     ;f
        bit iu+1     ;t
        bit ja-1     ;g
        bit jc-1     ;x
        bit zi-1     ;l
        bit ki+2     ;s
        bit kw+2     ;a
        bit gv       ;w
        bit jg-1     ;h


        ;Output in screen buffer

ein stx xreg
        sty yreg

        cmp #$0d     ;CRs filtered out
        bne cn1

        lda #$80
        sta sflag
        jsr print    ;Current line

        lda #0
        sta spalte ;
```

```
        jsr cnt6    ;Cursor down

        bit sflag   ;scroll ?
        bvc cna     ;No

        jsr print   ;Then display all

cna jmp cn2

cn1 jsr ok1

cn2 ldx xreg
    ldy yreg

    rts

    ;Read byte from TT buffer

ap jsr get
   cmp #$00
   beq aq       ;empty
   sec
   rts

aq lda #$20
ar clc
   rts


   ;Read Hex-Byte with max. 1 Blank
   ;If found: Carry = 1

as jsr ge       ;Get it and test for hex
   bcs aw       ;If hex,

   ;Read hex-byte

at jsr ge       ;Get and test
av bcc be       ;No hex number

aw jsr bf       ;Convert to number
   asl a
   asl a
   asl a
   asl a
   sta $0100    ;Hi-Nibble

   jsr ge       ;Get, and test for hex
   bcc ar       ;If not hex,
   jsr bf       ;Convert to number
   ora $0100    ;
ax sec
   rts          ;O.K.


   ;Bring address to P1
```

```
ay jsr ap        ;Skip blank
az jsr as        ;Hex number in accumulater?
ba bcc bc        ;None found
   pha
   jsr at        ;Second character
   bcc bb        ;none found
   bcc bk        ;Not found
   sta $80       ;After P2
   pla
   sta $81
   rts

bb pla
bc rts

   ;Skip two characters

bd jsr ap            ;Character from buffer
be jsr ap            ;          ""
   clc
   rts

   ; Hex -> Number

bf cmp #$3a
   php
   and #$0f
   plp
   bcc bg
   adc #$08
bg rts


   ;Read two addresses

bh jsr ay        ;Address after P1
   bcc bk        ;not found
   lda $80       ;at P2
   sta $82
   lda $81
   sta $83
   jsr ay        ;Address at P1

   ;Swap P1 w/ P2

bi ldx #$02
bj lda $7f,x
   pha
   lda $81,x     ;P2 after P1
   sta $7f,x
   pla
   sta $81,x     ;P1 to P2
   dex
   bne bj
   rts
```

```
bk lda #$3f     ;Error
   jmp dx

   ;Output value of P1

bl ldx #$01
   lda $7f,x
   pha
   lda $80,x
   jsr bm
   pla

   ;Output value in accumulator

bm pha
   lsr a
   lsr a
   lsr a
   lsr a
   jsr bo
   tax
   pla
   and #$0f
   jsr bo

   ;Output character in X and accumulator

bn pha
   txa
   jsr ein
   pla
   jmp ein

   ;Convert accumulator into hex

bo clc
   adc #$f6
   bcc bp
   adc #$06
bp adc #$3a
   rts

   ;P1 after $0200

bq lda $80
   sta $0201
   lda $81
   sta $0200
   rts


   ;Output bytes (P1)

br sta $8a      ;Number
   ldy #$00
```

```
bs jsr bu      ;Output space
   lda ($80),y
   jsr bm      ;output
   jsr bw      ;Increment P1
   dec $8a
   bne bs
   rts

bt jsr bu

bu lda #$7f    ;Output TAB

   .byte $2c

bv lda #$0d    ;output CR

   jsr ein     ;in Editor-Buffer

   rts

   ;Increment P1

bw inc $80
   bne bx
   inc $81
   bne bx
   inc $8b
bx rts

by tya         ;Mark code
   pha
   jsr bv      ;output CR
   pla
bz ldx #$2e    ;Point and
   jsr bn      ;value output
   jmp bt      ;2* TAB


   ;Read hex byte and store after P1

ca jsr at
   bcc cb
   ldx #$00
   sta ($80,x)
   cmp ($80,x)
   beq cb
   jmp bk

cb jsr bw
   dec $8a
   rts

cc lda #$03
   sta $80
   lda #$02
   sta $81
```

```
        lda #$04
        rts

        ;Output number in accumulator in binary

cd pha          ;store
   jsr bu       ;Tab
   pla

ce ldx #$07     ;8 Bit
   sta $8a      ;value
kq lda #$18     ;0 or 1
   asl $8a
   rol a
   jsr ein
   dex
   bmi cf
   tya          ;Formatted?
   beq kq       ;No

   jsr bu       ;Tab
   bne kq

cf rts


        ;Determine command length from address type

cg sta $8a
   ldx #$00
ch lda $8a
   and t1,x
   eor t2,x
   beq ci
   inx
   cpx #$14
   bcc ch
   ldx #$0e
ci lda t3,x
   pha
   lsr a
   lsr a
   lsr a
   lsr a
   tax
   pla
   and #$0f
   cmp #$0e
   rts


        ;Output P1 disassembly

cj ldy #68          ;Output "D"
   jsr by
ck jsr bl           ;Output P1
```

```
        jsr bu              ;Output Tab
        lda ($80),y         ;
        jsr cg              ;Address length
        stx $8a             ;Establish type
cl jsr bu                   ;Output tab
        da ($80),y          ;
        sr bm               ;Output value
        iny
        dec $8a             ;Continue?
        bne cl

cm jsr bu                   ;Output tab
        jsr bt              ;Output 2 tabs
        iny
        cpy #$04
        bcc cm              ;<4 ?

cn ldy #$00
        lda ($80),y
        ldx #$1f
co cmp t4,x
        beq cs
        dex
        bpl co
        ldx #$17
cp lda ($80),y
        and t5,x
        eor t6,x
        beq cr
        dex
        bpl cp

cq jmp di                   ;Output "??"

cr txa
        clc
        adc #$20
        tax
cs txa
        sta $8a
        asl a
        adc $8a
        tax
        ldy #$03
ct lda opcode,x
        jsr ein
        inx
        dey
        bne ct
        lda ($80),y
        jsr cg
        bcs cq
        tay
        lda #$7f            ;Tab
        jsr ein
        cpy #$08
```

```
                beq cx
                cpy #$09
                bcs cy
                lda #$24
                cpy #$02
                bne cu
                lda #$23
                jsr ein
        cu lda #$24
                cpy #$00
                bne cv
                lda #$7f        ;Tab
        cv cpy #$01
                bne cw
                lda #$41
        cw cpy #$05
                beq cx
                cpy #$06
                bne cz
        cx lda #$28
                jsr ein
        cy lda #$24
        cz jsr ein
                cpy #$0c
                bne da
                jsr dk
                txa
                jsr bm
                tya
                jmp bm

        da dex
                beq dc
                dex
                beq db
                ldy #$02
                lda ($80),y
                jsr bm
        db ldy #$01
                lda ($80),y
                jsr bm
        dc ldy #$00
                lda ($80),y
                jsr cg
                cmp #$03
                beq dh+1
                cmp #$04
                beq dg
                cmp #$05
                beq dd
                cmp #$06
                beq df
                cmp #$0a
                beq dh+1
                cmp #$0b
                beq dg
```

```
          cmp #$08
          beq de
          rts

       dd jsr dh+1
       de lda #$29
          jmp ein

       df jsr de
       dg lda #$59
       dh bit $58a9
          ldx #$2c
          jmp bn

          ;?? ausgeben

       di ldx #$02
          lda #$3f
       dj jsr ein
          dex
          bpl dj
          rts

       dk ldx $81
          ldy #$01
          lda ($80),y
          bpl dl
          dex
       dl clc
          adc #$02
          bcc dm
          inx
       dm clc
          adc $80
          bcc dn
          inx
       dn tay
          lda #$82
          sec
          rts

       do ldy #$00
          lda ($80),y
          jsr cg
          bcs ds
          cmp #$0c
          beq dk
          cmp #$03
          txa
          pha
          iny
          lda ($80),y
          pha
          dex
          dex
          beq dp
```

```
        iny
        lda ($80),y
        tax
  dp pla
        tay
        pla
        rts

  dq jsr bv
        lda #$3f     ;"?"
        jsr ein
        jsr dr
        ldx #$2b;

  dr lda t7,x     ;Output headline
        php
        and #$7f
        jsr ein
        inx
        plp
        bpl dr
        rts

  ds ldx #$49;
        jsr dq
        ldx #$31;
        jsr dr
        jsr bl
  dt jmp ag

  du ldx #$35;
  dv jsr dq
        bmi dt

  dw jsr ap          ;Read byte
        jsr as          ;Get hex value
        bcc dx          ;If none, then"?"
        sta $85
        jsr at          ;Hex value after P3
        sta $84
        bcs ea

  dx lda #$3f     ; "?"
        jsr ein
        jmp ag

  dy jsr bh          ;Get 2nd address
        bcc dx          ;Target address
        jsr dw
        jsr ap          ;Get byte
        cmp #$2d        ;
        beq dz
        cmp #$2b
        bne du          ;"missing +/- ..."
```

```
        lda  $84
        tax
        lda  $85
        tay
        lda  $82
        sbc  $80
        sta  $84
        lda  $83
        sbc  $81
        sta  $85
        txa
        sec
        sbc  $84
        sta  $84
        tya
        sbc  $85
             $85

dz      lda  $84
        cmp  $80
        lda  $85
        sbc  $81
        bcc  ea
        lda  $82
        sbc  $80
        tax
        lda  $83
        sbc  $81
        tay
        txa
        clc
        adc  $84
        sta  $84
        tya
        adc  $85
        sta  $85
        sec
ea      rts

eb      ldy  #$00
        lda  ($80),y
        jsr  cg        ;Addressing type (x)
ec      jsr  bw        ;P1 incremented accordingly
        dex
        bne  ec
        rts

ed      pha            ;if P2<P1: return
        lda  $82
        cmp  $80
        lda  $83
        sbc  $81
        bcc  ee
        lda  $8b
        beq  ef
ee      jmp  ag
```

```
      ef pla
         rts

      eg ldy #$00
         ldx #$00
         bcs ei
      eh jsr ed
         lda ($80,x)
         sta ($84),y
         jsr bw
         iny
         bne eh
         inc $85
         clc
         bcc eh
      ei jsr ed
         lda ($82),y
         sta ($84,x)
         lda $82
         bne ej
         dec $83
      ej dec $82
         lda $84
         bne ek
         dec $85
      ek dec $84
         clc
         bcc ei
      el jsr ap
      em jsr ap
         sta $b4
         ldx #$00
         stx $80
         stx $81
         cmp #$24
         beq ev
         cmp #$25
         beq es
         cmp #$21
         bne ew
      en jsr ap
         cmp #$3a
         bcs eq
         sbc #$2f
         bcc ep
         tay
         asl $80
         rol $81
         lda $80
         ldx $81
         asl a
         rol $81
         asl a
         rol $81
         clc
```

177

```
        adc $80
        bcc eo
        inx
eo sta $80
        tya
        clc
        adc $80
        sta $80
        txa
        adc $81
        sta $81
        jmp en

ep adc #$30
eq rts

er jsr ap
        cmp #$20
        bne et
es jsr ap
et lsr a
        eor #$18
        bne eu
        rol $80
        rol $81
        jmp er

eu eor #$18
        rol a
        rts

ev jsr ap
ew tax
        cmp #$47
        bcs ez
        sbc #$2f
        bcc ez
        cmp #$0a
        bcc ex
        cmp #$11
        bcc ez
        sbc #$07
ex ldx #$03
ey rol $80
        rol $81
        dex
        bpl ey
        ora $80
        sta $80
        jmp ev

ez txa
        rts

fa lda #$00
        ldx #$02
```

178

```
      fb sta $020a,x
         dex
         bpl fb
         ldx #$0f
      fc asl $80
         rol $81
         sed
         ldy #$02
      fd lda $020a,y
         adc $020a,y
         sta $020a,y
         dey
         bpl fd
         dex
         bpl fc
         cld
         ldx #$05
      fe jsr fh
         ora #$30
         sta $0100,x
         dex
         bpl fe
         ldx #$04
      ff lda $0100,x
         cmp #$30
         bne fg
         dex
         bne ff
      fg rts

      fh lda #$10
      fi asl $020c
         rol $020b
         rol $020a
         rol a
         bcc fi
      fj rts

      fk jsr fifo      ;Read STOP
         cmp #$16
         bne fj
         jmp ag        ;For input

      fl lda #$00
         sta spalte

         lda #68
         jsr bz        ;".D"
         jmp ck        ;disass.


         ;P1<->P3 and P2<->P4

      fn ldx #$03
      fo ldy $80,x
         lda $84,x
```

```
          sty $84,x
          sta $80,x
          dex
          bpl fo
          rts

          ;Test whether address lies in range

       fp cpy $84
          txa
          sbc $85
          bcc fq
          clc
          tya
          sbc $86
          txa
          sbc $87
          bcs fq
          clc
          rts

       fq sec
          rts

          ;Test whether P1 lies in range

       fr tya
          pha
          txa
          pha
          ldy $80
          ldx $81
          jsr fp
          pla
          tax
          pla
          tay
          rts

       fs tya
          sec
          sbc $80
          pha
          txa
          sbc $81
          tay
          clc
          pla
          adc #$7e
          bcs ft
          dey
       ft iny
          bne fu
          sec
          sbc #$80
          clc
```

```
           iny
           rts

    fu ldx #$5d;
    fv jsr dq
       ldx #$31;
       jsr dr
       jsr bl
       sec
       rts

    fw clc
       tya
       adc $88
       tay
       txa
       adc $89
       tax
       rts

       clc
       iny
       rts

    fx jsr fa
       nop
    fy lda $0100,x
       jsr ein
       dex
       bpl fy
       rts

       ;Read %

    fz jsr ap          ;Blank
    ga ldx #$07
    gb jsr ap          ;Look at blank
       cmp #$20
       beq gb
       lsr a
       eor #$18
       bne gd
       rol $8a
       dex
       bpl gb
       lda $8a

    gc sec
       rts

       ;Get character and test for hex

    ge jsr ap          ;Read character
    gf cmp #48          ;<"0"
       bcc gg          ;Yes, no hex
       cmp #58          ;<":"
```

```
        bcc gc          ;Yes, hex
        cmp #65         ;<"A"
        bcc gg          ;no hex
        and #$df        ;Clr Bit 5
        cmp #71         ;<"G"
        bcc gc          ;Yes, hex

gd clc                  ;no Hex
gg rts


    ;Get byte sequence and put at P1

gr ldy #$00
gs jsr ap               ;Read byte
        bcc gv          ;Ready
        cmp #$20        ;blank
        beq gs
        cmp #$22        ;='"' ?
        beq gt          ;Then text
        jsr gf          ;text for hex
        jsr av          ;Convert to number
        bcc gu          ;No hex format
        sta ($80),y
        jsr bw          ;Increment P1
        bne gs

    ;Get text

gt jsr ap               ;Read byte
        bcc gv          ;Ready
        cmp #$22        ;end of text
        beq gs

        sta ($80),y     ;Set it up
        jsr bw          ;increment P1
        bne gt


gu jmp dx               ;Error

gv rts

    ;-w-

        jsr ay          ;Address after P1
        jsr gr          ;Get and set up value
        ldy #119        ;"w"
        jsr by          ;Output
        jsr bl          ;Output P1

gx jmp ag               ;To command reading

    ;-r-

r  ldx #$00     ;Output headline
```

```
          jsr dr
          jsr bv        ;CR Output
          ldx #$2e      ;Point
          lda #82       ;period "R"
          jsr bn        ;output
          jsr bu        ;1* Tab
          lda $0200     ;output
          jsr bm        ;PC Hi
          lda $0201     ;output
          jsr bm        ;PC Lo
          jsr bu        ;Tab
          lda $0207     ;output
          jsr bm        ;IRQ Hi
          lda $0208     ;output
          jsr bm        ;IRQ Lo
          lda $0202     ;Status
          ldy #$ff      ;Output
          jsr cd        ;formatted binary
          jsr cc        ;Output regular
          jsr br        ;parameters
id jmp ag               ;Ready


          ;-m-

          jsr bh        ;Adresses -> P1/P2
ie jsr ed               ;Test for P2 > P1
          jsr fk        ;Test stop
          ldy #77       ;output ".M"
          jsr by
          jsr bl        ;Output address
          lda #$08      ;Output number
if jsr br               ;of bytes
ig beq ie               ;Unconditional jump


          ;-R- zu -r-

          jsr ay        ;Get PC
          bcc ik
          jsr bq        ;and note it
          jsr ay        ;Get IRQ
          bcc ik
          lda $80       ;and mark it
          sta $0208
          lda $81
          sta $0207
          jsr fz        ;Read %
          bcc ik
          sta $0202
          jsr cc
ih bne ii


          ;-M- to -m-
```

```
        jsr bd
        jsr az
        bcc ik
        lda #$08
ii sta $8a

ij jsr ap
        jsr ca
        bne ij

        jmp ag      ;O.K.

ik jmp dx           ;Error


        ;-d-

        jsr bh       ;Get addresses
        bcs im       ;2 addresses found
        lda #$ff     ;Else: $FFFF = END
        sta $82
        sta $83
        bne im

il jsr fifo          ;GET-function
        cmp #$00
        beq il
        cmp #$16     ;run/stop?
        bne im

        jmp ag       ;To input

im lda #$05          ;Output 5 bytes
        sta $b4
in dec $b4
        bmi il

io jsr fk            ;Read stop
        jsr ed       ;Test for end,
        jsr cj       ;otherwise, disassemble
        jsr eb       ;Increment P1 correctly
ip beq in


        ;-D- to -d-

        jsr ay       ;Address of P1
        bcc is
        jsr ap       ;Read byte
        jsr as       ;max. 1 Blank
        bcc ir
        tay
        jsr cg       ;length of other type
        tya
        ldy #$00
iq sta ($80),y
```

```
        iny
        dex
        beq ir
        jsr as
        bcs iq

ir jsr fl        ;New disassembly
   jsr bv        ;Output CR
   jsr eb        ;P1 incremented correctly
   jsr fl        ;disass.

   ldx #$0a
   stx spalte
   rts

is jmp dx        ;Error message


   ;-f-

   jsr bh
   bcc is
   jsr ap
   jsr as
   bcc is
   ldx #$00
it jsr ed
   sta ($80,x)
   jsr bw
   clc
iu bcc it


   ;-t-

   jsr dy        ;Set pointer
iv jmp eg        ;Copy

ja jsr ay        ;Get address
   bcc jb        ;Not available
   jsr bq        ;at $0200,$0300

jb ldx $0206
   txs
   sei

   lda $0207
   sta $0315
   lda $0208
   sta $0314

   lda $0200     ;PC
   pha
   lda $0201
   pha
```

```
        lda $0202        ;Status
        pha
        lda $0203
        ldx $0204
        ldy $0205
        rti


        ;-x-

jc
        ldx #$c2         ;Job: Return Swap
        ldy #$3e
        stx $849c
        sty $849b

nf jmp ag        ;To input


        ;-h-

jg jsr bh         ;Addresses in P1,P3
        jsr fn           ;P1,P2 <-> P3,P4
        ldx #$0b         ;Buffer in P1
        lda #$02
        stx $80
        sta $81
        jsr gr           ;Get byte sequence
        lda $80          ; in buffer
        clc
        sbc #$0b
        bcc ki           ;No data: "?"
        sta $020a
        jsr fn           ;P1,P2 <-> P3,P4
jh jsr ed         ;Test P1 for end
        jsr fk           ;Test for stop
        ldy $020a        ;Amount of data - 1
ji lda ($80),y    ;Data in RAM
        cmp $020b,y      ;Searched data
        bne jj           ;Unequal.: continue
        dey
        bpl ji
        ldx #$55         ;".Found"
        jsr dr
        jsr bl           ;Output address
jj jsr bw         ;Increment P1
        clc
        bcc jh           ;Continue search

ki jmp dx


        ;-s-

        lda #0
        tax
```

```
      ze sta $8100,x
         inx
         bne ze

         jsr za          ;Name -> Buffer
         bcc ki          ;Error

         jsr as          ;Get GEOS file type
         bcc ki          ;Not available.
         beq ki          ; = 0  -> ill.
         cmp #13
         bcs ki          ;>= 13 -> ill.

         sta $8145       ;in Info

         lda #$82        ;C-64 File-Type
         sta $8144

                         ;Structure is SEQ=0 !

         jsr bh          ;Get 2 addresses
         bcc ki          ;Error

         ldx $80         ;Start
         ldy $81
         stx $8147
         sty $8148

         stx $814b       ;= Initialization
         sty $814c

         ldx $82         ;End
         ldy $83
         stx $8149
         sty $814a

         ldx #<name      ;Pointer to names
         ldy #>name
         stx $8100
         sty $8101

         ldx #$03        ;Icon width
         ldy #$15        ;Icon height
         stx $8102
         sty $8103

         lda #$bf
         sta $8104       ;Number of bytes

         ldx #$3f
         lda #$55
      zf sta $8104,x     ;Icon pattern
         dex
         bne zf
```

187

```
        lda #0            ;At DIR side 0
        sta $16

        ldx #$81          ;Pointer to Info
        sta $14           ;Lo
        stx $15           ;Hi

        jsr $c1ed         ;save
        txa
        beq zg            ;O.K.

        cmp #$03
        bne zh

        ldx #$90;         ;disk full
        bne zm

zh txa                    ;i/o error
        ora #$30
        sta numb
        ldx #$a0;
zm jsr dq                 ;Output

zg jmp ag                 ;To Input



        ;-1-

zi lda #0

        ldx vers          ;Versions-Offset
        sta $885e,x       ;Address flag

v3 jsr za                 ;Name -> buffer
        bcc me            ;Error

        ldx #<name        ;Pointer to names
        ldy #>name
        stx $0e
        sty $0f

        jsr $c20b         ;Search file and
        txa               ;goto $8400
        bne zh            ;i/o error

        jsr as            ;:pad type byte?
        bcc q1            ;Else, goto info


        ;Other parameters available!

        cmp #$00          ;(GEOS, its own address)
        beq zk

        cmp #$01          ;(DOS-Format)
```

188

```
        bne me              ;Output "?"

        ldx $8401           ;Track and sector data
        ldy $8402
        stx $04
        sty $05

        ldx #$ff            ;with correction
        jsr zo
        bcs zp              ;found

        ldx #$00            :Buffer $8000
        ldy #$80
        stx $0a
        sty $0b

        jsr $c1e4           ;Load first sector

        lda $8002           ;Correct DOS address
        sec
        sbc #2
        tax
        lda $8003
        sbc #0
        tay

zp stx $10
        sty $11

        lda #$ff
        sta $06
        sta $07
        jsr $c1ff           ;Load DOS

        jmp zs              ;Continue as below

        ;*   *    *    *

zk ldx #0                   ;without correction
        jsr zo              ;Loading address

q1 ldx #$00                 ;Pointer to Dir
        ldy #$84
        stx $14
        sty $15
        jsr $c211           ;Load over Info

zs txa
        beq zz              ;Error

        jmp zh

zz jmp ag                   ;Ready

me jmp dx                   ;"?" Output
```

```
          ;Get name

za ldx #$00
   jsr ap           ;Read byte
   bcc zj           ;Error
   cmp #$20         ;Skip blank
   beq za

   cmp #$22         ;='"' ?
   bne zj           ;Else, error

zc jsr ap           ;Get character
   cmp #$22         ;='"' ?
   beq zd
   sta name,x
   inx
   cpx #16
   bne zc

   jsr ap           ;'"' Read

   cmp #$22         ;Correct Syntax ?
   bne zj           ;No

zd lda #0           ;Null closer
   sta name,x

   sec
   rts

zj clc              ;Error-Flag
   rts

   ;Get load address

zo jsr ay           ;Get
   bcc zl           ;without correction

   txa              ;With correction
   bne zr

   ldx $80          ;given
   lda $81
   bcs zq           ;Absolute

zr lda $80
   sbc #2
   tax
   lda $81
   sbc #0

zq ldy vers         ;Versions-Offset

   sta $8860,y
   txa
   sta $885f,y
```

```
        lda #$01        ;Flag
        sta $885e,y
        sec

zl rts

kw jmp dx               ;Output "?"


   ;-a-

kx jsr ap               ;Read byte
   bcc kw
   cmp #$20             ;Read blank
   beq kx
   jsr aw
   jsr ba               ;Address of P1
   bcc kw               ;Error

ky jsr ap               ;Get byte
   bcs kz               ;Available

   jsr bu               ;Output blank
   jsr cn               ;disassembling
   jmp md               ;Cursor pos.

kz cmp #$20             ;Read blanks
   beq ky
   and #$df
   sta $020c            ;left
   jsr ap
   and #$df
   sta $020b            ;middle
   jsr ap
   and #$df
   sta $020a            ;right

   lda #$39             ;Pointer to opcodes
   sta $8a              ; (57 decimal)

la dec $8a
   bmi kw               ;ill. Opcode

   ldx #$02             ;Counter
   lda $8a
   asl a                ;times 3
   adc $8a
   tay
lb lda $020a,x          ;Search for opcode
   cmp opcode,y
   bne la               ;Not equal

   iny
   dex
   bpl lb               ;Continue testing
```

```
          ldx $8a              ;Mark pointer
          stx $020a

          ldx #$04
          lda #$20
    lc sta $020b,x
          dex
          bpl lc
          ldx #$04
    ld jsr ap
          bcc ln
          cmp #$20
          beq ld
          cmp #$23
          bne ll
          jsr ge
          bcs lf
          cmp #$27
          bne le
          jsr ap
          clc
          bcc lj
    le cmp #$24
          bne lg
          jsr ge
    lf jsr av
          bcs lj
          bcc li
    lg cmp #$25
          bne lh
    lh jsr ga
    li bcc lv
    lj sta $82
          ldx #$02
          stx $b4
          bne lq
    lk jsr ap
    ll cmp #$24
          beq lk
          sta $020b,x
          jsr gf
          bcc lm
          jsr av
          bcc ln
          ldy $82
          sty $83
          sta $82
          lda #$00
    lm sta $020b,x
          dex
          bpl lk
    ln lda #$0c
          sta $b4
    lo dec $b4
          bmi lx
          lda $b4
```

```
          asl a
          asl a
          adc $b4
          adc #$04
          tay
          ldx #$04
lp  lda $020b,x
          cmp t8,y
          bne lo
          dey
          dex
          bpl lp
lq  ldy #$00
lr  lda t3,y
          and #$0f
          cmp $b4
          beq ly
ls  iny
          cpy #$14
          bcc lr
          lda $b4
          bne lt
          inc $b4
          bne lq
lt  cmp #$07
          bne lu
          clc
          bcc lw
lu  cmp #$09
lv  bne lx
          lda $020a
          cmp #$08
          bcs lx
          nop
          ldy $82
          ldx $83
          jsr fs
          sta $82
lw  lda #$0c
          sta $b4
          bcc lq
lx  jmp dx

ly  ldx $020a
          cpx #$20
          bcc lz+1
          lda t9,x

lz  .byte $2c

          lda #$ff

          eor #$ff
          and t2,y
          ora t4,x
          jsr cg
```

```
           cmp $b4
           bne ls
           dex
           beq mb
           dex
           beq ma
           ldy #$02
           lda $83
           sta ($80),y
ma         ldy #$01
           lda $82
           sta ($80),y
mb         ldy #$00
           lda $8a
           sta ($80),y
           jsr cg
           txa
           clc
           adc $80
           sta $80
           bcc mc
           inc $81
mc         jsr bv
           lda #97        ;Output "a"
           jsr bz
           jsr bl
md         lda #$0a
           sta spalte
           rts            ;Goto Input

t8
.byte $20,$20,$20,$20
.byte $20,$20,$20,$20,$20,$41,$20,$20
.byte $20,$00,$23,$20,$20,$58,$2c,$00
.byte $20,$20,$59,$2c,$00,$29,$58,$2c
.byte $00,$28,$59,$2c,$29,$00,$28,$20
.byte $20,$20,$20,$00,$20,$29,$00,$00
.byte $28,$20,$20,$20,$00,$00,$20,$58
.byte $2c,$00,$00,$20,$59,$2c,$00,$00

t7
.byte $0d,$7f,$7f,$7f,$7f
.text 'pc'
.byte $7f,$7f
.text 'irq'
.byte $7f,$7f
.text 'n'
.byte $7f
.text 'v'
.byte $7f
.text '#'
.byte $7f
.text 'b'
.byte $7f
.text 'd'
.byte $7f
```

```
              .text 'i'
              .byte $7f
              .text 'z'
              .byte $7f
              .text 'c'
              .byte $7f
              .text 'ac'
              .byte $7f
              .text 'xr'
              .byte $7f
              .text 'yr'
              .byte $7f
              .text 's'
              .byte 'p or $80

              * = * + 2

              .text ' erro'
              .byte 'r or $80
              .text ' in'
              .byte $a0
              .text 'missing +/'
              .byte '- or $80
              .text 'low stac'
              .byte 'k or $80
              .text 'illegal cod'
              .byte 'e or $80
              .byte $0d
              .text '.found'
              .byte $a0
              .text 'branc'
              .byte 'h or $80
              .text 'zero pag'
              .byte 'e or $80
              .text 'out of memor'
              .byte 'y or $80
              .text 'illegal line numbe'
              .byte 'r or $80
              .text 'loa'
              .byte 'd or $80
              .text 'disk or dir ful'
              .byte 'l or $80
              .text 'i/o #'
       numb   * = * + 1
              .byte $a0

       code   ;commands
              .byte 114,82,109,77,100,68,102,116
              .byte 103,120,108,115,97,119,104
       t4
              .byte $90,$b0,$f0,$30,$d0,$10,$50,$70
              .byte $00,$18,$d8,$58,$b8,$ca,$88,$e8
              .byte $c8,$ea,$48,$08,$68,$28,$40,$60
       t9
              .byte $aa,$a8,$ba,$8a,$9a,$98,$38,$f8
       t6
```

195

```
          .byte $41,$81,$e1,$22,$01,$42,$a0,$a2
          .byte $a1,$c1,$02,$21,$61,$4c,$84,$86
          .byte $62,$e6,$c6,$e0,$c0,$24,$20,$78
t5
          .byte $e3,$e3,$e3,$e3,$e3,$e3,$e3,$e3
          .byte $e3,$e3,$e3,$e3,$e3,$df,$e7,$e7
          .byte $e3,$e7,$e7,$f3,$f3,$f7,$ff,$ff
opcode ;56 Opcodes
          .text 'bccbcsbeqbmibnebplbvc'
          .text 'bvsbrkclccldcliclvdex'
          .text 'deyinx'
          .text 'inynopphaphpplaplprti'
          .text 'rtstaxtaytsxtxatxstya'
          .text 'secsedeorstasbcrolora'
          .text 'lsrldyldxldacmpasland'
          .text 'adcjmpstystxrorincdec'
          .text 'cpxcpybitjsrsei'
t2
          .byte $6c,$9e,$20,$a2,$96,$0a,$8a,$80
          .byte $09,$19,$10,$01,$11,$08,$03,$04
          .byte $00,$0c,$1c,$14
t1
          .byte $ff,$df,$ff,$ff
          .byte $df,$9f,$8f,$9f,$1f,$1f,$1f,$1f
          .byte $1f,$0f,$03,$1c,$9f,$1c,$1c,$1c
t3
          .byte $38,$3b,$39,$22,$24,$11,$10,$22
          .byte $22,$3b,$2c,$25,$26,$10,$1f,$27
          .byte $10,$39,$3a,$23,$00,$52,$41,$43

backd    .byte $00,$a0,$00,$00,$3f,$01



akku      *   = * + 1
xreg1     *   = * + 1
yreg1     *   = * + 1
xreg      *   = * + 1
yreg      *   = * + 1
lpoint *   = * + 1
spalte *   = * + 1
zeile     *   = * + 1
soll      *   = * + 1
name      *   = * + 17
vers      *   = * + 1   ;geos-vers.-offset
buff      *   = * + 600 ;Output range

          .end
```

4.2.2 Overview of the commands

The EDMON commands are very similar to those of standard C64 machine language monitors. The following is a list of the commands, and the syntax of each command.

In general, a space or a comma must follow each "input unit" (command character or operand). In the examples we use $1000 as the starting address and $2000 as the end address, if these addresses are needed.

r

The current contents of the processor registers will be printed. The contents can then be changed.

g 1000

Starts a program which starts at $1000. To return control to the monitor, the program must end with a BRK (not an RTS!).

d 1000 2000

Outputs a disassembly of the memory range. Five lines are printed at a time. Pressing a key will cause the output to continue.

The end address is optional. The hex portion of the printed lines can be edited and the changes will be accepted with RETURN. The functions are terminated with RUN/STOP.

a 1000 LDA #$FE

The assemble command allows a direct assembly to memory. The next address is automatically printed in order to make input easier. If only the address is entered, the current command at this location will be printed. It can then be changed or left alone by pressing RETURN.

m 1000 2000

Outputs the memory range as a hex dump. If no end address is given, only eight bytes will be displayed. The memory contents can be changed. RETURN accepts the modified line. Output is terminated with RUN/STOP.

t 1000 2000 3000-

Moves the memory range $1000-$2000 to $3000. The minus sign means that $3000 is the first address of the new data block. In most cases "t" is used in this mode.

t 1000 2000 3000+

Moves the memory range to $3000. Here $3000 is the last address of the new data block, however. The data is placed before $3000.

f a000 b000 55

Fills memory range from $A000 to $B000 with the value $55.

h 1000 2000 "Hello"

The hunt command searches the memory range for the occurrence of the string "Hello". All start addresses of this string will be printed.

h 1000 2000 2F 1E

Here the hunt function searches through a memory range for addresses which contain the bytes $2F and $1E in this order. The number of bytes in the input line is variable. All addresses found will be printed.

w 1000 "Hello mouse"

The word command allows a string to be entered directly into memory. The next unused memory location will then be displayed. In this case the following would appear:

        w 100A

w 1000 F5 E1 A1

In this case word allows bytes to be placed in memory directly. The specified data is stored in order. The first unused memory location will then appear. In our case: "w 1003".

l "program"

Loads a GEOS program at the address specified in the INFO sector.

l "program",00,2000

Loads a GEOS program. The address in the INFO sector will be ignored. The data will be placed at $2000.

l "program",01

Loads a program in the old DOS format. The start address must be at the start of the data block as usual. The program will be loaded at this address.

1 "program",01,2000

Loads a program at $2000. The data must be in the standard DOS format (bytes 1 and 2 of the file contain the load address). The load address in the file will be ignored.

s "program",ft,1000,2000

The data in the range $1000 to $1FFF will be stored on the disk. "ft" stands for the desired file type (00-09, must be two characters). If an accessory is to be created, for example, the file type must be five. For applications the file type is six.

An INFO sector is automatically created when saving and the start and end addresses are stored in it. In addition, the start address is entered as the entry address. The file thus created will have a striped icon.

x

You can exit EDMON with this command. To do this you must insert the disk from which you loaded EDMON. If "x" doesn't work, it's probably because the load disk is not in the drive.

When you enter an "x" EDMON will check to see if the disk in the drive contains a SWAP file. If it does not, the command will not be executed.

4.2.3 Tips for working with EDMON

EDMON is an accessory. This has the big advantage that it can be used from
any application as well as the deskTop. We want to show you some examples
in this section to give you a taste of the possibilities available with EDMON.
When you load EDMON, your screen will look something like this:

```
B*
    PC  IRQ  N V # B D I  Z C  AC XR YR SP
 .R 624F  FA9D 0 0 1 1 0 0 1 0 00 E3 60 F8
```
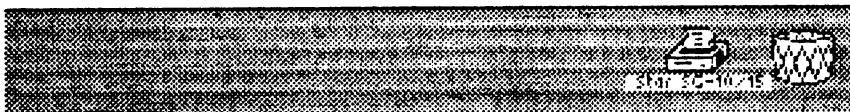

Figure 38: EDMON

When EDMON is executed it displays the contents of the registers. Naturally,
EDMON overwrites part of the current memory, but this is first saved on the
disk, so you don't have to worry about losing any data. In spite of this, you
shouldn't load EDMON if you have something irreplaceable in the computer.

The memory that EDMON occupies is chosen very carefully. It starts at $6000,
which is right in the second GEOS screen. This way none of the program is
overwritten and you can look at everything, disassemble it or save it. You have
complete access to GEOS.

Let's look at part of the memory. You may already know that from $8000-
$83FF there are four buffers for the disk drive. Let's look at the second buffer
at $8100 by entering

        M 8100 8140

You will see something like the following picture:

200

```
B*
    PC   IRQ   N V # B D I  Z C AC XR YR SP
.R 624F  FA9D  0 0 1 1 0 0 1  0 00 E3 60 F8
.m 8100 8140
.M 8100  00 FF 03 15 BF FF FF FF
.M 8108  80 00 01 9F FF 9D A0 00
.M 8110  41 A8 C6 5D A8 A9 41 A8
.M 8118  AF 5D A8 A9 41 AE C9 5D
.M 8120  A0 00 41 A4 00 41 A6 44
.M 8128  59 AC AA 59 A4 AA 59 A6
.M 8130  AA 59 AC 44 59 A4 00 59
.M 8138  9F FF 81 80 00 19 80 00
.M 8140  01 FF FF FF 82 0D 00 FE
.
```
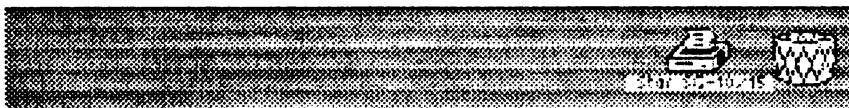
Figure 39: M 8100 8140

If you have used other machine language monitors before, you may notice that
EDMON is different from other monitors in that it displays just a hex dump
(with the contents of eight memory locations on one line), without the
corresponding ASCII characters. This is because the output of EDMON is done
in high-resolution graphics and is quite slow. If the ASCII characters were
printed as well, the output would become even slower.

If you have worked with GEOS for a while and are familiar with the
construction of the INFO sector, it may strike you the printed bytes bear a great
resemblance to an INFO sector. In fact, GEOS usually puts the INFO sector in
the buffer at $8100. The bytes from $8100-$8140 printed with the M command
determine the appearance of the icon.

But EDMON can do much more than just display the contents of memory
ranges. We will now look at part of the GEOS program itself, the jump table.
The jump table starts at $C100 and looks like this when viewed with EDMON:

```
.D C100   4C D7 C2   JMP $C2D7
.D C103   4C 16 CB   JMP $CB16
.D C106   4C BE CB   JMP $CBBE
.D C109   4C D9 CB   JMP $CBD9
.D C10C   4C E2 CB   JMP $CBE2
.D C10F   4C EA CB   JMP $CBEA
.D C112   4C F2 CB   JMP $CBF2
.D C115   4C FA CB   JMP $CBFA
.D C118   4C 4A C6   JMP $C64A
.D C11B   4C CF C6   JMP $C6CF
.D C11E   4C 63 C7   JMP $C763
.D C121   4C E2 C7   JMP $C7E2
.D C124   4C 4E C8   JMP $C84E
.D C127   4C BC C8   JMP $C8BC
```
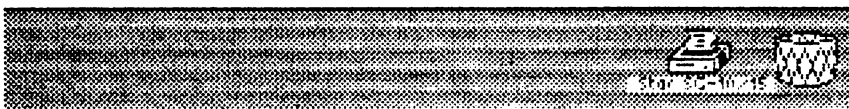
Figure 40: GEOS jump table

EDMON is so powerful because it is completely compatible with GEOS. This means that it disturbs GEOS as little as possible. For example, it will output characters in high-resolution mode just like GEOS. We could have written EDMON so that it first switched to the text mode and then displayed everything as normal text. But since the text screen is normally at $0400, and the GEOS applications start there, we wouldn't be able to look at the first parts of these applications.

As an example we want to show you how geoWrite is started. The start address is $0400. To do this we exit EDMON, load geoWrite, and then EDMON again (in geoWrite this is done from the geos menu). Now we can take a look at the beginning of geoWrite.

Enter the following:

        D 0400 0427 (RETURN)

This tells EDMON that you want a disassembled listing of the program in memory, in the memory range from $0400 to $0427.

After this is typed in, we see the first commands with which geoWrite is initialized:

```
.D 0403  4C 38 34    JMP $3438
.D 0406  20 BD C1    JSR $C1BD
.D 0409  20 26 22    JSR $2226
.D 040C  20 27 04    JSR $0427
.D 040F  20 9F 04    JSR $049F
.D 0412  4C 9F 05    JMP $059F
.D 0415  20 BD C1    JSR $C1BD
.D 0418  20 26 22    JSR $2226
.D 041B  20 27 04    JSR $0427
.D 041E  20 D7 05    JSR $05D7
.D 0421  20 09 05    JSR $0509
.D 0424  4C 2C C2    JMP $C22C
.D 0427  20 50 06    JSR $0650
.s "screen4",01,a000,bf40
.
```

The single point space (Commodore and *) is really nice.

Figure 41: geoWrite initialization

You can certainly imagine what a powerful tool EDMON is when you think that you can save areas of memory with it. You can not only save geoWrite with it, but also the GEOS KERNAL, in the normal DOS format or in GEOS format.

In conclusion we want to give you a little tip. It is very easy to write your own programs in GEOS with EDMON. You can then save them as accessories (file type 5) and then start and test them directly from GEOS with a double-click. You can develop a program that will later run under GEOS as an application or accessory as small modules, test them individually, and save them.

If you have problems with the initialization and end of your modules, you might like to look at the hardcopy program to see how we did it.

Chapter 5

GEOS for the Programmer

5.1. Examining GEOS

When we first used GEOS and saw this wonderful user interface, we naturally wanted to find out how it worked. But as we soon discovered that it isn't so easy to see GEOS behind the scenes. It was necessary to do so, however, or we wouldn't have been able to bring you most of the information in this book.

We will also help those who want to look behind the scenes themselves and want to be able to look at a disassembled accessory or deskTop.

You will need the following things:

- a machine language monitor

- a disk monitor

- and a copy of the original disk (never use the original itself)

For our work we used the monitor from the Abacus Software *Assembler Monitor-64* disk,the version with a starting address $C000. This monitor loads programs at various addresses and can look under the ROM. The disk monitor we used was from the Abacus book *The Anatomy of the 1541 Disk Drive*. *QuickCopy* was used to make a copy of the GEOS boot disk to examine the GEOS and GEOS BOOT programs.

5.2 Getting into GEOS

There are a number of difficulties when you try to get acquainted with the internal workings of GEOS. They start with the fact that GEOS has an autostart and ends with the KERNAL which is largely hidden (to save space) under the ROM. The KERNAL is also encrypted on the master diskette of GEOS V1.3.

We will only look briefly at the first two programs GEOS and GEOS BOOT. These two programs are only needed for the loading process, and both can be understood without much trouble. Please make a complete copy of the GEOS System diskette with a program such as *QuickCopy*. **Do not use the original system diskette.**

GEOS: Load with autostart

The loader "GEOS" starts at $0110, which loads it directly into the stack. This is the special part of memory ($0100-$01FF) in which the return addresses of machine language subroutines are stored. (In BASIC, when the interpreter encounters a GOSUB command, the interpreter saves the address to which it must RETURN. There are comparable commands in machine language, and the addresses are saved on the stack.)

This stack can be viewed with a machine language monitor, but the contents are constantly being changed by the monitor. If you want to look at the autostart "GEOS", load it at another address, such as $1110. This is a $1000 offset and it is easy to follow the program.

If you can't select the load address with your monitor, you can use the following program:

```
 1 REM LOADADDR CHAPT 2.1 GEOS TRICKS/TIPS
 5 X=4368
 : REM " = $1110"
10 OPEN 3,8,3,"GEOS"
20 GET#3,A$
30 GET#3,A$: REM SKIP LOAD ADDRESS
40 GET#3,A$:A=ASC(A$+CHR$(0))
50 POKE X,A : PRINT A;
60 IF ST=0 THEN X=X+1:GOTO 40
70 CLOSE 3
```

After running this program you can view GEOS at $1110 with a monitor.

GEOS BOOT: The quick-loader

GEOS BOOT starts at $6000. If your monitor resides at $6000 you can use the program above by replacing GEOS with GEOS BOOT and putting the new load address in line 5.

GEOS BOOT reduces the load time from two minutes to about 30 seconds.

Applications

Now we want to come to the interesting things. How can we look at an application or an accessory and even print a disassembly of it, in order to understand it and be able to change it?

Problem: USR file

A GEOS application or accessory is a USR file. If you try to load it with

```
LOAD "geoWrite",8
```

the disk drive will respond with "?FILE NOT FOUND ERROR".

Problem: Different ASCII coding

Some files can be loaded with

```
LOAD "NAME,U,R",8
```

But this does not work for many GEOS programs because the filename is in normal ASCII format instead of Commodore ASCII. The disk drive will respond:

```
"FILE NOT FOUND ERROR"
```

Problem: VLIR file

Moreover, geoWrite is a VLIR file, and these cannot be loaded by the normal floppy DOS. In normal Commodore DOS there is a pointer in the file entry to the first data block of a file. This block has a pointer to the next data block, and so on. The file consists of a chain of blocks.

A VLIR file can have several such "chains." Naturally, GEOS must know where they all begin. The pointer in the file entry points to a single sector. This sector contains the pointers for the start of the chains (pointer sector). A GEOS program which is stored in VLIR format consists of several parts, the starts of which are entered in the pointer sector.

209

Let's take a closer look at this complicated structure. Insert a copy of the GEOS applications diskette.

Using the disk monitor from the Abacus book *Anatomy of the 1541* we load track 18, sector 1. The file entry for geoWrite starts at $60.

```
>:60 00 00 C3 07 0B 47 45 4F  ..-..geo
>:68 57 52 49 54 45 A0 A0 A0  Write
>:70 A0 A0 A0 A0 A0 07 03 01
>:78 06 56 0A 05 0D 2D 5A 00
```

The important bytes for us at the moment are:

$63 = $07 = track

$64 = $0B = sector

For a normal DOS file this would mean that the first data block was at track 7, sector 11. But since geoWrite is a VLIR file ($77=1), this sector contains only the pointers to the individual records. In other words: In track $07, sector $0B are the pointers to the start of the "chains" which make up geoWrite.

Let's take a look at this sector, by loading track 7, sector 11:

```
>:00 00 FF 07 04 09 08 0C 13
>:08 0D 0F 0D 10 0E 11 0E 12
>:10 00 00 00 00 00 00 00 00
>:18 00 00 00 00 00 00 00 00
>:20 00 00 00 00 00 00 00 00
```

The first two bytes indicate that there is no block after this one (always the case with a VLIR pointer sector) and that $FF=255 valid data bytes are present. There are only fourteen bytes after this which are not zero:

$07	=	track record 1
$04	=	sector record 1
$09	=	track record 2
$08	=	track record 2
$0C	=	track record 3
$13	=	sector record 3
$0D	=	track record 4
$0F	=	track record 4
$0D	=	track record 5
$10	=	sector record 5
$0E	=	track record 6
$11	=	track record 6
$0E	=	track record 7
$12	=	sector record 7

If you find other numbers, write them down and use them later instead of the numbers printed. Since the remaining pointers are zero, geoWrite consists of seven parts (records).

Finding the start address of GEOS files

In conclusion we would like to give you some tips in case you want to disassemble the applications or accessories. You can find the start address in the INFO sector. The track and sector of the INFO sector is in the file entry after the last $A0 which pads the filename. Let's take a brief look at geoWrite:

```
>:60 00 00 C3 07 0B 47 45 4F ..-..geo
>:68 57 52 49 54 45 A0 A0 A0 Write
>:70 A0 A0 A0 A0 A0 07 03 01
>:78 06 56 0A 05 0D 2D 5A 00
```

The INFO sector is at track $07, sector $03. If we load this the load address is in:

$47 = $00 = load address LOW

$48 = $04 = load address HIGH

geoWrite therefore starts at $0400 in memory and is located exactly where the video RAM would be. To examine geoWrite, it must be loaded at a different address, like $1400.

Now we want to load geoWrite at a different address in memory. To do this we create a file entry called "WRITE1" and set its pointer to record 1 (track 7, sector 4). The easiest way to put an entry in the directory is to save a simple BASIC program. Exit the monitor and enter the following one-liner:

```
10 REM DIRECTORY ENTRY ONLY
```

Now save this program on your copy of the GEOS disk under the name "WRITE1".

```
SAVE "WRITE1",8
```

We search for this file entry with the disk monitor. It might look like this (your disk may have different values):

```
>:C0 00 00 82 19 00 57 52 49 .....WRI
>:C8 54 45 31 A0 A0 A0 A0 A0 TE1
>:D0 A0 A0 A0 A0 A0 00 00 00
>:D8 00 00 00 00 00 00 00 00
>:E0 00 00 00 00 00 00 00 00
```

The two memory locations which are important to us are:

$C3 = $19 = track

$C4 = $00 = sector

These two locations contain the pointer to the first data block in the program. We simply set this pointer to record 1 of geoWrite. Replace $19 with $07 and $00 with $04.

```
>:C0 00 00 82 07 04 57 52 49  .....WRI
>:C8 54 45 31 A0 A0 A0 A0 A0  TE1
>:D0 A0 A0 A0 A0 A0 00 00 00
>:D8 00 00 00 00 00 00 00 00
>:E0 00 00 00 00 00 00 00 00
```

Then write the modified directory sector back to the disk. If you have different values for the track and sector, use them here.

With a monitor that can load at any address you can load the first part of geoWrite. The first two bytes of the program will not be loaded. Normally these contain the start address, but GEOS gets the start address from the INFO sector, so the first two bytes actually belong to the program. We will look at these and write them down.

To do this we load the first data block of record 1. Our data block is at track $07, sector $04.

```
>:00 07 0C 4C AE 28 4C 38 34
>:08 20 BD C1 20 26 22 20 27
>:10 04 20 9F 04 4C 9F 05 20
>:18 BD C1 20 26 22 20 27 04
>:20 20 D7 05 20 09 05 4C 2C
```

The first two bytes ($07, $0C point to the track and sector of the next data block. Then the program starts with $4C $AE $28. If we would load "WRITE1" with

```
        LOAD "WRITE1",8,1
```

the program would be loaded at $AE4C and would start with $28. We write down the $4C and $AE and replace these two values with $02 and $10. This sets the start address to $1002. This value may seem a bit strange to you, but after we load the program with $C4 starting at $1002, we can easily put the $4C and $AE in front of it. After we have loaded "WRITE1", the first part of geoWrite is in memory.

The disk buffer should now look like this:

```
>:00 07 0C 02 10 28 4C 38 34
>:08 20 BD C1 20 26 22 20 27
>:10 04 20 9F 04 4C 9F 05 20
>:18 BD C1 20 26 22 20 27 04
>:20 20 D7 05 20 09 05 4C 2C
```

Write the modified sector back to the work disk. Using your monitor fill the area from $1000 to $5FFF with $F1. This will allow you to tell where record 1 ends. Load "WRITE1" either from the monitor or with:

```
LOAD "WRITE1",8,1
```

If you look at the memory from $1000 on with the monitor, you will see that the program starts at $1002. In front of this are two $F1's which you filled the memory with.

```
>:1000 F1 F1 28 4C 38 34 20 BD
```

Now you can enter the two missing bytes.

```
>:1000 4C AE 28 4C 38 34 20 BD
```

You can find out the end address by looking for the $F1's. The place where these begin is the end of record 1. With our monitor the command is:

```
H 1000 6000 F1 F1 F1
```

The first address which is printed is $354D. This means that record 1 runs from $1000 to $354D.

The best thing to do is save the area $1000-$354D on another disk. Then you can make record 2 thru 7 loadable. Just put the pointer ($09, $08) to record 2 (thru 7) in the file entry of "WRITE1" and follow the same procedure as record 1.

Now you can look at disassemble, and print the individual sections of geoWrite without any problems. We used another trick which made our work much easier. If you want to print a disassembly of geoWrite, it would produce a lot of paper since only one line is written per line. We sent the disassembly to disk instead of printing it directly. For our monitor it went like this:

Record 1 lies from $1000 to $354D. The monitor is at $C000. Insert a disk with lots of room (at least 540 free blocks) and enter the following BASIC program:

```
10 OPEN 3,8,3,"PART1,S,W": REM OPEN FILE
20 CMD 3: REM REDIRECT OUTPUT TO DISK
30 SYS 49152: REM START MONITOR
```

213

Start the program. The sequential file "PART1" will be opened for writing and
the output to the screen will be redirected to this file. Then the monitor is
started. When the cursor appears again (the initial message will be sent to the
file instead of the screen, enter:

```
D 1000 354D (RETURN)
```

After quite some time the cursor will appear again. Enter:

```
X (RETURN)
PRINT#3 (RETURN)
CLOSE 3 (RETURN)
```

Now you have a disassembly of "PART1" of geoWrite on the disk. You can
look at it with the following program:

```
 1 REM PART1 DISPLAYER CHAP 5.2
 5 PRINT CHR$(147)
10 OPEN 3,8,3,"PART1,S,R"
20 GET#3,A$:
25 IF PEEK(203)<> 64 THEN 50
30 PRINT A$;
40 IF ST=0 THEN 20
50 CLOSE 3
```

When you start the program, the following appears on the screen:

```
*** MONITOR 64 V2.0 ***
(C) 1984 ABACUS SOFTWARE
C*
   PC  IRQ  SR AC XR YR SP  NV-BDIZC
>*E145 EA31 4E 00 31 35 F8  01001110
>
>,1000   4C AE 28    JMP $28AE
>,1003   4C 38 34    JMP $3438
>,1004   20 BD C1    JSR $C1BD
```

This is followed by the rest of the disassembled record 1. The output will end
when you press a key. If you use a different monitor, the beginning will
naturally look different. We have written a program which reads this file into an
array and prints it in multiple columns on a page. On our Gemini 10X in elite
we could get four columns on a page. "PART1" was then quite easy to read.

```
10 OPEN3,8,3,"PART1,S,R"
15 OPEN1,4,1
20 PRINT CHR$(147)
22 PRINT#1,CHR$(15): REM COMPRESSED ON
30 GET#3,A$:IFA$="" THEN A$=CHR$(0)
35 IF B$=">"ANDA$="," THEN 100
40 PRINTA$;
45 B$=A$
50 X=X+1
```

```
90 IF X<200 AND PEEK(203)=64 AND ST=0 THEN 30
99 CLOSE3:PRINT"START OF DISSASSEMBLY NOT FOUND":CLOSE1:END
100 PRINT"------------------":
110 DIM A$(60,4)
120 IF ST<>0 THEN CLOSE3:CLOSE1:END
130 FOR J=1 TO 4
140 FOR I=1 TO 60
145 IF AN=0 THEN AN=1:GOTO 160
150 GET#3,B$:GET#3,B$
160 REM
200 GOSUB 900
205 IF ST<>0 THEN W$="END":GOTO 500
210 A$(I,J)=N$:PRINTCHR$(147);A$(I,J),I;J,ST
215 NEXTI
220 NEXTJ
250 GOTO 500
300 REM
400 STOP
499 REM ************************
500 FOR I - 1 TO 60
510 FOR J - 1 TO 4
520 PRINT#1,A$(J,I);"   ";
530 NEXT J
540 PRINT#1,CHR$(10);:REM LINEFEED
550 NEXT I
555 FORK=1TO6:PRINT#1,CHR$(10):NEXTK
560 IF W$ = "END" THEN CLOSE3:CLOSE1:END
570 IF ST=0 THEN GOTO 130
580 CLOSE3:CLOSE1:END
899 REM ******************
900 N$="": IF ST <> 0 THEN 950
901 IF PEEK(203)<>64 THEN PRINT"CANCEL":CLOSE3:CLOSE1:STOP
905 N$="":
910 GET#3,A$:IF A$="" THEN A$=CHR$(0)
920 IF A$=CHR$(95) OR A$=CHR$(13) THEN 950
930 N$=N$+A$
940 GOTO 910
950 IF LEN(N$)<23THEN N$=N$+" ":GOTO 950
960 RETURN
```

This program is intended just to give you some ideas. The filename for "PART1" must be entered in line 10. A channel is opened for the printer and the printer is set to elite (check your printer manual for the correct codes). Lines 30 to 99 skip past the opening message of the monitor. The end of the message is recognized when the first ">," occurs (this may be different depending on which monitor you are using).

An array is then dimensioned which can hold one complete page. Then the lines are read column by column until the page is completed. The variable-length commands are padded to the same length so that the columns line up. The page is then printed.

Accessories

Problem: USR file

A GEOS accessory is a USR file. If you try to load it with

```
LOAD "ALARM CLOCK",8
```

the disk drive will respond with "?FILE NOT FOUND ERROR".

Problem: Different ASCII coding

Such files can be loaded with

```
LOAD "NAME,U,R",8
```

But this is not possible for many GEOS programs, such as the ALARM CLOCK, because the filename is in standard ASCII format instead of Commodore ASCII. The disk drive just responds:

```
"FILE NOT FOUND ERROR"
```

The ALARM CLOCK may be loaded by using CHR$ codes:

```
LOAD CHR$(97)+CHR$(108)+CHR$(42)+",U,R",8,1: REM al*
```

If you want to look at a GEOS file which does not have the VLIR structure, the two bytes in the file entry behind the file type point directly to the first data block instead of to the sector which contains the pointers to the records. You can then place this track and sector directly in the file entry you created. Assuming you want to load the note pad, and the file entry for note pad looks like this:

```
>:A0 00 00 83 05 08 6E 6F 74 .....NOT
>:A8 65 20 70 61 64 A0 A0 A0 E PAD
>:B0 A0 A0 A0 A0 A0 05 00 00
>:B8 05 56 0A 02 0D 02 11 00
```

Create a file entry called "Notes1" with the one-line BASIC program mentioned earlier, and put $05 and $08 right after the file type. (Your note pad may have a different track and sector for the first data block. Use your values instead.)

By the way, the byte after the track and sector of the INFO block indicates whether a GEOS file has the structure VLIR or SEQ. In our example for note pad there is a 0 = SEQ in this byte, while a 1 would indicate VLIR:

```
>:B0 A0 A0 A0 A0 A0 05 00 00
```

216

This value is always 1 for GEOS KERNAL, geoWrite, and geoPaint

With these tricks and a lot of compact pages of listings, we were finally able to get to the inside of GEOS. We were able to find a number of locations that could be changed in order to get more out of GEOS. If you want this, sort of thing yourself, we recommend the Abacus book *GEOS Inside and Out* which contains information on the construction of the applications, accessories, INFO sectors, and VLIR files.

5.3 GEOS routines

GEOS offers a large number of routines for programming your applications and accessories. These routines handle a variety of tasks for the programmer, such as reading the keyboard and mouse control. In spite of this there are currently no programs on the market (other than those included with GEOS) that run under GEOS and make use of these possibilities. We think there are two reasons for this:

- There is little information about the GEOS routines and their functions. Therefore programmers have no opportunity to use them.

- Programming under GEOS is significantly different from programming with the usual C64 KERNAL routines. You will probably have difficulties writing executable programs with the GEOS routines.

Much necessary information can be found in *GEOS Inside and Out* from Abacus. We will not repeat topics that are described in detail there (job loop, window techniques, and the INFO sector). One thing that is very important for successful programming under GEOS is the single-step simulator printed in that book.

In this chapter we want to give you another tool that wasn't available previously: The jump table of the GEOS routines. It is at least as important as the jump table in the C64 KERNAL. If you know these routines and their parameters, you are half done with the program.

But even if you don't want to write any programs under GEOS you can find a lot of important information in this chapter: What range is available for BASIC programs when GEOS is in memory, which locations can be changed in memory to make certain modifications to GEOS, and how GEOS actually works.

This information was discovered by disassembling GEOS V1.2.

5.4 How GEOS works

Actually, the term GEOS in this book is not very precise. For example, when we say that GEOS displays a window with certain contents or GEOS copies a disk, we aren't quite correct. On the other hand, it isn't of much interest to those who just want to use GEOS. But if we want a given look at how GEOS works, we have to express ourselves a bit more accurately.

Actually GEOS is "only" the GEOS KERNAL. In addition there are two applications, six accessories and another system file (deskTop). The deskTop has a special status, although it has a strong resemblance to an application. There is a special routine to load the deskTop and when booting, a file with the name deskTop is automatically loaded after the GEOS KERNAL.

The GEOS KERNAL offers powerful routines for, among other things, creating windows, loading and saving disk sectors. The deskTop uses these routines to give messages before certain actions or to copy a file. But the KERNAL isn't just a collection of finished routines waiting to be accessed by the deskTop or an application. The GEOS KERNAL is itself an active part of the program. We will explain this small but fine distinction with an example.

On a C64 without GEOS there is a routine which reads the keyboard. If a machine language programmer needs input from the keyboard, he jumps to this routine and gets the code for a character back in the accumulator, or a 0 if no key was pressed. With GEOS the KERNAL isn't so passive. If a program wants to react to keyboard input, it doesn't read the keyboard at regular intervals; it passes an address to the KERNAL to which it is to branch if a key is pressed. The program itself doesn't have to worry about anything else.

This difference is so distinctive that we will describe it in general. A program that runs under GEOS is connected to the KERNAL by events. It passes various entry addresses to the KERNAL to which the KERNAL branches on specific events. Naturally, there must be routines ready to handle these events.

This difference has significant effects on the manner in which programs under GEOS are to be written. Your own programs should adhere to certain rules so that GEOS can function properly. It would be wrong, for example, to write a program that read the keyboard itself in a loop after the program was started, and jumped to the routine "load and start deskTop" when the "a" key was pressed. This style of programming does not use the possibilities that GEOS offers.

We would like to briefly explain how to start your own program and how it should end. Let's assume we are writing an accessory, with EDMON, for instance. At the beginning of this program there should be an initialization routine. The start address will automatically be placed in the INFO sector by

EDMON when the program is saved. This initialization header differs radically from the initialization of a machine language program that would run in the normal C64 mode.

In the latter case various memory locations would be set to the desired values, perhaps a few jump vectors modified, and then the program would proceed to the actual main routine. This main routine would then control the rest of the operation of the program.

Under GEOS an initialization routine must pass the addresses of various subroutines for certain events to the KERNAL and then end with an RTS. This is the main difference.

Let's assume we're writing an application. This will have pull-down menus and will process keyboard input. How would the initialization of such a program go? How would GEOS start this program?

If our application uses the standard GEOS features, it appears as an icon on the user interface. When the user then double-clicks this program or loads it with open, GEOS (or more accurately, the deskTop) first looks to see what kind of program it is. After it determines that the program is an application, a KERNAL routine is executed that is responsible for loading applications. This is necessary because deskTop itself is in the area that can be used by an application and will therefore be overwritten when the application is loaded.

The KERNAL load routine fetches the load address from the INFO sector and loads the program at this address. The entry address will then be fetched and the program will be executed with a JSR to this address.

Our application initializes itself at this entry address in the following manner: First, a routine is executed which constructs a menu and prepares it for use (DoMenu, see jump table). Our program passes a set of parameters concerning the construction, position and strings to this routine. In addition, values are passed for each menu option that determine if certain subroutines will be executed when the menu is clicked or if another submenu is to be constructed.

In addition, our program writes in two different places the address of the subroutine to be executed when a key is pressed. Finally, our program ends with an RTS. This returns control to the load routine and the GEOS KERNAL. Despite this, our program is not over yet, because it is connected to the GEOS KERNAL by certain memory locations.

When the user now clicks the menu with the mouse, the KERNAL builds it according to the parameters passed. If a menu option is selected, GEOS KERNAL gets the entry address from the values passed and jumps to this with a JSR. In the same manner it gets the passed address of the desired subroutine if a key is pressed and jumps to this subroutine.

220

Only when programs cooperate with GEOS in this manner can the full
capabilities of GEOS be used. For example, any accessory can be loaded at any
time and then exited without losing any data in the application. To do this our
program reads all file entries from the disk and checks them for file type 5. All
file entries with this file type are placed in a list and the filenames are displayed
in a menu. When the user then clicks the third menu option, for example, we get
the third filename from the table and pass it to a KERNAL routine.

This stores the memory that the accessory uses, and loads and starts the
accessory. When the work with the accessory is done, the KERNAL gets the
saved memory back from the disk and returns to our application with an RTS.

Naturally there are a number of other functions which must also be executed.
For example, the KERNAL saves the entire construction of our menu so that a
menu in the loaded accessory doesn't destroy our menu.

To end our program we provide a corresponding menu option QUIT with a
subroutine to go along with it. If this menu option is selected and our routine is
executed, we make all the preparations for the end of the program (data storage)
and end our program by calling the routine "load deskTop" from the jump
table (JMP C22C). The rest is handled by the GEOS KERNAL again.

5.5 Memory layout

Of course, if we want to write our own programs, we need to know more than just the basic operation of applications. To cooperate with GEOS fully, we have to know what memory locations we can use for our programs.

$0000-$007F	Zero-page used by GEOS.
$0080-$00FF	Zero-page that can be used by applications. An accessory should first save these locations if it uses them.
$0100-$01FF	Processor stack.
$0200-$03FF	Largely unused. The range $0300-$0333 contains the standard C64 operating system vectors. GEOS sets these pointers to its own values.
$0400-$5FFF	Space for applications. If you also want to use the deskTop routines, you can use the area from $4570-$496E.
$6000-$7F3F	Second screen. This is were GEOS saves the background before displaying a window and a menu.
$7F40-$7FFF	Space for applications.
$8000-$80FF	Buffer 1 for disk data.
$8100-$81FF	Buffer 2 (usually the current INFO sector).
$8200-$82FF	Buffer 3 (usually contains the BAM, which is managed by GEOS).
$8300-$83FF	Buffer 4 (often used for storing the pointers to the tracks and sectors for load and store operations).
$8400-$8BFF	GEOS KERNAL data (job addresses, filename, disk name, sprite data).
$8C00-$8FE9	Color RAM.
$8FF8-$8FFF	Pointers to the sprite data.
$9000-$9FFD	GEOS KERNAL (PRG).
$A000-$BF3F	First screen. This memory normally corresponds to what you see on the screen.
$BF40-$BFFF	GEOS KERNAL (data).
$C000-$FFFF	GEOS KERNAL (PRG).

What's interesting is that the GEOS KERNAL starts at $9000. If you exit GEOS with SPECIAL: BASIC and then want to do a warm start, you must protect this memory area from the BASIC interpreter by setting the upper limit of BASIC to $9000. Unfortunately, the GEOS KERNAL sets this value to $A000.

You must POKE the following values in the direct mode immediately after SPECIAL: BASIC:

        POKE 52,144: POKE 54,144: POKE 56,144

In no event may any string operations be performed before these POKEs are made.

5.6 The GEOS jump table

Do you still remember the first time you saw a description of the C64 KERNAL jump table? Suddenly programming was much easier. You no longer had to write a number of useful routines yourself for your own programs, you could just jump to the KERNAL.

Such a jump table also offers a decisive advantage: Different versions of a operating system for the Commodore computers will be compatible as long as the entry addresses in the jump table point to comparable subroutines.

GEOS also contains such a jump table. All of the important actions of the GEOS KERNAL can be used from this table. Moreover, it is very likely that programs which use the jump table routines will also work properly with new versions of GEOS KERNAL. For example, many routines were changed or placed at different addresses from GEOS V1.0 to GEOS V1.2. The table still contains pointers to the same routines, however.

The rest of this chapter contains more than just the jump table—for most routines you will also find their functions and parameters. With GEOS V1.2 you can make a disassembled listing of the GEOS KERNAL as described in Section 5.2, then you can take a closer look at these routines. Restrict yourself to the jump table when writing your own programs, so that your programs will also work with newer versions of GEOS. GEOS 1.3 is encrypted on the disk, so the disassembly as described in Section 5.2 will not work with this version.

C100 JMP $E360   Vector for IRQ

Job 1: Move mouse and test its area
Job 2: Decrement timer group A
Job 3: Decrement timer group B
Job 4: Service cursor if seventh bit in $84B4 is set
Job 5: Decrement $850A (LOW), $850B (HIGH)

C103 JMP $9E7F   ProcessInit

Write type 1 jobs into job buffer 1.

- Accumulator: number of jobs.
- $02, $03: address of table.

This table must have the following construction:

                Job address (job 1) LOW
                Job address HIGH
                Timer value LOW
                Timer value HIGH
                Job address (job 2) LOW
                Job address HIGH
                Timer value LOW
                Timer value HIGH

and so on. This table contains jobs which will be executed over and over again. The timer value determines the frequency at which the job is executed. In addition, each job is assigned a status byte by the GEOS KERNAL which can be used to deactivate jobs. The IRQ counts the timer values down to 0 (see $C100) and then executes the job. The individual bits in the status byte have the following meaning:

Bit 5   If this bit is set, the IRQ will no longer decrement the corresponding timer. The job is then frozen at its current state (FREEZE).
Bit 6   Prevents the job from being executed, even when the timer value runs out (BLOCK).
Bit 7   Set by the IRQ to indicate that the timer has run out. The job can then be processed by the job loop ($C2C8) (Enable). This immediately resets bit 7 (Disable) and starts this process.

Jobs are only entered by ProcessInit. Since bit 5 (FREEZE) is also set, the timer will not run. Before a job can be processed it must be released (StartProcess). It is important to enter all of the jobs at once. There is no provision for adding a job later.

C106 JMP $9F29   StartProcess

Clears bit 5 and bit 6 in the status byte and loads the timer values. Removes any blocking (UNFREEZE and UNBLOCK). The job number is passed in the X register.

C109 JMP $9F44   EnableProcess

Although jobs are normally started when the timer runs out, any job can be started "by hand." This job will then be executed on the next pass through the main loop. The job number is passed in the X register.

225

C10C JMP $9F4D   BlockProcess

Sets bit 6 in the status byte of the job passed in the X register. The timer values will still be decremented, but this job will not be executed at 0.

C10F JMP $9F55   UnblockProcess

Unblocks the job by clearing bit 6 in the status byte. Job number in the X register.

C112 JMP $9F5D   FreezeProcess

By setting the fifth bit the process will be frozen at its current timer state. The X register must contain the job number.

C115 JMP $9F65   UnfreezeProcess

Clears bit 5 in the status byte of the job passed in the X register. The previously stopped timer will start to run again.

Graphics:

When using the graphic routines byte $2F determines where information will be written or from where it will be fetched. The pointers to the various screens are set depending on $2F. The following values for $2F have the following effects:

$80:   output only to the visible screen ($A000)
$40:   output to the invisible screen ($6000)
$C0:   output to both screens
$00:   output always at $AF00

C118 JMP $ECD7   HorizontalLine

Draws horizontal lines on the high-resolution graphic screen.

Parameters (as X/Y coordinates):

Acc: bits determine the pattern directly
$08,$09:   left border (LOW, HIGH)
$0A,$0B: right border (LOW, HIGH)
$18:   Y position of the line
X value: (0-319)
Y value: (0-199)

C11B JMP $ED5C   InvertLine

Inverts a line on the graphics screen.

Parameters (as X/Y coordinates):

$08,$09:   left border (LOW, HIGH)
$0A,$0B: right border (LOW, HIGH)
$18:   Y position of the line
$12:   normally 0; if a different value is placed here, the set bits in the first
            byte to the left of the line will be retained
$13:   same for the right border
X value: (0-319)
Y value: (0-199)

C11E JMP $EDFE   RecoverLine

Gets a line back from the second screen. This is used to restore the original
screen after a menu, for example.

$08,$09:   left border (LOW, HIGH)
$0A,$0B: right border (LOW, HIGH)
$18:   Y position of the line
$12:   normally 0; if a different value is placed here, the set bits in the first
            byte to the left of the line will not be gotten
$13:   same for the right border

C121 JMP $EE87   VerticalLine

Draws a vertical line.
Acc: line pattern
$08:   upper bound
$09:   lower bound
$0A: X position LOW
$0B: X position HIGH

C124 JMP $EEF3 Rectangle

Fills a rectangle with a pattern.

$06:	upper bound
$07:	lower bound
$08,$09:	left bound (LOW, HIGH)
$0A,$0B:	right bound (LOW, HIGH)
$22,23:	pointer to the pattern

The pattern must consist of eight bytes, resulting in an 8x8 matrix. One of the sixteen KERNAL patterns can be used easily by first calling the routine SetPattern ($C139). To do this, SetPattern must be passed by the number of the desired pattern (0-15) in the accumulator.

C127 JMP $EF61 FrameRectangle

Draws a rectangular frame.

Acc:	write value
$06:	upper bound
$07:	lower bound
$08,$09:	left bound (LOW, HIGH)
$0A,$0B:	right bound (LOW, HIGH)
$22,23:	pointer to the pattern

C12A JMP $EF0A   InvertRectangle

Inverts a rectangle.

$06:	upper bound
$07:	lower bound
$08,$09:	left bound (LOW, HIGH)
$0A,$0B:	right bound (LOW, HIGH)
$12:	set bits on the left edge will not be inverted
$13:	set bits on the right edge will not be inverted

Bytes $12 and $12 prevent an icon from becoming smaller when it is inverted. Otherwise the inverted frame would become the color of the background.

228

C12D JMP $EF26   RecoverRectangle

Gets a rectangle from the second screen. This is used to restore the screen contents after a window is removed, for example.

$06:   upper bound
$07:   lower bound
$08,$09:   left bound (LOW, HIGH)
$0A,$0B: right bound (LOW, HIGH)
$12:   set bits on the left edge will not be restored
$13:   set bits on the right edge will not be restored

C130 JMP $DC10   DrawLine

This routine draws lines. It is much more flexible than HorizontalLine or VerticalLine, but is also considerably slower.

$08:   X position point 1 LOW
$09:   X position point 1 HIGH
$0A: X position point 2 LOW
$0B: X position point 2 HIGH
$18:   Y position point 1
$19:   Y position point 2

In addition the following status bits determine the execution of this function:

Bit 7 (negative flag) = 1: Bit 0 (carry flag) has no meaning. In this case things are a bit complicated. It must be handled first however, since the negative flag has priority.

S1   = screen in $0C,$0D (e.g. at $A000)
S2   = screen in $0E,$0F (e.g. at $6000)

Byte $2F again determines which screen is the destination of the output.

In this case the line on the active screen is erased. If two screens are enabled, this line will be coped from the second to the first.

geoPaint can use this function for an UNDO. The current action takes place only on S1. If you want to undo something, the same function is called again in this mode (bit 7 = 1). The modified part of the screen is then fetched from S2 and copied to S1.

If the negative flag = 0, the carry flag has the following effect:

Bit 0 (CY) = 1: draw a line
Bit 0   = 0: clear a line

C133 JMP $DDB4   DrawPoint

In this routine the same conditions apply for the status as in DrawLine ($C130).

$08:   X position of point LOW
$09:   X position of point HIGH
$18:   Y position of point

C136 JMP $EFEB   DoDefaultJobs

Executes jobs from a table:

Table of LOW values   = $EFFF
Table of HIGH values   = $F009

A pointer to a table of the desired job number is passed in $02, $03. The necessary data for the job follows immediately after the job number. End with job number 0.

Job number 0 = RTS = end

Job number 1 = $F013: GetStartPoint

Function:

Get three bytes and write them to $87D4 (X LOW), $87D5 (X HIGH), and $87D6 (Y). These coordinates will be used as the starting point for the following jobs.

Job number 2 = $F020: DrawLineToPoint

Function:

The coordinates loaded previously by job 1 are placed in $08, $09, and $18 and job 1 is executed to get the coordinates of another point (three bytes). These three new points are then written into $0A, $0B, and $19. The routine DrawLine is then called with the carry bit set and the value flag cleared (see $C130).

Job number 3 = $F03E: FillArea

Function:

Gets the second coordinate (3 bytes). Calculates a rectangle from the two points and fills this with a pattern.

Condition for job 3: The coordinates of the first corner point are set with job 1 and the desired pattern is selected with job 5. Then FillArea can be called.

Example for a table which uses job 3 to fill a rectangle from P1 to P2 with a pattern:

$03, $02 = pointer to the table = $1000

Then DoDefaultJobs.

Data in the table:

$1000 - $01 GetStartPoint (P1)

$1001 - $0A X LOW of P1

$1002 - $00 X HIGH of P1

$1003 - $10 Y of P1

$1004 - $05 DefPattern

$1005 - $01 pattern number (black)

$1006 - $03 FillArea

$1007 - $50 X LOW of P2

$1008 - $00 X HIGH or P2

$1009 - $60 Y of P2

$100A - $00 end of table

Job 4 = $F044: RTS

This job does nothing. Processing of the table simply continues.

Job 5 = $F045: DefPattern

Function:

The following byte specifies the number of a desired pattern (0-31).

Job 6 = $F04B: OutString

Function:

Outputs a string at a given location. Requires three bytes for the position: X LOW, X HIGH, Y, followed by a two-byte pointer to the string. The text must be terminated with a 0.

Job 7 = $F05E: FrameArea

Function:

Draw a rectangular frame. Requires three bytes again to specify the position of the second point. P1 must be set with job number 1.

Job 8 = $F06A: XOffset

Function:

Increments the X coordinates of P1 by the specified value. Requires two bytes: X LOW and X HIGH.

Job 9 = $F086: YOffset

Function:

Increments the Y coordinates of P1 by the specified value. Requires one byte (Y).

Job 10 = $F066: XYOffset

Function:

Like jobs 8 and 9. Requires three bytes: X LOW, X HIGH, and Y.

C139 JMP $F0F1   SetPattern

Set the pointer to a desired pattern. The number of the pattern (0-31) is passed in the accumulator.

C13C JMP $F11B   GetLineStart

Calculates the start address of the line whose number (0-199) is passed in the accumulator. $2F again determines the screen addressed.

The result in $0C, $0D and $0E, $0F can be used directly for additional graphic output.

C13F JMP $DDF1   TestPoint

Tests a graphics point: $08, $09 = X position, $18 = Y position. If the point is set, the carry flag will be set, else it will be cleared. $2F determines which screen will be tested.

C142 JMP $DAE6   DoIcon

Draws a picture of any size. The icons on the deskTop are drawn with this routine.

The following parameters must be passed:

$02,$03   = address of the picture data
$04   = X position (0-39)
$05   = Y position (0-199)
$06   = width of the picture in bytes
$07   = height of the picture in lines

The X position can only be specified in eight-bit steps and will automatically be multiplied by eight.

C145 JMP $DE11   PutChar

Outputs the character in the accumulator at the current position. Characters less than 32 ($20) will be interpreted as control characters.

C148 JMP $DFAD   PutString

Outputs a string. A pointer in $02, $03 must point to the start of the string, and the string must be terminated with a 0.

C14B JMP $DFC0   UseSystemFont

Switches to the BSW operating system (see the eight memory locations at $26).

C14E JMP $E5A4   InitMouse

Initializes the mouse.

If the carry flag is set and there is a value other than zero in $19, $18, the mouse position (X) will be taken from $19, $18 and the mouse position (Y) will be taken from the Y register. The speed will be set to zero.

Menu checking will be initialized regardless of the carry flag.

C151 JMP $E73C   DoMenu

This routine creates a complete pull-down menu. The following parameters must be passed:

```
$02,$03    = pointer to a data table.
Construction of the table:
Byte 0     = Y position up
Byte 1     = Y position down
Byte 2     = X position left LOW
Byte 3     = X position left HIGH
Byte 4     = X position right LOW
Byte 5     = X position right HIGH
Byte 6     = status byte (see below)
Byte 7     = pointer to menu label 1 LOW
Byte 8     = pointer to menu label 1 HIGH
Byte 9     = build flag
Byte 10    = pointer for menu option 1 LOW
Byte 11    = pointer for menu option 1 HIGH
```

The rest of the table continues like bytes 7-11 for the remaining menu options.

The status byte (byte 6) is constructed as follows:

Bits 0-4   Number of menu options
Bit 6   = 1, the mouse can not exit the menu borders
            = 0, mouse can be moved freely
Bit 7   = 1, menu will be constructed vertically, else horizontally as in the
            deskTop.

The bits of the construction flag have the following meanings:

Bit 7   = 1, then the pointer for this menu option points to the data for
            another submenu which has the same menu structure. This can be
            used to create any number of submenus.
Bit 7   = 0, the pointer for this menu option points to the entry address of
            the selected routine. This must remove the menu itself by calling
            $C190 and must end with RTS.
Bit 7   = 0 and bit 6 = 1, the pointer for this menu option also points to an
            entry address. After the routine there has been processed, a submenu
            will be created. The pointer to the submenu data must be passed in
            $02, $03 before the routine ends with RTS.

This can be used to execute jobs while the menu is active.

The individual menu labels must be terminated with 0.

C154 JMP $E996   ClrActMenu

Sets the background of the current menu when it is being removed. If $84B1,
$84B2 = 0, then the background will be colored white (see deskTop menu
bar), otherwise the address of a routine which restores the background must be
stored in the two memory locations.

C157 JMP $E985   GotoFirstMenu

Removes the displayed submenus. ClrActMenu is used to do this.

C15A JMP $EB4B   DoClickIcon

Creates clickable symbols, such as the disk symbol on the deskTop.

$02,$03   = address of the table
Construction of the table:
Byte 0   = number of desired click fields
Byte 1   = desired mouse position X LOW
Byte 2   = desired mouse position X HIGH
Byte 3   = desired mouse position Y
              for each click field
Byte 4   = pointer to pattern 1 LOW
Byte 5   = pointer to pattern 1 HIGH
Byte 6   = X position (0-39)
Byte 7   = Y position (0-199)
Byte 8   = width of the picture in bytes
Byte 9   = height of the picture in lines
Byte 10   = job address for click LOW
Byte 11   = job address for click HIGH

Same as bytes 4-9 for all additional click fields.

After drawing the pictures on the screen the click fields will be activated and continually tested for clicks.

C15D JMP $FA6D   DShiftLeft

The X register must point to a two-byte address in the zero page which will be shifted Y times to the left.

C160 JMP $FA83 BBMult

Multiplies two bytes on the zero page. The X and Y registers serve as pointers to these bytes. The result will be placed in the memory location pointed to by X.

C163 JMP $FAA4   BMult

Multiplies a two-byte value (X) by a byte (Y); result in (X).

C166 JMP $FAA9   DMult

Multiplies a two-byte value (X) by a two-byte value (Y); result in (X).

C169 JMP $FADC   DDiv

Divides a two-byte value (X) by a two-byte value (Y); result in (X).

C16C JMP $FB0A   Computation routine

C16F JMP $FB26   Computation routine

C172 JMP $FB2B   Computation routine

C175 JMP $FB3E   DDec

Decrement a two-byte value by 1.

C178 JMP $CE4D   ClrRam

Fills a memory area with 0.

Parameters:
$02,$03   = number of bytes
$04,$05   = start address.

C17B JMP $CE51   FillRam

Fills a memory area with the value from $06.

Parameters:

$02,$03   = number of bytes
$04,$05   = start address
$06   = start value

C17E JMP $FBCE   Copy

Parameters:

$02,$03   = source address
$04,$05   = destination address
$06,$07   = number of bytes

C181 JMP $CE73   TabCopy

Copies blocks which are designated in a table. The pointer to the table must be passed in $02,$03.

Construction of the table:

Byte 0   = destination address LOW
Byte 1   = destination address HIGH
Byte 3   = number of bytes in this block
Byte 4   = data block starts here

After the data block the table continues as byte 0 for the next block. The end of the table is marked with destination address = $0000.

C184 JMP $E265

C187 JMP $FB4B   IRQTask1

C18A JMP $E5E9   MouseOn

Turn mouse on.

C18D JMP $E5E0   MouseOff

Turn mouse off.

C190 JMP $E81C   DoPreviousMenu

Clear last submenu and make the previous menu the current one.

C193 JMP $E805   ReDoMenu

Reconstructs the current menu or submenu.

C196 JMP $FD28

C199 JMP $9FDE   WaitJob

Causes a delay in program execution. The delay time is passed in $02,$03. This time is then available for the other jobs in the main loop.

C19C JMP $E5D5   ClearMouseMode

Sets the mouse flag ($30) to zero and turns the sprite off.

C19F JMP $EEE7   DatRectangle

In contrast to the function Rectangle, the data is placed directly behind the command JSR DatRectangle. This looks as follows:

```
1000 JSR $C19F
1003 byte 1 for $06 (up)
1004 byte 2 for $07 (down)
1005 byte 3 for $08 (left)
1006 byte 4 for $09
```

1007 byte 5 for $0A (right)
1008 byte 6 for $0B
1009 ...the program continues here...

The operation is the same as Rectangle. The program execution automatically continues directly behind the data.

C1A2 JMP $EF52   DatFrameRectangle

Works like FrameRectangle ($C127); data must follow the command immediately.

C1A5 JMP $EF1A   DatRecoverRectangle

Works like RecoverRectangle ($C12D); data immediately after.

C1A8 JMP $EFD8   DatDoDefaultJobs

Works like DoDefaultJobs ($C136); jobs and data immediately following.

C1AB JMP $DAB8   DatDoIcon

See C142 JMP $DAE6; parameters immediately after.

C1AE JMP $DF74   DatPutString

See C148 JMP $DFAD; the data have the following format:

Byte 0   = X position LOW
Byte 1   = X position HIGH
Byte 2   = Y position
Byte 3-   = string to be outputted, must be terminated with a zero.

C1B1 JMP $D4F8   GetRealSize

Gets the size of the current character from the current font. The various type styles (**bold**, <u>underline</u>) are taken into account.

The ASCII value of the character must be passed in the accumulator and the current type style (from $2E) must be passed in the X register. The width will be stored in Y and the height in A (without descenders).

C1B4 JMP $FC68   DatFillRam

See FillRam (C17B JMP $CE51); the five data bytes must follow.

C1B7 JMP $FBA0   DatCopy

Like C17E JMP $FBCE, with six data bytes following.

C1BA JMP $E00B   ModString

Output a string with ability to modify it.

Parameters:

$02,$03   = pointer to the string to be printed, terminated with 0
$04   = end-of-line flag
$05   = line of output
$06   = maximum length of string
$0A,$0B   = job on end of line

If the end of the line is reached and $04 is not 0, the job in $0A, $0B will be executed. This allows editing.

C1BD JMP $E80B   DoFirstMenu

Clears the menu structure back to the main menu.

C1C0 JMP $E1BD   InitTextPrompt

Sets the length of the cursor bar independent of the current font. The height of the character set (stored in $29) must be passed in the accumulator. Enables the cursor.

C1C3 JMP $C2C8   JobLoop

The routines here play a central in GEOS. Here practically the entire job structure is controlled in one loop. See the book *GEOS Inside and Out* by Abacus Software for details.

C1C6 JMP $9DE7   InitSprite

Copies a sprite block into one of eight available buffers. From there the sprite can be made visible by turning it on (see $C1D2).

$08   = number of the block (0-7)
$0A,$0B   = pointer to the sprite data

C1C9 JMP $DFEF   GetSize

Gets the width of the current character (in the accumulator) and returns it in the accumulator. Type styles (**bold**, outline, ...) are not taken into account.

C1CC JMP $DFC8   InitFont

Enables a new font. The start address must be in $02, $03.

C1CF JMP $9E0D   PosSprite

The sprite is positioned at the given location.

$08   = number of the sprite
$0A   = X position LOW
$0B   = X position HIGH
$0C   = Y position

C1D2 JMP $9E4F   EnableSprite

Enables the sprite whose number is in $08.

C1D5 JMP $9E67   DisableSprite

Disables the sprite whose number is in $08.

C1D8 JMP $CEAB   JmpInd

Executes a JSR if the address is not $0000. The high byte is in the X register and the low byte in the accumulator.

C1DB JMP $985C   CalcBlocksFree

Calculates the number of free blocks from the BAM. The pointer to the BAM must be in $0C,$0D (normally the pointer is $8200). The result will be returned in $0A,$0B.

C1DE JMP $CB93   ChkDKGEOS

Tests the BAM (pointer = $0C,$0D) for GEOS format. The result will be returned in the accumulator and in $848B. If the acc = 0, the disk is not in GEOS format, while if acc = $FF, it is a GEOS disk.

C1E1 JMP $C435

C1E4 JMP $C469   ReadBlock

Reads a sector.

$04	= track
$05	= sector
$0A,$0B	= pointer to the data buffer

C1E7 JMP $924B   WriteBlock

Writes a sector to the disk.

$04	= track
$05	= sector
$0A,$0B	= pointer to the data

C1EA JMP $98A6   SetGEOSDisk

Changes the inserted disk to "GEOS Format V1.0". Also sets the flag in $848B.

C1ED JMP $92F1   SaveFile

Saves a file on the disk.

$15,$14	pointer to a buffer which must contain the future INFO sector (usually at $8100)
$8100,$8101	must contain a pointer to the filename
$8144	DOS file type (such as $83)
$8145	GEOS file type
$8146	file structure (SEQ=0 or VLIR=1)
$8147,$8148	start of the data
$8149,$814A	end of the data to be saved
$814B,$814C	entry point

C1F0 JMP $93A8   SetGDirEntry

This is a subroutine of SaveFile ($C1ED). It creates the directory entry of a file. The parameters correspond to those for $C1ED. The address of the data to be saved is not needed.

$0E,$0F must contain a pointer to a buffer (usually $8300) which contains enough free blocks on the disk (track, sector).

The number of blocks needed must be in $06,$07.

The new directory block will then be written to the disk.

C1F3 JMP $93EE   BldGDirEntry

Subroutine of GetGDirEntry ($C1F0). The directory entry will be created in $8400-$841D, but will not be written to the disk.

C1F6 JMP $9483   GetFreeDirBlk

Searches for a free directory entry from the directory page in $16.

The Y register points in $8000 to the DOS file type of the next free entry. The error message is passed in the X register (X=0: no error, X=4: directory full).

C1F9 JMP $9532   WriteFile

Saves a file on the disk without creating an entry in the directory.

$0E,$0F   = pointer to a buffer which must contain the necessary free blocks (track, sector).
$10,$11   = pointer to the data to be saved. It will be saved in the free block until a track 0 is encountered. This serves as the end marker.

C1FC JMP $95BE   AllocNeccBlocks

Allocates the blocks needed for a save procedure on the disk. The routine will look for as many free blocks, mark them as allocated, and store them in a buffer pointer to by $0E,$0F (usually $8300). In addition, the number of bytes to be stored must be passed in $06,$07.

The error message will be returned in the X register (X=0: no error, X=3: disk full).

C1FF JMP $CB00   ReadFile

Reads a chained file. The data will be placed in RAM.

$04	= track of the first data block
$05	= sector of the first data block
$06, $07	= maximum number of data to be read
$10,$11	= address for storing the data in RAM

The error message will be returned in the X register (X=0: no error, X=11: there was more data present than was supposed to be read according to $06,$07. $06,$07 can be used as an upper limit so as not to overwrite important parts of a program).

C202 JMP $DE98   DrawChar

Outputs a character in the accumulator at the current cursor position without advancing to the next write position.

C205 JMP $9030   FollowChain

Gets the tracks and sectors used by a chained file and puts them in a buffer to which $08, $09 point (usually $8100). The track and sector of the first data block must be passed in $04 and $05.

C208 JMP $C96C   LoadFtFile

Loads a file according to the file type.

$0E,$0F   = pointer to desired filename

GEOS loads the various file types in different ways. The file type is located in the file entry at position 22. Data is loaded differently than accessories, for example.

C20B JMP $C9A7   FindFile

Searches in the directory and the border block (the sector where GEOS manages the file entries whose icons are placed on the border) for a file entry. This is then passed in $8400.

$0E,$0F   = pointer to the filename

The error message will be returned in the X register (X=0: no error, X=5: file not found).

If the entry was found in the border block, $886E = $FF, else $00.

C20E JMP $DBD0

C211 JMP $CA80

This is a load routine. A pointer to the complete file entry must be passed in $14,$15 (such as through FindFile $C20B through $8400).

This routine gets the necessary information from the INFO sector, loads the file to the location found there, and then starts it if it is a program.

C214 JMP $C2F3   EnterTurbo

Enables the fast disk routines.

C217 JMP $911B   LdDeskAcc

Completes load procedure for an accessory. It is automatically called when the name of an accessory is passed to $C208. This routine creates a SWAP file and loads and starts the accessory.

C21A JMP $C479

Subroutine for reading a sector.

C21D JMP $91C6   LdAppl

Complete load procedure for an application. It is automatically called if the filename of an application is passed to $C208.

C220 JMP $9269

Subroutine for writing a sector.

C223 JMP $92A4   Verify

Executes a verify.

C226 JMP $9733   DeleteFile

Deletes a file; puts the BAM in $8200 and deletes a file. A pointer in $14,$15 must point to the complete directory entry. The blocks belonging to the file will be released in the BAM.

C229 JMP $CAC7

Loads the INFO sector of a file to $8100 and gets the following data from it:

$04   = start track of the file
$05   = start sector of the file
$10,$11   = load address

The routine must be passed in $14,$15 the pointer to the complete directory entry. Normally this is in the first 30 bytes at $8400.

C22C JMP $CC54   EnterDeskTop

This routine must be executed in order to exit an application. It clears the screen and starts the procedure that loads and starts the deskTop.

C22F JMP $CCB3   EnterProg

This reset routine is subordinate to EnterDeskTop ($C22C). It starts a program. The start address must be passed in $10,$11. The screen will not be cleared.

C232 JMP $C34B   SleepTurbo

The speeder in the current disk drive (number in $8489) will be disabled. Only one disk drive may have an active speeder (bus problems). The speeder is reactivated with EnterTurbo.

C235 JMP $C387   ExitTurbo

Function similar to SleepTurbo ($C232). The difference is that on a subsequent EnterTurbo the speeder routines will be recopied into the disk drive, while after SleepTurbo the speeder is just turned back on.

C238 JMP $9721   DeleteFile

Deletes a file from the directory and releases the blocks it occupied. The routine must be passed in $02,$03, a pointer to the null-terminated filename.

C23B JMP $9067   FindFTypes

Gets the name of the files with a certain GEOS file type into a list. An additional selection is possible by specifying the desired TYPE as listed in the INFO sector.

Parameters:

$0E,$0F	= pointer to the free data area for the filenames found
$10	= desired GEOS file type (0-9)
$11	= maximum number of desired files
$16,$17	= a pointer to the desired TYPE entry of the file. If this is not needed, the pointer must point to $0000.

C23E JMP $918F   RstrAppl

This is an important routine for programming accessories. Since this file creates a SWAP file on the disk when it is loaded, the program must be put back in memory when the accessory is done (QUIT). To do this, this routine is passed to the main loop. The end of an accessory must always look like this:

```
LDX #$C2    address of this routine
LDY #$3E
STX $849B    pass as job
STY $849C
RTS          end of the accessory
```

C241 JMP $9CA7   EnterBasicPrg

Routine loads and starts a BASIC program from GEOS.

C244 JMP $97FA   FastDelFile

Similar to DeleteFile ($C238).

In addition to the filename, this routine must be passed a pointer ($08,$09) to a table which contains the tracks and sectors occupied by the file.

C247 JMP $C957   GetDirHead

Loads the BAM to $8200.

C24A JMP $92EB   PutDirHeae

Writes the BAM from $8200 to the disk.

C24D JMP $95C6   VAllocNeccBlocks

This routine differs from AllocNeccBlocks ($C1FC) only in a somewhat higher entry address. This allows the programmer to set the track and sector (in $08, $09) at which the search for free blocks will begin. The command is not necessary for normal applications.

C250 JMP $EF42   CoverRectangle

Saves a rectangle on the "invisible" second screen (opposite: RecoverRectangle = $C12D).

$06	= upper bound
$07	= lower bound
$08,$09	= left bound (LOW, HIGH)
$0A,$0B	= right bound (LOW, HIGH)
$12	= set bits on the right edge omitted
$13	= set bits on the left edge omitted

C253 JMP $EF36   DatCoverRectangle

Like CoverRectangle. The eight data must follow immediately after the call.

C256 JMP $F1BB   DoDlgBox

Creates a window for outputting messages. For more information, see *GEOS Inside and Out* from Abacus Software.

C259 JMP $9909   RenameFile

Allows files on the disk to be renamed.

$02,$03   = pointer to the current filename
$0C,$0D   = pointer to the new filename

C25C JMP $C4C1   InitTurboBus

Called after every action with the fast data bus and saves important parameters (IRQ, NMI, etc.). This command is not very interesting for programmers because it is called automatically.

C25F JMP $C548   ExitTurboBus

Opposite of InitTurboBus ($C25C). Is usually executed automatically.

C262 JMP $FA78   DShiftRight

The X register must point to a two-byte address on the zero-page which will be shifted Y times to the right.

C265 JMP $FB82   CopyFString

Copies a string (terminated with a zero) to a memory location.

X register:

points to a pointer in the zero-page which points to the string.

Y register:

points to a pointer in the zero-page which points to the destination address. A maximum of 256 bytes can be copied.

C268 JMP $FB84   CopyString

Operation and parameters like CopyFString. Multiple strings, stored in succession, can be copied simultaneously. The number of strings must be passed in the accumulator (each string must be null-terminated).

C26B JMP $FC44   CmpFString

Compares two strings to each other. Parameters like CopyFString. If the accumulator contains $00, both strings are equal. Otherwise, it contains a pointer to the differing element.

C26E JMP $FC46   CmpString

Like CmpFString. Multiple strings, stored in succession, can be compared. The number of strings to be compared must be passed in the accumulator.

Result is like CmpFString. The X register contains the number of the string in which the inequality occurred.

C271 JMP $CBF5   Reset

Subroutine for initialization of GEOS.

C274 JMP $9942   OpenRecordFile

Opens a VLIR file on the disk.

$02,$03 must contain a pointer to the filename.

C277 JMP $99C3   CloseRecord

Closes a VLIR file. The directory entry is also updated (date, time). No parameters required.

C27A JMP $9A12   NextRecord

Positions to the next record. Error message in the X register (X=0: no error, X=8: record does not exist).

C27D JMP $9A1B   PreviousRecord

Positions to the previous record. Error message in the X register (X=0: no error, X=8: the current record is the first).

C280 JMP $9A21   PointRecord

Positions to a certain record. The accumulator must contain the record number. Error message in the X register (X=0: no error, X=8: record does not exit).

C283 JMP $9A3A   DeleteRecord

Removes the current record from the list. The list moved up and the blocks occupied by the entry will be released. There are no direct parameters to pass.

C286 JMP $9A7E   InstRecord

Inserts a new record in front of the current record. This new one is then the current record.

C289 JMP $9A95   InstRecord+

Inserts a new record behind the current record. This becomes the current record.

C28C JMP $9AAF   ReadRecord

Reads the current record. The desired load address must be in $10,$11. $06,$07 must contain the maximum number of data desired.

C28F JMP $9AC6   WriteRecord

Stores the current record.

$06,$07   number of bytes
$10,$11   start address of the data

C292 JMP $965E

C295 JMP $99CC

Subroutine for CloseRec.

C298 JMP $91E7   GetPtrCurDkNm

Sets a pointer to the name of the current disk. The X register must point to a pointer in the zero page which points to the name.

C29B JMP $E17F   PromptOn

Enables the cursor.

C29E JMP $E1A3   PromptOff

Disables the cursor.

C2A1 JMP $9000   NewDisk

Initializes a new disk.

C2A4 JMP $CEBB

Subroutine of all Data routines.

C2A7 JMP $E53D   GetNextChar

Gets a character from the keyboard. If the buffer is empty, a zero will be returned.

C2AA JMP $DA6B

C2AD JMP $9702   TestBlock

Tests if a given block is already allocated in the BAM. $0E,$0F must contain the track and sector of the block. If the value returned in the accumulator is zero, the block is allocated.

C2B0 JMP $C934

C2B3 JMP $C934

C2B6 JMP $9C44   GetByte

Gets a byte from the current record and returns it in the accumulator. Error message in the X register (X=0: no error, X=12: end of file).

C2B9 (three NOP's)

C2BC JMP $9204

C2BF JMP $F32F   RstrFrmDialogue

Automatically called when clearing a window. The jobs frozen and saved during window output will be reactivated (pull-down menus, etc.).

C2C2 JMP $FCBD

Outputs "System Error in $xxxx". The routine pulls the appropriate address off the stack and converts it to a four-digit hex number in ASCII.

C2C5 JMP $DA66

5.7 Variables used by the GEOS KERNAL

If you want to understand the operation of the GEOS KERNAL from a disassembled listing, and you use our jump table, you will come across memory locations again and again which the KERNAL uses. It would be a lot easier if you could look up the memory locations and their functions in some table. In addition, you have to know about the memory location used by GEOS when you are writing your own programs. Therefore, we have gathered all of the memory locations and the meanings we are aware of into a table.

$02,$03	General-purpose pointer for parameters
$04,$05	Usually the track and sector for disk operations: $05 also contains the line number for text output
$06,$07	Length of a file for graphics: Y position
$08,$09	Usually the left edge of a graphic output
$0A,$0B	Usually the right edge of a graphic output
$0C,$0D	The pointer to screen 1 for graphic output
$0E,$0F	Like $0C, $0D for screen 2
$10,$11	Often contains a start address in RAM for disk operations
$16	Current directory page
$18,$19	X position for text output
$2E	Status byte for the type style (bold, reverse, outline, etc.)
$2F	Bit 7 = 1 : visible screen ($A000) selected Bit 6 = 1 : invisible screen ($6000) (also bit 7 and 8 = 1)

$0030	Mouse flag
	        Bit 7 = 0 : mouse off
	        Bit 6 = 1 : area test on

$0039	Joystick flag
	        Bit 7 = 0 : no characters in the keyboard buffer
	        Bit 6 = 1 : joystick movement unchanged
	        Bit 5 = 1 : fire button pressed

$003A	Current mouse X position (LOW)
$003B	Current mouse X position (HIGH)
$003C	Current mouse Y position
$008E	Current value for bus operations
$8400	The directory entry is usually located here (30 bytes).
$8400	DOS file type
$8401	Track of the data
$8402	Sector of the data
$8403- $8412	Name of the file, padded with $A0
$8413	Track of the INFO block
$8414	Sector of the INFO block
$8415	Structure (0 SEQ, 1 VLIR)

$8416	GEOS file type (0-9)
$8417	Year
$8418	Month
$8419	Day
$841A	Hour
$841B	Minute
$841C/D	Length of the file in blocks
$8489	Number of the current output device
$848A	Table of status values of the output device (speeder status)
$848B	GEOS FORMAT V1.0? 0 no, $FF yes
$8496	Current record number
$8497	Number of records present
$8498	Flag determines whether an update is necessary when closing a record
$84AF/B0	Indirect entry address on BRK: Normally this vector points to the output routine for the message "System error near $XXXX"
$84A1/2	Job address: fire button pressed In this vector an address can be entered to which execution will branch when the fire button is pressed
$84A3/4	Job address for keyboard input. Here an address can be entered to which execution will branch if a character is entered: The character will be passed to this routine in the accumulator
$84A5/6	Job address: joystick moved
$84A7/8	Job address: mouse left the area ($84B6 not 0)
$84B4	Cursor flag: Bit 7 = 1 : cursor on Bit 6 = 1 : cursor bright Bit 6 = 0 : cursor dark
$84B5	Flag for: symbol not invertable (not zero)
$84B6	Mouse status register Bit 7 = 1 : contact with upper bound (area 1) Bit 6 = 1 : contact with lower bound Bit 5 = 1 : contact with left bound Bit 4 = 1 : contact with right bound Bit 3 = 1 : mouse outside the marked second area
$84B7	Pull-down menu: current subtable
$84B8	Upper bound for mouse
$84B9	Lower bound for mouse
$84BA/B	Left bound for mouse
$84BC/D	Right bound for mouse
$84C1	Upper bound for mouse (2nd area)
$84C2	Lower bound for mouse (2nd area)

$84C3/4	Left bound for mouse (2nd area)
$8C5/6	Right bound for mouse (2nd area)
$8501	Maximum mouse speed
$8502	Minimum mouse speed
$8503	Acceleration
$8504	Current character, read from keyboard buffer
$8505	Code for current fire-button status: 　　　Bit 7 = 0 : button pressed 　　　Bit 7 = 1 : button not pressed
$8506	Code for current joystick position: 　　　$00: right 　　　$02: up 　　　$04: left 　　　$06: down 　　　$FF: center position
$8507	Current mouse speed
$850A/B	Count register for IRQ
$8516	Year *System time:*
$8517	Month *System time:*
$8518	Day *System time:*
$8519	Hour *System time:*
$851A	Minute *System time:*
$851B	Second *System time:*
$86C0	Pull-down menu status byte: 　　　Bit 7 = 1 : menu vertical 　　　Bit 7 = 0 : menu horizontal 　　　Bit 6 = 0 : mouse bounds set on screen 　　　Bit 6 = 1 : old mouse bounds
$85C1	Upper edge of menu
$86C2	Lower edge of menu
$86C3/4	Left edge of menu
$86C5/6	Right edge of menu
$86C7/8	Address of the subtable
$86F1	Timer values (counting/active) for buffer 1 (LOW/HIGH)
$8719	from here on: status bytes of the program jobs
$872D	from here on: job addresses of the program jobs
$8755	Timer values (for init/new entry) of program jobs
$877D	Number of program jobs
$877E	from here on: timer values for delay (LOW)
$8793	from here on: timer values for delay (HIGH)
$87A7	Addresses of the delay jobs (LOW)
$87BB	Addresses of the delay jobs (HIGH)
$87D7	Read pointer for the keyboard buffer
$87D8	Write pointer for the keyboard buffer

$87D9	Repeat flag for keyboard input
$87DA-$87E9	Keyboard buffer
$87EA	Current character by input through IRQ in keyboard buffer
$880B	Temporary storage for accumulator on IRQ
$8850/1	Accessory return
$8852	Accessory return: STACK
$8853/4	Window return
$8855	Window return: STACK
$8860	Temporary storage for $01
$8861	Sprite status: eight bits for sprite on/off
$8863/4	Entry address for floppy RAM
$8865	Data (track) for floppy RAM
$8866	Data (sector) for floppy RAM
$886E	Flag for: read file from border ($FF, else $00)
$886F	Track: entry found in DIR
$8870	Sector: entry found in DIR
$8871/2	File entry found, points to location in buffer ($8000)
$8873	Track of the desired VLIR file
$8874	Sector of the desired VLIR file
$8875	Number of read attempts

Appendix

Error messages

When you are working with GEOS and an error occurs in connection with the disk drive, a message explaining the error will usually be printed (such as "Operation canceled due to disk error: Missing or unformatted disk").

But there are a number of errors for which only a number appears (such as "I:8"). The following list describes what these errors are:

I:2   Corresponds roughly to the error message in the old DOS "Illegal track and sector". A track less than one or greater than 35 occurred during a read or write operation.

I:6   An attempt was made to release a block which is already free.

I:7   Occurs if an invalid track (=0) is given for the pointer to the link block of a VLIR file.

I:8   An attempt was made to read the previous record in a VLIR file even though there is none.

I:9   An attempt was made to put more records than possible in a VLIR file.

I:A   An attempt was made to open a VLIR file although the file does not have VLIR structure or does not have the format USR.

I:B   During a read operation there were more data present than were read (not necessarily an error!). Marker: end of file.

Index

263

Optional Diskette

GEOS TRICKS and Tips
Optional diskette

For your convenience, the program listings contained in this book are available on an 1541 formatted floppy disk. You should order the diskette if you want to use the programs, but don't want to type them in from the listings in the book.

All programs on the diskette have been fully tested. You can change the programs for your particular needs. The diskette is available for $14.95 plus $2.00 ($5.00 foreign) for postage and handling.

When ordering, please give your name and shipping address. Enclose a check, money order or credit card information. Mail your order to:

Abacus Software
P.O. Box 7219
Grand Rapids, MI 49510

Or for *fast* service, call **(616) 241-5510.**

SpeedTerm

Terminal Software for the C-64 and C-128

SpeedTerm is a fast and easy-to-use terminal communication package for the Commodore 64 and 128 owner. You can use **SpeedTerm** to communicate with hundreds of thousands of computers users worldwide or dozens of exciting online services such as CompuServ, Genie, Delphi, the Source and others.

Even though **SpeedTerm**is simple in design, it packs numerous features that aren't found in others termnal packages. For instance, **SpeedTerm** supports both Xmodem and Punter file transfer protocols so large files can be uploaded and down loaded without error.
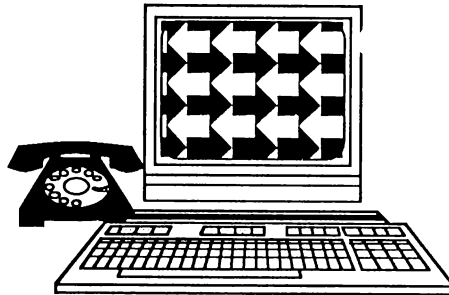
Other features of **SpeedTerm**:

* combined C-64 and C-128 version on one diskette
* manages large 24K (45K for C-128 version) capture buffer for recording long sessions
* permits flexible user defined function keys
* supports VT52 terminal emulation
* uses Xmodem and Punter protocols for error-free file transfer
* has powerful command mode with 30+ commands
* works with most popular modems
* uses either 40 or 80 (C-128 version) column monitors
* includes 70+ page manual with easy-to-understand tutorial

Suggested retail price: $39.95
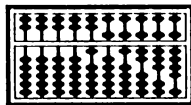
ABACUS SOFTWARE
P. O. Box 7219
Grand Rapids, MI 49510
Phone: (616) 241-5510

COBOL Language
for the C-64 or the C-128

COBOL is the most widely used commercial programming language today. Now you can learn the COBOL language using your Commodore 64 or Commodore 128 home computer. The COBOL language uses English-like sentences. This makes it an easy to learn language. And since **COBOL-64** and **COBOL-128** are designed with easy of use in mind, it's perfect for the beginner.

Our COBOL software includes a syntax checking editor, a compiler, an interpreter and symbolic debugging aids. So you'll be able to write and test your COBOL programs very quickly.

Complete with Editor, Interpreter and Debugger
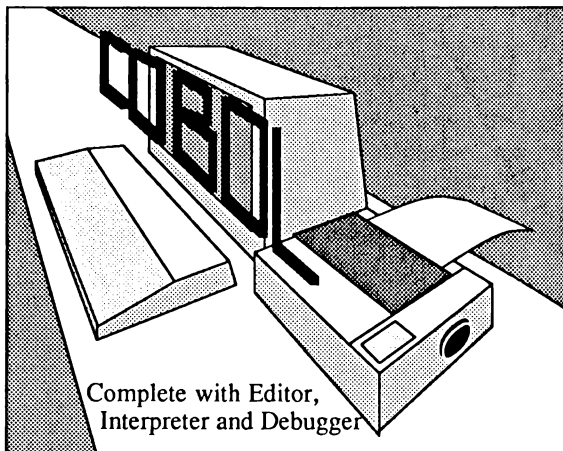
Features of **COBOL-64** and **COBOL-128**:

* includes integrated editor for creating COBOL source
* fast compiler/interpreter to transform source into excutable program
* features debugging tools: breakpoint, trace, single step.
* supports subset of ANSI COBOL '74
* includes sample programs demonstrating file handling
* with 150-page manual

Advantages of using COBOL:

* most widely used commercial programming language
* English-like syntax eases learning curve
* language common to many different computers

Hardware requirements:
Commodore 64 with 1541 or 1571 disk drive.
Commodore 128 with 1541 or 1571 disk drive (supports 40 or 80 column monitor)
Works with most printers.

Suggested retail price: C-64 version - $39.95
                               C-128 version - $39.95
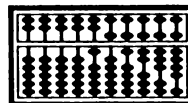
ABACUS SOFTV
P. O. Box 7219
Grand Rapids, MI
Phone: (616) 241

Expand your vocabulary—
Take on a new language

GEOS *Tricks & Tips*

GEOS Tricks & Tips has something for everyone who uses GEOS, whether you're a beginner, intermediate or expert. This companion volume to the book **GEOS Inside & Out** is an invaluable resource for information about GEOS shortcuts, applications and internals. Contains listings for many utility programs that can be used with GEOS. **GEOS Tricks & Tips** includes:

- Over 50 tricks and tips that you can use everyday with GEOS: setting up form letters with geoWrite, writing your own error messages, fast document movement in geoWrite, alter the notes icon, and many more
- Use the fast printing utilities to print the Notepad and geoWrite
- Modify the GEOS program operation to suit your preferences
- **CONVERTER**, a utility program, converts any word-processor files into geoWrite format
- **FONTEDIT**, modify existing GEOS fonts or create your own new fonts with this useful utility program
- **EDMON**, a GEOS-based machine language monitor, lets you discover the internals of the GEOS

Optional program diskette available for this title. Contains all of the programs listed in the book—complete, error-free and ready to run.

About the authors:
Manfred Tornsdorf and Ruediger Kerkloh previously combined their C-64 and GEOS talents to produce their best-selling **GEOS Inside & Out**. Tornsdorf is a physics student with an electronics background; Kerkloh is an electronics student with computer expertise.

ISBN 0-916439-86-0

Abacus

A Data Becker book